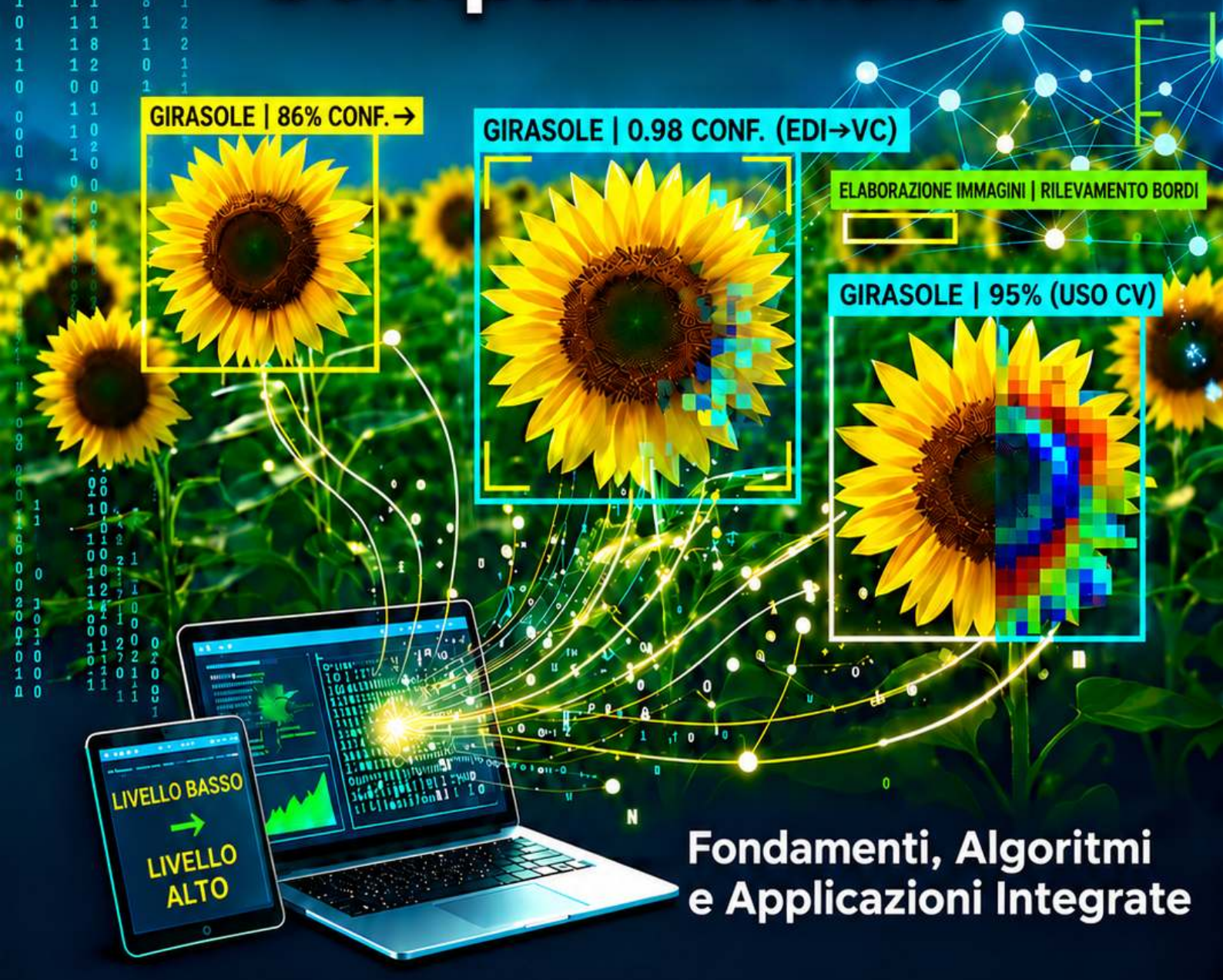
EDI & VC

Elaborazione Digitale delle Immagini e Visione Computazionale



Fondamenti, Algoritmi e Applicazioni Integrate

AUTORE: FRANCISCO DE ASSIS ZAMPIROLLI

Elaborazione Digitale delle Immagini e Visione Artificiale

Libro interattivo con Python

Francisco de Assis Zampirolli

Universidade Federal do ABC

20 settembre 2026

© 2026 Francisco de Assis Zampirolli da Universidade Federal do ABC (UFABC).
Todos os direitos reservados.

Este livro está sob a Licença:
Creative Commons Attribution-ShareAlike 4.0 International License
Detalhes no endereço: creativecommons.org/licenses/by-sa/4.0

Projeto gráfico: Francisco de Assis Zampirolli
Capa: Francisco de Assis Zampirolli, com suporte de IA (Gemini e ChatGPT)

CATALOGAÇÃO NA FONTE
SISTEMA DE BIBLIOTECAS DA UNIVERSIDADE FEDERAL DO ABC

[CÓDIGO CUTTER] Zampirolli, Francisco de Assis
Processamento Digital de Imagens e Visão Computacional / Francisco de Assis Zampirolli – Santo André, SP : Edição do Autor, 2026.

[xx], [NNN] p. : il.

ISBN: [ISBN A DEFINIR] (1ª Edição)

1. Processamento Digital de Imagens. 2. Visão Computacional. 3. Morfologia Matemática. 4. Python. 5. Correção Automática. 6. Moodle. I. Título.

CDD [XX] ed. – [CÓDIGO CDD]

Indice

EDI e VA — Python 1
Organizzazione . . . 1

Ficha Catalográfica 3

Prefazione 5
Contesto della prima edizione . . . 5
Come viene prodotto questo libro . . . 6
Uso di strumenti di Intelligenza Artificiale . . . 7
La libreria `morph.py` . . . 8
Esercizi di Programmazione (EP) e Validazione Automatica . . . 9
Codice aperto . . . 9
Prima di iniziare: *Notebook* in Python . . . 10
Guida per il Docente: Pubblicazione degli EP su Moodle . . . 10
Integrazione con Moodle (VPL) . . . 10
Come Pubblicare su Moodle . . . 11

I Parte I — Fondamenti di EAI 1

1 Fondamenti e Primi Passi 3
1.1 Obiettivi . . . 3
1.2 Prima di iniziare: *Notebooks* interattivi . . . 3
1.3 Fondamenti . . . 3
1.3.1 👁️ Visione Computazionale . . . 4
1.3.2 🖼️ Elaborazione Digitale delle Immagini (PDI) . . . 4
1.4 Fasi del PDI . . . 5
1.5 Formazione dell'Immagine e Spettro . . . 5
1.6 Cos'è un'Immagine Digitale? . . . 7
Rappresentazione Matematica . . . 7
1.7 Configurazione dell'Ambiente . . . 9
1.8 Versioni di `morph.py` . . . 9
1.9 Fondamenti di Matrici — attenzione alla copia di riferimenti . . . 10
1.10 Lettura e Visualizzazione delle Immagini . . . 11
Alternativa: *Download* dell'Immagine in Archiviazione Locale . . . 12
1.11 Conversione di Tipi e Soglia . . . 13
Conversione in Scala di Grigi . . . 13
Soglia (*Thresholding*) . . . 13
Esempio pratico di conversione e soglia . . . 14
1.12 Limiarizzazione con il metodo di Otsu . . . 14
1.13 Accesso ai Pixel . . . 15
1.14 Riassunto . . . 17
1.15 🤖 Uso del Gemini Notebook come Tutor Complementare . . . 18
Funzionalità Disponibili sulla Piattaforma . . . 18
1.16 Elenco di Esercizi . . . 19
Riferimenti del Capitolo . . . 19

1.17 💻 **Parte pratica con esercizi di programmazione** . . . 19
🎯 Obiettivo di questo Quaderno . . . 19
§ Istruzioni Passo a Passo . . . 20
⚠️ Importante: Regole e Buone Pratiche . . . 21
1.17.1 EP01_01 📏 Tre metriche di distanza nella PDA (Processamento Digitale delle Immagini) 21
1.17.2 EP01_02 📊 Prestazioni Predittive — Metriche di ML in VC . . . 29
1.17.3 EP01_03 ↗ *Mean Average Precision* (mAP) — Curva Precisione-Sensibilità . . . 30
1.17.4 EP01_04 🖼️ Lettura e Informazioni di un'Immagine in Matrice . . . 34
1.17.5 EP01_05 🔄 Negativo di un'Immagine in Ton di Grigio . . . 35
1.17.6 EP01_06 🎨 Conversione RGB → Toni di Grigio (ITU-R BT.601) . . . 37
1.17.7 EP01_07 ⚫ Soglia Manuale: Immagine Binaria . . . 39
1.17.8 EP01_08 🎨 Rimappatura per Intervallo di Intensità . . . 40
1.17.9 EP01_09 🏁 Schema a Scacchiera: Matrice a Quadretti . . . 41
1.17.10 EP01_10 📄 Metadati: Lettura di File PGM . . . 43
1.17.11 EP01_11 ↗ Analisi di Vicinato: Filtro di Massimo 1D . . . 45

2 Dalla Cattura al Pixel - Campionamento, Quantizzazione e Connettività **49**
2.1 Obiettivi . . . 49
2.2 L'Occhio e la Fotocamera - Elementi della Percezione Visiva . . . 49
2.2.1 Visione umana . . . 49
2.2.2 Sensori digitali . . . 49
2.3 Illusioni Ottiche: Le Sfide della Percezione Visiva . . . 50
2.3.1 Ambiguità e Contesto . . . 50
2.3.2 Luminosità e Contrasto Locale . . . 51
2.3.3 Geometria e Prospettiva . . . 51
2.4 Il Modello Matematico della Formazione dell'Immagine . . . 51
2.4.1 Rappresentazione Digitale e Canali Spettrali . . . 52
2.4.2 Preparazione dell'Ambiente Pratico . . . 53
2.4.3 Implementazione della Lettura delle Immagini in `morph.py` . . . 53
2.4.4 RGB e RGBA: Esempio Pratico con l'Immagine Mandrill . . . 54
2.5 Digitalizzazione: Campionamento e Quantizzazione . . . 55
2.5.1 Campionamento - Discretizzazione dello spazio . . . 55
2.5.2 Quantizzazione - Discretizzazione dell'intensità . . . 55
2.5.3 Effetti del campionamento e della quantizzazione . . . 56
2.5.4 Analisi Tecnica . . . 58
2.6 Relazioni tra Pixel - Topologia dell'Immagine . . . 59
2.6.1 Vicinato . . . 59
2.6.2 Adiacenza, connettività e percorsi . . . 59
2.6.3 Distanze tra pixel . . . 60
2.7 Archiviazione delle Immagini . . . 60
2.7.1 Esempio: Estrazione di Metadati e Localizzazione GPS . . . 61
2.7.2 Perché questa separazione è importante? . . . 63
2.8 Trasformazioni Geometriche Fondamentali . . . 63
2.8.1 Traslazione . . . 64
2.8.2 Rotazione . . . 65
2.8.3 Scala (Ridimensionamento) . . . 66
2.8.4 Taglio (*Shear*) . . . 67
2.9 Riassunto . . . 68
2.10 🤖 Uso del Gemini Notebook come Tutor Complementare . . . 68
2.11 Elenco di Esercizi . . . 69
Riferimenti del Capitolo . . . 69
2.12 💻 **Parte pratica con esercizi di programmazione** . . . 69
🎯 Obiettivo di questo Quaderno . . . 69
2.12.1 EP02_01 ⊙ Regolazione di Luminosità e Contrasto . . . 70

2.12.2 EP02_02 🔬 Sottocampionamento Spaziale . . . 72
2.12.3 EP02_03 🎨 Quantizzazione dei Livelli di Grigio . . . 73
2.12.4 EP02_04 📐 Trasformata di Distanza in Immagine Binaria . . . 75
2.12.5 EP02_05 Traslazione dell'Immagine . . . 77
2.12.6 EP02_06 🔄 Rotazione dell'Immagine . . . 80
2.12.7 EP02_07 🔍 Ridimensionamento (Scala) . . . 82
2.12.8 EP02_08 🔀 Taglio (Shear) . . . 84
2.12.9 EP02_09 🧩 Trasformazione Affine Generica . . . 85
2.12.10 EP02_10 🎯 Correzione della Prospettiva (Omografia) . . . 87
2.12.11 EP02_11 🏆 Correzione della Prospettiva (Omografia) su Immagine Reale . . . 91

3 Operazioni Spaziali: Intensità, Istogramma e Filtraggio 97
3.1 Obiettivi . . . 97
3.2 Operazioni a Livello di Intensità . . . 97
3.2.1 Preparazione dell'Ambiente Pratico . . . 98
3.2.2 Operazioni Aritmetiche . . . 99
3.2.3 Miscelazione Ponderata (*Alpha Blending*) . . . 100
3.2.4 Operazioni Logiche e Maschere Bit a Bit . . . 101
3.3 Istogramma delle Immagini . . . 102
3.3.1 Equalizzazione dell'Istogramma . . . 103
3.3.2 Specifica dell'istogramma . . . 107
3.4 Fondamenti Spaziali: Intorno, Convoluzione e *Kernel* . . . 108
3.4.1 Intorno . . . 109
3.4.2 Trattamento dei Bordi . . . 109
3.4.3 Correlazione vs. Convoluzione . . . 111
3.4.4 Il Ruolo del *Kernel* . . . 112
3.4.5 Esempio Numerico: Correlazione Passo per Passo . . . 115
3.5 Filtraggio Spaziale di Levigatura . . . 116
3.5.1 Filtro a Media (*Box Filter*) . . . 116
3.5.2 Filtro Gaussiano . . . 117
3.6 Filtraggio Spaziale di Enfasi . . . 119
3.6.1 Laplaciano . . . 119
3.6.2 Operatore di Sobel . . . 123
3.6.3 Operatore di Prewitt . . . 125
3.6.4 *Unsharp Masking* (USM) . . . 126
3.6.5 Rilevatore di Canny . . . 128
3.7 Filtri d'Ordine: Filtro Mediano . . . 129
3.7.1 Rumore Impulsivo: Sale e Pepe . . . 129
3.7.2 Filtro della Mediana . . . 130
3.8 Applicazione Pratica: Pre-elaborazione per la Segmentazione . . . 133
3.9 Riassunto . . . 134
3.10 🤖 Uso di Gemini Notebook come Tutor Complementare . . . 134
3.11 Lista di Esercizi . . . 135
Riferimenti del Capitolo . . . 136
3.12 💻 **Parte pratica con esercizi di programmazione** . . . 136
🎯 Obiettivo di questo Quaderno . . . 136
3.12.1 EP03_01 ➕ Addizione Saturata di una Costante . . . 137
3.12.2 EP03_02 🔀 Alpha *Blending* di Due Immagini . . . 138
3.12.3 EP03_03 🎭 Inversione dell'Immagine (Negativo Fotografico) . . . 140
3.12.4 EP03_04 📊 Equalizzazione dell'istogramma (L bit) . . . 141
3.12.5 EP03_05 🔲 Applicazione della Maschera AND Binaria . . . 143
3.12.6 EP03_06 Filtro di Media con *Kernel* N×N . . . 145
3.12.7 EP03_07 🔍 Operatore Laplaciano (w4) per l'Enfatizzazione dei Bordi . . . 149
3.12.8 EP03_08 🧭 Gradiente di Sobel: Gx e Gy . . . 151

3.12.9 EP03_09 📡 Filtro della Mediana 3×3 . . . 152
3.12.10 EP03_10 ✨ *Unsharp Masking* (USM) . . . 154

4 Morfologia Matematica e Segmentazione delle Immagini 159
4.1 Obiettivi . . . 159
4.2 Soglie . . . 160
4.2.1 Immagine di Monete . . . 161
4.2.2 Pre-processamento per l'Otsu . . . 161
4.2.3 Risultato: CLAHE come Miglior Preprocessamento . . . 164
4.3 Morfologia Matematica . . . 165
4.3.1 Erosione e Dilatazione . . . 165
4.3.2 Apertura e Chiusura . . . 173
4.3.3 Operatori Geodetici . . . 175
4.3.4 Ricostruzione Morfologica . . . 179
4.3.5 Riempimento di Buchi e Rimozione dei Bordi . . . 181
4.3.6 *Pipeline* di Pulizia Binaria con CLAHE . . . 184
4.3.7 Morfologia in Toni di Grigio . . . 185
4.4 Segmentazione delle Immagini: Fondamenti e Tassonomia . . . 188
4.4.1 Etichettatura . . . 189
4.4.2 Trasformata della Distanza . . . 192
4.4.3 Trasformata della Distanza Euclidea in quattro passaggi . . . 196
4.4.4 Segmentazione per *Watershed* . . . 198
4.5 Estrazione di Componenti e Descrittori di Forma . . . 203
4.5.1 Etichettatura e Statistiche dei Componenti . . . 204
4.5.2 Descrittori di Forma . . . 205
4.5.3 Collegamento con la Rilevazione Moderna degli Oggetti . . . 207
4.6 Riassunto . . . 210
4.7 🤖 Uso del Gemini Notebook come Tutor Complementare . . . 210
4.8 Elenco di Esercizi . . . 211
Riferimenti del Capitolo . . . 212
4.9 💻 **Parte Pratica con Esercizi di Programmazione** . . . 212
🎯 Obiettivo di questo Quaderno . . . 212
4.9.1 EP04_01 Soglia Globale con Soglia Fissa . . . 213
4.9.2 EP04_02 📊 Soglia Automatica di Otsu . . . 215
4.9.3 EP04_03 🌱 Dilatazione Binaria Piana (mm.dil0) . . . 217
4.9.4 EP04_04 🪨 Erosione Binaria Piatta (mm.ero0) . . . 218
4.9.5 EP04_05 🧹 Apertura Morfologica (Rimozione del Rumore) . . . 222
4.9.6 EP04_06 🧩 Chiusura Morfologica (Riempimento delle Lacune) . . . 224
4.9.7 EP04_07 Dilatazione ed Erosione con Pesi (mm.dil1 / mm.ero1) . . . 226
4.9.8 EP04_08 🌋 Gradiente Morfologico, Top-hat e Black-hat . . . 227
4.9.9 EP04_09 Trasformata della Distanza e il "Nucleo" dell'Oggetto . . . 230
4.9.10 EP04_10 🪙 Separazione dei *Blob*, Etichettatura e Descrittori . . . 233

5 Trasformate e Compressione 235
5.1 Obiettivi . . . 235
5.2 Configurazione dell'Ambiente . . . 236
5.3 Trasformata di Fourier Discreta 2D . . . 236
5.3.1 Simulatore: Ricostruire Segnali con Sinusoidi . . . 236
5.3.2 Interpretazione dello spettro di frequenza . . . 238
5.3.3 L'Esperimento della Griglia: Costruire un'Immagine da un Singolo Coefficiente . . . 238
5.3.4 Definizione Matematica . . . 240
5.3.5 Magnitudine e Fase . . . 240
5.3.6 Cosa trasportano l'ampiezza e la fase? . . . 242
5.4 Teorema della Convoluzione e Strategie di Filtraggio . . . 243

5.4.1 *Kernel* Separabile vs. Non Separabile . . . 243
5.4.2 Analisi dell'Efficienza Computazionale . . . 244
5.4.3 Discussione dei risultati sperimentali . . . 244
5.5 Filtri nel Dominio delle Frequenze . . . 248
5.5.1 Filtri Passa-Basso . . . 249
5.5.2 Filtri Passa-Alto e Passa-Banda . . . 249
5.5.3 Rimozione del Rumore Periodico . . . 253
📌 Sintesi — Filtri Spettrali . . . 254
5.6 *Wavelet* e Multirisoluzione . . . 255
5.6.1 Il Limite della Trasformata di Fourier: localizzazione spaziale . . . 255
5.6.2 Trasformata *Wavelet* Discreta 2D . . . 256
5.6.3 Famiglie di *Wavelet* . . . 256
5.6.4 Analisi Multirisoluzione con la DWT 2D . . . 259
Sintesi — Fourier vs. *Wavelet*: quando utilizzare ciascun approccio? . . . 265
5.7 Compressione delle Immagini . . . 266
5.7.1 Tassonomia delle Ridondanze . . . 266
5.7.2 Trasformata del Coseno Discreta (DCT-II 2D) . . . 267
5.7.3 Le Funzioni di Base della DCT . . . 267
5.7.4 Concentrazione di Energia e Ricostruzione Progressiva . . . 268
5.7.5 Il *Pipeline* di Compressione JPEG . . . 270
5.7.6 Simulatore Interattivo: Quantizzazione DCT . . . 273
5.8 Confronto dei Formati Immagine . . . 274
5.8.1 Caratteristiche dei Formati . . . 274
5.8.2 Metriche di Valutazione della Qualità . . . 274
5.8.3 Ispezione Visiva: Natura degli Artefatti di Compressione . . . 275
5.8.4 Valutazione Quantitativa e Spaziale della Compressione . . . 276
Sintesi — Compressione JPEG . . . 281
5.9 Applicazione Pratica: Rimozione del Rumore mediante Filtraggio Ibrido . . . 282
5.9.1 Analisi delle Prestazioni e Conclusione del Capitolo . . . 282
5.10 Riepilogo del Capitolo . . . 284
Fondamenti Essenziali . . . 284
5.11 🤖 Uso del Gemini Notebook come Tutore Complementare . . . 286
5.12 Elenco di Esercizi . . . 286
Riferimenti del Capitolo . . . 287
5.13 💻 **Parte Pratica con Esercizi di Programmazione** . . . 288
🎯 Obiettivo di questo Quaderno . . . 288
5.13.1 EP05_01 🟢 Filtro Passa-Basso Ideale per Distanza nello Spettro . . . 289
5.13.2 EP05_02 🟡 Filtro *Notch*: Rimozione dei Picchi Periodici . . . 290
5.13.3 EP05_03 🟠 Quantizzazione DCT: la Vera Fonte di Compressione . . . 292
5.13.4 EP05_04 🔴 Implementazione della DFT 2D a partire dalla definizione . . . 295
5.13.5 EP05_05 🏆 *Pipeline* JPEG Completo: DCT, Quantizzazione e Ricostruzione . . . 297

II Parte II — Visione Artificiale **301**

6 Analisi dei Documenti e Ispezione Industriale **303**
6.1 Obiettivi del Capitolo . . . 303
6.2 Configurazione dell'Ambiente . . . 304
6.3 Banche di Immagini per la Sperimentazione . . . 305
6.3.1 Immagini Pubbliche con `skimage.data` . . . 306
6.4 Normalizzazione del Fondo e Equalizzazione Locale del Contrasto . . . 308
6.5 Riconoscimento Ottico dei Caratteri (OCR) . . . 310
6.6 Traduzione Automatica del Testo Riconosciuto . . . 312
6.7 Fondamenti di OMR e Ispezione Industriale . . . 314

6.8 Progetti Pratici: Costruzione di un *Pipeline* di Analisi Documentale . . . 314
6.8.1 Allineamento Automatico dei Documenti (*OCR/OMR Pre-processing*) . . . 314
6.8.2 Algoritmo di Rettifica dell'Inclinazione (*Deskew*) . . . 316
6.8.3 Modellazione Matematica . . . 316
6.8.4 Limiti Pratici . . . 320
6.8.5 Rilevamento dei Bordi e dei Contorni . . . 320
6.8.6 Isolamento, Segmentazione e Decodifica del *QRCode* . . . 324
6.8.7 Decodifica dei Codici a Barre . . . 326
6.9 Il MCTest come Caso di Studio: dal Prototipo al Sistema in Produzione . . . 329
6.9.1 Ottenimento e Preparazione del Modulo . . . 329
6.9.2 Estrazione dell'Area delle Risposte . . . 330
6.9.3 Segmentazione del *QRCode* . . . 332
6.9.4 Localizzazione dei Riquadri delle Risposte . . . 334
6.9.5 Lettura Automatica delle Risposte . . . 334
6.10 Ispezione Industriale Automatizzata . . . 336
6.10.1 Riferimenti e *Dataset* Pubblici . . . 337
6.10.2 Rilevamento di Difetti tramite Analisi della Texture . . . 338
6.11 Riassunto . . . 341
Prossimi Passi . . . 342
6.12 🤖 Uso del Gemini Notebook come Tutor Complementare . . . 342
6.13 Lista di Esercizi . . . 343
Riferimenti del Capitolo . . . 344
6.14 💻 Parte Pratica con Esercizi di Programmazione . . . 344
Legenda della Difficoltà . . . 345
🎯 Obiettivo di questo Quaderno . . . 345
6.14.1 EP06_01 🟢 Valutazione della Segmentazione tramite IoU (*Intersection over Union*) . 346
6.14.2 EP06_02 🟢 Filtro di Marcatori per Circolarità . . . 348
6.14.3 EP06_03 🟡 Classificazione delle Marcature nei Fogli di Risposta (OMR) . . . 350
6.14.4 EP06_04 🟡 Stimatore dell'Inclinazione tramite Mediana Angolare (*Deskew*) . . . 352
6.14.5 EP06_05 🟠 Normalizzazione del fondo per divisione (Correzione dell'illuminazione) . 354
6.14.6 EP06_06 🔴 Mappa di Varianza Locale per il Rilevamento di Trama . . . 356
6.14.7 EP06_07 🟣 *Pipeline* di Ispezione Industriale: Registrazione per Traslazione e Sottrazione 358
6.14.8 EP06_08 ⚫ Segmentazione e Decodifica Reale di *QRCode* con OpenCV . . . 360

7 Classificazione delle Immagini e Riconoscimento di Pattern 365
7.1 Obiettivi del Capitolo . . . 365
7.2 Configurazione dell'Ambiente . . . 365
7.3 Un Problema Concreto: Classificazione della Frutta . . . 367
7.4 Panoramica del Capitolo: Il *Pipeline* Classico di Classificazione delle Immagini . . . 369
7.5 Fondamenti del Riconoscimento di Pattern . . . 369
7.6 Estrazione di Descrittori Classici . . . 370
7.6.1 *Local Binary Patterns* (LBP) . . . 371
7.6.2 *Histogram of Oriented Gradients* (HOG) . . . 373
7.6.3 L'Impatto della Scala e la Normalizzazione delle Caratteristiche . . . 375
7.7 📌 Mappa Concettuale . . . 377
7.8 Descrittori nella Pratica . . . 377
7.9 Un Problema Concreto: Simulando Descrittori di Frutta . . . 382
7.10 Classificatore k-NN: Come Funziona Internamente . . . 385
7.10.1 Metrica di Distanza . . . 386
7.10.2 Regola di Decisione . . . 386
7.10.3 Il Ruolo del Parametro k . . . 388
7.11 Progetto Pratico 1: Classificazione di Cifre Scritte a Mano con k-NN . . . 389
7.11.1 Classificazione con Vettori di Intensità . . . 390
7.11.2 Classificazione con descrittori HOG . . . 391

7.12 Valutazione dei Classificatori . . . 393
7.12.1 Scelta di k tramite Validazione Incrociata . . . 393
7.12.2 Il Compromesso tra Bias e Varianza: Diagnosi di *Overfitting* e *Underfitting* . . . 394
7.13 Progetto Pratico 2: Confronto di Descrittori per la Classificazione di Trame . . . 396
7.13.1 *Pipeline* di estrazione delle caratteristiche e valutazione comparativa . . . 398
7.13.2 Diagnóstico Fino del Classificatore: Precisione, Recall e F1-Score . . . 401
7.14 Limiti dei Descrittori Artigianali . . . 403
7.15 Riassunto . . . 404
7.16 🤖 Uso del Gemini Notebook come Tutor . . . 404
7.17 Lista di Esercizi . . . 404
Riferimenti del Capitolo . . . 405
7.18 💻 **Parte Pratica con Esercizi di Programmazione** . . . 406
🎯 Obiettivo di questo Quaderno . . . 406
Download . . . 406
Riepilogo dei Metodi di `morph.py` (Cap. 7) . . . 407
7.18.1 EP07_01 🟢 Classificatore k-NN Passo dopo Passo . . . 407
7.18.2 EP07_02 🟡 Normalizzazione *Z-score* e Robustezza del k-NN a Scale Diverse . . . 409
7.18.3 EP07_03 🟡 Valutazione tramite Matrice di Confusione . . . 412
7.18.4 EP07_04 🟠 Codifica Manuale del Descrittore LBP . . . 414
7.18.5 EP07_05 🔴 Istogramma delle Orientazioni di una Cella HOG . . . 416
7.18.6 EP07_06 🟣 *Pipeline* Completo: Descrittori + k-NN + Valutazione Multi-Classe . . . 419
7.18.7 EP07_07 ⚫ Classificazione Reale di un Mosaico di Trame tramite LBP + k-NN . . . 422

8 Comprendere le Scene: Correspondenza delle Caratteristiche, Rilevamento e Segmentazione 429
8.1 Obiettivi del Capitolo . . . 429
8.2 Configurazione dell'Ambiente . . . 429
8.3 Rilevatori e Descrittori Locali di Caratteristiche . . . 431
8.3.1 L'Algoritmo ORB (*Oriented FAST and Rotated BRIEF*) . . . 431
8.3.2 Esplorando il Simulatore dell'ORB . . . 432
8.4 Corrispondenza delle Caratteristiche e Omografia con ORB . . . 433
8.4.1 Rilevamento e Corrispondenza delle Caratteristiche con ORB . . . 435
8.5 Modellazione Matematica: Omografia e RANSAC . . . 437
8.5.1 Omografia . . . 437
8.5.2 RANSAC . . . 437
8.5.3 Stima dell'Omografia con RANSAC . . . 439
8.5.4 Registrazione dell'Immagine . . . 440
8.6 Rilevamento di Oggetti: *Haar Cascade* (Viola-Jones) . . . 442
8.7 Rilevamento di Oggetti: Bounding Box, IoU e NMS . . . 444
8.7.1 *Intersection over Union* (IoU) . . . 444
8.7.2 Soppressione Non-Massima (NMS) . . . 445
8.8 Segmentazione Visuale: Semantica, per Istanza e Panottica . . . 446
8.8.1 Segmentazione Semantica per Soglia . . . 447
8.8.2 Segmentazione delle istanze mediante etichettatura . . . 447
8.8.3 Alternativa: Trasformata della Distanza + *Watershed* . . . 449
8.8.4 Misurazione ed Estrazione di Attributi con `mm.measure` . . . 451
8.8.5 Esportazione delle Annotazioni: da `mm.measure` al Formato YOLO . . . 453
8.8.6 Validazione delle Annotazioni tramite IoU: `mm.verifyBoundBox` . . . 454
8.8.7 Visualizzazione delle Annotazioni Salvate: `mm.showBoundBox` . . . 455
8.9 Panoramica dei Modelli Moderni . . . 456
8.10 Limiti degli Approcci Classici e Motivazione per il *Deep Learning* . . . 457
8.11 Riassunto . . . 457
Prossimi Passi . . . 458
8.12 🤖 Uso del Gemini Notebook come Supporto allo Studio . . . 458

8.13 Elenco di Esercizi . . . 458
Riferimenti del Capitolo . . . 459
8.14 💻 **Parte Pratica con Esercizi di Programmazione** . . . 460
🎯 Obiettivo di questo Quaderno . . . 460
Download . . . 460
Riepilogo dei Metodi di `morph.py` (Cap. 8) . . . 461
8.14.1 EP08_01 🟢 Distanza di Hamming e Corrispondenza di Descrittori Binari . . . 461
8.14.2 EP08_02 🟢 Omografia e RANSAC: Il Voto per *Inlier* . . . 463
8.14.3 EP08_03 🟢 Immagine Integrale: Somme Rettangolari a Tempo Costante . . . 465
8.14.4 EP08_04 🟢 IoU e Soppressione Non-Massima (NMS) . . . 467
8.14.5 EP08_05 🟡 Etichettatura dei Componenti Connessi: Segmentazione delle Istanze . . . 469
8.14.6 EP08_06 🟡 *Bounding Boxes*, Centroidi e Proprietà delle Istanze con `mm.measure` . . . 471
8.14.7 EP08_07 🟡 Rimozione del Rumore Sale e Pepe e Misurazione degli Oggetti . . . 474
8.14.8 EP08_08 🟡 Immagine in Scala di Grigi e Soglia Dinamica . . . 476
8.14.9 EP08_09 🟠 Gradiente di Illuminazione e Soglia Adattativa . . . 477
8.14.10 EP08_10 🔴 Contrasto Basso e Separazione degli Oggetti Tangenti (*Watershed* / Distanza) . . . 479
8.14.11 EP08_11 🔴 Classificazione e Validazione di Oggetti con Riferimento di *Bounding Box* 481
8.14.12 EP08_12 🔴 Segmentazione di Istanza in Immagine Reale: Oggetti Geometrici . . . 484

9 Apprendimento Profondo per la Visione Artificiale **489**
9.1 Obiettivi del Capitolo . . . 489
9.2 Panoramica: Classificazione, Rilevamento e Segmentazione . . . 490
9.3 Configurazione dell'Ambiente . . . 491
9.4 Fondamenti di Deep Learning per la VC . . . 493
9.4.1 Dalla Convoluzione Fissa alla Convoluzione Appresa . . . 493
9.4.2 Neurone Artificiale . . . 493
9.4.3 Livello Convoluzionale . . . 494
9.4.4 Funzione di Attivazione . . . 496
9.4.5 *Pooling* . . . 496
9.4.6 Addestramento delle Reti Neurali: Come le CNN Imparano . . . 498
9.4.7 Architettura di una CNN . . . 501
9.4.8 Come il Gradiente Regola i *Kernel* della Convoluzione . . . 501
9.5 Applicazioni Pratiche in VC . . . 506
9.5.1 Classificazione delle Immagini con CNN . . . 508
9.5.2 Rilevamento di Oggetti . . . 538
9.5.3 Segmentazione degli Oggetti . . . 549
9.5.4 Ingegneria dei Dati per la Visione Artificiale (*Roboflow*) . . . 558
9.5.5 Applicazioni Geometriche e Integrate . . . 563
9.6 Riassunto . . . 568
9.7 🤖 Uso del Gemini Notebook come Tutor Complementare . . . 569
9.8 Elenco di Esercizi . . . 569
9.9 Chiusura della Parte II . . . 571
Riferimenti del Capitolo . . . 571
9.10 💻 **Parte Pratica con Esercizi di Programmazione** . . . 572
🎯 Obiettivo di questo Quaderno . . . 573
9.10.1 EP09_01 🟢 Convoluzione 2D Manuale (*Forward* di un Livello Appreso) . . . 573
9.10.2 EP09_02 🟢 *Pooling* Manuale (Massimo e Media) . . . 577
9.10.3 EP09_03 🟡 Conteggio dei Parametri Addestrabili di una CNN . . . 580
9.10.4 EP09_04 🟡 Intersezione su Unione (IoU) e Soppressione dei Non-Massimi (NMS) . . . 585
9.10.5 EP09_05 🟠 Valutazione della Segmentazione: IoU e Dice Pixel per Pixel . . . 586
9.10.6 EP09_06 🔴 *Pipeline* Integrato: Dalla Rilevazione alla Misurazione nel Mondo Reale . 591

Riferimenti **595**

Appendici 597

A Sobre o MCTest 597
A.1 Instalação do MCTest . . . 597
A.2 Criando um exame no MCTest . . . 597
A.3 Criando uma questão paramétrica . . . 598
A.4 Exemplo real: uma questão do Simulado 4 . . . 600
A.5 Considerações finais . . . 602

B Sobre o Moodle (VPL) 603
B.1 Configurando uma atividade VPL . . . 603
B.2 Publicando os EPs como atividades VPL . . . 603
B.3 Considerações finais . . . 604

C Uso do SEB nos simulados quinzenais e avaliações 605
C.1 Configurando a aplicação no *SEB Tool* . . . 605
C.2 Vinculando a chave do SEB à atividade VPL no Moodle . . . 606
C.3 Restrição por endereço IP (opcional) . . . 606
C.4 Considerações finais . . . 606

D Geração de material didático com o *Gemini Notebook* 607
D.1 Vídeos conceituais e de resolução de EPs . . . 607
D.2 *Slides* de apoio . . . 607
D.3 Material principal e considerações finais . . . 607

E Uso de IA na Avaliação de Estudantes: o Projeto vpl-ai-feedback 609
E.1 Contexto de uso . . . 609
E.2 Arquitetura do projeto . . . 610
E.3 O *prompt* universal e a rubrica em duas etapas . . . 611
E.4 Duas camadas adicionais de evidência . . . 612
E.5 Fluxo de execução . . . 613
E.6 Exemplo ilustrativo: rubrica de uma prova com três questões . . . 613
E.7 Envio do *feedback* por e-mail . . . 616
E.8 Riscos, limites éticos e responsabilidade docente . . . 617
E.9 Considerações finais . . . 617

Elenco delle Figure

1 Pulsante cliccabile **Esegui Colab**, disponibile all'inizio delle parti **Teorica** e **Pratica** di ogni capitolo, che consente l'esecuzione interattiva degli esempi e degli esercizi. Nella versione PDF, esiste un *link* HTML diretto tra l'immagine del simulatore e la sua didascalia. 7

1.1 Diagramma relazionale che illustra le distinzioni fondamentali, le sinergie e le interconnessioni tra EDI e VA nel contesto dei sistemi basati su immagini. 4
1.2 Rappresentazione del flusso sequenziale di elaborazione: l'output di ciascun livello diventa l'input del livello successivo. 6
1.3 Flusso dettagliato del PDI: dall'acquisizione sensoriale all'estrazione di attributi e al riconoscimento automatizzato, includendo l'elaborazione a colori e la compressione. 6
1.4 Rappresentazione dello spettro visibile e della sua posizione rispetto alle altre radiazioni elettromagnetiche, evidenziando la variazione delle lunghezze d'onda da 380 nm a 750 nm. . . 7
1.5 (A) Spettro elettromagnetico completo in scala logaritmica, con evidenza della fascia visibile. (B) Dettaglio della luce visibile (380-750 nm) e dei suoi colori. (C) Scomposizione della luce bianca tramite prisma: la lunghezza d'onda minore subisce una rifrazione maggiore, separando UV, visibile e infrarosso. 8
1.6 Esempi di immagini sintetiche rappresentate matricialmente. 11
1.7 Mandrill (Mandrillus sphinx) in ambiente naturale. Credito: Julien Renoult (CC BY 4.0). . . 12
1.8 Elaborazione base delle immagini: (a) immagine originale, (b) immagine in scala di grigi, (c) immagine binarizzata con soglia (T=128). 14
1.9 Confronto tra sogliatura manuale (T=128) e automatica (Otsu) sull'immagine in scala di grigi. 15
1.10 Accesso a un pixel specifico (r, c) e la rappresentazione del suo intorno nell'immagine. 17
1.11 Simulatore EP01_01: Distanze Euclidea, City-block e Chessboard 23
1.12 Simulatore EP01_02: Prestazioni Predittive — Metriche di ML 30
1.13 Simulatore EP01_03: Mean Average Precision (mAP) e Curva P-S 33
1.14 Simulatore EP01_04: Statistiche dei pixel in matrice discreta 5x5 35
1.15 Simulatore EP01_05: Trasformazione del Negativo dell'Immagine (Inversione di Intensità) . . 37
1.16 Simulatore EP01_06: Conversione RGB in Scala di Grigi (Ponderazione Percettiva ITU-R BT.601) 38
1.17 Simulatore EP01_07: Soglia Globale Interattiva (Binarizzazione p > T) 40
1.18 Simulatore EP01_08: Rimappatura per Intervallo Condizionale (Luminosità e Contrasto per Soglia) 42
1.19 Simulatore EP01_09: Generatore di Motivo a Scacchiera (Logica di Parità (i + j) % 2) . . . 43
1.20 Simulatore EP01_10: Struttura del file PGM (Intestazione ASCII P2 e mappatura a tupla Python) 45
1.21 Simulatore EP01_11: Filtro di Massimo Locale 1D (Vicinato 1x3 con Condizione di Bordo) . 47

2.1 Confronto didattico tra il sistema visivo biologico e quello elettronico: in alto, l'anatomia dell'occhio umano che evidenzia la retina e i fotorecettori (coni e bastoncelli); in basso, la struttura di una fotocamera digitale che dettaglia il sensore CMOS, la matrice di filtri di Bayer (RGGB) e il processo di quantizzazione digitale eseguito dall'ADC. 50
2.2 Raccolta di sfide percettive: (in alto a sinistra) Vaso di Rubin — ambiguità figura-sfondo; (in alto al centro) Elefante di Shepard — incongruenza geometrica; (in alto a destra) Ombra di Adelson — costanza della luminosità basata sul contesto; (in basso a sinistra) Griglia di punti — inibizione laterale; (in basso a destra) Scala di Schröder. 51

2.3 Immagine RGB (3 canali, *MimesNimes*3) e versione RGBA (4 canali, *MimesNimes*4) con trasparenza alfa graduale da sinistra a destra. Immagine Mandrill — USC SIPI Image Database (dominio pubblico). 55
2.4 Effetto del sottocampionamento. I titoli mostrano le dimensioni (W x H) e la dimensione della matrice in memoria (KB). 57
2.5 Effetto della riduzione della profondità di bit. I titoli mostrano il numero di bit, i livelli e la dimensione in memoria (KB). 58
2.6 Illustrazione delle vicinanze 4 in una matrice 3x3. Al centro (1,1), il pixel di interesse. 59
2.7 Area de Protezione Ambientale Quilombos do Médio Ribeira - Barbudo-rajado (Malacoptila striata). Credito: Thomas Fuhrmann (CC BY-SA 4.0). 62
2.8 Esempio di traslazione dell'immagine dell'uccello con spostamenti (50,50) e (100,50). 64
2.9 Esempio di rotazione dell'immagine dell'uccello a 30° e 45° utilizzando l'interpolazione bilineare. 65
2.10 Confronto tra interpolazione con zoom sul dettaglio dell'occhio (ritaglio 60×60, ingrandito 4×). Si noti l'effetto a blocchi nel vicino più prossimo rispetto all'ammorbidimento nella bilineare. 67
2.11 Esempio di taglio dell'immagine dell'uccello: orizzontale (shx=0.3), verticale (shy=0.3) e combinato (shx=0.2, shy=0.2). 68
2.12 Simulatore EP02_01: Regolazione di Luminosità e Contrasto Lineare (p' = p +) 72
2.13 Simulatore EP02_02: Sottocampionamento Spaziale (Riduzione di Risoluzione per Salto f) . 74
2.14 Simulatore EP02_03: Quantizzazione e Profondità di Bit (Riduzione del Numero di Livelli di Grigio) 76
2.15 Simulatore EP02_04: Trasformata della Distanza in Immagine Binaria (Scacchiera, City-block ed Euclidea) 78
2.16 Simulatore EP02_05: Traslazione Geometrica dell'Immagine (Spostamento tx e ty con Riempimento del Bordo) 79
2.17 Simulatore EP02_06: Rotazione dell'immagine attorno all'origine di un angolo 81
2.18 Simulatore EP02_07: Ridimensionamento Spaziale e Interpolazione (Nearest Neighbor vs Bilineare) 83
2.19 Simulatore EP02_08: Trasformazione Geometrica di Taglio 2D (Shear Orizzontale e Verticale) 86
2.20 Simulatore EP02_09: Trasformazione Affine 2D (Matrice 2×3) 88
2.21 Simulatore EP02_10: Correzione della Prospettiva (Trasformazione di Omografia 3×3) . . . 90
2.22 Acquisizione dell'immagine di un Sudoku a sinistra. A destra, conversione in scala di grigi e ridimensionamento. Credito: Héctor Rodríguez de Guardamar, Spagna (CC BY 2.0). 92
2.23 Correzione prospettica: originale e vista frontale raddrizzata. 93
2.24 Simulatore EP02_11: Correzione della Prospettiva del Sudoku (Omografia 3×3 con Ricampionamento Bilineare) 94

3.1 *Mandrill* (*Mandrillus sphinx*) fotografato in ambiente naturale in Sudafrica. Credito: Carlos Guilherme Rodrigues (CC BY-SA 3.0). 99
3.2 Operazioni aritmetiche sature: addizione di costante (schiarimento) e sottrazione di costante (scurimento con saturazione a 0). 100
3.3 Ritratto di un leopardo (*Panthera pardus*) in ambiente naturale. Credito: C. Brück (CC BY-SA 4.0). 101
3.4 *Alpha blending* tra ritagli allineati del mandrillo e del leopardo (Figura 3.3) per diversi valori di . Con =1 si vede solo il mandrillo; con =0, solo il leopardo; valori intermedi fondono proporzionalmente gli sguardi delle due immagini. 101
3.5 Operazioni logiche bit a bit con maschera circolare: AND (isolamento della ROI), OR (illuminazione della ROI) e NOT (negativo). 102
3.6 Istogrammi dell'immagine originale, di una versione scurita (−80) e di una schiarita (+80). La sottrazione/addizione satura a 0 e 255. 103
3.7 Equalizzazione dell'istogramma in un'immagine 5×5 a 3 bit (L=8): immagine originale con toni concentrati in {2,3,4}, immagine equalizzata con toni redistribuiti in {1,5,7}, e i rispettivi istogrammi che evidenziano la dispersione delle frequenze. 106
3.8 Equalizzazione dell'istogramma: mm.equalize (globale tramite CDF) vs. CLAHE (adattativa con limitazione del contrasto). Gli istogrammi rivelano la dispersione progressiva delle intensità. 107

3.9 Specifica dell'istogramma: mandrillo mappato sul profilo tonale del leopardo. La CDF dell'uscita approssima la CDF di riferimento. 108
3.10 Correlação com *kernel* de média 3×3: versão didática (mm.conv0) vs. vetorizada (mm.conv via cv2.filter2D). A diferença máxima de 39 ocorre nas bordas. 115
3.11 *Patch* 5×5 estratto dall'immagine del leopardo (posizione [250:255, 250:255]). La finestra gialla evidenzia il vicinato 3×3 centrato sul pixel [1,1] dove la correlazione verrà calcolata. . . 115
3.12 Filtro medio con *kernel* di dimensione crescente (3×3, 7×7, 15×15). La sfocatura dei bordi aumenta con la dimensione del *kernel*. 117
3.13 *Kernel* Gaussiano 5×5 (=1): pesi normalizzati, maggiori al centro e decrescenti radialmente. La griglia facilita la lettura di ciascun coefficiente. 118
3.14 Confronto tra filtro medio e Gaussiano (*kernel* 9×9, =0). Il Gaussiano preserva meglio i bordi, visibile nel dettaglio del viso. 119
3.15 Simulatore: Filtraggio Spaziale di Enfasi 1D (Unsharp Masking e High-Boost) 120
3.16 *Kernel* laplaciano w4 applicato al *patch* 5×5: la finestra gialla evidenzia l'intorno 3×3 dove viene calcolato l'operatore di derivata seconda. 122
3.17 Laplaciano applicato all'immagine del leopardo: risposta grezza (bordi) con w4 e w8, e immagini evidenziate tramite sottrazione del Laplaciano. w8 è più sensibile alle diagonali. . . 123
3.18 Operatore di Sobel sull'immagine del leopardo: Gx rileva i bordi verticali, Gy rileva i bordi orizzontali, e la magnitudine | f| combina entrambi, rivelando tutti i bordi indipendentemente dalla direzione. 125
3.19 Operatore di Prewitt: Gx, Gy e magnitudine — paragonabile a Sobel, ma senza ponderazione Gaussiana perpendicolare. 125
3.20 Rafforzamento della nitidezza tramite *Unsharp Masking* nell'immagine del leopardo: l'immagine sfocata viene sottratta da quella originale per generare la maschera ad alta frequenza, che viene poi reintrodotta per enfatizzare bordi e dettagli. 127
3.21 *Unsharp Masking* sull'immagine del leopardo con =1 e k da 0.5 a 8.0. Per k>2 compaiono aloni sui bordi e appare il rumore di sfondo. 128
3.22 Rivelatore di Canny con diverse coppie di soglia: soglie basse catturano più bordi (incluso il rumore); soglie alte mantengono solo i bordi più forti. 129
3.23 Rumore sale e pepe con densità crescenti (2%, 5%, 10%). Il parametro prob indica la frazione totale di pixel corrotti, metà sale (255) e metà pepe (0). 130
3.24 Comparazione dei filtri per la rimozione del rumore sale e pepe (10%): Gaussiano, Media, Mediana, Bilaterale e Morfologico. Riga superiore: immagini complete; riga inferiore: crop della regione di interesse. 133
3.25 *Pipeline* di pre-processamento: CLAHE → Gaussiano → Canny. Ogni fase prepara l'immagine per la successiva, ottenendo bordi puliti e ben definiti. 134
3.26 Simulatore EP03_01: Addizione Saturata di Costante (p' = clip(p + k)) 138
3.27 Simulatore EP03_02: Alpha Blending di Due Immagini (g = · f1 + (1−) · f2) 140
3.28 Simulatore EP03_03: Inversione dell'immagine — Negativo fotografico (p' = 255 − p) 142
3.29 Simulatore EP03_04: Equalizzazione dell'istogramma (Livelli L = 2^B) 144
3.30 Simulatore EP03_05: Applicazione della Maschera AND Binaria 146
3.31 Simulatore EP03_06: Filtro Medio con Kernel N×N 148
3.32 Simulador EP03_07: Operador Laplaciano (w4) para Realce de Bordas 150
3.33 Simulatore EP03_08: Gradiente di Sobel (Gx e Gy) 153
3.34 Simulatore EP03_09: Filtro della Mediana 3×3 155
3.35 Simulatore EP03_10: *Unsharp Masking* (USM) 157

4.1 Imagem com moedas de vários tipos. Crédito: GAZI.MD.AHAD (CC BY-SA 4.0). 161
4.2 Confronto dei pre-processamenti: immagine | istogramma+T* | ²B(T) | Otsu. La migliore separazione bimodale indica la soglia più affidabile. 164
4.3 Elemento strutturante a forma di croce (B_{cruz}) utilizzato per la connettività-4. 166
4.4 Erosione e dilatazione su immagine binaria 10×10 con elemento strutturante 'L' asimmetrico 3×3. Validazione della dualità erosione–dilatazione. 172
4.5 Simulatore: Erosione Morfologica (A B_L) 172

4.6 Apertura, chiusura, composizione e filtro sequenziale alternato applicati alla binarizzazione Otsu delle monete (preprocessate con miglioramento del contrasto). Elemento strutturante: disco 13×13. 174
4.7 Simulatore interattivo di dilatazione geodetica: il marker f (verde) si propaga passo dopo passo attraverso i percorsi liberi della maschera g, senza attraversare pareti. 176
4.8 Risoluzione del labirinto tramite Operazioni Complementari: invertendo la maschera e il marker all'inizio del *pipeline*, il flusso viene risolto direttamente nel dominio complementare tramite mm.cero e mm.suprec, eliminando le re-inversioni ridondanti. 179
4.9 *Pipeline* di Ricostruzione Morfologica per Dilatazione Condizionata: la maschera contiene due oggetti, il marcatore isola solo il nucleo dell'oggetto principale, e le iterazioni ricostruiscono la sua forma esatta fino alla convergenza. 181
4.10 *Pipeline* di riempimento dei buchi (*clohole*): il marker proviene dal bordo dell'immagine, ristretto al complemento f^c. La dilatazione geodesica ricostruisce lo sfondo esterno; dopo la complementazione, i buchi interni vengono riempiti. 183
4.11 *Pipeline* di eliminazione delle strutture di bordo (*edgeoff*): il marcatore cattura le radici connesse alle estremità, la ricostruzione delimita questi elementi e la sottrazione preserva solo gli oggetti completamente interni. 184
4.12 *Pipeline* completo di segmentazione con CLAHE: Otsu → apertura (r=9) → clohole → apertura (r=33) → edgeoff. 185
4.13 Simulatore interattivo avanzato di morfologia con elemento strutturante modificabile e profilo 1D. 186
4.14 Morfologia in toni di grigio e i rispettivi istogrammi. L'erosione riduce le intensità locali, la dilatazione le amplifica, il gradiente evidenzia i bordi, e gli operatori Top-hat e Black-hat evidenziano dettagli locali luminosi e scuri. Elemento strutturante: disco di diametro 19. . . . 188
4.15 Algoritmo di etichettatura per *flood-fill* con pila. 191
4.16 Simulatore interattivo di etichettatura delle componenti connesse (*connected component labeling*): visualizzazione dell'espansione flood-fill, connettività-4 e connettività-8. 192
4.17 Effetto della connettività sull'etichettatura: l'immagine binaria 10×10 contiene pixel diagonalmente adiacenti. Con connettività-4, questi pixel formano componenti distinte; con connettività-8, si fondono in un'unica componente. 192
4.18 Algoritmo della Trasformata della Distanza. 194
4.19 Simulatore interattivo della Trasformata della Distanza (TD) iterativa tramite erosione in toni di grigio. I pixel fuori dall'immagine assumono il valore massimo (144), propagando i costi a partire dallo sfondo interno. 195
4.20 Trasformata della Distanza in immagine binaria 10×10. A sinistra: originale (*foreground* = 255). Al centro: mm.dist1 iterativa (erosioni con croce). A destra: mm.dist (L2). 195
4.21 Simulatore interattivo 2D (4x4) con sincronizzazione rigorosa delle code di propagazione orizzontale (b) per ottenere la convergenza esatta descritta nell'articolo. 196
4.22 Percorso geodetico minimo in un labirinto. Le distanze geodetiche sono calcolate dall'ingresso e dall'uscita usando mm.gdist. I pixel la cui somma delle due distanze è uguale alla distanza minima tra i marcatori appartengono a un percorso ottimale. 198
4.23 Algoritmo Didático di *Watershed* per Crescita di Regioni Limitato da Maschera. 199
4.24 Simulatore interattivo dell'Algoritmo *Watershed* per propagazione morfologica. Disegna la maschera, posiziona i marcatori e regola l'Elemento Strutturante per osservare l'inondazione. Quando i bacini si incontrano simultaneamente, il pareggio viene risolto assumendo una delle regioni in modo casuale. 200
4.25 *Pipeline watershed* delimitato da maschera su immagine binaria 20×20. 202
4.26 *Pipeline Watershed* per separazione di monete sovrapposte: dalla maschera binaria dilatata fino ai contorni finali annotati sull'immagine originale. 203
4.27 Componenti connessi estratti dopo il *pipeline* CLAHE → Otsu → pulizia morfologica. Ogni oggetto è colorato con un colore distinto e annotato con la propria area in pixel. 205
4.28 Contorni estratti con *findContours*. Ogni moneta è annotata con la sua circolarità — valori vicini a 1 confermano la forma circolare. 207

4.29 *Bounding boxes* derivate dai componenti connessi sovrapposti all'immagine originale. Le annotazioni vengono esportate nel formato YOLO (*classe cx cy w h*), con coordinate normalizzate nell'intervallo [0,1]. 209
4.30 Simulatore EP04_01: Soglia Globale con Soglia Fissa (p' = (p > T) ? 255 : 0) 214
4.31 Simulatore EP04_02: Sogliatura Automatica di Otsu (T* = argmax ²_B(T)) 216
4.32 Simulatore EP04_03: Dilatazione Binaria Piana (g = f B) 219
4.33 Simulatore EP04_04: Erosione Binaria Piana (g = f B) 221
4.34 Simulatore EP04_05: Apertura Morfologica (g = (f B) B) 223
4.35 Simulatore EP04_06: Chiusura Morfologica (g = (f B) B) 225
4.36 Simulatore EP04_07: Dilatazione ed Erosione con Pesi (mm.dil1 / mm.ero1) 228
4.37 Simulatore EP04_08: Gradiente morfologico, Top-hat e Black-hat 230
4.38 Simulatore EP04_09: Trasformata della Distanza (Livelli di Erosione) 232
4.39 Simulatore EP04_10: Separazione di Blob, Etichettatura e Descrittori 234

5.1 Decomposizione di Fourier 1D: un'onda quadra (linea tratteggiata) è approssimata dalla somma delle prime sinusoidi (linee colorate). Più termini ci sono, migliore è l'approssimazione. 236
5.2 Simulatore interattivo della decomposizione di Fourier 1D: visualizzazione della somma di sinusoidi con diverse frequenze, ampiezze e fasi. Aggiungi termini e osserva la convergenza verso forme d'onda arbitrarie. 237
5.3 Ogni frequenza nello spettro (punto isolato) corrisponde a un'onda sinusoidale 2D ruotata nel dominio spaziale. 239
5.4 Simulatore interattivo della sintesi di Fourier 2D. Modifica la posizione orizzontale (u) e verticale (v) del coefficiente nello spettro di frequenze centrato e osserva come la distanza dal centro determina la frequenza spaziale (spessore) e l'angolo determina l'orientamento dell'onda sinusoidale generata. 239
5.5 Diagramma concettuale dello spettro di Fourier 2D centrato. 241
5.6 Esperimento di cambio di fase: Immagine A (monete) e Immagine B (motivo geometrico) ricostruite con magnitudini e fasi scambiate. Il risultato mostra che la struttura visiva è **molto più sensibile alla fase** che alla magnitudine: quando la fase di B viene mantenuta, l'immagine risultante preserva l'organizzazione spaziale di B, anche con la magnitudine di A. La magnitudine, a sua volta, influenza principalmente il contrasto e la texture. Si noti che la qualità della ricostruzione non è perfetta — ci sono artefatti visibili —, evidenziando l'interdipendenza tra fase e magnitudine per una rappresentazione fedele dell'immagine. . . . 243
5.7 Confronto di efficienza: Convoluzione Non Separabile (Spaziale 2D), Separabile (Spaziale 1D) e tramite FFT. 245
5.8 Senza *padding*, un dislocamento severo fa sì che l'immagine fuoriesca sul lato opposto (convoluzione circolare). 246
5.9 Verifica del Teorema della Convoluzione: la differenza pixel per pixel tra la convoluzione spaziale (cv2.filter2D) e la moltiplicazione in frequenza (FFT) è numericamente nulla — confermando l'equivalenza teorica. 247
5.10 La Dualità Pericolosa: Il taglio netto in Frequenza (Cilindro) si trasforma obbligatoriamente in una Sinc spaziale. Le sue ondulazioni causano il *ringing* fantasma sui bordi dell'immagine. 248
5.11 Filtri passa-basso. 249
5.12 Simulatore interattivo di filtri nel dominio della frequenza. 250
5.13 Confronto tra filtri passa-basso: Ideale (D =30), Gaussiano (D =30) e Butterworth (D =30, n=2). Profili di H(u,v) lungo una linea centrale e immagini filtrate corrispondenti. 252
5.14 Filtro passa-alta Gaussiano. (a) Original; (b) Filtro passa-alta (D =30) - as bordas das moedas e fundo texturizado são realçados. 253
5.15 Rimozione del rumore periodico tramite filtro *notch* nel dominio della frequenza: (a) immagine con rumore sinusoidale, (b) spettro che mostra i picchi del rumore, (c) maschera *notch* centrata sui picchi, (d) immagine restaurata. 254
5.16 Fourier globale è cieco per la posizione. Gli spettri non dicono dove sono i bordi. 256
5.17 Funzioni della *Wavelet* ( ). Nota come rapidamente decadano a zero (supporto compatto), al contrario delle sinusoidi infinite di Fourier. 258

5.18 Diagramma della decomposizione *wavelet* 2D su due livelli. 258
5.19 Simulazione della decomposizione *wavelet* 2D. 260
5.20 Decomposição *wavelet* 2D de 2 níveis com *wavelet* Haar: subbandas LL, LH, HL, HH em cada nível. As subbandas de detalhe revelam estruturas orientadas em diferentes escalas utilizando um padrão sintético. 262
5.21 Confronto tra famiglie di *wavelet*: Haar, db4, sym4 e bior2.2. Sottobanda LL (approssimazione) e HH (diagonale) per ciascuna scelta, illustrando il compromesso tra compattezza e levigatezza basato sul motivo sintetico. 263
5.22 Ricostruzione *wavelet* con sogliatura dei coefficienti (*hard thresholding*): all'aumentare della soglia, più dettagli vengono azzerati, producendo immagini progressivamente più morbide. La metrica PSNR quantifica la perdita di qualità sul pattern sintetico. 265
5.23 L'alfabeto visivo del JPEG: Le 64 funzioni di base della DCT-II. Il coefficiente DC si trova in alto a sinistra (liscio). Scendendo e avanzando verso destra, l'oscillazione spaziale aumenta drasticamente. 268
5.24 DCT 2D in blocco 8×8: coefficienti e ricostruzione progressiva. 270
5.25 *Pipeline* JPEG semplificato applicato alla classica immagine del *Cameraman*: DCT in blocchi 8×8, quantizzazione con diversi fattori di qualità e ricostruzione tramite IDCT. Gli artefatti a blocchi (*blocking artifacts*) diventano visivamente evidenti con fattori di qualità ridotti ($Q = 10$ e $Q = 25$). 273
5.26 Simulatore interattivo di compressione DCT-JPEG: regola il fattore di qualità e visualizza in tempo reale i coefficienti azzerati, il blocco ricostruito e l'errore di quantizzazione. 274
5.27 Analisi comparativa degli artefatti di compressione sotto fattore di qualità ridotto ($Q = 10$). A sinistra, si osserva l'artefatto di blocco caratteristico della discretizzazione per DCT nel JPEG. A destra, si evidenzia l'effetto di attenuazione e smussamento dei bordi intrinseco allo standard WebP. 276
5.28 Curva rateo-distorsione: PSNR vs dimensione del file per JPEG, WebP e PNG applicata all'immagine del *Cameraman*. 278
5.29 Analisi spaziale della degradazione: immagini ricostruite e rispettive mappe di errore assoluto normalizzate per diversi fattori di qualità JPEG. 280
5.30 *Pipeline* completo di rimozione del rumore misto: (1) aggiunta di rumore gaussiano e periodico; (2) identificazione dei picchi di interferenza nello spettro di frequenze; (3) applicazione di una maschera *notch* con attenuazione gaussiana morbida; (4) post-elaborazione tramite filtro bilaterale per l'eliminazione del rumore stocastico residuo. 284
5.31 Mappa concettuale delle trasformazioni e delle proprietà nel dominio delle frequenze. 285
5.32 Simulatore EP05_01: Filtro Passa-Basso Ideale nello Spettro 291
5.33 Simulatore EP05_02: Filtro Notch 293
5.34 Simulatore EP05_03: Quantizzazione DCT (*round-trip*) 294
5.35 Simulatore EP05_04: DFT 2D manuale 296
5.36 Simulatore EP05_05: *Pipeline* JPEG completo a blocchi 298

6.1 Campione di immagini pubbliche di *skimage.data*, raggruppate per area di applicazione. . . . 308
6.2 Confronto tra strategie di preprocessamento per la binarizzazione dell'immagine del testo: (a) immagine originale; (b) sogliatura diretta con il metodo di Otsu; (c) sfondo stimato mediante filtraggio gaussiano; (d) immagine dopo la normalizzazione dello sfondo (divisione per l'immagine smussata); (e) CLAHE seguito da sogliatura con Otsu; e (f) normalizzazione dello sfondo seguita da sogliatura con Otsu. Le immagini illustrano l'effetto di ciascuna tecnica nella compensazione delle variazioni di illuminazione e nella qualità della segmentazione. . . . 309
6.3 Confronto del testo estratto da Tesseract da quattro versioni della stessa immagine: (a) immagine originale; (b) CLAHE seguito da sogliatura di Otsu; (c) normalizzazione dello sfondo seguita da sogliatura di Otsu; e (d) normalizzazione dello sfondo in scala di grigi, senza sogliatura. Il confronto evidenzia che la binarizzazione esterna non sempre favorisce il riconoscimento, poiché Tesseract esegue già internamente la propria binarizzazione adattiva. . 312

6.4 Flusso di riconoscimento e traduzione automatica. L'immagine preprocessata con normalizzazione dello sfondo, senza sogliatura, viene utilizzata come input per Tesseract OCR, e il testo riconosciuto viene tradotto dall'inglese al portoghese tramite la libreria *deep-translator*. 313
6.5 *Pipeline* di ingestione dei documenti: rasterizzazione adattiva delle pagine PDF in matrici discrete in formato PNG, mostrando la seconda pagina. 316
6.6 Simulatore interattivo di correzione dell'inclinazione (*deskew*): sposta lo *slider* per inclinare il documento e osserva le tre fasi del *pipeline* — immagine inclinata, bordi Canny e risultato corretto. 319
6.7 *Pipeline* di rettificazione assiale: visualizzazione comparativa tra l'ingresso ruotato originale, la mappa dei gradienti strutturali di Canny e il risultato finale allineato con sfondo normalizzato in bianco. 319
6.8 Rilevamento dei dischi marcatori, estrazione dei contorni e raddrizzamento tramite trasformazione prospettica. 322
6.9 Simulatore interattivo di filtraggio per circularità: muovi lo *slider* per regolare la soglia C e osserva quali componenti sono accettati (verde) o rifiutati (rosso). 323
6.10 *Pipeline* di elaborazione del *QRCode*: sogliatura, apertura morfologica, inversione e ritaglio finale per la decodifica. 326
6.11 Simulatore interattivo del *pipeline* di isolamento del QRCode: naviga tra le fasi di filtraggio, regola l'elemento strutturante della chiusura morfologica e osserva il rilevamento del contorno quadrato sull'immagine originale del modello. 327
6.12 Decodifica del codice a barre lineare con *pyzbar*: immagine di input e dati estratti. 329
6.13 Immagine del foglio delle risposte in scala di grigio caricata dal PDF rasterizzato. 331
6.14 Area di risposte estratta da *getAnswerArea*: regione raddrizzata contenente i riquadri di marcatura. 332
6.15 Regione del *QRCode* isolata da *segmentQRcode* all'interno dell'area delle risposte rettificata. . 333
6.16 Ritaglio inferiore dell'area di risposta, concentrando i riquadri delle bolle da segmentare. . . . 334
6.17 Risposte lette automaticamente da MCTest dopo la segmentazione e classificazione di tutte le bolle. 336
6.18 Rilevamento di difetti per sottrazione dell'immagine: prodotto di riferimento, immagine con difetto simulato e maschera dell'anomalia rilevata. 338
6.19 Simulatore interattivo di ispezione industriale per sottrazione di immagini: controlla le distorsioni geometriche di disallineamento (registrazione) e la soglia di rilevamento per osservare l'impatto sui falsi positivi. 339
6.20 Rilevamento dell'eterogeneità della texture: mappa di varianza locale e maschera di anomalia. 341
6.21 Simulatore EP06_01: IoU tra maschera di riferimento e maschera predetta 348
6.22 Simulatore EP06_02: Filtro di Marcatori per Circolarità 350
6.23 Simulatore EP06_03: Classificazione delle Marcature OMR 352
6.24 Simulatore EP06_04: Stimatore della Pendenza per Mediana Angolare 354
6.25 Simulatore EP06_05: Normalizzazione dello Sfondo per Divisione 356
6.26 Simulatore EP06_06: Mappa della Varianza Locale per il Rilevamento della Trama 358
6.27 Simulatore EP06_07: *Pipeline* di Ispezione — Registrazione per Traslazione e Sottrazione . . 360
6.28 Simulatore EP06_08: Segmentazione Geometrica + Verifica tramite Decodifica di QRCode . 364

7.1 Esempio motivazionale: diversi frutti che formano raggruppamenti distinti in uno spazio delle caratteristiche. 368
7.2 Panoramica del *pipeline* classico di classificazione delle immagini: estrazione dei descrittori (LBP, HOG), formazione dello spazio delle caratteristiche e classificazione tramite k-NN. Non si applica ai modelli di Deep Learning (CNN, YOLO), che apprendono *feature end-to-end* direttamente dai pixel. **Fonte:** elaborato con l'ausilio di Gemini Notebook ({GOOGLE}, 2025). 369
7.3 Confronto visivo di diversi descrittori applicati alla stessa immagine. Ogni descrittore rivela aspetti distinti della scena. 371
7.4 Istogramma dei codici LBP dell'immagine del cameraman. Ogni barra rappresenta la frequenza relativa di un codice LBP, formando il vettore di caratteristiche utilizzato per descrivere la texture. 372

7.5 Primi 100 componenti del vettore di caratteristiche HOG. 374
7.6 Importanza della normalizzazione delle caratteristiche per il classificatore k-NN. 377
7.7 Mappa concettuale del processo di classificazione delle immagini utilizzando descrittori classici (LBP, HOG, pixel grezzi) e classificatore tradizionale (k-NN). Non si applica a modelli di Apprendimento Profondo (CNN, YOLO), che apprendono *feature end-to-end* direttamente dai pixel. 378
7.8 Matrici di confusione ottenute dal classificatore k-NN utilizzando tre descrittori diversi. Le righe rappresentano la classe reale (Mela, Banana e Arancia) e le colonne la classe predetta. Maggiore è la concentrazione di valori sulla diagonale principale, migliore è la prestazione del descrittore. 381
7.9 Confronto dettagliato dell'accuratezza globale in uno scenario realistico. Si noti come i descrittori estratti strutturalmente (LBP e HOG) superino l'uso delle intensità pure dei pixel grezzi. 382
7.10 Illustrazione del processo di generazione di dati sintetici con *make_classification*. A sinistra, visualizzazione delle tre classi in uno spazio bidimensionale formato da due caratteristiche informative. A destra, importanza relativa di ciascuna caratteristica per la classificazione, evidenziando che solo 7 delle 10 caratteristiche sono effettivamente discriminative, mentre le altre sono ridondanti (2) o ripetute (1). 385
7.11 Classificazione di un nuovo campione tramite l'algoritmo k-NN. Il punto di test (stella) viene classificato in base ai tre vicini più prossimi, evidenziati da cerchi. 388
7.12 Simulatore interattivo della frontiera di decisione del k-NN: aggiungi punti di addestramento e regola il valore di k per osservare l'effetto sulla regione di decisione. 389
7.13 Campione di cifre scritte a mano dalla base *load_digits*, utilizzato come caso di studio di classificazione. 390
7.14 Matrice di confusione del classificatore k-NN addestrato con vettori di intensità grezzi (pixel). 391
7.15 Confronto di accuratezza tra descrittori di pixel grezzi e HOG per il classificatore k-NN (k=3) sulla base di cifre. 392
7.16 Precisione media per cross-validazione (5 pieghe) in funzione del parametro k, per il dataset di cifre con vettori di intensità grezzi. 394
7.17 Precisione sui set di training e test per diversi valori di k. Le regioni colorate rappresentano, in modo concettuale, tendenze di comportamento del classificatore: maggiore rischio di overfitting (rosso), compromesso tra bias e varianza (verde) e maggiore rischio di underfitting (blu). . . . 396
7.18 Campioni sintetici delle tre classi di texture utilizzate nell'esperimento di classificazione, generati con rumore gaussiano e variabilità intra-classe. 398
7.19 Acurácia média obtida via validação cruzada (5-fold) para os descritores de Pixels Brutos, LBP e HOG aplicados à base de texturas sintéticas. 400
7.20 Métricas di valutazione dettagliate per il classificatore k-NN con descrittori LBP 403
7.21 Simulatore EP07_01: Classificatore k-NN Passo a Passo 409
7.22 Simulatore EP07_02: Effetto della Normalizzazione *Z-score* sulla Distanza k-NN 411
7.23 Simulatore EP07_03: Precisione x Richiamo 413
7.24 Simulatore EP07_04: Codice LBP di un vicinato 3×3 415
7.25 Simulatore EP07_05: Istogramma HOG di una Cella (mappatura degli angoli ai bins) 418
7.26 Simulatore EP07_06: *Pipeline* k-NN Multi-Classe (votazione, spareggio e matrice di confusione) 421
7.27 Mosaici di riferimento (formato PGM ASCII) usati nell'EP07_07. Caso 1: griglia 2x2 con un campione di ogni classe. Caso 2: griglia 3x3 con classi ripetute e maggiore variabilità. 426
7.28 Simulatore EP07_07: Classificazione di un Mosaico di Trame tramite LBP + k-NN 427

8.1 Panoramica dei principali concetti trattati in questo capitolo, inclusi la corrispondenza delle caratteristiche, il rilevamento degli oggetti, la segmentazione delle immagini e le loro relazioni con gli approcci moderni basati sul *Deep Learning*. **Fonte:** elaborato con l'ausilio di Gemini Notebook ({GOOGLE}, 2025). 430
8.2 Confronto delle prestazioni di diverse implementazioni della distanza di Hamming. 433

8.3 Simulatore interattivo del descrittore ORB: esplora la logica di rotazione e costruzione del descrittore binario. Modifica la rotazione per osservare come il pattern di campionamento dei test binari del BRIEF (linee verdi e rosse) si orienta dinamicamente per garantire l'invarianza angolare. . . . 434
8.4 Imagem original gerada pelo Gemini e versão rotacionada (20°), simulando uma mudança de ponto de vista. . . . 435
8.5 Le 5 migliori corrispondenze delle caratteristiche ORB tra l'immagine originale e la scena sintetica, prima del filtraggio con RANSAC. . . . 437
8.6 Simulatore interattivo dell'algoritmo RANSAC: regola la soglia di distanza ed esegui l'algoritmo per osservare la separazione tra *inliers* e *outliers*. . . . 438
8.7 Simulatore interattivo del RANSAC applicato alla registrazione di immagini: i punti chiave di due immagini vengono accoppiati tramite un descrittore, alcune corrispondenze sono spurie (*outlier*), e il RANSAC stima la trasformazione di similarità che allinea la maggior parte di esse. 439
8.8 Corrispondenze classificate come *inliers* (verde) dal RANSAC stimando l'omografia tra le due immagini. . . . 440
8.9 Registrazione dell'immagine della scena usando l'omografia inversa stimata da RANSAC. . . . 441
8.10 Rilevamento di volti e occhi con *Haar Cascade*. . . . 444
8.11 Effetto della Soppressione Non-Massimale (NMS): più rilevamenti ridondanti (a sinistra) vengono ridotti a un rilevamento per oggetto (a destra). . . . 446
8.12 Segmentazione classica (non basata su deep learning): maschera semantica (moneta vs sfondo) tramite sogliatura di Otsu, e segmentazione di istanze tramite etichettatura di componenti connessi. . . . 449
8.13 Separazione di monete che si toccano tramite trasformata della distanza + *watershed*, senza erosione del contorno delle monete. . . . 451
8.14 *Bounding boxes* ricostruite a partire dal file di annotazioni YOLO esportato da *mm.saveMeasures*, sovrapposte all'immagine originale delle monete. . . . 455
8.15 Simulatore EP08_01: Distanza di Hamming tra Due Descrittori Binari . . . 463
8.16 Simulatore EP08_02: RANSAC — Votazione per Inlier tra Modelli Candidati . . . 465
8.17 Simulatore EP08_03: Somma Rettangolare in O(1) — Molteplici Situazioni di Bordo . . . 467
8.18 Simulatore EP08_04: IoU e Soppressione Non-Massimale . . . 469
8.19 Simulatore EP08_05: Etichettatura delle Componenti Connesse — Connettività 4 vs. 8 . . . 471
8.20 Simulatore EP08_06: Estrazione di *Bounding Boxes*, Centroidi e Proprietà con *mm.measure* 473
8.21 Simulatore EP08_07: Morfologia con Connettività Configurabile e Misurazione . . . 475
8.22 Simulatore EP08_08: Filtraggio Mediano in Scala di Grigi e Misurazione di Oggetti OpenCV 477
8.23 Simulatore EP08_09: Gradiente di Illuminazione, Soglia Adattativa e Misurazione OpenCV . 478
8.24 Simulatore EP08_10: Separazione di Dischi Tangenti via Trasformata della Distanza L2 e *Watershed* . . . 480
8.25 Simulatore EP08_11: Classificazione Geometrica con Controlli Indipendenti di *Overlay BBox* (GT e DET) . . . 483
8.26 Simulatore EP08_12: Immagine utilizzata nell'EP08_12. A sinistra, l'immagine originale. A destra, l'immagine con le *bounding boxes* del file di riferimento. . . . 486
8.27 Simulatore EP08_12: Accuratezza della Segmentazione su Multipli Oggetti (IoU) . . . 487

9.1 Panoramica dei principali concetti trattati in questo capitolo. **Fonte:** elaborato con l'ausilio di *Gemini Notebook* ({GOOGLE}, 2025). . . . 490
9.2 Simulatore interattivo di convoluzione 2D classica: scegli tra le tre immagini sintetiche di input 12×12 (casa, faccia felice o triste) e un filtro 3×3 (Sobel V, Sobel H, Nitidezza o Identità) e procedi passo dopo passo per osservare come i prodotti interni locali del campo recettivo costruiscono la mappa delle caratteristiche cella per cella. . . . 494
9.3 Simulatore interattivo del neurone artificiale in un contesto di CNN: passa da un neurone di un livello convoluzionale (dove x_i sono intensità di pixel in un campo ricettivo e w_i sono pesi del *kernel*) a un neurone di un livello completamente connesso, regolando input, pesi, bias e funzione di attivazione per visualizzare il calcolo di z e dell'output y in tempo reale. . . . 495

9.4 Simulador interativo da Camada Convolucional: escolha entre três imagens de entrada 12×12 (casa, rosto feliz ou rosto triste) para observar como os mesmos *kernels* fixos reagem a diferentes bordas e formas. Navegue pelo campo receptivo com os botões ou reprodução automática, ajuste a função de ativação e alterne o zero-padding, acompanhando o mapa de características sendo revelado célula a célula, com o cálculo detalhado termo a termo e a fórmula da dimensão de saída em tempo real. 497
9.5 Simulatore interattivo della funzione di attivazione ReLU: trascina il controllo per vedere come i valori negativi vengono azzerati e i valori positivi vengono preservati, sia nella curva che in una mappa delle caratteristiche reale. 498
9.6 Simulatore interattivo di *pooling*: scegli tra *max-pooling* e *average-pooling* e procedi passo dopo passo per osservare la riduzione della risoluzione spaziale della mappa delle caratteristiche. 499
9.7 Simulatore interattivo della Funzione di Perdita (*Cross-Entropy*): seleziona la classe reale dell'immagine (Casa, Felice o Triste) e regola le probabilità stimate dalla Softmax per visualizzare il calcolo della penalizzazione scalare e il grafico del logaritmo negativo in tempo reale. 500
9.8 Simulatore interattivo di *Backpropagation*: procedi attraverso i passaggi della Regola della Catena per seguire il flusso del segnale di errore nella direzione inversa della rete, osservando il calcolo delle derivate parziali del gradiente in ogni strato. 500
9.9 Simulador interativo dos Algoritmos de Otimização: compare a trajetória do SGD e do Adam sobre uma superfície de perda não convexa com mapa de calor e curvas de nível. A linha sólida mostra o otimizador selecionado avançando passo a passo; a linha tracejada mostra, para comparação instantânea, o caminho completo que o outro otimizador percorreria a partir do mesmo ponto inicial. Observe como o SGD fica retido no Mínimo Local à direita, enquanto o Adam pode ou não transpor a barreira central dependendo do impulso acumulado e da taxa de aprendizado. Clique em qualquer ponto do mapa para redefinir o ponto inicial dos pesos. . 502
9.10 Architettura semplificata di una CNN per la classificazione delle immagini, evidenziando il *forward pass* e le fasi di addestramento tramite la funzione di perdita, la retropropagazione e l'algoritmo di ottimizzazione. 503
9.11 Simulatore interattivo dell'architettura di una CNN: scegli una delle immagini di ingresso 12×12 (casa, viso felice o triste), clicca su ciascun blocco del *pipeline* — Ingresso, *Conv+ReLU*, *Pooling*, *Flatten*, *FC* e *Softmax* — ed esegui passi di addestramento reali (*forward pass* + retropropagazione) per osservare la perdita e l'accuratezza evolversi, i *kernel* essere regolati e il *Softmax* iniziare a puntare alla classe corretta. 504
9.12 Simulatore interattivo del calcolo del gradiente di un peso del *kernel* convoluzionale. 507
9.13 Campioni di cifre dal set di addestramento dopo la conversione in *tensori* PyTorch e normalizzazione. 510
9.14 Rappresentazione del flusso delle trasformazioni dimensionali dei *tensor* lungo l'architettura *CNNDigitos*. 510
9.15 Curve di training e valutazione della CNN sul dataset di cifre: evoluzione della perdita di entropia incrociata sul set di training e dell'accuratezza sul set di test nel corso di 50 epoche. 514
9.16 Flusso di attivazioni della CNN addestrata durante l'elaborazione di un esempio reale della cifra 3, dal *dataset load_digits*: dimensioni dei tensori strato per strato, dall'input al *logit* di output. 515
9.17 Grafo computacional della *CNNDigitos* generato tramite *torchviz*, che mostra le operazioni di *forward* e i nodi di gradiente (*backward*) associati a ciascun parametro addestrabile. 516
9.18 Grafo dettagliato della classe CNNDigiti generato tramite *torchviz*. 518
9.19 Confronto di accuratezza tra i classificatori classici del Capitolo 7 (pixel grezzi e HOG con k-NN) e la CNN addestrata in questo capitolo, sulla stessa base di cifre. 519
9.20 Campioni dei set di addestramento dopo il preprocessamento e l'adattamento dimensionale ai tensori PyTorch: Dominio A (cifre da 0 a 4, compito di origine) e Dominio B (cifre da 5 a 9 reindicizzate da 0 a 4, compito di destinazione). 522

9.21 Comparação de acurácia no conjunto de teste do Domínio B (dígitos 5 a 9) sob restrição de dados (20 exemplos de treino): demonstração do impacto do congelamento rígido e da transferência negativa em redes de baixa capacidade. 526
9.22 Flusso di attivazioni e trasformazioni dimensionali dei tensori nel modello di transfer learning durante l'elaborazione di un campione di test del Dominio B (cifra 7, reindicizzato come classe 2). 527
9.23 528
9.24 530
9.25 531
9.26 Campioni sintetici del *dataset* di diagnosi fogliare: foglia sana, foglia malata (macchie scure) e sintomo sconosciuto (clorosi giallastra) con rumore sale e pepe. 534
9.27 536
9.28 537
9.29 540
9.30 Campioni iniziali del *dataset* sintetico rumoroso di oggetti geometrici con le bounding box in formato YOLO sovrapposte. 543
9.31 Confronto tra l'immagine originale con rumore sale e pepe e l'immagine dopo l'applicazione del Filtro Mediano (3x3). Le *bounding box* in formato YOLO rimangono invariate. 545
9.32 548
9.33 550
9.34 Campioni del dataset sintetico di noduli: immagine in scala di grigi sotto rumore e rispettiva maschera binaria di riferimento mostrate. 551
9.35 Linea di base classica di segmentazione: smoothing, sogliatura di Otsu e apertura morfologica mostrate. 552
9.36 Evoluzione dell'allenamento della U-Net compatta: riduzione della perdita e aumento dell'indice IoU medio sul set di validazione lungo 35 epoche. 556
9.37 Confronto quantitativo (IoU medio) e qualitativo tra l'approccio classico e la U-Net addestrata su tre campioni di validazione. 558
9.38 561
9.39 561
9.40 563
9.41 Realtà aumentata basata su marker ArUco: marker inserito in scena reale e ruotato, rilevato e sostituito con immagine virtuale tramite omografia. 565
9.42 Misurazione di dimensioni fisiche reali utilizzando una carta di riferimento di scala nota (8,56 cm) renderizzata. Il rettangolo grigio rappresenta la carta di riferimento e il rettangolo blu rappresenta l'oggetto bersaglio. Le dimensioni stimate dell'oggetto sono visualizzate in centimetri. 568
9.43 Simulatore EP09_01: Convoluzione 2D Manuale (correlazione incrociata + bias + ReLU, con *padding* e *stride* regolabili) 576
9.44 Simulatore EP09_02: Pooling Manuale (massimo vs. media, con finestra k e stride s regolabili) 579
9.45 Simulatore EP09_03: Conteggio dei Parametri — Convoluzione vs. Strato Totalmente Connesso 584
9.46 Simulatore EP09_04: IoU e Soppressione Non-Massima (NMS) 587
9.47 Simulatore EP09_05: Valutazione della Segmentazione — IoU e Dice Pixel per Pixel 590
9.48 Simulatore EP09_06: Pipeline Integrato — Rilevamento alla Misurazione del Mondo Reale . 593

A.1 Questão do tabuleiro de xadrez gerada pelo MCTest, ilustrando o enunciado parametrizado com o valor de `height` sorteado para a variação e o exemplo de entrada/saída do primeiro caso de teste. 600
A.2 Questão real gerada pelo MCTest para o Simulado 4 — tópico "Operadores Morfológicos". . . 602

Elenco delle Tabelle

1 Diversi formati di pubblicazione generati automaticamente dallo stesso codice sorgente. . . . 6
2 Numero di pagine del PDF e tempo di rendering per combinazione (parallelismo reale medio di ~5 processi, picco di 8). La combinazione base in Python/Portoghese si limita a renderizzare le uscite già registrate nei *notebook* sorgente; le altre rieseguono tutto il codice. 6

1.1 Connessione tra PDI, VC e altre aree della scienza. 5
1.2 Principali tipi di immagine digitale e i rispettivi insiemi di valori possibili per ogni pixel. . . . 9
1.3 Principali librerie e moduli utilizzati in questo libro per il calcolo matriciale, PDI, VC, visualizzazione e validazione degli esercizi. 9

2.1 Convenzioni di scala e ordine dei canali nelle principali librerie di PDI. 52
2.2 Confronto tra strategie di compressione ed efficienza di elaborazione. 58
2.3 Confronto tra metriche di distanza applicate alla maglia di pixel. 60
2.4 Principali formati di archiviazione delle immagini digitali e le loro applicazioni nell'elaborazione digitale delle immagini (PDI). 60
2.5 Metodi di interpolazione disponibili in mm.resize e i rispettivi numeri di vicini utilizzati nel calcolo. 66

3.1 Algoritmo di equalizzazione dell'istogramma. 104
3.2 Interpretazione tipica dei coefficienti del *kernel*. 112
3.3 Fasi dell'*Unsharp Masking*. 126
3.4 Media vs. mediana con un pixel corrotto. La mediana ignora il valore anomalo; la media è spostata di ~40 livelli. 129

4.1 Tecniche di pre-processamento valutate per migliorare la separazione tra monete e sfondo prima dell'applicazione del metodo di Otsu. 162
4.2 Proprietà di apertura e chiusura. 173
4.3 Sequenze del filtro sequenziale alternato mm.asf. 174
4.4 Confronto tra gli operatori *clohole* e *edgeoff*. 182
4.5 Tassonomia semplificata dei principali approcci di segmentazione e raffinamento studiati in questo capitolo. 189
4.6 *Pipeline* completo del *watershed* basato su marcatori per immagini reali. 201
4.7 *Pipeline* semplificato utilizzato nell'esempio didattico della Figura 4.25. 201
4.8 Confronto tra gli approcci basati su componenti connessi e su contorni. 204

5.1 Corrispondenza tra le regioni dello spettro di ampiezza dopo l'applicazione dell'FFT Shift. . . 238
5.2 Confronto della complessità della convoluzione diretta, separabile e tramite Trasformata Rapida di Fourier (FFT). 244
5.3 Sottobande prodotte dalla Trasformata *Wavelet* Discreta 2D (DWT), indicando i filtri applicati in ciascuna direzione e il contenuto predominante di ciascuna componente. 256
5.4 Confronto tra famiglie di *wavelet*, evidenziando la lunghezza del supporto, il numero di momenti nulli, la simmetria e le applicazioni tipiche. 257
5.5 Confronto tra la Trasformata Discreta di Fourier (DFT) e la Trasformata *Wavelet* Discreta (DWT), evidenziandone le principali caratteristiche e applicazioni. 265
5.6 Categorie di ridondanza nelle immagini digitali e i rispettivi meccanismi di sfruttamento. . . 266
5.7 Fasi del *pipeline* di compressione JPEG e i rispettivi fondamenti di progetto. 270
5.8 Confronto strutturale tra i principali formati di immagine rasterizzati. 274

5.9 Sintesi delle fasi del *pipeline* di compressione JPEG e dei rispettivi impatti. 281

6.1 Immagini di skimage.data utilizzate negli esperimenti di questo capitolo. 306
6.2 Selezione di immagini pubbliche rappresentative disponibili nel modulo *skimage.data*, organizzate per ambito di applicazione. 306

7.1 Insieme di descrittori cromatici, testurali e geometrici utilizzati per rappresentare immagini di frutta. 382
7.2 Esempio semplificato di classificazione della frutta utilizzando due caratteristiche normalizzate. 385

8.1 Confronto tra i principali paradigmi di segmentazione visuale. 447
8.2 Architetture rappresentative di Deep Learning per il rilevamento e la segmentazione delle immagini. 456

9.1 Confronto tra i principali compiti di VC secondo la granularità dell'informazione prodotta. . 490
9.2 Confronto tra i tre scenari di trasferimento dell'apprendimento presentati in questa sezione. . 538

C.1 Expressões de filtro de URL utilizadas na configuração do SEB. 605

EDI e VA — Python

Benvenuti nel libro EAI e Visione Artificiale — versione Python / Italiano.

Organizzazione

- **Parte I — Fondamenti di EAI** — Rappresentazione, istogrammi, filtraggio, morfologia
- **Parte II — Visione Artificiale** — Segmentazione, descrittori, rilevamento, apprendimento profondo

Ficha Catalográfica

Prefazione

Questo libro costituisce un'**opera in costante aggiornamento** nei settori dell'**Elaborazione Digitale delle Immagini (EDI)** e della **Visione Artificiale (VA)**, concepita come materiale didattico interattivo per corsi di laurea triennale e magistrale in Informatica, Ingegneria e settori affini.

> **!** In evidenza
>
> L'opera si fonda sulla metodologia descritta in Zampirolli et al. (2025) — estensione di Zampirolli et al. (2024), lavoro premiato nel filone **Risorse e Ambienti Educativi** dell'**EduComp 2024**. Il contenuto integra la libreria `morph.py`, sviluppata dall'autore.

Questo non è un libro statico. Il suo contenuto evolve continuamente man mano che gli esempi vengono migliorati, nuove sezioni vengono incorporate e gli approcci pedagogici vengono affinati sulla base dell'esperienza d'uso e del riscontro di studenti e docenti. In tal modo, l'opera è trattata come un *progetto in costante evoluzione*, cercando di seguire sia i progressi tecnologici sia le migliori pratiche didattiche in EDI e VA.

Contesto della prima edizione

Il contenuto di questa edizione è stato elaborato tra maggio e agosto 2026, durante la prima offerta del corso di Elaborazione Digitale delle Immagini basata su questo materiale didattico. Il corso è stato tenuto in due classi di laurea triennale — composte prevalentemente da studenti di Informatica, nel periodo mattutino — e in una classe di laurea magistrale in Informatica, tutte presso l'UFABC. Questa prima edizione rappresenta il risultato di tale offerta iniziale, consolidando il contenuto sviluppato, testato e continuamente perfezionato nel corso del periodo accademico.

La metodologia di insegnamento adottata ha seguito una sequenza strutturata in ogni lezione. Inizialmente, veniva mostrato un video breve, della durata media di 7 minuti, generato in **Gemini Notebook**, alternando la presentazione della parte concettuale del capitolo e la risoluzione degli Esercizi di Programmazione (EP). In questo secondo caso, quasi tutti gli EP venivano risolti durante la stessa lezione. Successivamente, veniva presentato un insieme di circa 12 *slide*, anch'esse generate in **Gemini Notebook**, aventi come fonti il PDF completo del libro e il file `morph.py`. Queste *slide* venivano prodotte a partire da un *prompt* specifico per la generazione del contenuto teorico e pratico di ciascun capitolo.

Dopo questa fase iniziale, restavano circa 100 minuti di lezione per l'esplorazione dei due *notebook* Colab del capitolo, uno teorico e uno pratico, con enfasi sui simulatori interattivi e sull'esecuzione dei blocchi di codice, consentendo agli studenti di modificare i parametri e osservarne gli effetti. Nelle lezioni svolte in laboratorio, gli studenti trasferivano le risposte degli EP sviluppate in Colab direttamente nelle attività VPL del Moodle, dove venivano sottoposte alla valutazione automatica. Questo processo di pubblicazione e utilizzo degli EP è presentato alla fine di questa prefazione.

La valutazione continua includeva simulazioni quindicinali, somministrate con il **SEB (*Safe Exam Browser*)** in Moodle, contenenti EP simili a quelli delle liste di esercizi. Ogni simulazione concedeva un bonus del 5% sul voto finale. Anche le due prove del corso utilizzavano il SEB, ma erano composte da quesiti parametrizzati generati in **MCTest**, la cui descrizione combina LaTeX e parametri nel formato `[[code:variabile]]`, definiti in sezioni di Python delimitate da `[[def: ... ]]` nella stessa domanda. Questo processo ha permesso di generare 110 varianti di prova, sorteggiate individualmente tra gli studenti.

L'Appendice A descrive in dettaglio la creazione di tali domande in MCTest e la loro esportazione in Moodle;

l'Appendice B descrive la configurazione delle attività VPL, inclusi il processo di pubblicazione degli EP; l'Appendice C presenta la configurazione del SEB per le simulazioni e le prove; l'Appendice D descrive la generazione dei video e delle *slide* utilizzati nelle lezioni con **Gemini Notebook**; e l'Appendice E dettaglia l'uso dell'**Intelligenza Artificiale (IA)** nella generazione di *feedback* socratico per attività formative e valutative, integrando la correzione automatica effettuata dal VPL.

Come viene prodotto questo libro

Il contenuto di questo libro è sviluppato in **Quarto** ed è archiviato nella cartella `all` del repository **github.com/fzampirolli/pdi-vc**. A partire da un unico codice sorgente, il libro viene generato automaticamente e pubblicato in diversi formati, presentati nella Tabella 1.

Sebbene l'edizione in PDF registri lo stato dell'opera al termine della prima offerta del corso nel 2026, lo sviluppo del libro continua in modo permanente. A ogni aggiornamento del repository GitHub, vengono generate automaticamente nuove versioni delle pagine HTML, del PDF e dei notebook, mettendo immediatamente a disposizione dei lettori i miglioramenti.

Tabella 1: Diversi formati di pubblicazione generati automaticamente dallo stesso codice sorgente.

Formato	Descrizione
HTML	Versione per il *web*, con simulatori interattivi e navigazione tra i capitoli: fzampirolli.github.io/pdi-vc/
PDF	Versione adatta alla stampa o alla lettura *offline*: livro.pt.py.pdf
Notebook (`.ipynb`)	Compatibili con **Jupyter** e **Google Colab**, che consentono di eseguire, modificare e sperimentare gli esempi di codice e gli Esercizi di Programmazione (EP). Un filtro personalizzato mantiene i riferimenti incrociati, la numerazione di figure e tabelle, oltre a formattare automaticamente le citazioni secondo lo standard ABNT.

L'opera è pubblicata in **dieci combinazioni** — ogni percorso di codice (**Python** e **C++**) in cinque lingue (portoghese, inglese, francese, spagnolo e italiano). Con la cache di traduzione già popolata, una rigenerazione completa (traduzione, riesecuzione dei *notebook*, rendering di HTML e PDF e pubblicazione) richiede circa **72 minuti**, con un parallelismo reale medio di ~5 processi (picco di 8); la **prima** generazione di una nuova lingua, con la cache vuota, è molto più lenta — circa **80 minuti** per combinazione, come osservato nelle prime generazioni di spagnolo e italiano. La Tabella 2 riassume ogni combinazione (con la cache già popolata): numero di pagine del PDF e tempo di rendering. Questo tempo totale reale è molto inferiore alla somma dei tempi di ciascuna combinazione presa singolarmente (oltre **5 ore** se eseguite una alla volta): su una CPU con molti core, più combinazioni vengono elaborate contemporaneamente, quindi il tempo totale non è la semplice somma dei tempi individuali.

Tabella 2: Numero di pagine del PDF e tempo di rendering per combinazione (parallelismo reale medio di ~5 processi, picco di 8). La combinazione base in Python/Portoghese si limita a renderizzare le uscite già registrate nei *notebook* sorgente; le altre rieseguono tutto il codice.

Combinazione	Percorso	Lingua	Pagine	Tempo di rend.
`py.pt`	Python	Portoghese (base)	654	~7 min
`py.en`	Python	Inglese	644	~31 min
`py.fr`	Python	Francese	666	~31 min
`py.es`	Python	Spagnolo	666	~31 min
`py.it`	Python	Italiano	658	~31 min
`cpp.pt`	C++	Portoghese	434	~26 min
`cpp.en`	C++	Inglese	426	~34 min
`cpp.fr`	C++	Francese	434	~38 min

Combinazione	Percorso	Lingua	Pagine	Tempo di rend.
`cpp.es`	C++	Spagnolo	430	~37 min
`cpp.it`	C++	Italiano	428	~35 min

Stato di validazione. Solo la combinazione `py.pt` (Python, portoghese) è la fonte curata: è lì che l'autore scrive, revisiona e valida tutto il contenuto. Le altre nove combinazioni — i percorsi in **C++** e le traduzioni in inglese, francese, spagnolo e italiano — sono **generate automaticamente**, tramite traduzione con un modello linguistico e transpilazione del codice, e **richiedono ancora una revisione dettagliata**. Il punto più delicato è il **testo incorporato in alcune figure**: dove la versione tradotta non è ancora stata prodotta, l'immagine appare con testo in portoghese (ogni figura tradotta segue lo stesso suffisso di lingua degli altri file generati, ad esempio `immagine_en.png`).

Le pagine HTML, il PDF e i *notebook* disponibili nel progetto riflettono sempre lo stato più recente dello sviluppo. Quando viene pubblicata un'**edizione ufficiale**, essa viene identificata da un *tag* nel repository GitHub, consentendo di riprodurre esattamente il contenuto corrispondente a quell'edizione. In questo modo, il lettore può seguire sia l'evoluzione continua del progetto sia recuperare qualsiasi edizione ufficiale precedentemente pubblicata.

```
git clone https://github.com/fzampirolli/pdi-vc.git
cd pdi-vc
git checkout <tag-da-versao>
```

Ogni capitolo mette a disposizione due pulsanti **Esegui Colab** (Figura 1), che indirizzano ad ambienti complementari:

1. **Parte Teorica**, situata all'apertura di ogni capitolo, contenente i concetti presentati ed esempi di codice eseguibili;
2. **Parte Pratica**, nella sezione **Parte Pratica con Esercizi di Programmazione** (EP), dedicata agli esercizi e ai loro simulatori interattivi.

Figura 1: Pulsante cliccabile **Esegui Colab**, disponibile all'inizio delle parti **Teorica** e **Pratica** di ogni capitolo, che consente l'esecuzione interattiva degli esempi e degli esercizi. Nella versione PDF, esiste un *link* HTML diretto tra l'immagine del simulatore e la sua didascalia.

La generazione dei *notebook* è completamente automatizzata dallo *script* `gerar_notebooks_alunos.py`, che adatta il contenuto ad ambienti interattivi, consentendone l'esecuzione senza la necessità di installare Quarto.

Uso di strumenti di Intelligenza Artificiale

La concezione, il progetto pedagogico, la struttura concettuale e il contenuto fondamentale di questo libro sono di **esclusiva autorship dell'autore**.

Nel processo di redazione e supporto allo sviluppo, sono stati utilizzati, nelle loro versioni gratuite, strumenti di **Intelligenza Artificiale (IA)**, come **ChatGPT**, **Claude**, **DeepSeek** e **Gemini**, impiegati strettamente come risorse ausiliarie. Il loro uso si è limitato al supporto nella revisione e nel miglioramento dello stile testuale, all'ottimizzazione della sintassi dei codici e alla generazione assistita di illustrazioni concettuali ed esempi.

Tutte le risposte e i contenuti suggeriti da questi strumenti sono stati sottoposti a **analisi critica, verifica, validazione tecnica, adattamento e integrazione da parte dell'autore**, che si assume la responsabilità per il testo, i codici e le risorse pedagogiche presentate. Gli strumenti di IA **non sono considerati autori o coautori dell'opera**, né responsabili delle decisioni intellettuali, tecniche o pedagogiche che ne fondano il contenuto.

Si sottolinea, inoltre, che i **simulatori** e le risorse interattive presenti nel libro — disponibili nelle versioni HTML e Jupyter Notebook (IPYNB) — sono stati progettati e implementati come parte dell'approccio pedagogico dell'opera, con l'obiettivo di offrire una sperimentazione diretta dei concetti presentati e favorire l'autonomia di apprendimento del lettore.

Distinto dall'uso editoriale sopra descritto, il *pipeline* di generazione multilingua (vedi "Come questo libro è prodotto") impiega un modello linguistico come **componente ingegneristica del sistema**: la traduzione automatica del contenuto tra il portoghese e altre lingue, con cache incrementale che evita di ritradurre porzioni non modificate. Questa fase utilizza l'API a pagamento di **DeepSeek** (modello `deepseek-v4-flash`); il libro completo è già stato tradotto dal portoghese in **inglese**, **francese**, **spagnolo** e **italiano**, e i **Capitoli da 1 a 5** dispongono inoltre di un percorso in **C++** che compila ed esegue realmente — a un costo complessivo dell'ordine di pochi dollari, grazie alla cache incrementale.

La libreria `morph.py`

La libreria **`morph.py`** (Zampirolli et al., 2025) accompagna l'intera opera come strumento di supporto all'insegnamento. Il suo obiettivo non è sostituire librerie consolidate, come **OpenCV**, **scikit-image**, **scikit-learn**, **NumPy** o **Matplotlib**, né competere con esse in prestazioni o ampiezza. Piuttosto, cerca di **rendere trasparenti gli algoritmi fondamentali di Elaborazione Digitale delle Immagini e Visione Computazionale (PDI-VC)**, consentendo allo studente di comprenderne l'implementazione, modificare il codice e sviluppare nuove funzionalità.

Quando possibile, la libreria offre due versioni per lo stesso algoritmo: una **implementazione didattica**, scritta passo dopo passo per riflettere direttamente le definizioni matematiche presentate nel libro, e un'altra **implementazione ottimizzata**, basata su librerie specializzate per applicazioni pratiche.

> **Eredità Tecnica**
>
> La struttura della `morph.py` ha come base strumenti di Morfologia Matematica sviluppati in Brasile, come **MMachLib** (Lotufo et al., 1997) e **MMach** (Barrera et al., 1998), oltre alla **mmorph**, utilizzata nell'opera di Dougherty; Lotufo (2003). Queste librerie sono state un punto di riferimento per l'insegnamento del settore per decenni.

Questa filosofia può essere osservata, ad esempio, negli operatori morfologici di dilatazione:

- `mm.dil`: implementazione ottimizzata, indicata per applicazioni pratiche;
- `mm.dil0`: implementazione didattica della dilatazione per **elementi strutturanti planari**, seguendo direttamente la definizione classica della Morfologia Matematica;
- `mm.dil1`: implementazione didattica della dilatazione per **funzioni strutturanti (*kernel* non planari)**, seguendo rigorosamente la loro formulazione matematica.

La libreria offre anche funzioni per lettura, visualizzazione, conversione dei colori, filtraggio e morfologia matematica tramite un'interfaccia semplice:

```
from morph import mm

img = mm.gray(mm.read("lena.jpg"))  # lettura e conversione in scala di grigi
grad = mm.gradm(img)                # gradiente morfologico
mm.show(grad)                       # visualizzazione
```

Nel corso del libro, nuovi metodi vengono introdotti nella libreria per semplificare lo svolgimento degli Esercizi di Programmazione (EP). Tra questi ci sono `mm.readTrain()` e `mm.readTest()`, che automatizzano la lettura delle basi di addestramento e test utilizzate nei capitoli dedicati al riconoscimento di pattern.

Inoltre, tutti i progetti del **Gemini Notebook** includono il file `morph.py`, consentendo di consultare e discutere direttamente l'implementazione degli algoritmi presentati nel libro. In questo modo, lo studente può utilizzare il Gemini Notebook stesso come assistente di studio, ponendo domande come:

Spiega come è stato implementato l'algoritmo `mm.knn0` della `morph.py`, mostrando il codice e commentando ogni fase in modo didattico. Inoltre, spiega le differenze tra `mm.knn0()` e `mm.knn()`.

! Implementazioni didattiche e implementazioni ottimizzate

In diversi capitoli, lo stesso algoritmo viene presentato in due versioni. Le funzioni con suffisso `0`, `1`, ecc. (ad esempio, `mm.knn0()`, `mm.dil0()` e `mm.dil1()`) sono implementazioni didattiche, sviluppate per facilitare la comprensione degli algoritmi. Le funzioni senza suffisso (come `mm.knn()` e `mm.dil()`), invece, utilizzano implementazioni ottimizzate, spesso basate su librerie specializzate come OpenCV, scikit-image e scikit-learn, risultando più adatte per applicazioni pratiche.
Nelle attività valutate dal **VPL del Moodle**, librerie come **scikit-learn** e, in alcuni casi, **scikit-image**, potrebbero non essere utilizzate a causa dei limiti di memoria dell'ambiente di esecuzione. Per questo motivo, gli Esercizi di Programmazione (EP) impiegano preferibilmente le implementazioni didattiche della `morph.py`, che producono gli stessi risultati, sebbene con minori prestazioni computazionali. Questa restrizione non esiste quando i programmi vengono eseguiti localmente, ad esempio in **VS Code**, **Jupyter Notebook** o **Google Colab**.

Il codice sorgente della libreria è disponibile presso:

https://github.com/fzampirolli/pdi-vc/tree/master/morph

Esercizi di Programmazione (EP) e Validazione Automatica

Ogni unità del libro include **Esercizi di Programmazione (EP)** pratici e di complessità crescente, progettati per consolidare i concetti presentati nel corso del capitolo. Ogni EP è accompagnato da un **simulatore interattivo**, disponibile nelle versioni HTML e IPYNB, che consente allo studente di manipolare parametri, visualizzare il comportamento degli algoritmi e sviluppare una comprensione intuitiva del problema prima di iniziare la sua implementazione. In questo modo, il processo di apprendimento combina sperimentazione, programmazione e validazione automatica.

La validazione delle soluzioni viene effettuata localmente dalla classe `TestSuite` (`testsuite.py`), che confronta l'output del programma con i file dei casi di test (`.cases`):

```
TestSuite("EP01_01.py").run()
```

Il sistema supporta più linguaggi (Python, Java, C++, C, JavaScript e R) e può essere integrato direttamente in Moodle. Per i docenti interessati alla procedura completa passo passo di pubblicazione degli EP come attività VPL, si consulti la **Guida per il Docente**, alla fine di questa prefazione, e l'Appendice B.

Codice aperto

Il progetto è disciplinato da principi di codice aperto. Il repository pubblico riunisce il testo, i codici, le immagini e gli *script*: **github.com/fzampirolli/pdi-vc**

Ogni versione del libro è archiviata in modo permanente su **Zenodo** con un *DOI citabile*. Per citare questo materiale in lavori accademici:

> ZAMPIROLLI, Francisco de Assis. **PDI+VC — Elaborazione Digitale delle Immagini e Visione Artificiale**. UFABC, 2026. DOI: 10.5281/zenodo.20784605

Va altresì registrato, a proposito, che il contenuto esposto in quest'opera riflette lo sguardo critico, la proposta pedagogica e l'esperienza didattica del suo autore. Pertanto, le analisi, gli approcci e le opinioni espressi lungo questo libro rappresentano esclusivamente la comprensione del suo ideatore, non costituendo né riflettendo una posizione ufficiale o istituzionale dell'Università Federale dell'ABC (UFABC).

Prima di iniziare: *Notebook* in Python

Il concetto di ***Literate Programming*** (Programmazione Letterata), proposto da Donald Knuth (Knuth, 1984), costituisce il fondamento della struttura di questo materiale. La logica inverte il paradigma tradizionale: il programma è scritto per la lettura umana, assimilabile a un saggio, mentre il codice viene estratto separatamente per l'esecuzione computazionale.

Il contenuto è strutturato in *notebook* — documenti che intercalano **celle di testo** (in Markdown) e **celle di codice** (in Python).

- **Esecuzione:** le celle di codice sono identificate da `[ ]`. L'esecuzione può essere effettuata con `Shift + Enter` o tramite il pulsante ▷ dell'interfaccia.
- **Ambienti:** i *notebook* possono essere eseguiti localmente, tramite Jupyter o Visual Studio Code (VS Code), nonché in ambienti cloud, come Google Colab.

> **Nota sul formato**
>
> In ambienti interattivi, il codice può essere modificato ed eseguito. Nelle versioni statiche (HTML o PDF), i blocchi di codice hanno finalità di lettura e riferimento, senza pregiudicare l'integrità delle spiegazioni.

Guida per il Docente: Pubblicazione degli EP su Moodle

*Questa sezione è destinata ai docenti che desiderano integrare gli Esercizi di Programmazione (EP) nelle attività **VPL (Virtual Programming Lab)** di Moodle, integrando l'Appendice B.*

Integrazione con Moodle (VPL)

Gli EP dei *notebook* possono essere convertiti in attività **VPL** su Moodle. Lo *script* `ep_tools.py` (eseguito tramite `make build`) automatizza l'esportazione degli EP dalla fine di ogni capitolo in due fasi:

1. **Estrazione** (`make eps`): genera un HTML interattivo autonomo per ogni EP, con enunciato, esempi e simulatore, in `gen/book/eps/<versione>/`. Vedi l'elenco completo su: https://fzampirolli.github.io/pdi-vc/eps/py.pt/index.html
2. **Conversione** (`make moodle`): trasforma l'HTML in un frammento autocontenuto, senza dipendenza da CSS esterni, pronto per essere incollato nell'editor di Moodle, in `gen/book/eps/<versione>_moodle/`. Vedi un esempio su: https://fzampirolli.github.io/pdi-vc/eps/py.pt_moodle/EP01_01.html

Inoltre, tutti i simulatori interattivi sviluppati nel corso del libro possono essere consultati direttamente su https://fzampirolli.github.io/pdi-vc/simuladores/py.pt/.

> **Altri percorsi e lingue**
>
> I link sopra usano `py.pt` come esempio. Per accedere agli EP o ai simulatori di un'altra combinazione linguaggio/lingua, basta sostituire quel segmento nell'URL — ad esempio, `cpp.it` per il percorso C++ in italiano: `https://fzampirolli.github.io/pdi-vc/eps/cpp.it/index.html`. La struttura è la stessa per tutte le combinazioni disponibili, in `/eps/<versione>/`, `/eps/<versione>_moodle/` e `/simuladores/<versione>/`.

Pubblicando con `make publish`, queste **due versioni** di ogni EP sono disponibili nei *link* indicati. Il docente può combinare le due strategie: incollare la versione Moodle nell'editor VPL (con simulatore funzionante) e includere nell'enunciato un *link* alla versione completa, dove le formule vengono renderizzate correttamente da MathJax.

Revisione obbligatoria prima della pubblicazione su Moodle

Sebbene automatizzata, la conversione richiede una revisione da parte del docente a causa delle limitazioni dell'editor TinyMCE/HTML Purifier di Moodle:

- **Formule matematiche** — TinyMCE elimina le barre rovesciate (\), corrompendo le equazioni LaTeX. Per questo motivo, la versione Moodle converte le formule semplici in HTML puro (entità come `≥`, `×`). Le formule complesse (`\begin{cases}`, matrici, integrali) possono risultare incomplete — verifica e, se necessario, riscrivile in testo o indica la versione completa tramite *link*.
- **Figure esterne** — Le immagini complementari devono essere inserite manualmente nell'attività VPL.
- **Simulatori** — Funzionano perfettamente purché utilizzino JavaScript standard con stili *inline* e manipolazione diretta del DOM (`getElementById`). **Attenzione:** se si includono formule in LaTeX contenenti il carattere \ in Moodle, i simulatori potrebbero smettere di funzionare.

Come Pubblicare su Moodle

1. Accedi alla versione Moodle dell'EP desiderato:

   ```
   https://fzampirolli.github.io/pdi-vc/eps/py.pt_moodle/EPXX_YY.html
   ```

2. Copia il codice sorgente completo della pagina (`Ctrl+U` → `Ctrl+A` → `Ctrl+C`).
3. Su Moodle, crea l'attività VPL, accedi all'editor HTML, incolla il contenuto (`Ctrl+V`) e salva.
4. Importa i file `.cases` dalla cartella `all/capXX/casos/` nelle impostazioni dei test VPL.

Riutilizzo dei Casi di Test

I file `.cases` sono compatibili con VPL, consentendo di utilizzare lo stesso set di test sia nello sviluppo locale che nella correzione automatizzata su Moodle.

Parte I

Parte I — Fondamenti di EAI

Capitolo 1

Fondamenti e Primi Passi

Executar Colab Abrir GitHub

Questo capitolo apre la **Parte 1** del libro, dedicata ai fondamenti dell'Elaborazione Digitale delle Immagini (EDI). Vengono presentati la rappresentazione matematica delle immagini digitali e i principali metodi per la loro manipolazione.

Si utilizza il linguaggio Python e la libreria `morph.py` (ZAMPIROLLI, 2025).

1.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- Comprendere la natura fisica e matematica dell'immagine digitale $f(x, y)$.
- Identificare le bande dello spettro elettromagnetico rilevanti per l'elaborazione digitale delle immagini (PDI).
- Eseguire operazioni di base: lettura, visualizzazione e salvataggio delle immagini.
- Accedere e modificare le intensità dei pixel singolarmente.
- Applicare la sogliatura manuale.
- Configurare l'ambiente di sviluppo in Python.
- Gestire strutture di array (NumPy) senza incorrere in trappole di memoria.

1.2 Prima di iniziare: *Notebooks* interattivi

Questo materiale è stato realizzato secondo il concetto di ***Literate Programming*** (Programmazione Letteraria), ideato da Donald Knuth negli anni '80 (KNUTH, 1984). Knuth — anche creatore del sistema **TeX** per la tipografia digitale — propose che i programmi fossero scritti come una narrazione logica per gli esseri umani, alternando codice e documentazione.

Per eseguire una cella, premere Shift + Invio oppure fare clic sul pulsante ▷.

> **Nota sul formato**
>
> Nelle versioni renderizzate (PDF o HTML), il codice è presentato in blocchi statici a scopo di lettura e riferimento. L'esecuzione interattiva richiede l'accesso tramite Google Colab (disponibile in cima alla pagina) o in ambiente locale con VSCode o Jupyter Notebook.

1.3 Fondamenti

Lo studio dei sistemi basati su immagini comprende un ecosistema di discipline integrate che trasformano dati visivi grezzi in conoscenza strutturata. Mentre alcune aree si concentrano sulla generazione di rappresentazioni,

altre si dedicano al trattamento e all'analisi di tali dati per supportare applicazioni tecnologiche complesse.

Il diagramma presentato in Figura 1.1 stabilisce la distinzione e la complementarità tra l'Elaborazione Digitale delle Immagini (EDI) e la Visione Artificiale (VA). L'EDI, evidenziata in **verde**, si concentra sulla trasformazione da immagine a immagine, mirando al miglioramento della qualità o alla pre-elaborazione, come la rimozione del rumore e l'accentuazione del contrasto.

Al contrario, la VA, contrassegnata in **blu**, si concentra sull'interpretazione del contenuto visivo per estrarre modelli o informazioni, come il riconoscimento di oggetti e gesti. La regione di intersezione illustra la sinergia tra le aree, dove l'EDI prepara i dati visivi per l'interpretazione da parte della VA. La mappa dimostra inoltre le interconnessioni di entrambe le discipline con aree quali la Robotica, la Grafica Computerizzata, l'Intelligenza Artificiale (IA) e le Neuroscienze.

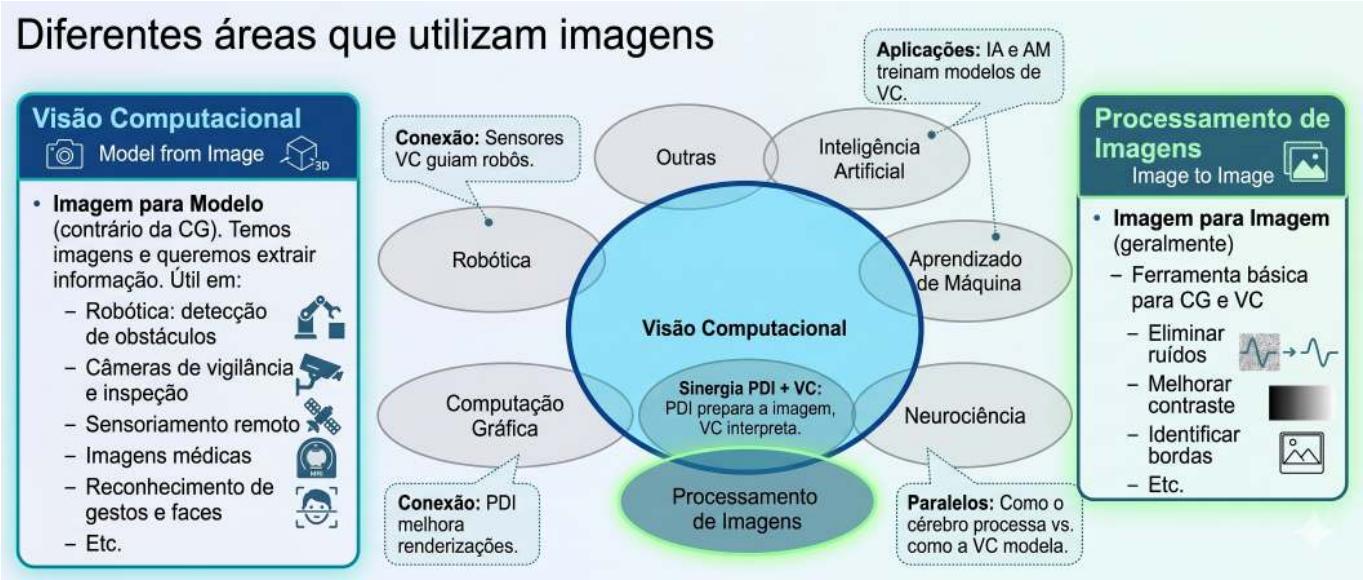


Figura 1.1: Diagramma relazionale che illustra le distinzioni fondamentali, le sinergie e le interconnessioni tra EDI e VA nel contesto dei sistemi basati su immagini.

1.3.1 Visione Computazionale

- **Focus:** *Immagine* → *Modello* (percorso inverso rispetto alla Computer Grafica).
- **Obiettivo:** Estrarre informazioni di alto livello da immagini o video.
- **Applicazioni tipiche:**
  - Robotica: rilevamento di ostacoli, localizzazione e navigazione autonoma.
  - Sorveglianza e ispezione: riconoscimento di eventi, lettura di targhe, controllo qualità.
  - Telerilevamento: analisi di immagini satellitari, mappatura ambientale.
  - Imaging medico: rilevamento di tumori, segmentazione di organi, supporto alla diagnosi.
  - Interazione uomo-computer: riconoscimento di gesti, espressioni facciali, eye tracking.
- **Relazione con altri ambiti:** utilizza tecniche di Apprendimento Automatico e IA per classificare e interpretare le scene; funge da "occhi" della Robotica.

1.3.2 Elaborazione Digitale delle Immagini (PDI)

- **Obiettivo:** *Immagine* → *Immagine* (generalmente - trasformazione di un'immagine in un'altra).

- **Scopo:** Migliorare la qualità visiva o estrarre caratteristiche di basso livello.

- **Applicazioni comuni:**
  - Rimozione del rumore (filtri media, mediana, gaussiano).
  - Miglioramento del contrasto (equalizzazione dell'istogramma, regolazione gamma).

  - Rilevamento dei bordi (Sobel, Canny, Laplaciano).
  - Segmentazione (sogliatura, crescita delle regioni, *watershed*).
  - Trasformazioni geometriche (ridimensionamento, rotazione, correzione prospettica).
- **Relazione con altre aree (vedi Tabella 1.1):**
  - È la **base** per la maggior parte dei sistemi di visione artificiale (pre-elaborazione).
  - La Computer Grafica applica spesso PDI per la post-elaborazione (es.: smussatura, miglioramento).
  - Le tecniche di IA possono ottimizzare i parametri di elaborazione (es.: apprendimento di filtri).

Tabella 1.1: Connessione tra PDI, VC e altre aree della scienza.

Área	Relação com PDI e VC
Intelligenza Artificiale	Fornisce modelli (reti neurali, SVM) che interpretano le uscite della VC.
Robotica	Consuma dati di VC per prendere decisioni (navigazione, manipolazione).
Apprendimento Automatico	Utilizza descrittori estratti da PDI/VC per addestrare classificatori.
Computer Grafica	Percorso inverso: *modello* → *immagine*; spesso applica PDI per una resa realistica.
Neuroscienze	Ispira modelli di PDI (es.: filtri simili alle cellule gangliari della retina).

1.4 Fasi del PDI

Le fasi del PDI sono presentate nella Figura 1.2 e possono essere interpretate come una catena di trasformazioni che riduce la ridondanza dei dati per ricercarne il significato:

- **Basso Livello:** opera direttamente sui pixel dell'immagine disturbata per apportare miglioramenti e filtraggi, producendo come output un'immagine pulita o esaltata.
- **Medio Livello:** riceve l'immagine trattata ed esegue la segmentazione e la descrizione, trasformando la matrice dei pixel in attributi strutturati (forma, dimensione e texture).
- **Alto Livello:** utilizza la tabella degli attributi per alimentare processi di logica e intelligenza artificiale, portando alla decisione o al riconoscimento finale (come la diagnosi medica).

La Figura 1.3 descrive l'intera sequenza dell'elaborazione delle immagini digitali (PDI), dall'acquisizione all'interpretazione. Il flusso inizia con l'Acquisizione dell'Immagine (1) e prosegue con il Miglioramento (2) e il Restauro (3). Successivamente, il contenuto viene isolato tramite la Segmentazione (4) e raffinato mediante la Morfologia (5). La transizione cruciale avviene nella Rappresentazione e Descrizione (6), dove gli oggetti visivi vengono convertiti in dati matematici (area, perimetro, ecc.), consentendo il Riconoscimento (7). I processi ausiliari includono l'Elaborazione dell'Immagine a Colori e la Compressione, che contribuiscono all'efficienza dell'archiviazione e dell'analisi.

1.5 Formazione dell'Immagine e Spettro

Il processo di formazione di un'immagine si basa sull'interazione tra materia ed energia radiante. Essenzialmente, un'immagine viene concepita quando un **sensore registra la radiazione** risultante dall'interazione con un oggetto fisico. Nel contesto della visione umana e della fotografia convenzionale, questo fenomeno dipende da una sorgente luminosa che illumina la scena; le caratteristiche degli oggetti vengono quindi codificate attraverso le **variazioni di intensità e colore** della luce che raggiunge il sensore, come illustrato nella Figura 1.4.

La **luce visibile** occupa solo una piccola banda dello spettro elettromagnetico — tra **380 nm (viola)** e **750 nm (rosso)** — come illustrato nella Figura 1.5. I sensori digitali convenzionali operano in questa stessa finestra, ma apparecchiature specializzate possono captare radiazioni invisibili all'occhio umano, come

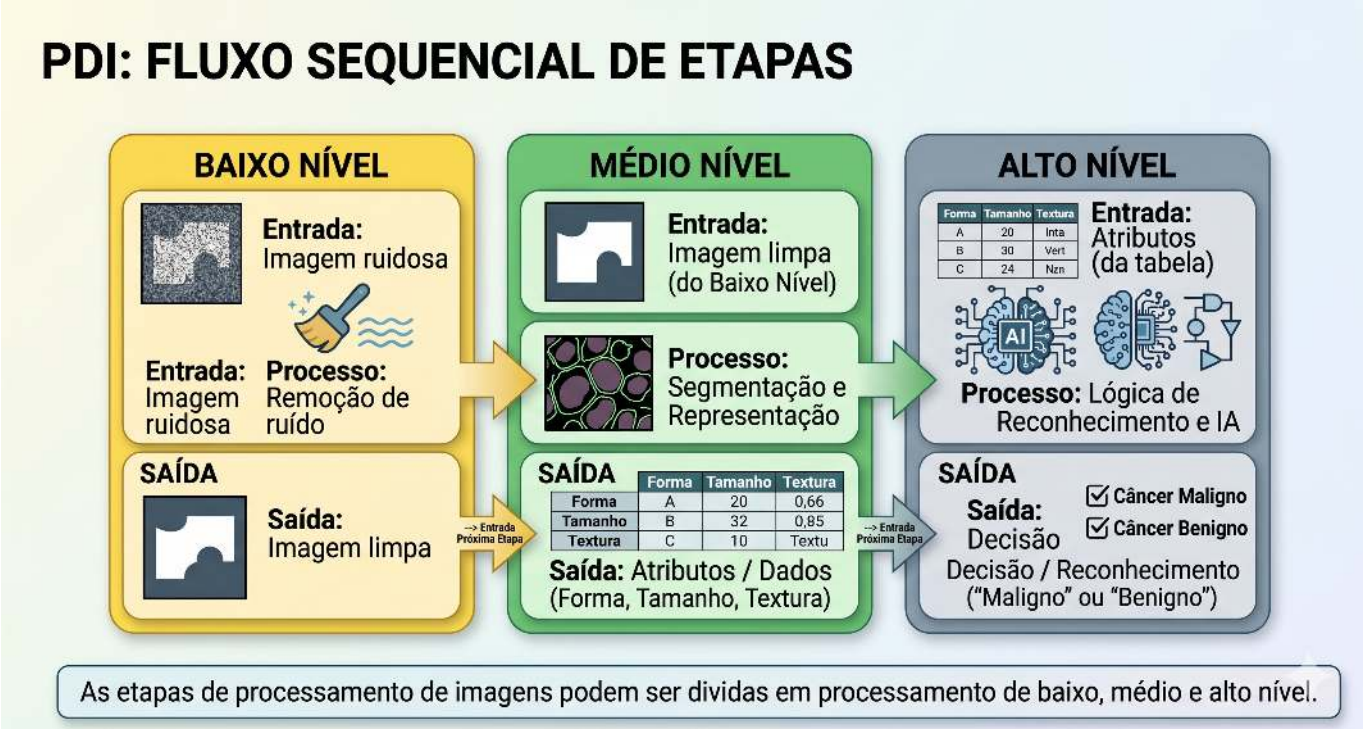


Figura 1.2: Rappresentazione del flusso sequenziale di elaborazione: l'output di ciascun livello diventa l'input del livello successivo.

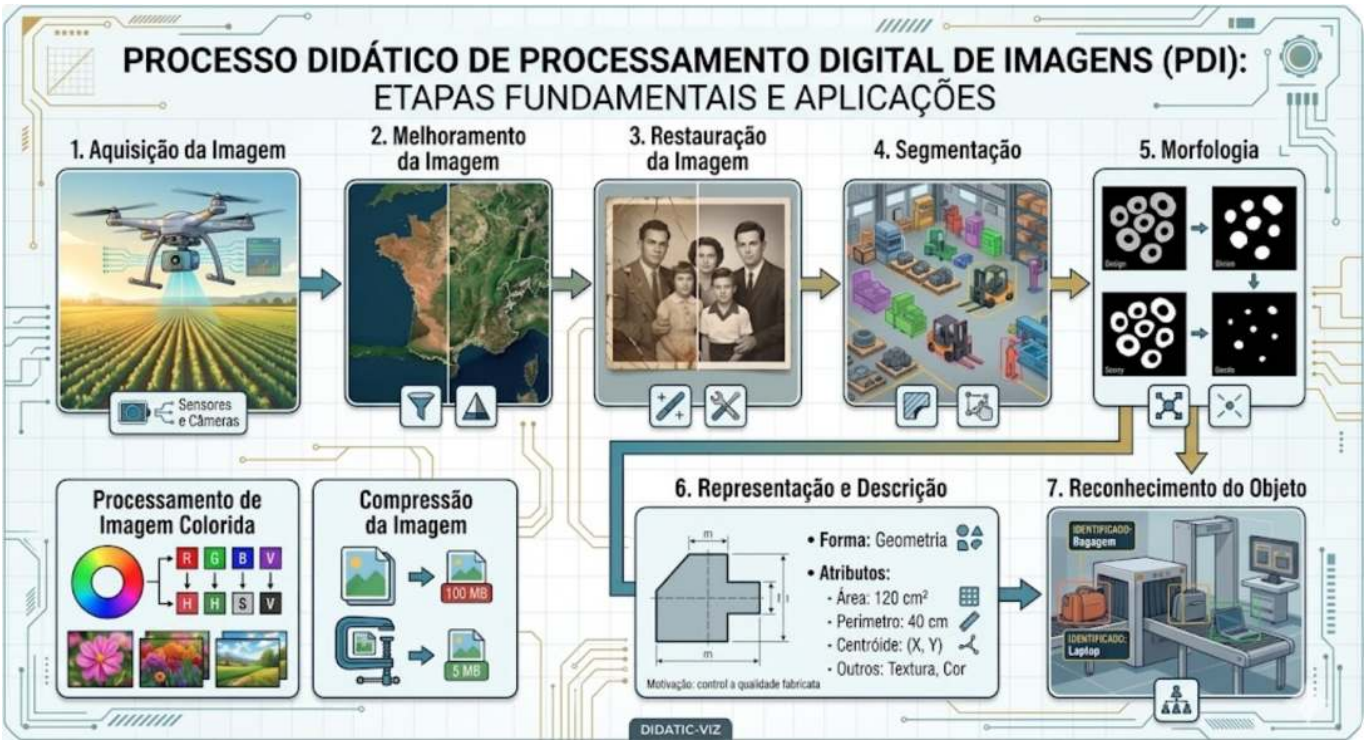


Figura 1.3: Flusso dettagliato del PDI: dall'acquisizione sensoriale all'estrazione di attributi e al riconoscimento automatizzato, includendo l'elaborazione a colori e la compressione.

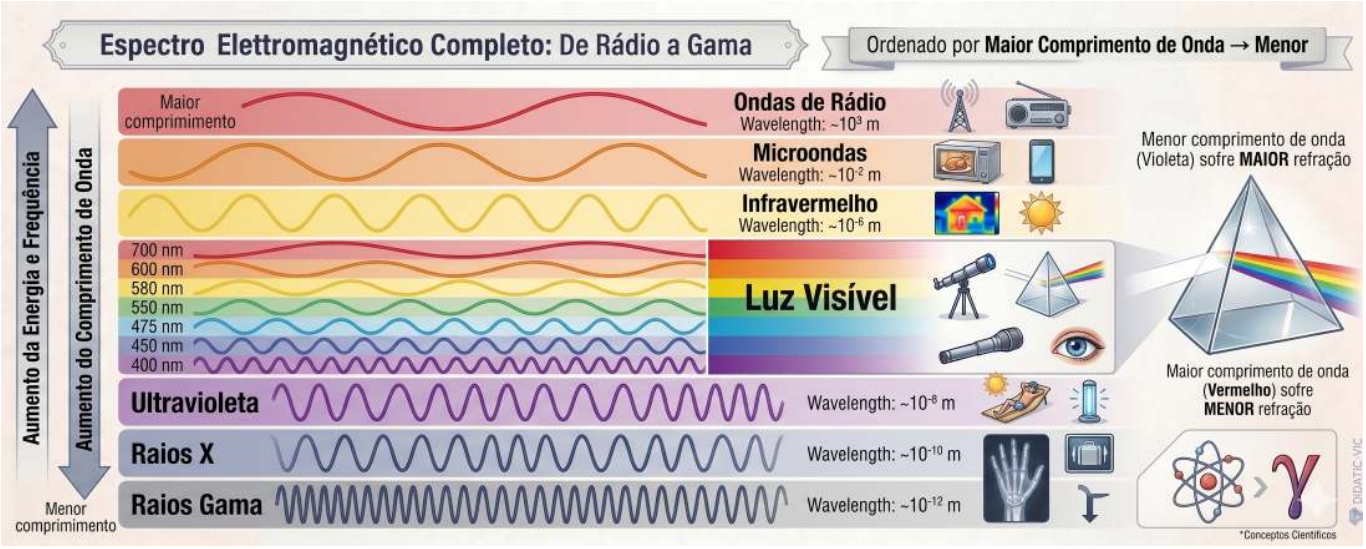


Figura 1.4: Rappresentazione dello spettro visibile e della sua posizione rispetto alle altre radiazioni elettromagnetiche, evidenziando la variazione delle lunghezze d'onda da 380 nm a 750 nm.

l'infrarosso e i raggi X. In PDI, l'immagine formata dipende direttamente dalla sensibilità spettrale del sensore utilizzato.

A partire da questo processo fisico di acquisizione, diventa possibile modellare matematicamente l'immagine digitale come una funzione bidimensionale discreta, nella quale ogni punto della scena è rappresentato da campioni numerici di intensità luminosa, formalizzando così i concetti di pixel e di immagine digitale presentati nella sezione successiva.

1.6 Cos'è un'Immagine Digitale?

Un'**immagine digitale** è formata da una griglia di **pixel** (*Picture Elements*), dove ogni pixel è la più piccola unità elementare dell'immagine.

> Cos'è un Pixel?
>
> Un **pixel** è la più piccola unità indirizzabile che compone un'immagine digitale. Ogni pixel occupa una posizione unica nella griglia e memorizza uno o più valori numerici che rappresentano la sua intensità o colore.

Rappresentazione Matematica

A differenza di una funzione continua, il dominio di un'immagine digitale è un piano rettangolare finito $\mathbb{E} \subset \mathbb{Z}^2$, che rappresenta la griglia di campionamento. Questo dominio è indicizzato da coordinate intere:

$$\mathbb{E} = \{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x < L,\ 0 \leq y < H\} \tag{1.1}$$

Dove:

- L: rappresenta la larghezza dell'immagine (numero di colonne).
- H: rappresenta l'altezza dell'immagine (numero di righe).

L'immagine digitale è una funzione che associa a ogni coppia di coordinate (x, y) uno o più valori che descrivono l'aspetto del pixel.

$$f : \mathbb{E} \to \mathcal{V} \tag{1.2}$$

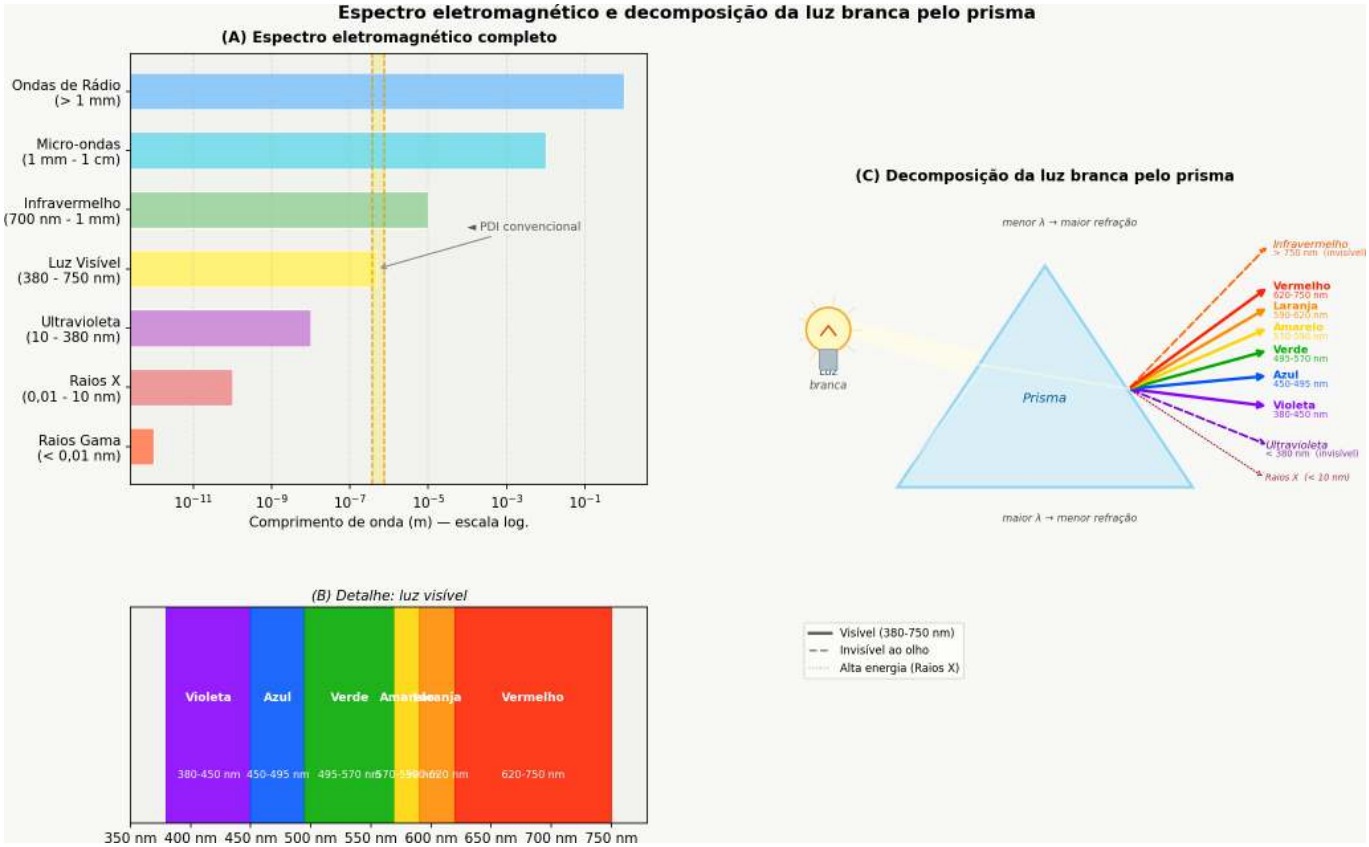


Figura 1.5: (A) Spettro elettromagnetico completo in scala logaritmica, con evidenza della fascia visibile. (B) Dettaglio della luce visibile (380-750 nm) e dei suoi colori. (C) Scomposizione della luce bianca tramite prisma: la lunghezza d'onda minore subisce una rifrazione maggiore, separando UV, visibile e infrarosso.

L'insieme $\mathcal{V}$ definisce i valori possibili per il pixel (codominio), variando a seconda del tipo di immagine, come dimostrato nella Tabella 1.2.

Tabella 1.2: Principali tipi di immagine digitale e i rispettivi insiemi di valori possibili per ogni pixel.

Tipo di immagine	$\mathcal{V}$ (valori del pixel)	Rappresentazione
Binaria	$\{0, 1\}$ o $\{0, 255\}$	■□
Scala di grigi	$\{0, 1, \ldots, 255\}$	`[ ][=][#]`■
A colori (RGB)	$\{0, \ldots, 255\}^3$ (terne ordinate di valori)	■■■

Esempio pratico: Un'immagine a colori nel modello RGB può essere rappresentata matematicamente da una funzione che associa tre valori di intensità a ogni pixel. Computazionalmente, questa rappresentazione corrisponde a tre matrici sovrapposte — i canali rosso (*Red*), verde (*Green*) e blu (*Blue*) — nelle quali ogni elemento memorizza l'intensità luminosa del rispettivo canale in una determinata posizione dell'immagine.

Per rendere possibili gli esperimenti pratici di PDI e VC, questo libro utilizza l'ecosistema scientifico di Python, integrando librerie orientate alla manipolazione matriciale, all'elaborazione delle immagini e alla visualizzazione dei risultati, presentate di seguito.

1.7 Configurazione dell'Ambiente

Questo materiale utilizza l'ecosistema scientifico di Python, con particolare attenzione a **NumPy** (calcolo matriciale), **OpenCV** (visione computazionale) e alla libreria **morph.py** (ZAMPIROLLI, 2025), sviluppata per scopi didattici e utilizzata lungo tutto questo libro, come presentato nella Tabella 1.3.

Il progetto è attualmente disponibile in **Python** e **Portoghese**. La sua organizzazione, tuttavia, consente l'espansione ad altri linguaggi di programmazione e idiomi.

Gli **Esercizi di Programmazione (EP)** presentati alla fine dei capitoli possono essere validati dal modulo `testsuite.py`, che consente di eseguire casi di test in diversi linguaggi, tra cui Python, C, C++, Java, JavaScript e R. In questo modo, gli stessi casi di test possono essere utilizzati per verificare le soluzioni sviluppate nei *notebook* e nel **Moodle/VPL**.

Tabella 1.3: Principali librerie e moduli utilizzati in questo libro per il calcolo matriciale, PDI, VC, visualizzazione e validazione degli esercizi.

Libreria	Funzione principale
`numpy`	Rappresentazione matriciale e operazioni numeriche
`opencv-python`	Lettura, scrittura e operazioni di visione computazionale
`matplotlib`	Visualizzazione di immagini e grafici
`morph.py`	Astrazione didattica delle operazioni di PDI
`testsuite.py`	Esecuzione e validazione automatica degli EP

1.8 Versioni di `morph.py`

La libreria `morph.py` ha due versioni pubbliche:

- **Versione 1.0** — versione originale, pubblicata insieme all'articolo presentato all'EduComp 2024: repository della versione 1.0
- **Versione 1.1** — versione utilizzata nelle attività di questo libro: repository della versione 1.1

La versione 1.1 è stata adattata per soddisfare anche i vincoli di memoria del **Moodle/VPL** (Laboratorio Virtuale di Programmazione). Nella versione 1.0, alcune librerie, come `matplotlib`, `requests` e `skimage`, venivano caricate al momento dell'`import morph`. In ambienti con risorse limitate, questo caricamento poteva superare il limite di memoria e produrre errori come `Jail: out of memory, 128MiB`.

Nella versione 1.1, questi *import* sono stati sostituiti da **caricamento *lazy***: ogni libreria viene importata solo dal metodo che effettivamente la utilizza. Così, il comando `import morph` mantiene il caricamento iniziale ridotto, utilizzando principalmente `numpy` e `cv2`, mentre le risorse aggiuntive vengono caricate su richiesta.

La configurazione dei *notebook* è centralizzata nel file `config.py`, disponibile nello stesso repository. Questo file verifica le dipendenze, cerca di garantire l'uso di **OpenCV 5.0** e mette a disposizione `morph.py` e, quando necessario, `testsuite.py`. Quando l'ambiente non dispone della versione adeguata di OpenCV, `config.py` può reinstallare la versione richiesta e, se necessario, richiedere il riavvio dell'ambiente di esecuzione.

In questo modo, i *notebook* utilizzano un'unica routine di configurazione, riducendo la ripetizione di codice e facilitando la riproduzione dell'ambiente computazionale utilizzato in questo libro.

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

1.9 Fondamenti di Matrici — attenzione alla copia di riferimenti

Poiché un'immagine digitale può essere rappresentata da una matrice, è importante comprendere come creare e manipolare correttamente le matrici. In Python, esiste una trappola comune nell'utilizzo dell'operatore `*` con le liste:

> ⚠ Attenzione alla copia di riferimenti
>
> Eseguendo `m = [[0]*2]*3`, non vengono create tre righe indipendenti. Vengono invece create tre riferimenti alla **stessa lista**. Di conseguenza, la modifica di un elemento in una delle righe modificherà anche l'elemento corrispondente nelle altre righe.

Per visualizzare questo comportamento, si può eseguire il codice su Python Tutor e confrontarlo con il metodo corretto per creare una matrice utilizzando le liste: `m = [[0]*2 for _ in range(3)]`.

In pratica, per l'elaborazione digitale delle immagini (EDI), si raccomanda di utilizzare **NumPy**, che fornisce strutture dati multidimensionali efficienti e operazioni vettorizzate adatte alla manipolazione delle immagini. Il codice seguente presenta diversi modi di creare immagini sintetiche, i cui risultati sono mostrati nella Figura 1.6.

```
import numpy as np

# Creando un'immagine nera (zeri) di 4, 6 pixel
img_preta = np.zeros((4, 6), dtype='uint8')
img_preta[0,0] = 255  # pixel bianco nell'angolo in alto a sinistra

# Creando un'immagine bianca (255) di 4, 6 pixel
```

```
img_branca = np.ones((4, 6), dtype='uint8') * 255
img_branca[3,5] = 0  # pixel nero nell'angolo in basso a destra

# Creando un'immagine casuale per test (rumore)
img_random = mm.randomImage(4, 6, maxValue=255)

print("Matrice casuale generata:")
print(mm.drawImg(img_random))

mm.show(
    [img_preta, img_branca, img_random],
    titles=[
        "Prevalentemente nera\n(con 1 pixel bianco in (0,0))",
        "Prevalentemente bianca\n(con 1 pixel nero in (3,5))",
        "Immagine casuale\n(simulazione di rumore)"
    ],
    cols=3,
    figsize=(9, 3),
    axis=True
)
```

```
Matrice casuale generata:
 20  66  67 220 200  44
132  33  74   0 156   2
 78   7 211 213  34 207
194  68 103  10 180 152
```

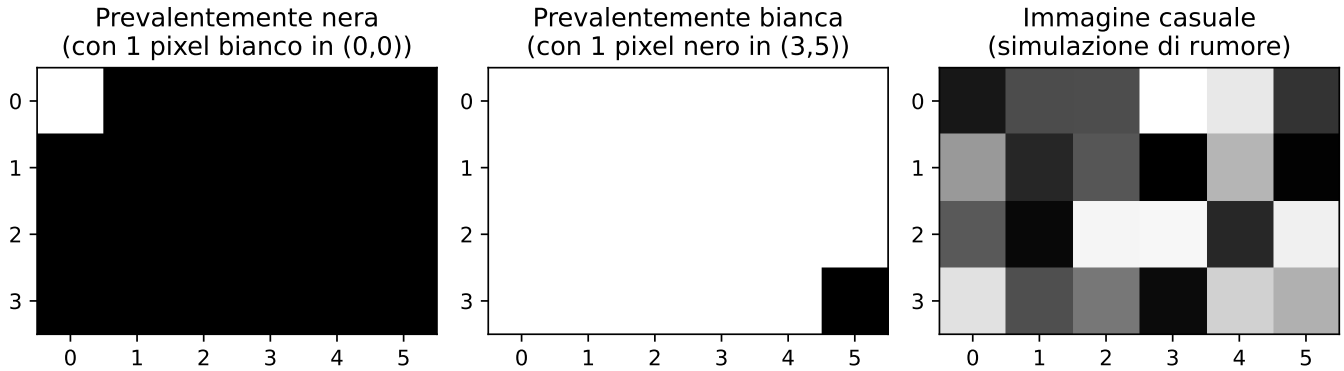


Figura 1.6: Esempi di immagini sintetiche rappresentate matricialmente.

1.10 Lettura e Visualizzazione delle Immagini

Nelle librerie come NumPy, OpenCV e scikit-image, le immagini digitali sono spesso rappresentate computazionalmente come *array* di NumPy. In questo modo, operazioni matriciali e concetti di algebra lineare possono essere applicati direttamente all'elaborazione digitale delle immagini (EDI).

Una delle operazioni fondamentali nell'elaborazione delle immagini digitali è la lettura delle immagini.

Nella libreria `morph.py`, la funzione `mm.read()` consente di caricare immagini sia da file locali che da URL, come illustrato nella Figura 2.7..

```
import os
import numpy as np

url     =
"https://upload.wikimedia.org/wikipedia/commons/8/87/Mandrillus_sphinx_339428057.jpg"
```

```
caminho = "imagens/mandrill.png"

if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    img_obj = mm.read(url, pil=True)
    mm.write(img_obj, caminho)
else:
    img_obj = mm.read(caminho, pil=True)

img = np.array(img_obj)

print(f"Dimensioni (H, W, Canali): {img.shape}")
print(f"Tipo di dato: {img.dtype}")

mm.show(img, title="Esempio: Cattura RGB")
```

```
Dimensioni (H, W, Canali): (1365, 2048, 3)
Tipo di dato: uint8
```

Figura 1.7: Mandrill (Mandrillus sphinx) in ambiente naturale. Credito: Julien Renoult (CC BY 4.0).

Alternativa: *Download* dell'Immagine in Archiviazione Locale

In ambienti in cui la lettura diretta degli URL non è disponibile — a causa di restrizioni di rete, policy di *firewall* o assenza di connettività — un'alternativa consiste nell'eseguire preventivamente il *download* dell'immagine sul file system locale. Dopo l'archiviazione, l'immagine può essere caricata normalmente tramite la funzione `mm.read()`. Nell'esempio seguente, il file viene salvato localmente con il nome `mandrill.png`.

```
!wget -O mandrill.png \
    https://upload.wikimedia.org/wikipedia/commons/8/87/Mandrillus_sphinx_339428057.jpg
```

```
img = mm.read('mandrill.png')
mm.show(img, title="Esempio: Acquisizione RGB (file locale)")
```

Spiegazione

- `wget -O mandrill.png <URL>` esegue il *download* dell'immagine e la archivia localmente con il nome specificato;
- `mm.read('mandrill.png')` effettua la lettura del file direttamente dal file system, senza la necessità di ulteriori richieste HTTP;
- questa strategia riduce la dipendenza dalla connettività durante l'esecuzione degli esperimenti ed evita *download* ripetuti della stessa immagine.

Nota

Il prefisso `!` viene utilizzato in ambienti basati su *notebook*, come Jupyter Notebook, JupyterLab e Google Colab, per eseguire comandi del sistema operativo direttamente in celle di codice. Su terminali convenzionali, il comando deve essere utilizzato senza il prefisso `!`.
Qualora l'utilità `wget` non sia installata, si può utilizzare in alternativa:

```
!curl -o mandrill.png \
    https://upload.wikimedia.org/wikipedia/commons/8/87/Mandrillus_sphinx_339428057.jpg
```

1.11 Conversione di Tipi e Soglia

Come abbiamo visto, un'immagine a colori nello spazio RGB è rappresentata dalla funzione:

$$f : \mathbb{E} \to \{0, 1, \dots, 255\}^3$$

Cioè, per ogni pixel (x, y), abbiamo tre valori (R, G, B) che ne definiscono il colore.

Conversione in Scala di Grigi

Per convertire un'immagine RGB in scala di grigi (*grayscale*), è necessario combinare i tre canali in un unico valore di intensità g, che rappresenta la luminosità percepita. Poiché l'occhio umano non è ugualmente sensibile al rosso, al verde e al blu, si utilizza una **media ponderata**. Lo standard ITU-R BT.601 ({ITU-R}, 2011) definisce i seguenti pesi:

$$g = 0.299\,R + 0.587\,G + 0.114\,B \tag{1.3}$$

Dopo il calcolo, il valore g viene arrotondato all'intero più vicino e adattato all'intervallo $[0, 255]$. Il risultato è una nuova immagine, ora in scala di grigi, rappresentata da:

$$f_{\text{grigi}} : \mathbb{E} \to \{0, 1, \dots, 255\}$$

Soglia (*Thresholding*)

A partire dall'immagine in scala di grigi $f_{\text{grigi}}(x, y)$, un'operazione fondamentale è la **soglia**, che produce un'immagine **binaria** (solo bianco e nero). A tale scopo, si sceglie un valore di taglio T (generalmente nell'intervallo $[0, 255]$) e si definisce:

$$f_{\text{bin}}(x, y) = \begin{cases} 255 & \text{se } f_{\text{grigi}}(x, y) > T \\ 0 & \text{altrimenti} \end{cases} \tag{1.4}$$

Esempio: Con $T = 128$, i pixel con intensità superiore a 128 diventano bianchi (255); gli altri diventano neri (0).

La soglia è ampiamente utilizzata per **segmentare gli oggetti dallo sfondo**, estrarre i bordi o creare maschere binarie per l'elaborazione successiva.

> **Nota:** Il valore 255 rappresenta il bianco massimo nelle immagini a 8 bit, mentre 0 rappresenta il nero assoluto.

Esempio pratico di conversione e soglia

La Figura 1.8 illustra i passaggi principali per trasformare un'immagine a colori in scala di grigi e, successivamente, convertirla in un'immagine binaria mediante soglia. Il codice seguente implementa questi passaggi:

```
# 1. Converti in Scala di Grigi
img_gray = mm.gray(img)

# 2. Applica soglia (Pixel > 128 diventano 255, altri 0)
limiar = 128
img_binaria = mm.threshold(img_gray, limiar)

# Uso della nuova funzione
mm.show(
    [img, img_gray, img_binaria],
    titles=["Originale", "Scala di Grigi", f"Binaria (T={limiar})"],
    cols=3
)
```

Figura 1.8: Elaborazione base delle immagini: (a) immagine originale, (b) immagine in scala di grigi, (c) immagine binarizzata con soglia (T=128).

1.12 Limiarizzazione con il metodo di Otsu

Come presentato in Equazione 1.4, la limiarizzazione converte un'immagine in toni di grigio in un'immagine binaria utilizzando un valore di soglia T. Finora abbiamo fissato manualmente $T = 128$.

```
img_bin_fixo = mm.threshold(img_gray, T=128)
```

Tuttavia, la scelta manuale di T non è sempre banale. La libreria `mm` offre un'alternativa automatica: quando il parametro `limiar` **non viene fornito**, la funzione `mm.threshold(img_gray)` calcola il valore di T mediante il **metodo di Otsu** (OTSU, 1979). Questo metodo, che verrà dettagliato nei capitoli successivi, massimizza la varianza tra le classi dell'istogramma (frequenza di ciascun tono di grigio), separando automaticamente i pixel dell'oggetto e dello sfondo.

Il codice seguente confronta la limiarizzazione manuale ($T = 128$) con quella automatica (Otsu), mostrando anche il valore di T calcolato:

```
import cv2
# Sogliatura con T fisso (manuale)
T_fixo = 128
img_bin_fixo = mm.threshold(img_gray, T_fixo)

# Sogliatura con il metodo di Otsu (T automatico)
T_otsu, img_bin_otsu = cv2.threshold(img_gray, 0, 255,
                                     cv2.THRESH_BINARY + cv2.THRESH_OTSU)
print(f'Soglia calcolata da Otsu: T = {T_otsu}')
# o semplicemente:
# img_bin_otsu = mm.threshold(img_gray)

# Visualizzazione affiancata
mm.show(
    [img_gray, img_bin_fixo, img_bin_otsu],
    titles=["Scala di Grigi", f"Binaria (T={T_fixo})", f"Binaria (Otsu, T={T_otsu})"],
    cols=3
)
```

```
Soglia calcolata da Otsu: T = 95.0
```

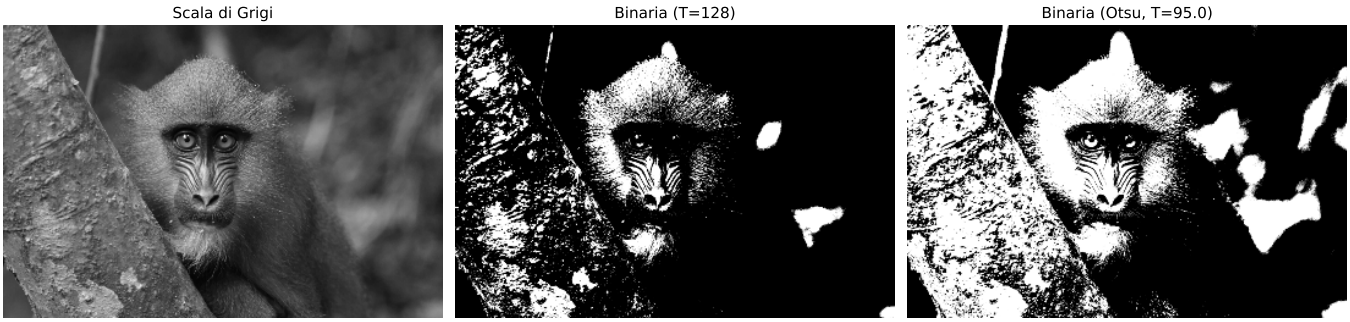


Figura 1.9: Confronto tra sogliatura manuale (T=128) e automatica (Otsu) sull'immagine in scala di grigi.

La Figura 1.9 mostra che la soglia ottenuta mediante Otsu si adatta automaticamente all'immagine, producendo una binarizzazione più efficace rispetto a un valore fisso, soprattutto quando le intensità dell'oggetto e dello sfondo sono ben separate nell'istogramma. Questa tecnica è ampiamente utilizzata nei sistemi di visione artificiale (VC) per la binarizzazione di documenti, il rilevamento di oggetti e il pre-processing delle immagini.

Semplicità della libreria `morph.py`

Mentre OpenCV richiede la chiamata completa:

```
T_otsu, img_bin = cv2.threshold(img_gray, 0, 255,
                                cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

la libreria `mm` astrae tutta questa complessità: basta chiamare `mm.threshold(img_gray)`. La soglia di Otsu viene calcolata automaticamente e l'immagine binaria viene restituita direttamente. Questo approccio consente di concentrarsi sul concetto, non sui dettagli implementativi.

1.13 Accesso ai Pixel

In Python con NumPy, un'immagine è strutturata come un **array multidimensionale**. L'accesso a un pixel specifico utilizza la convenzione matriciale **riga** (asse Y) e **colonna** (asse X): `img[riga, colonna]`.

Il codice della Figura 1.10 dimostra come estrarre questi valori in immagini a colori (RGB) e in scala di grigi, oltre a isolare l'intorno immediato del punto di interesse.

```
import matplotlib.pyplot as plt

# 1. Definizione delle coordinate del pixel target
r, c = 600, 800

# 2. Accesso diretto ai valori dei pixel
pixel_cinza = img_gray[r, c]
pixel_rgb   = img[r, c]

print(f"  VALORI NEL PIXEL TARGET ({r}, {c}):")
print(f"   • Toni di grigio (Scalare) : {pixel_cinza}")
print(f"   • Colorato (Vettore RGB)    : R={pixel_rgb[0]}, G={pixel_rgb[1]},
B={pixel_rgb[2]}")
print("-" * 50)

# 3. Visualizzazione didattica dell'intorno 3x3 attorno al pixel (r, c)
# Ritaglia dalla riga r-1 alla r+1, e dalla colonna c-1 alla c+1
vizinhanca_gray = img_gray[r-1:r+2, c-1:c+2]

print(f"  MATRICE DELL'INTORNO 3x3 IN TONI DI GRIGIO:")
print(f"   (Il pixel target centrale è evidenziato con tra parentesi quadre)\n")

# Stampa formattata per evidenziare il pixel centrale
for i, linha in enumerate(vizinhanca_gray):
    linhas_str = []
    for j, valor in enumerate(linha):
        if i == 1 and j == 1:
            linhas_str.append(f"[{valor:3d}]") # Evidenzia il centro (100, 100)
        else:
            linhas_str.append(f" {valor:3d} ")
    print("   " + " ".join(linhas_str))

# 4. Dimostrazione visiva usando il mm.show (opzionale, ma eccellente per il libro)
# Mostra l'immagine intera con un punto che delimita la regione
plt.figure(figsize=(5, 5))
plt.imshow(img)
plt.plot(c, r, 'ro', markersize=8, label=f'Pixel ({r},{c})') # Traccia (x, y) -> (c, r)
plt.title(f"Localização do Pixel ({r}, {c}) na Imagem")
plt.legend()
plt.axis('on') # Mantiene gli assi per lo studente per vedere le coordinate 100, 100
plt.show()
```

```
  VALORI NEL PIXEL TARGET (600, 800):
   • Toni di grigio (Scalare) : 81
   • Colorato (Vettore RGB)    : R=92, G=78, B=67
--------------------------------------------------
  MATRICE DELL'INTORNO 3x3 IN TONI DI GRIGIO:
   (Il pixel target centrale è evidenziato con tra parentesi quadre)

     83    96   105
     85  [ 81]    96
     83    77    84
```

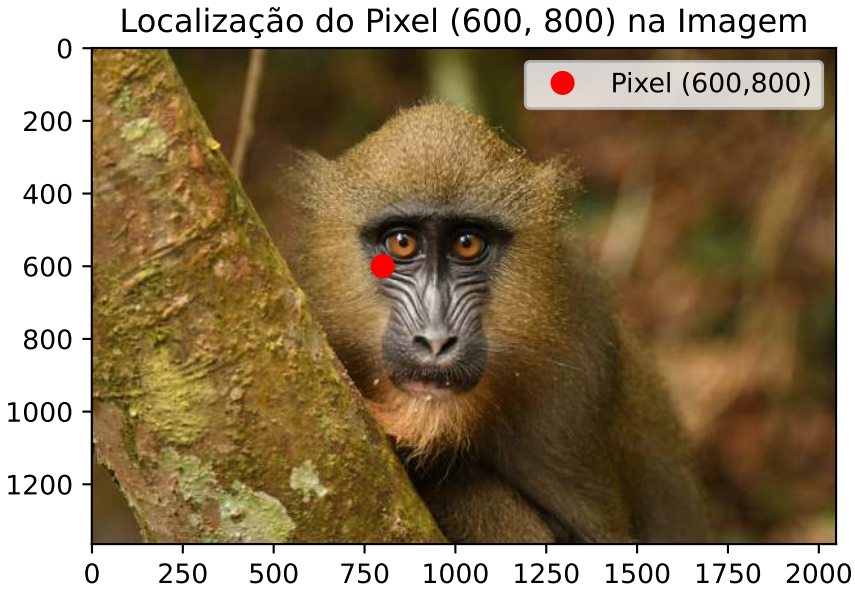


Figura 1.10: Accesso a un pixel specifico (r, c) e la rappresentazione del suo intorno nell'immagine.

Spiegazione riga per riga:

1. `img_gray[r, c]` — Restituisce un singolo valore intero (scalare) compreso tra 0 e 255, che rappresenta l'intensità di luminosità del grigio.
2. `img[r, c]` — Restituisce un vettore con tre valori `[R, G, B]`, corrispondenti alle intensità dei canali Rosso, Verde e Blu.
3. `img_gray[r-1:r+2, c-1:c+2]` — Esegue un'affettatura (*slicing*) per estrarre la sottomatrice 3×3 attorno al pixel. Questa operazione è alla base per l'implementazione di filtri spaziali e convoluzioni.
4. Il ciclo `for` successivo formatta semplicemente tale sottomatrice nella console, mostrando il pixel centrale tra parentesi quadre `[ ]` a scopo didattico.
5. La funzione `plt.plot(c, r, 'ro')` traccia un punto rosso sull'immagine per correlare la coordinata numerica con la sua posizione visiva. Si noti che Matplotlib utilizza l'ordine predefinito dello schermo `(X, Y)`, invertendolo in `(c, r)`.

Poiché l'indicizzazione in Python inizia da zero (*zero-based*), l'angolo superiore sinistro dell'immagine è la coordinata `(0,0)`. Ricorda sempre questa regola: la prima dimensione della matrice controlla l'**altezza** (righe/Y) e la seconda controlla la**larghezza** (colonne/X).

1.14 Riassunto

In questo capitolo sono stati presentati i fondamenti della rappresentazione delle immagini digitali: la definizione di pixel, la strutturazione delle immagini in matrici e l'impatto del campionamento e della quantizzazione sulla qualità finale:

- **Immagine digitale** = funzione $f(x, y)$ che mappa coordinate in intensità (scalari o vettoriali).
- **Dominio**: insieme finito $\mathbb{E} = \{(x, y) \in \mathbb{Z}^2 \mid 0 \leq x < L,\ 0 \leq y < H\}$.
- **Tipi principali**: binaria ($\mathcal{V} = \{0, 255\}$), toni di grigio ($\mathcal{V} = [0, 255]$) e RGB ($\mathcal{V} = [0, 255]^3$).
- **Sogliatura** converte i toni di grigio in immagine binaria; il **metodo di Otsu** determina automaticamente il valore di soglia massimizzando la varianza tra le classi.
- La libreria `morph.py` (o `mm`) offre funzioni didattiche per operazioni di base dell'elaborazione delle immagini, come `mm.gray()`, `mm.threshold()`, `mm.show_multiple()`.
- **Insidia NumPy**: non usare mai `[[0]*n]*m` per creare matrici — usa sempre `np.zeros()` o `np.ones()`.

- Accesso ai pixel tramite `img[riga, colonna]`, con indicizzazione ***zero-based***.

Il Capitolo 2 tratterà **istogrammi ed equalizzazione del contrasto**.

1.15 🤖 Uso del Gemini Notebook come Tutor Complementare

In questa edizione, oltre ai *notebook* interattivi su Google Colab, il **Gemini Notebook** è disponibile come strumento complementare di studio. La piattaforma utilizza esclusivamente i documenti forniti dall'autore come base di conoscenza, garantendo risposte coerenti con il contenuto del libro.

🎓 Studia con il Tutor Intelligente

Accedi all'ambiente del capitolo tramite il *link* qui sotto ed esplora in particolare le opzioni **Guida allo Studio** e **Conversazione** per approfondire la tua comprensione.
🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 01

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato costruito esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, oppure seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ **Avvertenza sul Contenuto Generato dall'IA**

L'IA è un potente alleato nello studio, ma il contenuto generato può contenere **errori o imprecisioni**. Consulta sempre **libri, articoli scientifici e altre fonti accademiche affidabili** per validare le informazioni. Quando possibile, esegui gli esempi pratici forniti in questo capitolo per verificare i risultati.

Funzionalità Disponibili sulla Piattaforma

Il **Gemini Notebook** offre una suite avanzata di strumenti basati sull'IA per trasformare il contenuto statico del libro in un'esperienza di apprendimento dinamica e multimediale. La piattaforma utilizza tecniche di **RAG** (*Retrieval-Augmented Generation*), fondate sul lavoro di Lewis (2020), per basare le risposte strettamente sui documenti forniti e minimizzare il verificarsi di allucinazioni.

Le principali funzionalità includono:

- **Sintesi Multimodali (Audio e Video)**: Generazione di conversazioni naturali tra esperti nel formato di **Sintesi Audio** (stile *podcast*) e **Sintesi Video**, che discutono i temi centrali del capitolo, come le differenze tra PDI e VC, o l'interpretazione di trasformazioni come la sogliatura e il metodo di Otsu.
- **Visualizzazione di Strutture (Mappa Mentale e Infografica)**: Creazione automatica di diagrammi che collegano visivamente i concetti, ad esempio, il flusso di elaborazione dalla cattura dell'immagine digitale, passando per la conversione in toni di grigio, la sogliatura e la segmentazione binaria.
- **Strumenti di Valutazione (Test e Schede Didattiche)**: Generazione di **Test** a scelta multipla e **Schede Didattiche** (*flashcards*) per il consolidamento delle conoscenze, basati sul testo d'autore (es.: domande sulla formula di conversione RGB→grigio o sul funzionamento della soglia globale e di Otsu).
- **Supporto alla Presentazione (*Slide* e Relazioni)**: Assistenza nella strutturazione di **Presentazioni di *Slide*** e nella redazione di **Relazioni** tecniche, facilitando la comunicazione dei risultati di esperimenti con le immagini.
- **Analisi dei Dati (Tabella Dati)**: Organizzazione dei dati estratti dal testo in tabelle strutturate, aiutando la comprensione di esempi pratici, come il confronto tra diversi valori di soglia.
- ***Chat* Contestualizzata**: Consente di porre domande dirette sul codice e sulla teoria, come: *"Come implementare la conversione da RGB a toni di grigio utilizzando i pesi dello standard ITU-R BT.601?"*

o *"Cosa succede all'immagine binaria se scelgo una soglia T=200 invece di T=128?"*.

1.16 Elenco di Esercizi

1. (15%) Con le tue parole, definisci **immagine digitale** e **pixel**. Fornisci un esempio concreto di come un'immagine a colori (RGB) è rappresentata in forma matriciale nel computer.
2. (15%) Spiega le differenze tra **immagine binaria**, **toni di grigio (8 bit)** e **a colori RGB**, indicando l'intervallo di valori possibili per ogni pixel in ciascun tipo.
3. (20%) Considerando la formula di conversione RGB → toni di grigio dello standard ITU-R BT.601:

   $$g = 0.299\,R + 0.587\,G + 0.114\,B$$

   Calcola il valore del pixel in toni di grigio per $(R, G, B) = (80, 180, 30)$. Arrotonda al numero intero più vicino.
4. (20%) Che cos'è la **sogliatura** (*thresholding*)? Spiega la differenza tra scegliere una soglia T fissa (es.: $T = 128$) e utilizzare il **metodo di Otsu** per la determinazione automatica della soglia. In poche parole, come sceglie la soglia il metodo di Otsu?
5. (15%) Nel contesto della libreria didattica `mm` discussa nel capitolo, rispondere:
   - a) (7,5%) Come si accede al valore del pixel nella posizione (riga=50, colonna=60) di un'immagine in scala di grigi `img_gray`?
   - b) (7,5%) Qual è il vantaggio di usare `mm.threshold(img_gray)` senza passare la soglia? Confrontare con la chiamata equivalente in OpenCV.
6. (15%) Cosa restituisce la proprietà `img.shape` per un'immagine NumPy in formato RGB? Fornire un esempio concreto con un'immagine di 640×480 pixel.

Riferimenti del Capitolo

La base teorica di questo capitolo comprende le seguenti opere di PDI e VC:

- Gonzalez (2018) per i fondamenti di **Elaborazione Digitale delle Immagini** (EDI).
- Singh (2019) per l'implementazione pratica di metodi di **elaborazione e analisi delle immagini**.
- Szeliski (2022) per lo studio di VC e algoritmi fondamentali.
- Bradski (2008) per l'applicazione della libreria **OpenCV** in ambiente Python.
- Lewis (2020) per il concetto di **generazione aumentata da recupero (RAG)**, utilizzato a supporto dell'elaborazione delle informazioni di questo materiale.

Executar Colab Abrir GitHub

1.17 💻 Parte pratica con esercizi di programmazione

🎯 Obiettivo di questo Quaderno

Gli **Esercizi di Programmazione (EPs)** presentati di seguito possono essere sottomessi anche in attività Moodle (attività VPL) che forniscono *feedback* automatico.

Questo quaderno è stato sviluppato per superare le limitazioni d'uso di Moodle. Con esso, si deve:

1. **Sviluppare:** Scrivere e modificare la propria soluzione direttamente nell'ambiente Colab.
2. **Validare:** Testare il proprio codice localmente utilizzando gli **stessi casi di test** di Moodle.
3. **Organizzare:** Salvare in modo sicuro i propri codici delle attività VPL.

4. **Valutare:** Quando si è connessi a Moodle, basta copiare la propria soluzione e cliccare su **Valuta** in Moodle (**se si è nella rete UFABC**) per registrare il proprio voto ufficiale.

§ Istruzioni Passo a Passo

In un ambiente di esecuzione (come VSCode, Jupyter o Colab), seguire l'ordine seguente per configurare l'ambiente e validare i propri esercizi:

Preparazione dell'Ambiente

Eseguire la cella di codice seguente per scaricare `morph.py` e `testsuite.py` dal repository della disciplina — solo se non sono già presenti nella directory locale. Con entrambi in `./`, il *notebook* e i sottoprocessi del `TestSuite` trovano il modulo senza configurazioni aggiuntive di percorso.

Nota: Lo script `testsuite.py` cercherà automaticamente i casi di test in `all/{cap}/cases` su GitHub.

Scrittura del Codice

Salvare la propria soluzione in una cella di codice utilizzando il comando magico `%%writefile`. Il nome del file deve seguire il pattern `EPX_Y.*`, dove `X` è il capitolo, `Y` è l'esercizio e `*` è l'estensione del linguaggio.

Esempio: `%%writefile EP01_01.py`

Download

Scaricare `morph.py` e `testsuite.py` eseguendo la cella seguente:

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Dopo aver salvato il file con la tua soluzione, esegui il comando seguente (in una nuova cella) per valutare i test automatici:

```
TestSuite("EP01_01.extensão").run()
```

Sostituisci l'estensione in base al linguaggio utilizzato:

Linguaggio	Estensione
Python	`.py`
Java	`.java`
C	`.c`
C++	`.cpp`
JavaScript	`.js`
R	`.r`

Come funziona: Il `TestSuite` scarica i casi di test da GitHub, esegue il tuo programma con ciascun input e confronta l'output con quello atteso – calcolando automaticamente il tuo voto.

Per testare codice Python direttamente, senza salvare file, usa `run_code(codice)` passando il codice come *stringa* in una variabile `codice`:

```
codice = """
from morph import mm
# ... il tuo codice qui ...
"""
TestSuite("EP04_01").run_code(codice)
```

⚠️ Importante: Regole e Buone Pratiche

🔹 Sull'Inserimento dei Dati

Il tuo programma deve leggere l'input standard (tastiera).

- **Python:** Utilizza `input()`.
- **Altri linguaggi:** Utilizza il comando di lettura standard equivalente (`cin`, `Scanner`, ecc.).

🔹 Configurazione dell'IA su Colab

Per un migliore apprendimento, si consiglia di disattivare il completamento automatico del codice tramite IA, poiché non sarà disponibile durante le valutazioni. Ad esempio, nel browser Chrome:

- Vai su: **Strumenti > Impostazioni > IA generativa**
- Deseleziona: **Abilita generazione di codice**

🔹 Integrità Accademica (Plagio)

Questa risorsa di test locali si applica a ES **senza variazioni**. Tuttavia:

- **Individualità:** Ogni studente deve sviluppare la propria soluzione.
- **Rilevamento di Similarità:** Il professore utilizza strumenti che rilevano copie, **anche con modifica di nomi di variabili o spazi bianchi**.

1.17.1 EP01_01 📏 Tre metriche di distanza nella PDA (Processamento Digitale delle Immagini)

In questa attività, devi scrivere un programma che calcoli le tre distanze classiche nella PDA: **Euclidea (L2)**, **City-Block (L1)** e ***Chessboard* (L∞)**.

- Leggi **4 numeri reali** che rappresentano le coordinate: A_x, A_y, B_x, B_y.
- Calcola le tre distanze utilizzando le formule:

$$d_{\text{Euclidea}} = \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2}$$

$$d_{\text{City-block}} = |B_x - A_x| + |B_y - A_y|$$

$$d_{\text{Chessboard}} = \max\left(|B_x - A_x|,\ |B_y - A_y|\right)$$

- Stampa i tre risultati, ciascuno su una riga, formattati con **due cifre decimali**, nell'ordine: **Euclidea**, ***City-block***, ***Chessboard***.

📌 **Importante**:

- Utilizza le funzioni matematiche standard del tuo linguaggio: `math.sqrt`, `abs` (o `fabs`) e `max`.
- L'output deve contenere **solo i numeri** (uno per riga), senza testi aggiuntivi.
- Consulta un simulatore interattivo per questo esercizio nella **Figura 1.11** (grafico con trascinamento dei punti e visualizzazione delle tre metriche).

1.17.1.1 🖼️ Perché è importante? – Costo computazionale

In un'immagine **1000×1000 pixel** (1 milione di pixel), calcolare la distanza di ogni pixel da un punto di riferimento richiede **1 milione di operazioni**. La scelta della metrica influisce sulle prestazioni:

Metrica	Operazioni per pixel	Costo relativo (1M pixel)	Quando utilizzarla
Euclidea (L2)	2 sottrazioni, 2 moltiplicazioni, 1 somma, 1 `sqrt`	🔴 Più costosa – `sqrt` è dispendiosa	Distanza "reale" nello spazio continuo
***City-block* (L1)**	2 sottrazioni, 2 `abs`, 1 somma	🟡 Moderata – senza radice quadrata	Griglie, robotica, immagini binarie
***Chessboard* (L∞)**	2 sottrazioni, 2 `abs`, 1 `max`	🟢 Più efficiente	Movimenti di pezzi, morfologia

La funzione `sqrt` è computazionalmente più costosa rispetto ad operazioni come addizione, sottrazione, moltiplicazione e valore assoluto. Nelle CPU moderne, la differenza può essere piccola (circa 1,5× a 3×), ma nei sistemi embedded o nei cicli di milioni di iterazioni, qualsiasi guadagno conta. Per questo motivo, **quando l'obiettivo è solo confrontare le distanze** (es.: trovare il punto più vicino), utilizza la **distanza euclidea al quadrato**.

1.17.1.2 📋 Compito (specifica per VPL)

Input:
Un'unica riga con quattro numeri reali: `Ax Ay Bx By`

Output:
Tre righe, ciascuna con un numero reale con due cifre decimali (Euclidea, City-block, Chessboard).

1.17.1.3 📌 Esempi

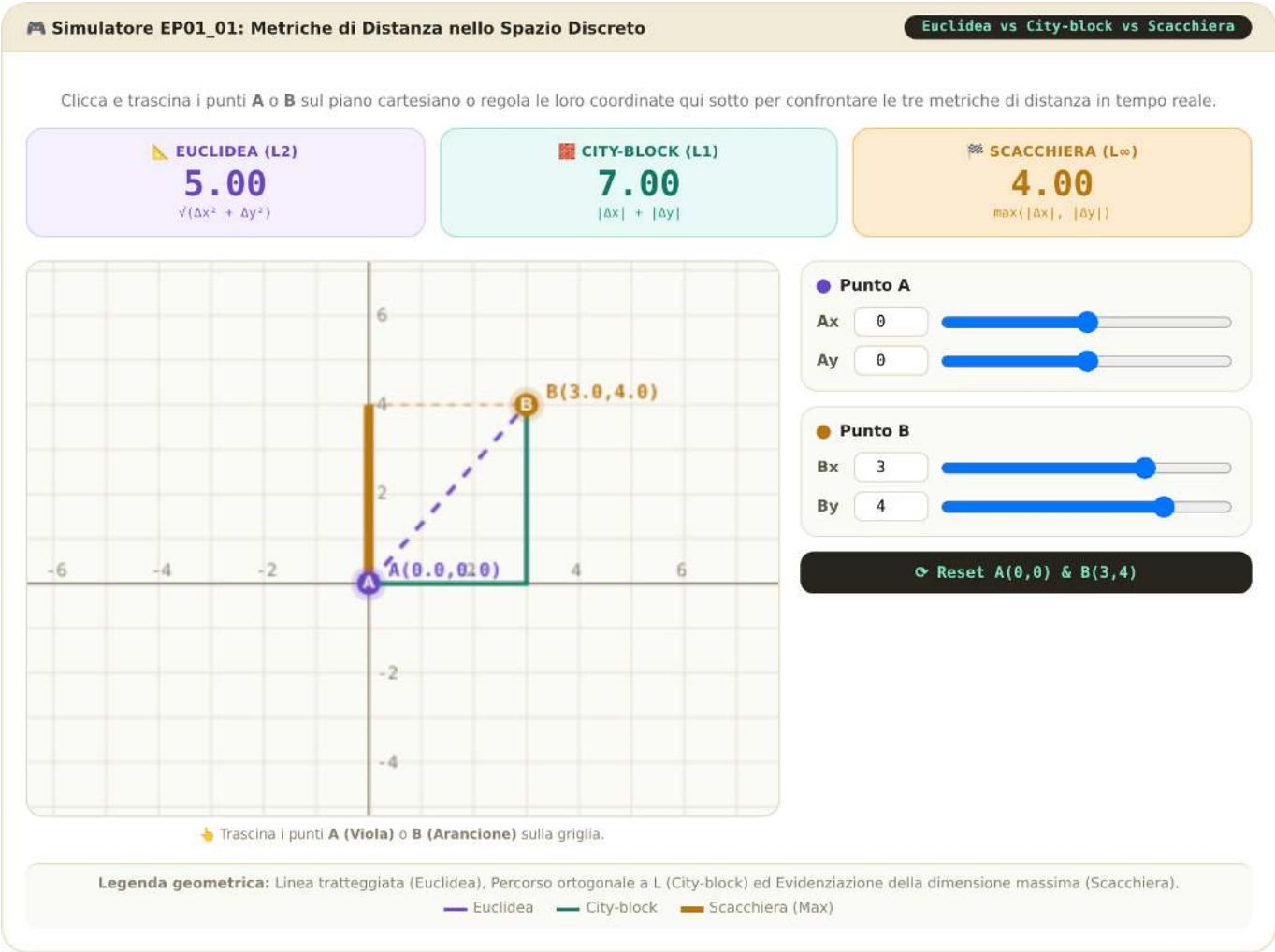
Input	Output	Osservazione
`0`	`5.00`	Triangolo 3-4-5
`0`	`7.00`	
`3`	`4.00`	
`4`		
`0`	`1.41`	Diagonale unitaria
`0`	`2.00`	
`1`	`1.00`	
`1`		

Esempio di test di `sqrt` in Python, con `timeit` che isola ogni operazione:

```python
import math
import timeit

N = 50_000_000

def apenas_soma():
```

```
    a, b = 3.0, 4.0
    return a + b

def soma_e_sqrt():
    a, b = 3.0, 4.0
    return math.sqrt(a*a + b*b)

t_soma = timeit.timeit(apenas_soma, number=N)
t_sqrt = timeit.timeit(soma_e_sqrt, number=N)

print(f"Somma semplice       : {t_soma:.3f} s")
print(f"Somma + sqrt        : {t_sqrt:.3f} s")
print(f"Rapporto (sqrt/somma)  : {t_sqrt/t_soma:.2f}x")
```

```
Somma semplice       : 2.876 s
Somma + sqrt        : 5.073 s
Rapporto (sqrt/somma)  : 1.76x
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0101-distancia.html

Figura 1.11: Simulatore EP01_01: Distanze Euclidea, City-block e Chessboard

1.17.1.4 🐍 Python

Basta creare una normale cella di codice e inserire il codice Python. L'input può essere simulato usando `input()`, che funziona normalmente.

Esempio di cella:

```
%%writefile EP01_01.py
# Codice Python
x1,y1,x2,y2 = int(input()), int(input()), int(input()), int(input())
# Calcolo delle differenze
dx = abs(x2 - x1)
dy = abs(y2 - y1)

# 1. Distanza Euclidea (L2)
dist_euclidiana = (dx**2 + dy**2)**0.5

# 2. Distanza City-block / Manhattan (L1)
dist_city_block = dx + dy

# 3. Distanza Scacchiera / Chebyshev (Linf)
dist_chessboard = max(dx, dy)

# Output formattato secondo i casi di test
print(f"{dist_euclidiana:.2f}")
print(f"{dist_city_block:.2f}")
print(f"{dist_chessboard:.2f}")
```

```
Overwriting EP01_01.py
```

```
# Aspetta che tu inserisca 4 numeri interi eseguendo questa cella.
# In Jupyter o Google Colab, la magia %run -i permette allo script di leggere dalla tastiera.
# In un terminale comune, useresti: python3 EP01_01.py (senza il '!' e senza '%run').

# %run -i EP01_01.py
```

```
# Invia 4 interi come input standard (stdin) allo script EP01_01.py usando una pipe
!echo -e "0\n0\n4\n4" | python3 EP01_01.py
```

```
5.66
8.00
4.00
```

```
TestSuite("EP01_01.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.5 ☕ Java

Per eseguire Java in Colab, è necessario utilizzare una cella con il prefisso `%%writefile` per salvare il codice in un file, compilarlo ed eseguirlo.

```
%%writefile EP01_01.java
import java.util.Scanner;

class EP01_01 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        double x1 = s.nextDouble(), y1 = s.nextDouble();
        double x2 = s.nextDouble(), y2 = s.nextDouble();

        double dx = Math.abs(x2 - x1);
        double dy = Math.abs(y2 - y1);
```

```
        System.out.printf("%.2f\n", Math.sqrt(dx*dx + dy*dy));
        System.out.printf("%.2f\n", dx + dy);
        System.out.printf("%.2f\n", Math.max(dx, dy));
    }
}
```

```
Overwriting EP01_01.java
```

```
!javac EP01_01.java
!echo -e "0\n0\n4\n4" | java EP01_01
```

```
5,66
8,00
4,00
```

```
TestSuite("EP01_01.java").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.6 💻 C

Analogamente, usa `%%writefile` per salvare il codice, poi compila ed esegui. Per C, utilizziamo il compilatore **GCC**.

```
# Installare il compilatore GCC e gli strumenti di build
# build-essential include gcc, g++, make, ecc.
import platform, shutil, subprocess

def instalar_gcc():
    if shutil.which("gcc"):
        print("  GCC già disponibile."); return
    if platform.system() != "Linux":
        print("  Mac: xcode-select --install | Windows: WSL o MinGW"); return
    try: import google.colab; cmd = ["apt-get", "install", "-y", "build-essential"]
    except ImportError:       cmd = ["sudo", "apt-get", "install", "-y", "build-essential"]
    subprocess.run(cmd, check=True)
    print("  build-essential installato!")

instalar_gcc()
```

```
  GCC già disponibile.
```

```
%%writefile EP01_01.c
#include <stdio.h>
#include <math.h>

int main() {
    double x1, y1, x2, y2;
    if (scanf("%lf %lf %lf %lf", &x1, &y1, &x2, &y2) != 4) return 0;

    double dx = fabs(x2 - x1);
    double dy = fabs(y2 - y1);

    // Euclidiana, City-block e Chessboard
    printf("%.2f\n", sqrt(dx * dx + dy * dy));
    printf("%.2f\n", dx + dy);
    printf("%.2f\n", fmax(dx, dy));
```

```
    return 0;
}
```

```
Overwriting EP01_01.c
```

```
# Compila il file .c generando l'eseguibile EP01_01
# -lm viene usato per linkare la libreria matematica (math.h) se necessario

!gcc EP01_01.c -o EP01_01 -lm
!echo -e "0\n0\n4\n4" | ./EP01_01
```

```
5.66
8.00
4.00
```

```
TestSuite("EP01_01.c").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.7 💻 C++

Analogamente al Java, usa `%%writefile` per salvare il codice, poi compila ed esegui. Ricorda che, in Colab, è necessario installare anche quanto segue:

```
# Installa il compilatore G++ per C++
# Il build-essential include g++, make, ecc.
import platform, shutil, subprocess

def instalar_gpp():
    if shutil.which("g++"):
        print("  G++ già disponibile."); return
    if platform.system() != "Linux":
        if platform.system() == "Darwin":
            print("  Mac: xcode-select --install")
        else:
            print("  Windows: usa WSL o MinGW (https://www.mingw-w64.org)")
        return
    try: import google.colab; cmd = ["apt-get", "install", "-y", "build-essential"]
    except ImportError:       cmd = ["sudo", "apt-get", "install", "-y", "build-essential"]
    subprocess.run(cmd, check=True)
    print("  Compilatore C++ pronto.")

instalar_gpp()
```

```
  G++ già disponibile.
```

```
%%writefile EP01_01.cpp
#include <iostream>
#include <iomanip>
#include <cmath>
#include <algorithm>

int main() {
    double x1, y1, x2, y2;
    if (!(std::cin >> x1 >> y1 >> x2 >> y2)) return 0;

    double dx = std::abs(x2 - x1);
    double dy = std::abs(y2 - y1);
```

```
    std::cout << std::fixed << std::setprecision(2);

    // Euclidiana, City-block e Chessboard
    std::cout << std::sqrt(dx*dx + dy*dy) << std::endl;
    std::cout << (dx + dy) << std::endl;
    std::cout << std::max(dx, dy) << std::endl;

    return 0;
}
```

```
Overwriting EP01_01.cpp
```

```
!g++ EP01_01.cpp -o EP01_01
!echo -e "0\n0\n4\n4" | ./EP01_01
```

```
5.66
8.00
4.00
```

```
TestSuite("EP01_01.cpp").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.8 🌐 JavaScript (Node.js)

Per JavaScript, usa `%%writefile` per creare il file ed eseguilo con Node:

```
%%writefile EP01_01.js
function escreva(s) { try { document.write(s + "<br>"); } catch(e) { console.log(s); } }

process.stdin.once('data', data => {
  const valores = data.toString().trim().split(/\s+/);
  const x1 = parseFloat(valores[0]);
  const y1 = parseFloat(valores[1]);
  const x2 = parseFloat(valores[2]);
  const y2 = parseFloat(valores[3]);

  const dx = Math.abs(x2 - x1);
  const dy = Math.abs(y2 - y1);

  // Euclidiana, City-block e Chessboard
  escreva(Math.sqrt(dx * dx + dy * dy).toFixed(2));
  escreva((dx + dy).toFixed(2));
  escreva(Math.max(dx, dy).toFixed(2));
});
```

```
Overwriting EP01_01.js
```

```
TestSuite("EP01_01.js").run()
```

```
<IPython.core.display.HTML object>
```

1.17.1.9 📊 R

In Colab è possibile eseguire codice R direttamente utilizzando la magia `%%R`.
Il programma deve leggere **quattro numeri** (x1, y1, x2, y2) e mostrare la distanza euclidea con **due cifre decimali**.

Esempio di cella:

```
%%R
dados <- scan(file = "stdin", n = 4, quiet = TRUE)
x1 <- dados[1]; y1 <- dados[2]; x2 <- dados[3]; y2 <- dados[4]
dist <- sqrt((x2 - x1)^2 + (y2 - y1)^2)
cat(sprintf("%.2f", dist))
```

```
# Installare R e Rscript
# Il pacchetto r-base installa l'ambiente R completo, inclusi Rscript
import platform, shutil, subprocess

def instalar_r():
    if shutil.which("R"):
        print("  R già disponibile."); return
    if platform.system() != "Linux":
        if platform.system() == "Darwin":
            print("  Mac: https://cran.r-project.org/bin/macosx/")
        else:
            print("  Windows: https://cran.r-project.org/bin/windows/base/")
        return
    try: import google.colab; cmd = ["apt-get", "install", "-y", "r-base"]
    except ImportError:       cmd = ["sudo", "apt-get", "install", "-y", "r-base"]
    subprocess.run(cmd, check=True)
    print("  Ambiente R pronto.")

instalar_r()
```

```
  R già disponibile.
```

Para testare nello stesso modo degli esempi precedenti, si deve utilizzare l'input standard (stdin) nel terminale o adattare il codice come segue:

```
%%writefile EP01_01.r
dados <- scan(file = "stdin", n = 4, quiet = TRUE)
dx <- abs(dados[3] - dados[1])
dy <- abs(dados[4] - dados[2])

# Output: Euclidea, City-block e Scacchiera
cat(sprintf("%.2f\n%.2f\n%.2f\n",
    sqrt(dx^2 + dy^2),
    dx + dy,
    max(dx, dy)))
```

```
Overwriting EP01_01.r
```

```
!echo -e "0\n0\n4\n4" | Rscript EP01_01.r
```

```
5.66
8.00
4.00
```

```
TestSuite("EP01_01.r").run()
```

```
<IPython.core.display.HTML object>
```

1.17.2 EP01_02 📊 Prestazioni Predittive — Metriche di ML in VC

In questa attività entrerai nel mondo dell'**Apprendimento Automatico** (*Machine Learning*). Il tuo obiettivo è valutare le prestazioni di un classificatore binario calcolando le metriche a partire da una **Matrice di Confusione**.

1.17.2.1 🧠 Perché la metrica giusta è importante?

Immagina un rilevatore di **monete da R$ 1,00**. L'impatto dell'errore definisce la metrica prioritaria:

Metrica	Esempio Pratico	Importanza in PDI/VC
Accuratezza	**Conteggio dei Grani**	Utile quando le classi sono bilanciate (es: metà dei grani difettosi, metà sani).
Precisione	**Sicurezza/Biometria**	Fondamentale per evitare **Falsi Positivi** (es: non consentire a un impostore di accedere a un sistema per un errore di riconoscimento).
Sensibilità	**Salute (Tumori)**	Fondamentale per evitare **Falsi Negativi** (es: non lasciare che un tumore passi inosservato in una radiografia).
F1-score	**Banconote**	Ideale per un **equilibrio** tra non rifiutare banconote autentiche e non accettare banconote false.

1.17.2.2 📊 La Matrice di Confusione

	Predetta Positiva	**Predetta Negativa**
Reale Positiva	**VP** (Vero Positivo)	**FN** (Falso Negativo)
Reale Negativa	**FP** (Falso Positivo)	**VN** (Vero Negativo)

Compito:

1. Leggi **4 valori interi** nell'ordine: `VP`, `FN`, `FP`, `VN`.
2. Calcola le metriche utilizzando le formule:

$$\text{Accuratezza} = \frac{VP + VN}{VP + VN + FP + FN}$$

$$\text{Precisione} = \frac{VP}{VP + FP}$$

$$\text{Sensibilità (Recall)} = \frac{VP}{VP + FN}$$

$$\text{F1-score} = \frac{2 \times \text{Precisione} \times \text{Sensibilità}}{\text{Precisione} + \text{Sensibilità}}$$

3. Stampa i risultati formattati con **due cifre decimali**, uno per riga.

📌 **Importante**:

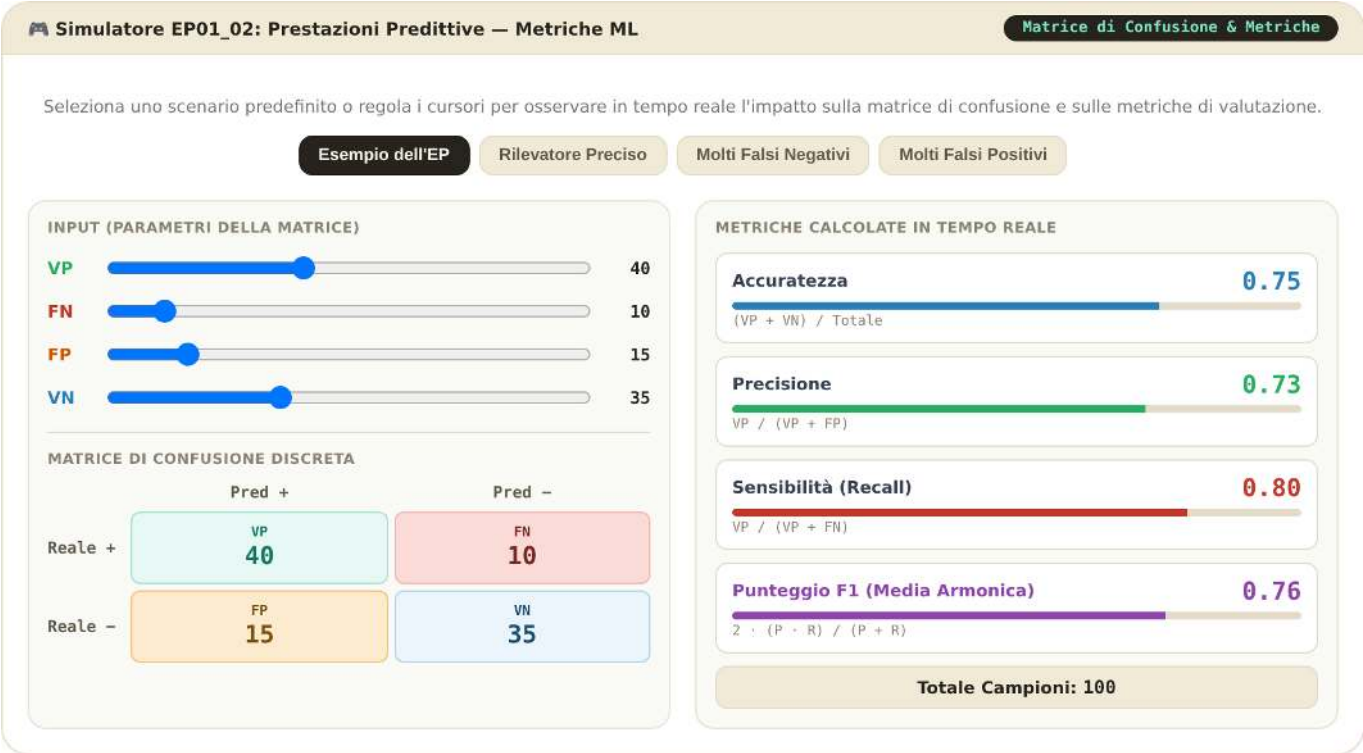
- Usa la divisione in virgola mobile per evitare risultati troncati.

- L'ordine di uscita deve essere: **Accuratezza**, **Precisione**, **Sensibilità** e **F1-score**.
- Consulta la simulazione interattiva di questo EP alla Figura 1.12.

1.17.2.3 📌 Esempio di Esecuzione

Input	Output	Osservazione
40	0.75	Accuratezza
10	0.73	Precisione
15	0.80	Sensibilità
35	0.76	F1-score

(Nota: VP=40, FN=10, FP=15, VN=35. Totale dei casi = 100)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0102.html

Figura 1.12: Simulatore EP01_02: Prestazioni Predittive — Metriche di ML

```
%%writefile EP01_02.py
# la tua soluzione
```

```
Overwriting EP01_02.py
```

```
TestSuite("EP01_02.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.3 EP01_03 ↗ *Mean Average Precision* (mAP) — Curva Precisione-Sensibilità

In questa attività valuterai un classificatore binario (es.: rilevamento della deforestazione in immagini satellitari, vedi dgi.inpe.br) tramite la **curva Precisione-Sensibilità** e la metrica **mAP** (***Mean Average Precision***). L'mAP è uno standard in competizioni come COCO (*Common Objects in Context*) e PASCAL VOC (*Visual Object Classes*) e nei modelli YOLO (*You Only Look Once*).

1.17.3.1 🧠 Perché l'mAP è la metrica standard?

Nella EP01_02 hai visto che la scelta della soglia altera significativamente Precisione e Sensibilità. L'**mAP (*Mean Average Precision*)** risolve questo problema: valuta il modello su **più soglie** (ogni soglia deve generare una matrice di confusione diversa) e riassume le prestazioni tramite l'**area sotto la curva Precisione-Sensibilità (P-S)**.

Mentre l'F1-*Score* considera un singolo punto di equilibrio, l'mAP considera l'intera curva. Più il valore è vicino a **1,0**, migliore è il rilevatore su tutte le soglie e classi (es.: monete da 25, 50 centesimi e 1 euro).

Metrica	Cosa riassume	Limitazione
F1-*Score*	Equilibrio P × S su una singola soglia	Dipende dalla soglia scelta
AP	Area sotto la curva P-S di una classe	Valida solo per una classe
mAP	Media delle AP su tutte le classi	Più complesso da implementare

Riferimenti: Roboflow — mAP · Video esplicativo

1.17.3.2 Come viene calcolato l'mAP — passo dopo passo

1. **Soglie fisse** (usa sempre questa lista):

```
limiares = [0.00, 0.09, 0.21, 0.31, 0.39, 0.52, 0.60, 0.71, 0.81, 0.89, 1.00]
```

2. Per ogni soglia (t), classifica i campioni: `predito = 1 se confiança  t, altrimenti 0`. Calcola VP, FP, FN, VN e ottieni Precisione((t)) e Sensibilità((t)).
3. Costruisci la curva P-S: coppie (Sensibilità((t)), Precisione((t))), ordinate per Sensibilità crescente.
4. **Rendi monotona la Precisione**:

$$P_{\text{mono}}[i] = \max_{j \geq i} P[j]$$

5. Calcola l'**AP** (area sotto la curva monotona) usando la **regola del trapezio** (approssimazione più accurata rispetto alla semplice somma di Riemann):

$$AP = \sum_{i=1}^{m-1} \frac{P_{\text{mono}}[i-1] + P_{\text{mono}}[i]}{2} \cdot (S[i] - S[i-1])$$

6. **mAP** = media delle AP di tutte le classi. In questa EP c'è solo **1 classe**, quindi mAP = AP.

> ℹ Nota
>
> 📐 **Differenza riassunta:**
> La *somma di Riemann* approssima l'area tramite rettangoli, potendo sottostimare o sovrastimare. La **regola del trapezio** usa trapezi, riducendo l'errore considerando la media tra i valori agli estremi dell'intervallo, risultando generalmente più precisa per funzioni continue a tratti, come la curva Precisione-Sensibilità.

1.17.3.3 📋 Compito

Leggi un intero `n` (quantità di campioni). Poi leggi `n` righe, ciascuna con: `verdade` (0 o 1) e `confiança` (float 0.0–1.0).

Calcola e stampa, per la **soglia 0.85** (indice 9 della lista):

- Matrice di Confusione (VP, FN, FP, VN)
- Accuratezza, Precisione, Sensibilità e F1-Score

Successivamente, per **tutte le soglie**, stampa:

- Precisioni grezze, Precisioni monotone e Sensibilità, separate da **,**
- mAP finale

1.17.3.4 📌 Importante

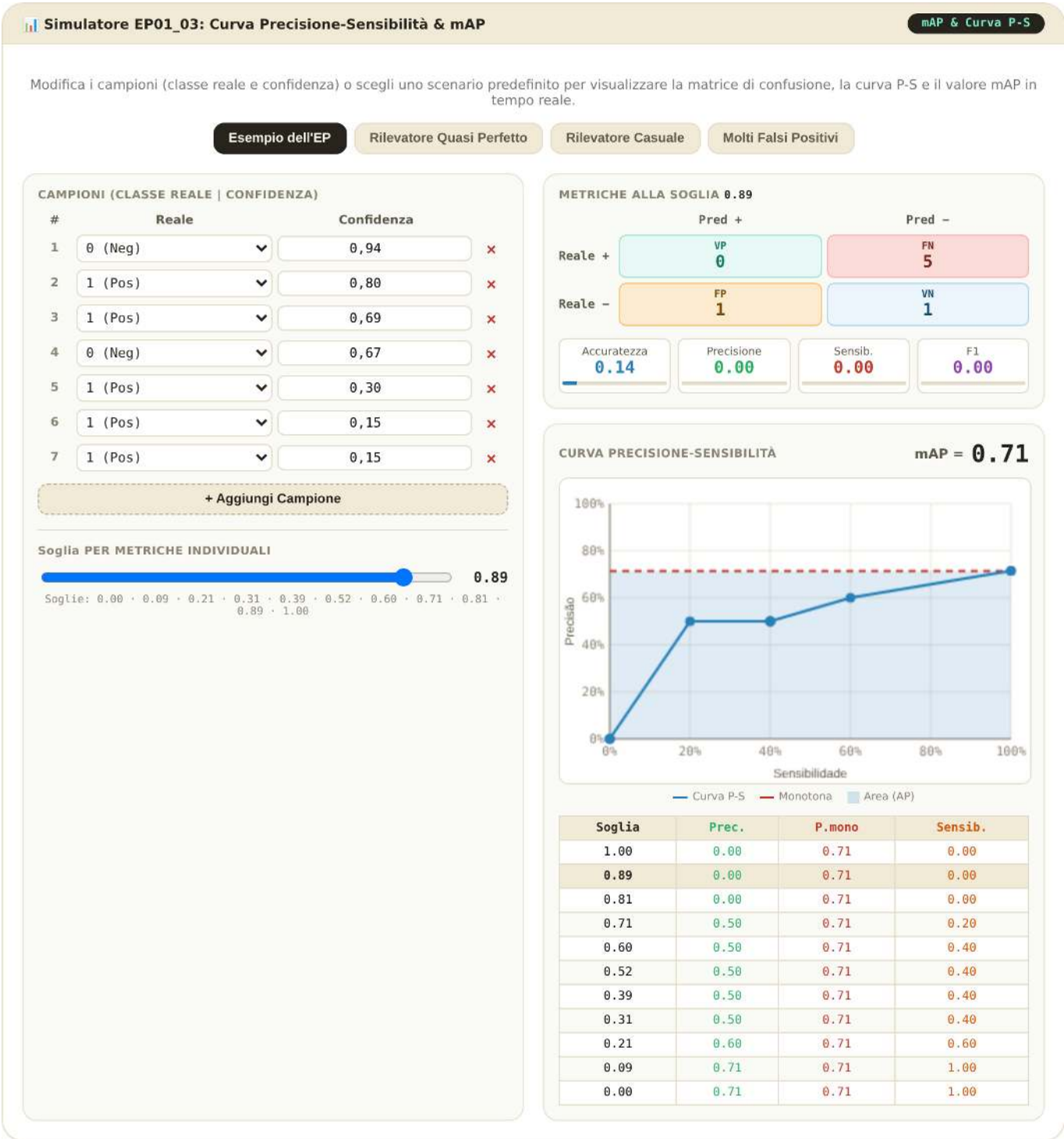
- Soglia fissa per le metriche individuali: **0.85**
- Divisione sicura: se il denominatore è zero, usa 0
- Formattazione: due cifre decimali
- Rendi monotona da destra verso sinistra
- La Figura 1.13 presenta una simulazione di questa domanda

1.17.3.5 📌 Esempio di Esecuzione

Input	Output Atteso
7 0 0.94 1 0.80 1 0.69 0 0.67 1 0.30 1 0.15 1 0.15	# MÉTRICHE PER LA SOGLIA 0.85 # Matrice di Confusione: VP = 0, FN = 5 FP = 1, VN = 1 Metriche di Valutazione: Accuratezza: 0.14 Precisione: 0.00 Sensibilità: 0.00 F1-Score: 0.00 # MÉTRICHE PER TUTTE LE SOGLIE # Precisioni: 0.00, 0.00, 0.00, 0.50, 0.50, 0.50, 0.50, 0.50, 0.60, 0.71, 0.71 Precisioni mon.: 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71, 0.71 Sensibilità: 0.00, 0.00, 0.00, 0.20, 0.40, 0.40, 0.40, 0.40, 0.60, 1.00, 1.00 mAP: 0.71

1.17.3.6 🐍 Suggerimento per calcolare l'AP (con regola del trapezio)

```python
def calcular_AP(verdades, confiancas, limiares):
    m = len(limiares)
    precisoes = [0.0] * m
    sensibilidades = [0.0] * m
    for i in range(m):
        p, s = calcular_metricas(verdades, confiancas, limiares[i])
        precisoes[m-1-i] = p
        sensibilidades[m-1-i] = s
    prec_mono = precisoes.copy()
    for i in range(m-2, -1, -1):
        if prec_mono[i] < prec_mono[i+1]:
            prec_mono[i] = prec_mono[i+1]
    AP = 0.0
    for i in range(1, m):
        # Regola del trapezio: media delle altezze per la base
        area_trapezio = (prec_mono[i-1] + prec_mono[i]) / 2.0
        AP += area_trapezio * (sensibilidades[i] - sensibilidades[i-1])
    return precisoes, prec_mono, sensibilidades, AP
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0103-map.html

Figura 1.13: Simulatore EP01_03: Mean Average Precision (mAP) e Curva P-S

```
%%writefile EP01_03.py
# la tua soluzione
```

```
Overwriting EP01_03.py
```

```
TestSuite("EP01_03.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.4 EP01_04 🖼️ Lettura e Informazioni di un'Immagine in Matrice

In questa attività, devi scrivere un programma che elabori un'immagine digitale rappresentata come una matrice di pixel in scala di grigi.

- Leggi due interi **L** e **C**, che rappresentano il numero di righe e di colonne.
- Leggi i **L * C** valori interi che compongono la matrice dell'immagine (ogni valore tra 0 e 255).
- Calcola e stampa le seguenti informazioni:

1. Il numero di righe.
2. Il numero di colonne.
3. Il valore del pixel più grande (**Massimo**).
4. Il valore del pixel più piccolo (**Minimo**).
5. La **Media** aritmetica di tutti i pixel.

📌 **Importante**:

- L'output deve seguire esattamente il formato etichettato (es: `Righe: X`).
- Il valore della media deve essere formattato con **due cifre decimali**.
- Vedi un simulatore interattivo per questa domanda nella **Figura 1.14** (griglia interattiva per la visualizzazione delle intensità e dei calcoli in tempo reale).

1.17.4.1 🧠 Perché è importante? – L'Immagine come Dato

Ogni immagine digitale è, in fondo, una struttura dati. In scala di grigi a 8 bit, ogni pixel è un valore scalare. Estrarre statistiche di base è il primo passo per:

Operazione	Utilità Pratica
Massimo/Minimo	Identificare se l'immagine è "slavata" (basso contrasto) o saturata.
Media	Calcolare la luminosità globale della scena per regolazioni dell'esposizione.
Normalizzazione	Ridimensionare i valori in intervalli come $[0, 1]$ nelle reti neurali.

1.17.4.2 📋 Attività (specifica per VPL)

Input:

La prima riga contiene l'intero `L` (righe).

La seconda riga contiene l'intero `C` (colonne).

Le righe successive contengono gli elementi della matrice.

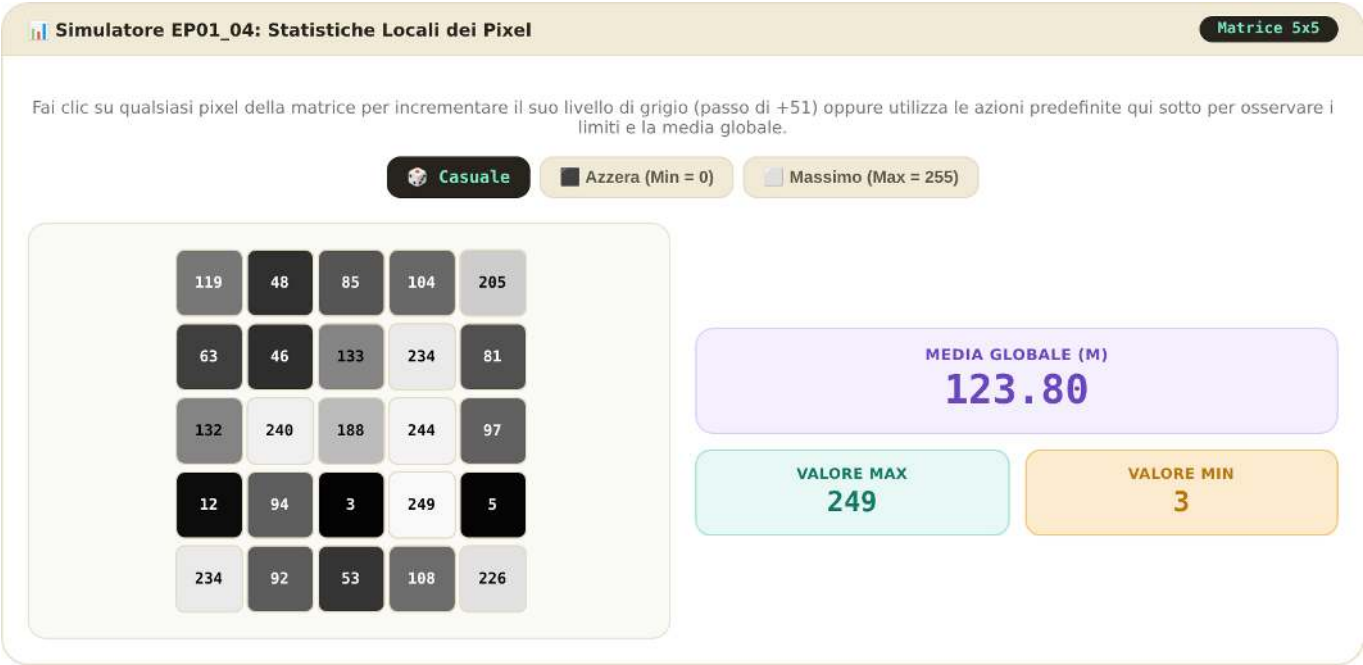
Output:

Cinque righe formattate secondo l'esempio:

```
Righe: L

Colonne: C

Max: V

Min: V

Media: V.VV
```

1.17.4.3 📌 Esempi

Input	Output	Osservazione
2 3 0 128 255 50 100 200	Righe: 2 Colonne: 3 Max: 255 Min: 0 Media: 122.17	Immagine piccola ad alto contrasto



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0104-estatisticas.html

Figura 1.14: Simulatore EP01_04: Statistiche dei pixel in matrice discreta 5x5

```
%%writefile EP01_04.py
# la tua soluzione
```

```
Overwriting EP01_04.py
```

```
TestSuite("EP01_04.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.5 EP01_05 🔄 Negativo di un'Immagine in Ton di Grigio

In questa attività, si deve scrivere un programma che calcoli il negativo di un'immagine digitale.

- Leggere due interi **L** e **C**, che rappresentano il numero di righe e colonne.

- Leggere i valori interi che compongono la matrice dell'immagine.
- Per ogni pixel, applicare la trasformazione di inversione:

$$pixel_{negativo} = 255 - pixel_{originale}$$

- Stampare la matrice risultante, mantenendo il formato originale (L righe e C colonne).

📌 **Importante**:

- I valori di ciascuna riga nell'output devono essere separati da uno **spazio bianco**.
- L'output deve contenere **solo i numeri** della matrice risultante.
- Vedere un simulatore interattivo per questa questione nella **Figura 1.15** (confronto in tempo reale tra la matrice originale e il suo negativo).

1.17.5.1 🧠 Perché è importante? – Inversione di Intensità

Il **negativo** è una trasformazione lineare di base che inverte la scala di luminosità. È uno strumento essenziale per l'occhio umano per identificare dettagli chiari "nascosti" in sfondi più scuri, ed è ampiamente utilizzato in:

Applicazione	Utilità
Immagini Mediche	Migliora la visualizzazione di anomalie in tessuti densi (es. Radiografie).
Astronomia	Evidenziare galassie e nebulose deboli contro il vuoto dello spazio.
Arti Digitali	Effetti estetici e preparazione di maschere di selezione.

1.17.5.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene l'intero `L`.

La seconda riga contiene l'intero `C`.

Le righe successive contengono gli elementi della matrice.

Output:

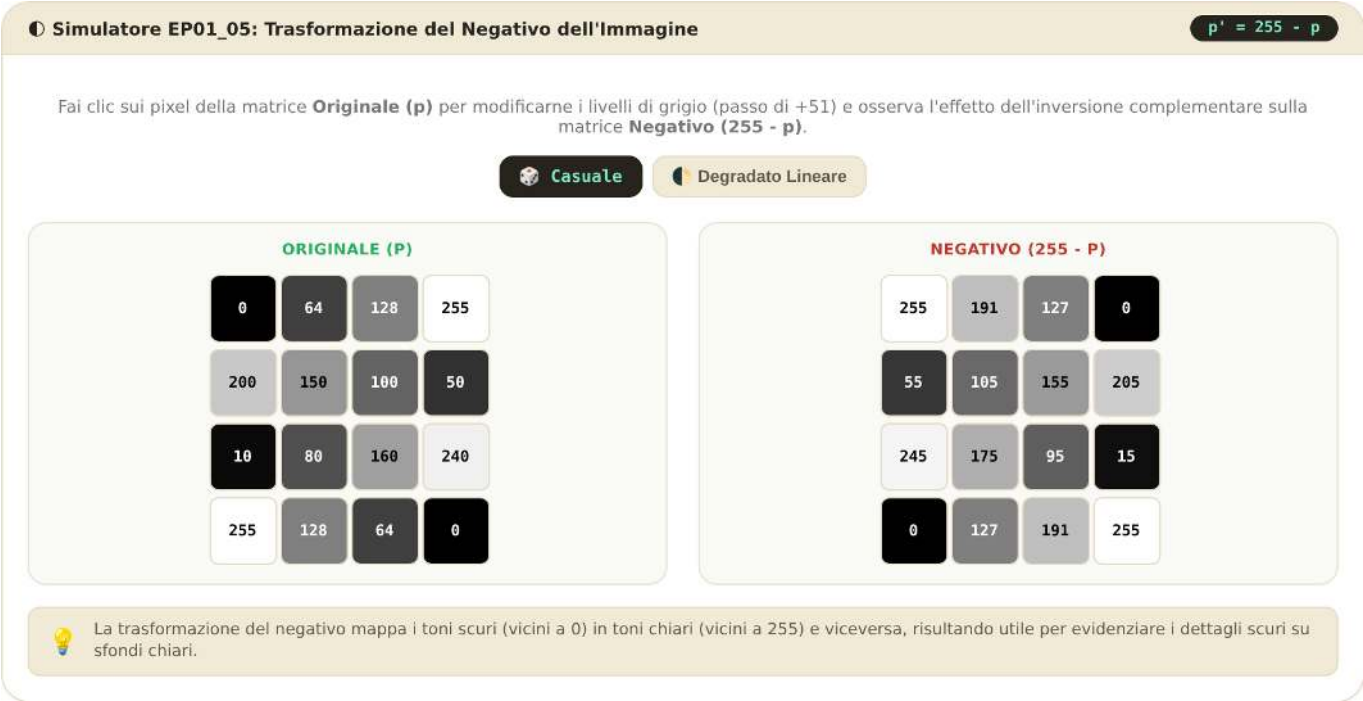
La matrice invertita con `L` righe e `C` colonne.

1.17.5.3 📌 Esempi

Input	Output	Osservazione
2 3 0 128 255 50 100 255	255 127 0 205 155 55	Dove c'era 0 (nero) diventa 255 (bianco)

```
%%writefile EP01_05.py
# la tua soluzione
```

```
Overwriting EP01_05.py
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0105-negativo.html

Figura 1.15: Simulatore EP01_05: Trasformazione del Negativo dell'Immagine (Inversione di Intensità)

```
TestSuite("EP01_05.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.6 EP01_06 🎨 Conversione RGB → Toni di Grigio (ITU-R BT.601)

In questa attività, devi scrivere un programma che converta pixel colorati (RGB) in toni di grigio utilizzando la ponderazione fisiologica dello standard ITU-R BT.601.

- Leggi due interi **L** e **C**, che rappresentano il numero di righe e colonne.
- Leggi **L** × **C** triple di interi, dove ogni terna rappresenta i canali **R** (Rosso), **G** (Verde) e **B** (Blu) di un pixel.
- Per ogni pixel, calcola il valore di grigio (g) usando la formula:

$$g = \text{round}(0.299 \times R + 0.587 \times G + 0.114 \times B)$$

- Stampa la matrice risultante (L righe e C colonne) contenente i valori interi convertiti.

📌 **Importante**:

- Utilizza la funzione `round()` del tuo linguaggio per garantire l'arrotondamento corretto all'intero più vicino.
- L'output deve contenere solo i valori di grigio, mantenendo la struttura a matrice (separati da spazio sulla riga).
- Vedi un simulatore interattivo per questa domanda nella **Figura 1.16** (regola gli slider per osservare come ogni colore contribuisce alla luminosità finale).

1.17.6.1 🧠 Perché non usare semplicemente la media?

L'occhio umano non percepisce tutti i colori con la stessa intensità. Siamo molto più sensibili al **Verde** che al **Blu** a causa della nostra evoluzione biologica. Lo standard ITU-R BT.601 utilizza pesi specifici per creare un'immagine in grigio che appaia naturalmente corretta alla nostra visione:

Canale	Peso	Percezione Umana
🟢 **Verde**	**58.7%**	Massima sensibilità (distinzione del fogliame).
🔴 **Rosso**	**29.9%**	Sensibilità media.
🔵 **Blu**	**11.4%**	Bassa sensibilità (toni più scuri).

1.17.6.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene l'intero `L`.

La seconda riga contiene l'intero `C`.

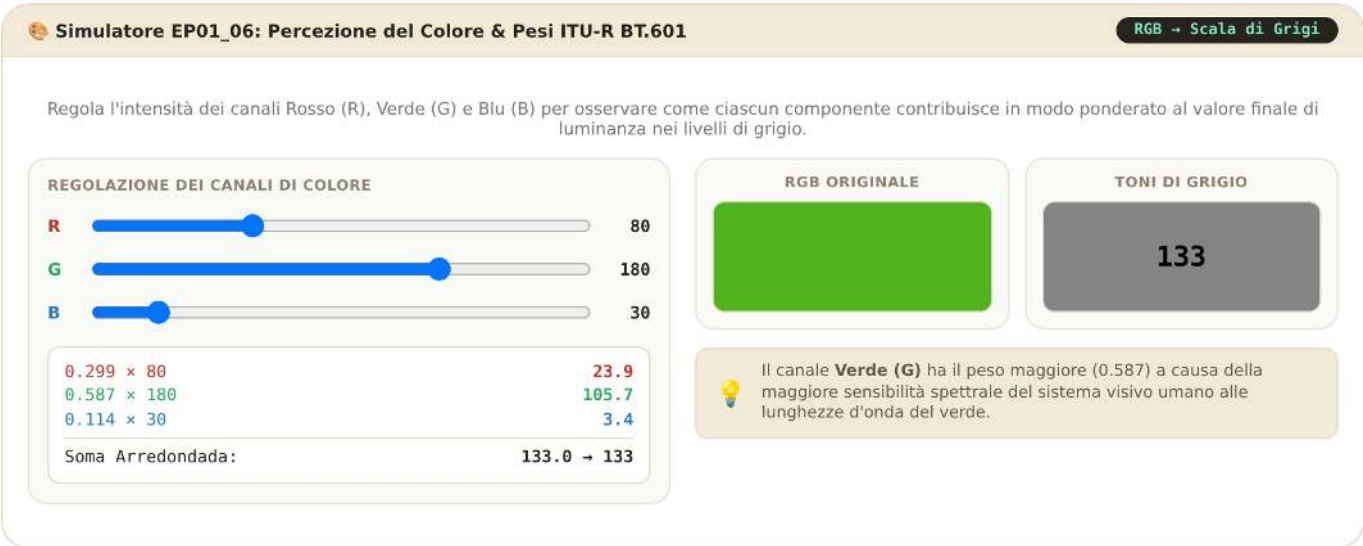
Le righe successive contengono triple di interi `R G B` per ogni pixel.

Output:

La matrice di grigi con `L` righe e `C` colonne.

1.17.6.3 📌 Esempi

Input	Output	Osservazione
1 3 255 0 0 0 255 0 0 0 255	76 150 29	Nota come il Verde (150) sia più luminoso del Blu (29)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0106-rgb-gray.html

Figura 1.16: Simulatore EP01_06: Conversione RGB in Scala di Grigi (Ponderazione Percettiva ITU-R BT.601)

```
%%writefile EP01_06.py
# la tua soluzione
```

```
Overwriting EP01_06.py
```

```
TestSuite("EP01_06.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.7 EP01_07 ⚫ Soglia Manuale: Immagine Binaria

In questa attività, devi scrivere un programma che esegua la segmentazione di un'immagine tramite sogliatura (*thresholding*).

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi un intero **T**, che sarà il valore della soglia (taglio).
- Leggi i valori interi della matrice.
- Per ogni pixel p, applica la seguente regola di binarizzazione:

$$\text{risultato} = \begin{cases} 255 & \text{se } p > T \\ 0 & \text{se } p \leq T \end{cases}$$

- Stampa la matrice risultante contenente solo i valori 0 o 255.

📌 **Importante**:

- Presta attenzione all'operatore: il pixel diventa bianco (255) solo se è **strettamente maggiore** di T.
- L'output deve mantenere la struttura della matrice (L righe e C colonne).
- Consulta un simulatore interattivo per questo problema su **Figura 1.17** (regola il cursore di T per osservare come gli oggetti vengono isolati dallo sfondo).

1.17.7.1 🧠 Cos'è la Segmentazione?

La sogliatura è il metodo più semplice per separare gli oggetti di interesse dallo sfondo dell'immagine. Trasformando i toni di grigio in bianco e nero puro, creiamo una mappa binaria che facilita il conteggio degli oggetti o l'identificazione delle forme:

Valore del Pixel (p)	Condizione	Risultato Finale
Scuro ($p \leq T$)	Sfondo/Rumore	**0** (Nero)
Chiaro ($p > T$)	Oggetto/In evidenza	**255** (Bianco)

1.17.7.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene l'intero `L`.

La seconda riga contiene l'intero `C`.

La terza riga contiene l'intero `T` (soglia).

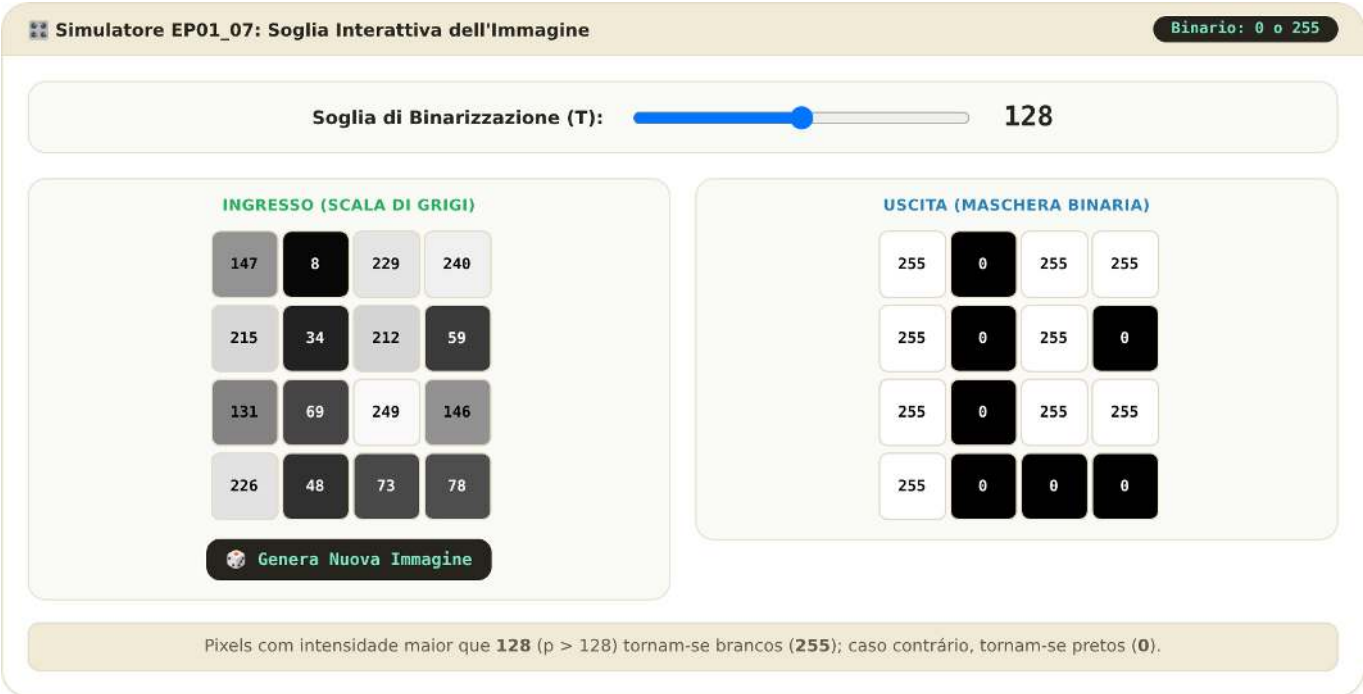
Le righe successive contengono gli elementi della matrice.

Output:

La matrice binarizzata (0 o 255) con `L` righe e `C` colonne.

1.17.7.3 📌 Esempi

Input	Output	Osservazione
2 4 128 0 100 128 200 50 129 255 64	0 0 0 255 0 255 255 0	Nota che il valore 128 è diventato 0 (poiché $128 \leq 128$)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0107-limiarizacao.html

Figura 1.17: Simulatore EP01_07: Soglia Globale Interattiva (Binarizzazione p > T)

```
%%writefile EP01_07.py
# la tua soluzione
```

```
Overwriting EP01_07.py
```

```
TestSuite("EP01_07.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.8 EP01_08 🎨 Rimappatura per Intervallo di Intensità

In questa attività, devi scrivere un programma che applichi trasformazioni lineari distinte a diverse regioni di intensità dell'immagine.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi gli interi **T** (soglia), (delta 1) e (delta 2).
- Leggi i valori interi della matrice.
- Per ogni pixel p, applica la regola di rimappatura condizionale:

$$\text{risultato} = \begin{cases} p + \delta_1 & \text{se } p < T \\ p + \delta_2 & \text{se } p \geq T \end{cases}$$

- Stampa la matrice risultante con i nuovi valori di intensità.

📌 **Importante**:

- è l'offset applicato ai pixel scuri (al di sotto della soglia).
- è l'offset applicato ai pixel chiari (maggiori o uguali alla soglia).
- I casi di test garantiscono che il risultato sia sempre nell'intervallo valido da **0 a 255**, quindi non è necessario gestire saturazione o arrotondamenti.
- L'output deve mantenere la struttura di matrice (**L** righe e **C** colonne).

1.17.8.1 🧠 Trasformazione Condizionale dei Pixel

Nell'elaborazione digitale delle immagini (PDI), spesso dobbiamo trattare le regioni in modo indipendente. Questa tecnica consente, ad esempio, di schiarire solo le ombre di una fotografia (aumentando i pixel scuri) senza bruciare la luminosità delle aree già chiare, o viceversa.

Intervallo di Intensità	Condizione	Operazione
Pixel scuri	$p < T$	$p + \delta_1$
Pixel chiari	$p \geq T$	$p + \delta_2$

1.17.8.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene gli interi `L` e `C`.

La seconda riga contiene gli interi `T`, e .

Le righe successive contengono gli elementi della matrice.

Output:

La matrice trasformata con `L` righe e `C` colonne, con valori separati da spazi.

1.17.8.3 📌 Esempi

Input	Output	Osservazione

`2 4`
`128 60 -40`
`0 100 150 255`
`80 128 200 30`
	`60 160 110 215`
	`140 88 160 90`
		I pixel < 128 sommano 60. I pixel 128 sottraggono 40.

```
%%writefile EP01_08.py
# la tua soluzione
```

```
Overwriting EP01_08.py
```

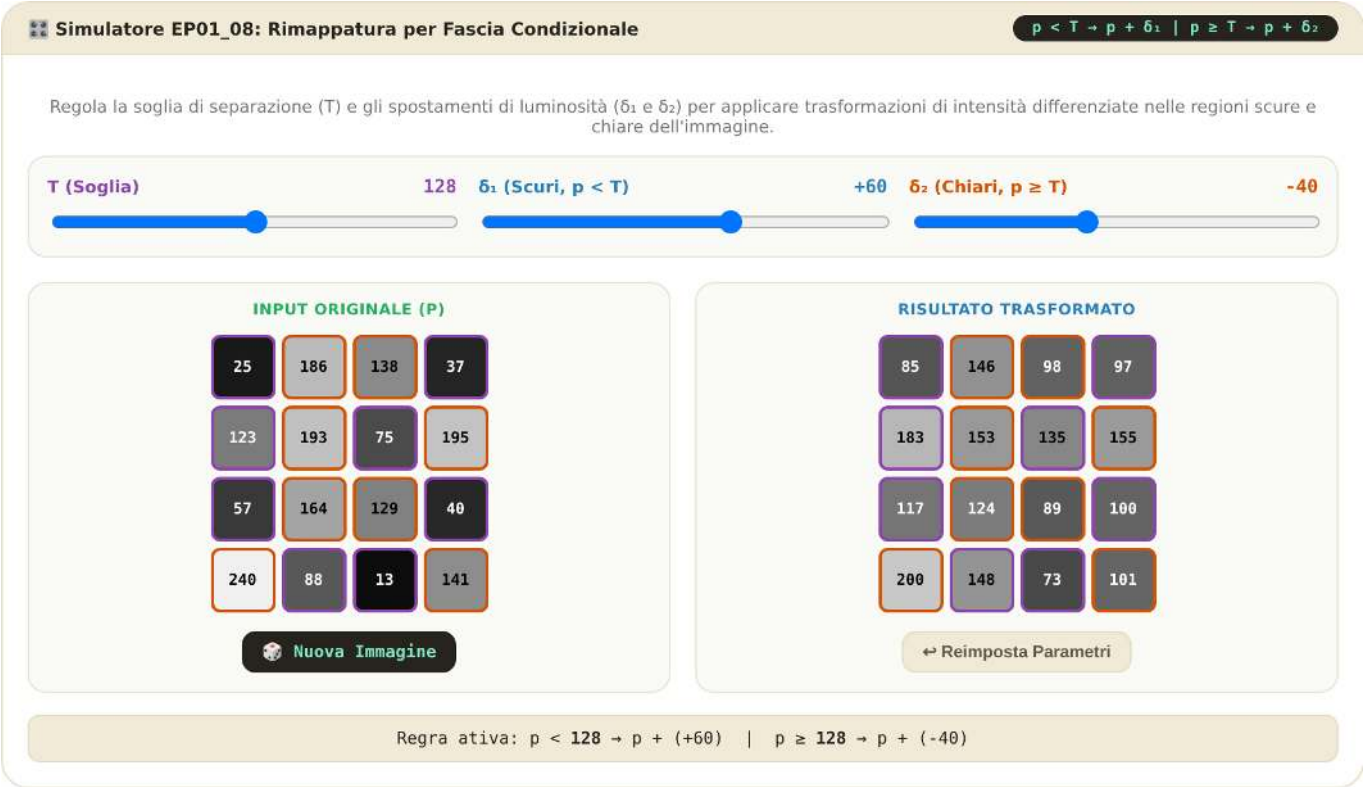
```
TestSuite("EP01_08.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.9 EP01_09 🏁 Schema a Scacchiera: Matrice a Quadretti

In questa attività, devi scrivere un programma che generi un'immagine sintetica con il motivo di una scacchiera.

- Leggi due interi **L** (righe) e **C** (colonne).



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0108-brilho-contraste.html

Figura 1.18: Simulatore EP01_08: Rimappatura per Intervallo Condizionale (Luminosità e Contrasto per Soglia)

- Genera una matrice in cui i valori alternano tra **0** (nero) e **1** (bianco).
- La logica di riempimento deve seguire la regola di parità:
- L'elemento nella posizione $(0, 0)$ è sempre **0**.
- Un pixel nella posizione (i, j) sarà **1** se la somma degli indici $(i + j)$ è **dispari**.
- Un pixel nella posizione (i, j) sarà **0** se la somma degli indici $(i + j)$ è **pari**.

📌 **Importante**:

- I colori devono alternarsi correttamente sia orizzontalmente che verticalmente.
- L'output deve essere la matrice stampata riga per riga, con gli elementi separati da uno spazio.
- Consulta un simulatore interattivo per questo problema su **Figura 1.19** (regola le dimensioni per visualizzare la costruzione della griglia e l'output testuale corrispondente).

1.17.9.1 🧠 Motivi Sintetici

Creare motivi geometrici è un esercizio fondamentale per padroneggiare la **logica degli indici** nelle matrici. Nell'elaborazione digitale delle immagini, il motivo a scacchiera non è solo estetico; è ampiamente utilizzato per:

Applicazione	Utilità
Calibrazione della fotocamera	Stimare i parametri intrinseci ed estrinseci dell'obiettivo.
Correzione della distorsione	Identificare e correggere l'effetto "a barile" o "a cuscino" negli obiettivi grandangolari.
Mappatura 3D	Proiettare motivi noti per ricostruire superfici in sistemi a luce strutturata.

1.17.9.2 📋 Compito (specifica per VPL)

Input:

Una riga contenente l'intero `L` (righe).

Una riga contenente l'intero `C` (colonne).

Output:

La matrice a scacchiera con `L` righe e `C` colonne, stampata con spazi tra gli elementi.

1.17.9.3 📌 Esempi

Input	Output	Osservazione
3 4	0 1 0 1 1 0 1 0 0 1 0 1	Nota che ogni riga inizia con l'inverso della precedente

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0109-xadrez.html

Figura 1.19: Simulatore EP01_09: Generatore di Motivo a Scacchiera (Logica di Parità (i + j) % 2)

```
%%writefile EP01_09.py
# la tua soluzione
```

```
Overwriting EP01_09.py
```

```
TestSuite("EP01_09.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.10 EP01_10 📄 Metadati: Lettura di File PGM

In questa attività, devi leggere un file immagine nel formato **PGM (Portable Gray Map)** ed estrarne le dimensioni dall'intestazione.

- Il formato **PGM (P2)** è un file di testo semplice (ASCII) che memorizza immagini in scala di grigi.
- Il file possiede un'**intestazione** strutturata nel seguente modo:

1. **Versione:** L'identificatore `P2`.
2. **Commenti:** Righe opzionali che iniziano con `#` (devono essere ignorate).
3. **Dimensioni:** Due interi che rappresentano **Larghezza** e **Altezza**.
4. **Massimo:** Un intero che rappresenta l'intensità massima (generalmente 255).

- Dopo l'intestazione, seguono i dati dei pixel.

📌 **Importante**:

- **Lettura del File:** Devi aprire il file indicato nell'esempio utilizzando la funzione `open()` di Python.
- **Ordine di Uscita:** Contrariamente all'ordine presente nel file, l'output atteso deve essere nel formato di tupla: `(Altezza, Larghezza, Canali)`.
- Poiché i file PGM sono in scala di grigi, il numero di **Canali** è sempre **1**.
- Vedi un simulatore interattivo per questa domanda alla **Figura 1.20** (regola le dimensioni per vedere come viene generata l'intestazione ASCII).

1.17.10.1 🧠 Comprendere il formato PGM

Il formato PGM è uno dei più semplici per l'elaborazione delle immagini. Essendo in testo puro, consente di visualizzare i metadati e persino i valori dei pixel aprendo il file in un blocco note:

Componente	Esempio	Significato
Numero Magico	`P2`	Identifica che è un PGM in formato testo (ASCII).
Commento	`# CREATOR...`	Riga informativa ignorata dal processore.
Dimensioni	`397 343`	**397** colonne (Larghezza) e **343** righe (Altezza).
Intensità	`255`	Definisce il valore del bianco puro (scala da 0 a 255).

1.17.10.2 📋 Compito (specifica per VPL)

Input:

Nessun input da tastiera. Il programma deve leggere il file `"aula01fig03b.pgm"` presente nella directory di esecuzione.

Output:

Una tupla contenente `(Altezza, Larghezza, 1)`.

1.17.10.3 📌 Esempi

Nome del File	Output Atteso	Osservazione
"aula01fig03b.pgm"	(343, 397, 1)	Nota l'inversione dell'ordine: Altezza prima

> ℹ Nota
>
> Il file necessario per questo EP verrà scaricato automaticamente dal repository tramite il seguente codice:

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0110-pgm.html

Figura 1.20: Simulatore EP01_10: Struttura del file PGM (Intestazione ASCII P2 e mappatura a tupla Python)

```
import os, urllib.request

BASE_URL = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/all/cap01/imagens"
file = "aula01fig03b.pgm"

opener = urllib.request.build_opener()
opener.addheaders = [('User-agent', 'Mozilla/5.0')]
urllib.request.install_opener(opener)

for f in [file]:
    if not os.path.exists(f):
        url = f"{BASE_URL}/{f}"
        try:
            urllib.request.urlretrieve(url, f)
            print(f"  File scaricato: {f}")
        except urllib.error.HTTPError as e:
            print(f"  Erro {e.code}: Non trovato in {url}")
```

```
%%writefile EP01_10.py
# la tua soluzione
```

```
Overwriting EP01_10.py
```

```
TestSuite("EP01_10.py").run()
```

```
<IPython.core.display.HTML object>
```

1.17.11 EP01_11 ↗ Analisi di Vicinato: Filtro di Massimo 1D

In questa attività, devi implementare un semplice filtro morfologico di massimo che opera su un segnale unidimensionale (vettore).

- Leggi un intero **n**, che rappresenta la dimensione del vettore.
- Leggi i **n** elementi interi che compongono il vettore originale **v1**.
- Crea un nuovo vettore **v2**, in cui ogni posizione i è il risultato del confronto tra l'elemento corrente e i suoi vicini immediati:

$$v2[i] = \max(v1[i-1],\ v1[i],\ v1[i+1])$$

📌 **Importante**:

- **Bordi:** Alle estremità del vettore (indici 0 e $n-1$), il vicinato ha solo due elementi (quello stesso e l'unico vicino disponibile). All'indice 0, confronta solo $v1[0]$ e $v1[1]$. All'ultimo indice, confronta solo $v1[n-2]$ e $v1[n-1]$.
- **Output:** Stampa l'intestazione "v2:" seguita dai valori del vettore risultante, uno per riga.
- Per un simulatore interattivo su questo problema, vedi **Figura 1.21** (passa il mouse sui risultati per visualizzare la finestra di vicinato utilizzata nel calcolo).

1.17.11.1 🧠 Perché analizzare i vicini?

Nell'elaborazione delle immagini, il valore di un pixel raramente è isolato; dipende dal contesto circostante. Il **Filtro di Massimo** è la base dell'operazione di **Dilatazione** nella morfologia matematica e serve a:

Funzione	Effetto Visivo
Enfatizzazione	Espande strutture luminose e "ingrossa" oggetti chiari.
Rimozione del Rumore	Elimina piccoli punti neri (rumore "sale e pepe" scuro).
Riempimento	Chiude piccoli buchi o lacune in forme binarie.

1.17.11.2 📋 Compito (specifica per VPL)

Input:

Un intero `n`.

Nelle righe successive, i `n` elementi interi del vettore.

Output:

La *stringa* `v2:` sulla prima riga.

Nelle righe successive, ogni elemento di `v2` (uno per riga).

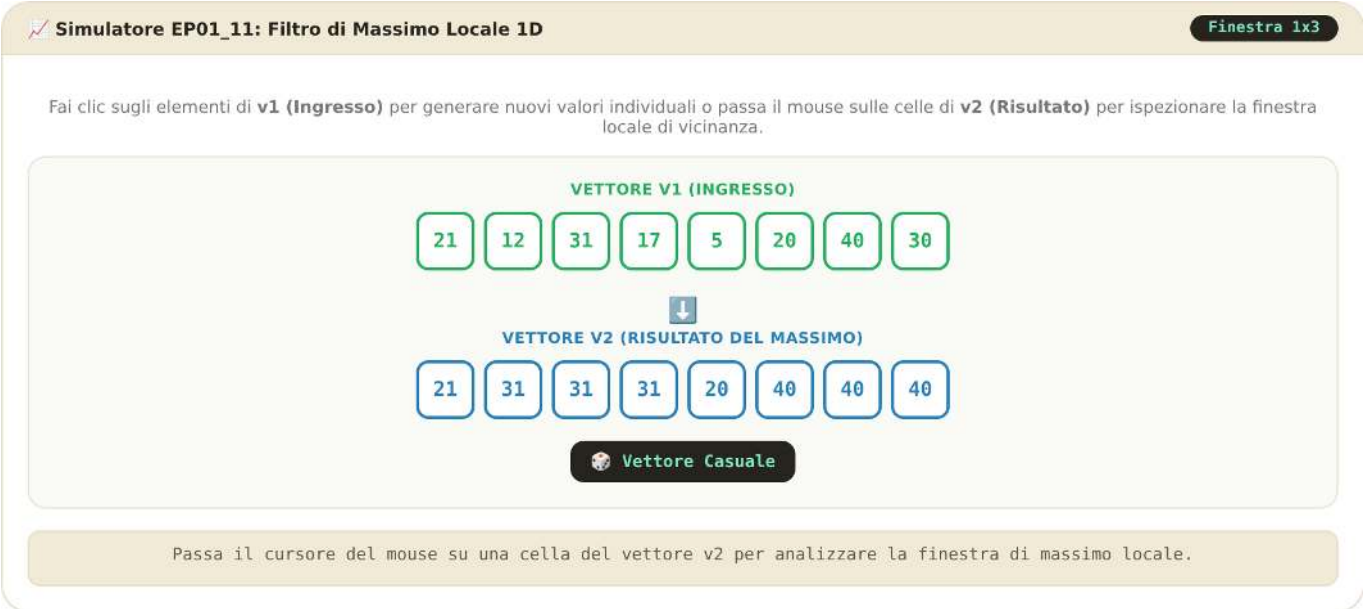
1.17.11.3 📌 Esempi

Input	Output	Osservazione
5	v2:	All'indice 1: max(10, 20, 5) = 20
10	20	
20	20	
5	30	
30	30	
15	30	

```
%%writefile EP01_11.py
# la tua soluzione
```

```
Overwriting EP01_11.py
```

```
TestSuite("EP01_11.py").run()
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap01/sim-ep0111-filtro-maximo.html

Figura 1.21: Simulatore EP01_11: Filtro di Massimo Locale 1D (Vicinato 1x3 con Condizione di Bordo)

```
<IPython.core.display.HTML object>
```

Capitolo 2

Dalla Cattura al Pixel - Campionamento, Quantizzazione e Connettività

Executar Colab Abrir GitHub

Questo capitolo approfondisce la comprensione dell'immagine digitale, passando dalla natura fisica della cattura alla sua rappresentazione matematica discreta. Indaghiamo come la luce diventa dato e come l'organizzazione spaziale dei pixel definisce le relazioni di vicinanza e connettività essenziali per algoritmi avanzati di Visione Artificiale (VC).

2.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- Spiegare il modello fisico della formazione dell'immagine basato su illuminazione e riflettanza.
- Distinguere i meccanismi della visione umana da quelli dei sensori digitali.
- Comprendere i processi di **campionamento** (discretizzazione dello spazio) e **quantizzazione** (discretizzazione dell'intensità).
- Descrivere le relazioni topologiche tra pixel: vicinanza, adiacenza, connettività e distanze.
- Eseguire trasformazioni geometriche di base (traslazione, rotazione, scala) preservando la qualità.
- Visualizzare in pratica gli effetti della variazione della risoluzione spaziale e della profondità di bit.

2.2 L'Occhio e la Fotocamera - Elementi della Percezione Visiva

La formazione di un'immagine digitale inizia con la cattura della luce riflessa dagli oggetti. Come illustrato nella Figura 2.1, sia l'occhio umano che le fotocamere digitali seguono principi ottici simili per mettere a fuoco la luce su una superficie sensibile, sebbene utilizzino meccanismi biologici ed elettronici distinti per la trasduzione del segnale.

2.2.1 Visione umana

L'occhio funziona come un sistema ottico complesso: la luce attraversa la cornea, l'umor acqueo, la pupilla (controllata dall'iride) e il cristallino — che regola dinamicamente la messa a fuoco — fino a raggiungere la retina. Sulla retina si trovano i fotorecettori: i **coni** (6 milioni), concentrati nella fovea, sono responsabili della visione dei colori e dei dettagli, mentre i **bastoncelli** (120 milioni) garantiscono la visione in condizioni di bassa luminosità (visione scotopica), rilevando solo intensità di grigio. Il punto cieco è la regione da cui origina il nervo ottico, priva di recettori.

2.2.2 Sensori digitali

Nelle fotocamere, i sensori di immagine svolgono il ruolo della retina. I due tipi più comuni sono il **CCD** (*Charge-Coupled Device* - Dispositivo a Carica Accoppiata) e il **CMOS** (*Complementary Metal-Oxide-Semiconductor* - Semiconduttore a Ossido Metallico Complementare). Il sensore è composto da una matrice di fotositi (pixel) che accumulano carica elettrica proporzionale alla luce incidente.

Per la ricostruzione dei colori, si utilizza il **Filtro di Bayer**, una matrice di filtri colorati che consente a ciascun pixel di catturare una sola componente cromatica: rosso, verde o blu (RGGB - *Red, Green, Green, Blue*). Successivamente, un **ADC** (*Analog-to-Digital Converter* - Convertitore Analogico-Digitale) quantizza questa carica in valori numerici, definiti da una profondità di bit (es.: 8 bit, che producono 256 livelli di intensità).

> **Curiosità**
>
> Sebbene l'occhio umano abbia milioni di recettori, la risoluzione ad alta definizione è limitata alla fovea (visione centrale), equivalente a circa 120 × 120 pixel. La percezione di una scena completa in alta risoluzione è il risultato di un intenso post-elaborazione svolto dal cervello.

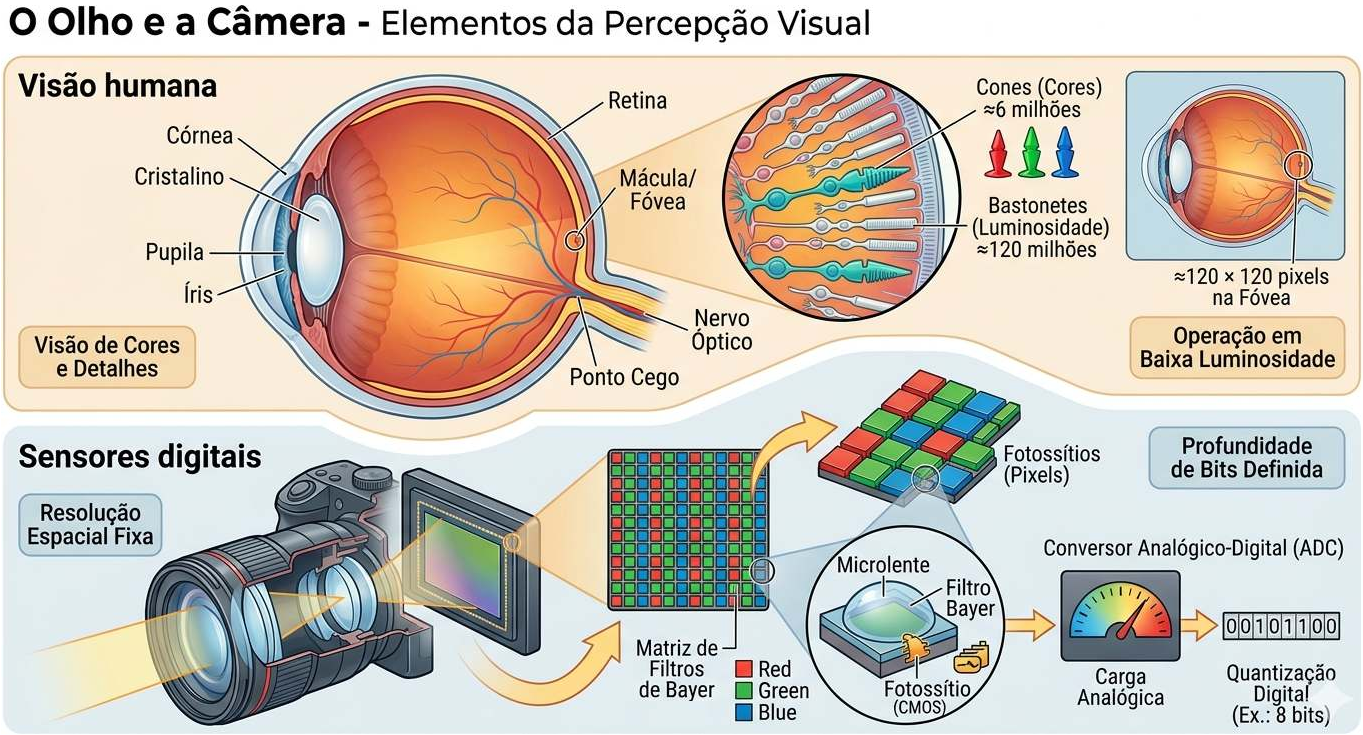


Figura 2.1: Confronto didattico tra il sistema visivo biologico e quello elettronico: in alto, l'anatomia dell'occhio umano che evidenzia la retina e i fotorecettori (coni e bastoncelli); in basso, la struttura di una fotocamera digitale che dettaglia il sensore CMOS, la matrice di filtri di Bayer (RGGB) e il processo di quantizzazione digitale eseguito dall'ADC.

2.3 Illusioni Ottiche: Le Sfide della Percezione Visiva

Mentre i sensori digitali catturano l'intensità della luce in modo lineare e oggettivo, il sistema visivo umano interpreta la scena basandosi su contesto, esperienze pregresse e meccanismi biologici di sopravvivenza. Le illusioni ottiche non sono "errori" dell'occhio, ma evidenze dell'intenso **post-elaborazione cerebrale** svolto nella corteccia visiva.

2.3.1 Ambiguità e Contesto

Il cervello cerca costantemente di dare senso a pattern ambigui. Nell'esempio del **Vaso di Rubin** (vedi Figura 2.2), la percezione alterna tra la figura (vaso) e lo sfondo (due volti), dimostrando che non riusciamo a elaborare entrambe le interpretazioni simultaneamente. L'illusione dell'**Elefante di Shepard**, invece, gioca con la nostra incapacità di riconciliare linee di contorno che suggeriscono volume in posizioni logicamente impossibili.

2.3.2 Luminosità e Contrasto Locale

Molte illusioni derivano dall'**inibizione laterale**, meccanismo mediante il quale neuroni vicini nella retina competono tra loro per evidenziare i bordi. Nella **Griglia Scintillante**, punti scuri "fantasma" sembrano apparire alle intersezioni bianche a causa di questa elaborazione locale del contrasto.

L'illusione dell'**Ombra sulla Scacchiera di Adelson** è forse la più impressionante per la VC: il quadrato "A" e il quadrato "B" hanno esattamente lo stesso valore di grigio nel sensore (o nel file digitale), ma il cervello "corregge" la luminosità di "B" comprendendo che si trova sotto un'ombra proiettata, percependo quest'ultimo come più chiaro.

2.3.3 Geometria e Prospettiva

La percezione della profondità può essere ingannata da costruzioni geometriche che sfidano la logica tridimensionale da un particolare angolo di osservazione. La **Scala di Schröder** sfrutta l'ambiguità della prospettiva per creare un oggetto che sembra salire o scendere a seconda di come viene osservato, evidenziando come la nostra interpretazione di "sopra" e "sotto" dipenda dal punto di fuga.

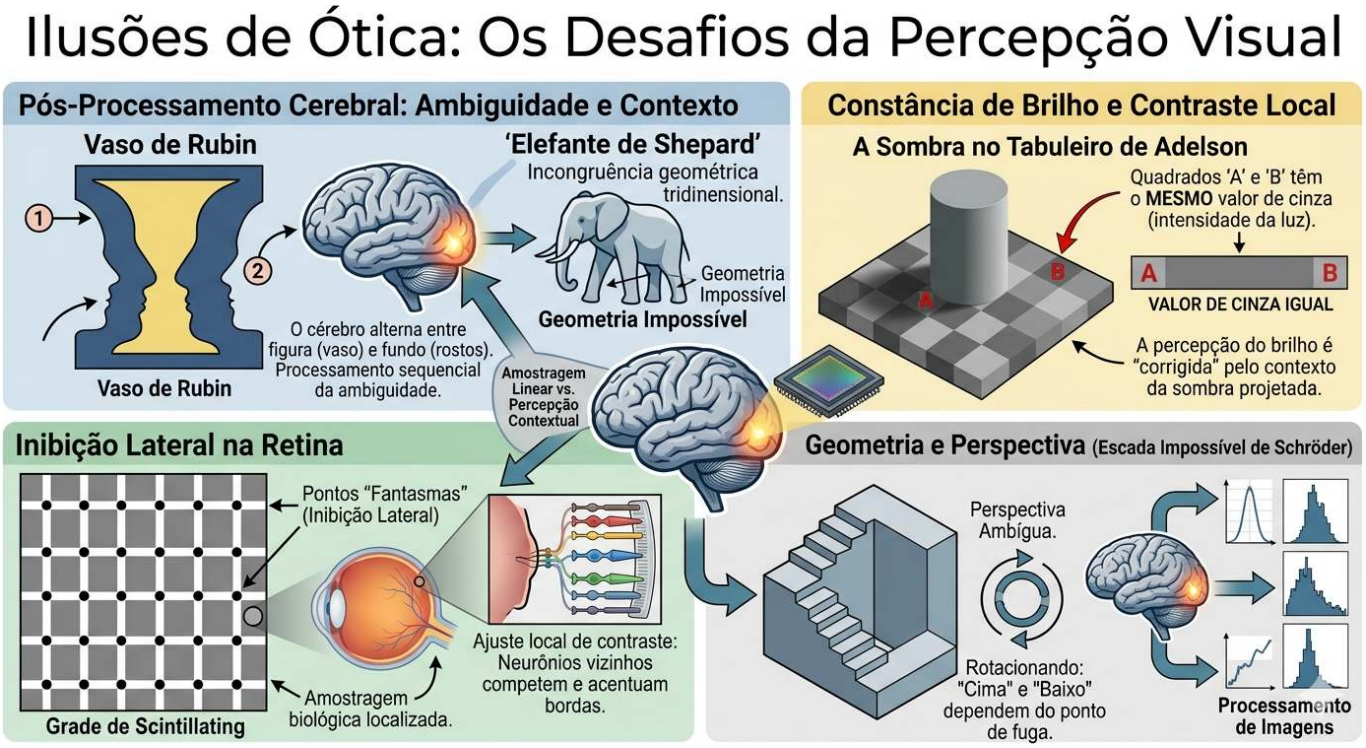


Figura 2.2: Raccolta di sfide percettive: (in alto a sinistra) Vaso di Rubin — ambiguità figura-sfondo; (in alto al centro) Elefante di Shepard — incongruenza geometrica; (in alto a destra) Ombra di Adelson — costanza della luminosità basata sul contesto; (in basso a sinistra) Griglia di punti — inibizione laterale; (in basso a destra) Scala di Schröder.

2.4 Il Modello Matematico della Formazione dell'Immagine

Un'immagine può essere modellata come il prodotto di due funzioni:

$$f(x, y) = i(x, y) \cdot r(x, y) \tag{2.1}$$

dove:

- $i(x, y)$ è l'**illuminazione** incidente sulla scena (energia luminosa per unità di area), determinata dalla sorgente di luce;

- $r(x, y)$ è la **riflettanza** dell'oggetto (frazione di luce riflessa), determinata dalle proprietà ottiche della superficie, con $0 < r(x, y) < 1$.

Nella pratica, le due componenti variano su scale spaziali distinte: $i(x, y)$ tende a variare lentamente nello spazio, mentre $r(x, y)$ può presentare variazioni brusche associate a trame, bordi e dettagli fini. Le tecniche di elaborazione delle immagini digitali (PDI) spesso cercano di separare o compensare queste componenti, come nei metodi di correzione dell'illuminazione non uniforme.

La Figura 2.3 illustra come questo modello si manifesti nelle immagini digitali a colori, rappresentate da molteplici canali spettrali (RGB) e, opzionalmente, da un canale aggiuntivo di trasparenza (RGBA).

2.4.1 Rappresentazione Digitale e Canali Spettrali

Nelle immagini digitali a colori, la funzione $f(x, y)$ della Equazione 2.1 è rappresentata da molteplici canali spettrali. Nello standard **RGB**, ogni pixel memorizza tre campioni indipendenti:

$$\mathbf{f}(x, y) = [R(x, y),\ G(x, y),\ B(x, y)]$$

corrispondenti alle intensità delle componenti rossa (*Red*), verde (*Green*) e blu (*Blue*). Nelle immagini di tipo **RGBA**, si aggiunge un quarto canale:

$$\mathbf{f}(x, y) = [R(x, y),\ G(x, y),\ B(x, y),\ A(x, y)]$$

dove $A(x, y)$ rappresenta il canale **alfa** (*Alpha*), responsabile di codificare l'opacità del pixel. Per convenzione, il valore $A = 0$ indica un pixel completamente trasparente, mentre $A = 255$ (o 1) rappresenta un pixel completamente opaco.

Pertanto, un'immagine digitale a colori è strutturata come una matrice multidimensionale di dimensioni:

- **RGB:** $M \times N \times 3$
- **RGBA:** $M \times N \times 4$

in cui ogni posizione (x, y) memorizza i campioni associati alle proprietà ottiche di quella coordinata spaziale.

La conversione tra intervalli di intensità è diretta ed è data da:

$$f_{\text{norm}}(x, y) = \frac{f(x, y)}{255}$$

dove $f(x, y) \in [0,\ 255]$ e $f_{\text{norm}}(x, y) \in [0,\ 1]$.

Convenzione di scala tra librerie: Diversi ecosistemi di programmazione adottano intervalli di valori e ordinamenti distinti per rappresentare i canali digitali, come sintetizzato nella Tabella 2.1.

Tabella 2.1: Convenzioni di scala e ordine dei canali nelle principali librerie di PDI.

Libreria	Tipo predefinito	Intervallo di valori	Ordine delle bande	Forma dell'array/tensore
OpenCV	`uint8`	$[0, 255]$	**BGR**	$H \times W \times C$
Pillow	`uint8`	$[0, 255]$	RGB	$H \times W \times C$
NumPy	`uint8/float32`	$[0, 255]$ o $[0, 1]$	—	$H \times W \times C$
scikit-image	`float64`	[0,0, 1,0]	RGB	$H \times W \times C$
PyTorch	`float32`	[0,0, 1,0]	RGB	$C \times H \times W$
TensorFlow/Keras	`float32`	[0,0, 1,0]	RGB	$H \times W \times C$

Trascurare queste discrepanze è una fonte frequente di errori nei flussi di lavoro di VC — come, ad esempio, iniettare un'immagine letta tramite OpenCV (standard BGR, tipo `uint8`) direttamente in un modello profondo di PyTorch che presuppone il formato RGB normalizzato.

2.4.2 Preparazione dell'Ambiente Pratico

Il blocco seguente carica il modulo `morph.py` dal repository e dimostra, per un pixel sintetico, le diverse convenzioni di scala e ordine dei canali adottate dalle principali librerie di elaborazione delle immagini (PDI).

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(demo=True)
from morph import mm
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0

 Convenzioni di scala per libreria
 NumPy / Pillow / OpenCV (uint8): [200 100  50] → [0, 255]
 scikit-image / PyTorch (float):  [0.78431373 0.39215686 0.19607843] → [0.0, 1.0]
 OpenCV: attenzione - legge in BGR: [ 50 100 200] (canali invertiti)
```

2.4.3 Implementazione della Lettura delle Immagini in `morph.py`

Il frammento seguente mostra il codice sorgente della funzione `mm.read`, permettendo di verificare direttamente come la libreria `morph.py` implementa la lettura delle immagini.

```
import inspect
print(inspect.getsource(mm.read))
```

```
    @staticmethod
    def read(file, pil=False, grayscale=False):
        """Lê imagem (local, URL ou Google Drive) → PIL.Image, ndarray 2D ou RGB 3D."""
        import re, requests
        from PIL import Image
        from io import BytesIO
        from urllib.request import urlopen, Request
        np = mm._get_np()

        # - fonte: URL / Google Drive -
        if isinstance(file, str) and file.startswith(("http://", "https://", "id=")):
            m = re.search(r"id=([\w-]+)", file) or re.search(r"/d/([\w-]+)", file)
            url = f"https://drive.google.com/uc?export=view&id={m.group(1)}" \
                if m and ("id=" in file or "drive.google.com" in file) else file
            hdr = {"User-Agent": "Mozilla/5.0 AppleWebKit/537.36 Chrome/124 Safari/537.36"}
            try:
                r = requests.get(url, headers=hdr, timeout=20)
                if r.status_code == 429: raise requests.exceptions.HTTPError()
                r.raise_for_status()
                file = BytesIO(r.content)
            except:
                file = BytesIO(urlopen(Request(url, headers=hdr), timeout=20).read())

        img = Image.open(file)
        img.load()
```

```
        if pil:                     return img
        if grayscale:               return np.array(img.convert("L"))   # (H,W)
        if img.mode == "L":         return np.array(img)                # (H,W)
        if img.mode == "RGBA":                                          # fundo branco
            bg = Image.new("RGB", img.size, (255, 255, 255))
            bg.paste(img, mask=img.split()[3])
            return np.array(bg)
        return np.array(img.convert("RGB"))                             # (H,W,3)
```

2.4.4 RGB e RGBA: Esempio Pratico con l'Immagine Mandrill

L'esempio seguente carica l'immagine Mandrill in formato RGB, costruisce artificialmente un canale alfa con gradiente orizzontale e visualizza entrambe le rappresentazioni, illustrando concretamente le strutture matriciali $M \times N \times 3$ e $M \times N \times 4$.

```python
import os
import numpy as np

# https://commons.wikimedia.org/wiki/File:Mandrill-k-means.png
url    = "https://upload.wikimedia.org/wikipedia/commons/a/ab/Mandrill-k-means.png"
caminho = "imagens/mandrill.png"

#   Lettura RGB
if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    img_pil = mm.read(url, pil=True)
    mm.write(img_pil, caminho)
else:
    img_pil = mm.read(caminho, pil=True)

img_rgb = np.array(img_pil.convert("RGB"))          # shape: (M, N, 3), uint8

#   Creazione del canale alfa (gradiente orizzontale)
h, w, _ = img_rgb.shape
alpha    = np.tile(np.linspace(0, 255, w, dtype=np.uint8), (h, 1))
img_rgba = np.dstack([img_rgb, alpha])               # shape: (M, N, 4), uint8

#   Diagnostica
print(f"RGB   → shape={img_rgb.shape}, dtype={img_rgb.dtype}, "
      f"intervallo=[{img_rgb.min()}, {img_rgb.max()}]")
print(f"RGBA  → shape={img_rgba.shape}, dtype={img_rgba.dtype}, "
      f"intervallo=[{img_rgba.min()}, {img_rgba.max()}]")
print(f"Pixel (0,0): RGB={img_rgb[0,0]}  |  RGBA={img_rgba[0,0]}")

#   Normalizzazione float32 (standard scikit-image / PyTorch)
img_rgb_norm = img_rgb.astype(np.float32) / 255.0
print(f"\nNormalizzato (float32): intervallo=[{img_rgb_norm.min():.2f},
{img_rgb_norm.max():.2f}]")
print(f"Pixel (0,0) normalizzato: {img_rgb_norm[0,0]}")

#   Visualizzazione
mm.show(
    [img_rgb, img_rgba],
    title=["RGB  -  $M \times N \times 3$", "RGBA  -  $M \times N \times 4$"],
    cols=2, axis=True
)
```

```
RGB   → shape=(512, 512, 3), dtype=uint8, intervallo=[14, 248]
RGBA  → shape=(512, 512, 4), dtype=uint8, intervallo=[0, 255]
Pixel (0,0): RGB=[125 110  58]  |  RGBA=[125 110  58   0]
```

```
Normalizzato (float32): intervallo=[0.05, 0.97]
Pixel (0,0) normalizzato: [0.49019608 0.43137255 0.22745098]
```

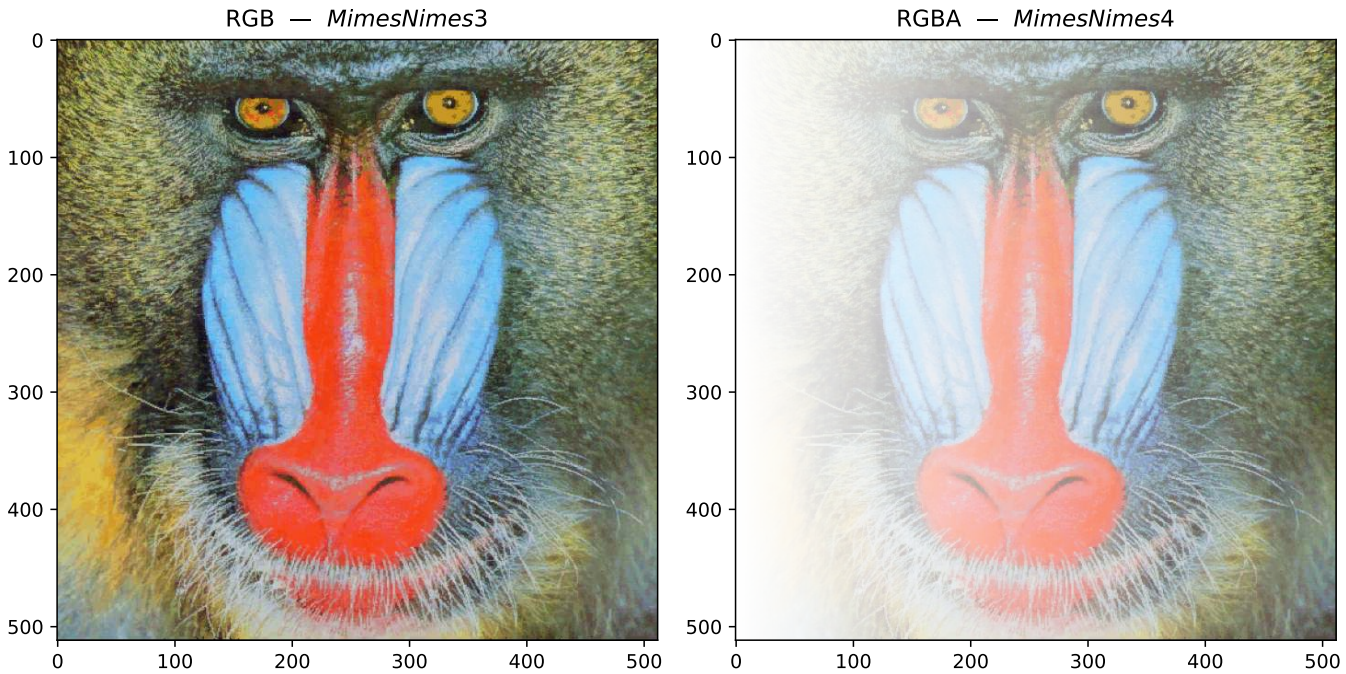


Figura 2.3: Immagine RGB (3 canali, $MimesNimes3$) e versione RGBA (4 canali, $MimesNimes4$) con trasparenza alfa graduale da sinistra a destra. Immagine Mandrill — USC SIPI Image Database (dominio pubblico).

2.5 Digitalizzazione: Campionamento e Quantizzazione

Per trasformare una scena continua in un'immagine digitale, sono necessari due processi: campionamento e quantizzazione.

2.5.1 Campionamento - Discretizzazione dello spazio

Il campionamento consiste nel misurare il valore della funzione $f(x, y)$ in punti equispaziati, formando una matrice di M righe (altezza) e N colonne (larghezza). Ciascun elemento di questa matrice è un **pixel**. La **risoluzione spaziale** è data da $M \times N$. Maggiore è la risoluzione, più dettagli spaziali vengono preservati, ma maggiore è anche il costo computazionale e di archiviazione.

2.5.2 Quantizzazione - Discretizzazione dell'intensità

La quantizzazione associa a ciascun pixel un valore numerico discreto, generalmente rappresentato da un intero di b bit. La **profondità di bit** definisce il numero di livelli di intensità: 2^b. Le immagini in scala di grigi utilizzano solitamente 8 bit (256 livelli). Le immagini a colori utilizzano tre canali da 8 bit (24 bit in totale).

Illustrazione: Se si utilizza solo 1 bit per pixel (bianco e nero), si perdono tutte le tonalità intermedie. Con 2 bit (4 livelli) si percepiscono già degradé grossolani. Con 8 bit, l'occhio umano difficilmente percepisce la discretizzazione (visione continua).

⚠ Errore di quantizzazione

L'errore di quantizzazione è la differenza tra il valore analogico reale e il valore discreto attribuito. Si manifesta come rumore di quantizzazione, visibile in regioni con gradiente uniforme quando si utilizzano pochi bit.

2.5.3 Effetti del campionamento e della quantizzazione

Gli esperimenti che seguono mostrano come la riduzione della risoluzione spaziale (sottocampionamento) e della profondità di bit degradino la qualità visiva. Utilizzare il codice per esplorare diversi fattori e livelli di grigio.

```
# Immagine di esempio (barbuto-striato) - base delle figure di campionamento,
# quantizzazione e trasformazioni geometriche di questo capitolo.
base    = "https://upload.wikimedia.org/wikipedia/commons"
arquivo = (
    "Area_de_Prote%C3%A7%C3%A3o_Ambiental_"
    "Quilombos_do_M%C3%A9dio_Ribeira_-_Thomas-"
    "Fuhrmann_%282023-_02%29_Malacoptila_striata.jpg"
)
url     = f"{base}/c/c5/{arquivo}"
caminho = "imagens/barbudo-rajado.jpg"

if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    mm.write(mm.read(url), caminho)

img_color = mm.read(caminho)
img_gray0 = mm.gray(img_color)
img_gray  = mm.crop(img_gray0, 820, 1850, 890, 1550)  # ritaglio per vedere i dettagli
print(f"Immagine originale: {img_color.shape}")
```

```
Immagine originale: (3067, 2047, 3)
```

```
import numpy as np

# Rilettura come oggetto PIL per l'accesso ai metadati EXIF (solo Python).
img_pil   = mm.read(caminho, pil=True)   # restituisce l'immagine PIL con EXIF
img_numpy = np.array(img_pil)
exif      = img_pil._getexif()           # ora funziona - img_pil è un'immagine PIL
print(f"Tipo dell'immagine: {type(img_color)}")
```

```
Tipo dell'immagine: <class 'numpy.ndarray'>
```

L'esperimento presentato in Figura 2.4 illustra il compromesso tra la **risoluzione spaziale** e il **costo di archiviazione** in memoria. Il codice utilizza la tecnica di sottocampionamento per fette (*slicing*) per ridurre la matrice originale di pixel secondo un fattore f, risultando in un risparmio di memoria — ad esempio, un fattore $f = 8$ riduce la dimensione del dato di 64 volte (8^2). Per fini di confronto visivo, le immagini ridotte vengono ripristinate alle dimensioni originali (512×512) tramite l'interpolazione per **vicino più prossimo** (`nearest`). Questo processo non recupera l'informazione persa, ma rende evidente l'effetto di ***aliasing*** e la struttura a blocchi (pixelizzazione) generata dalla bassa densità di dati della matrice campionata.

```
def subsample_simple(image, f):
    # Sottocampionamento tramite affettamento (slicing)
    reduced = mm.subsample(image, f)

    # Calcolo della memoria in KB
```

```
    mem_kb = reduced.nbytes / 1024
    label = f"{reduced.shape[1]}x{reduced.shape[0]}, {int(mem_kb)} KB\n(Fator {f})"

    # Ripristina la dimensione per la visualizzazione (H, W originali)
    res = mm.resize(reduced, (image.shape[1], image.shape[0]), method='nearest')
    return res, label

factors = [1, 4, 8, 12]

# Genera i risultati e li separa in liste per mm.show
results = [subsample_simple(img_gray, f) for f in factors]
imgs_list = [r[0] for r in results]
titles_list = [r[1] for r in results]

mm.show(imgs_list, titles=titles_list, cols=4, figsize=(16, 12))
```

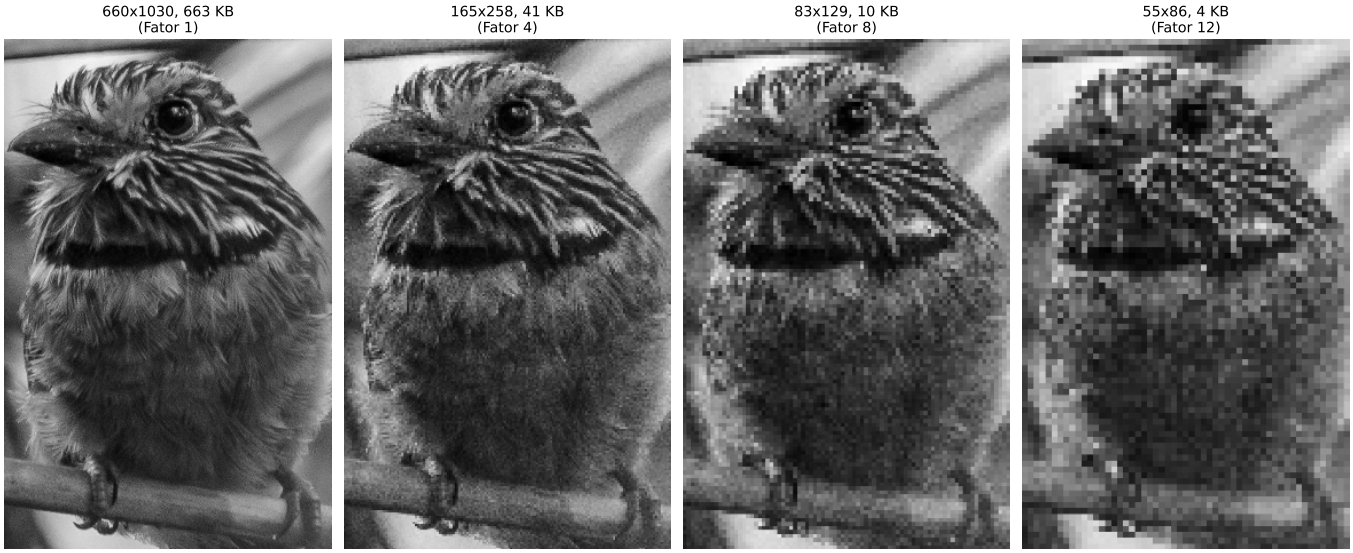


Figura 2.4: Effetto del sottocampionamento. I titoli mostrano le dimensioni (W x H) e la dimensione della matrice in memoria (KB).

L'esperimento in Figura 2.5 si concentra sulla **quantizzazione dell'intensità**, il processo di discretizzazione dell'ampiezza della funzione $f(x, y)$. Mentre il sottocampionamento influisce sulla griglia spaziale, la riduzione della profondità di bit limita la quantità di livelli di grigio disponibili per rappresentare la luminosità.

Riducendo la profondità da 8 bit (256 livelli) a valori inferiori, emerge l'effetto di **posterizzazione**, in cui i gradienti morbidi di una scena vengono sostituiti da transizioni brusche. Al limite di 1 bit, l'immagine diventa strettamente binaria, preservando solo la silhouette e perdendo i dettagli di texture e volume.

```
def quantize_simple(image, bits):
    """Riduce la profondità di bit e calcola i metadati di memoria."""
    levels = 2 ** bits
    # Normalizzazione e quantizzazione uniforme
    quantized = (np.floor(image / 256 * levels) / levels * 255).astype(np.uint8)

    # Calcolo della memoria in KB
    mem_kb = quantized.nbytes / 1024
    label = f"{bits} bits ({levels} níveis)\n{int(mem_kb)} KB"

    return quantized, label

# Lista di bit per il test (8 è il valore predefinito, 1 è il binario)
```

```
bits_test = [8, 4, 2, 1]

# Genera i risultati e li separa in liste per mm.show
results_q = [quantize_simple(img_gray, b) for b in bits_test]
imgs_q = [r[0] for r in results_q]
titles_q = [r[1] for r in results_q]

mm.show(imgs_q, titles=titles_q, cols=4, figsize=(16, 12))
```

Figura 2.5: Effetto della riduzione della profondità di bit. I titoli mostrano il numero di bit, i livelli e la dimensione in memoria (KB).

2.5.4 Analisi Tecnica

- **Dominio vs. Codominio**: Si noti che la risoluzione spaziale (dimensioni della matrice) rimane costante a 512x512; ciò che cambia è solo il codominio della funzione dell'immagine.
- **Costanza della Memoria**: Si osservi nei titoli che la dimensione in KB non diminuisce. Ciò accade perché NumPy memorizza ogni pixel quantizzato in un contenitore a 8 bit (`uint8`), indipendentemente dal fatto che il valore reale sia solo 0 o 1.
- **Percezione**: Il degrado visivo diventa critico al di sotto di 4 bit, dove l'occhio umano inizia a percepire i "confini" artificiali creati dalla mancanza di tonalità intermedie.

La limitazione dei tipi di dati più piccoli di un byte nell'ecosistema Python/NumPy è dovuta all'architettura hardware, che indirizza la memoria in blocchi di 8 bit (byte). Per mantenere la compatibilità con OpenCV e garantire l'efficienza, anche gli elementi binari vengono mappati in contenitori da 1 byte (`uint8` o `bool8`).

Sebbene linguaggi come l'ANSI C consentano la compressione di 8 pixel per byte (*bit-packing*), tale approccio richiede una decompressione costante per i calcoli e impone un'elevata complessità nella gestione dei puntatori. Come indicato in Tabella 2.2, si privilegia l'uso di `uint8` per la facilità di accesso ai vicini e per la versatilità nelle trasformazioni geometriche. Inoltre, i metodi nativi di NumPy e OpenCV eseguono l'elaborazione internamente a basso livello (C/C++), rendendo le operazioni vettorizzate più rapide rispetto alle implementazioni manuali con cicli annidati in Python.

Tabella 2.2: Confronto tra strategie di compressione ed efficienza di elaborazione.

Caratteristica	Python (NumPy/OpenCV)	ANSI C (Bit-packing)
Unità minima	1 Byte (8 bit)	1 Bit
Memoria (binaria)	256 KB (per 512x512)	32 KB (per 512x512)

Caratteristica	Python (NumPy/OpenCV)	ANSI C (Bit-packing)
Velocità	Elevata (Vettorizzazione in C)	Variabile (Lenta in caso di bit-shift)
Complessità	Bassa: Metodi pronti	Alta: Puntatori e Maschere

2.6 Relazioni tra Pixel - Topologia dell'Immagine

I pixel non sono elementi isolati; le loro posizioni relative definiscono concetti importanti per l'elaborazione.

2.6.1 Vicinato

Dato un pixel di coordinate (x, y), si definiscono due tipi principali di vicinato (per immagini su *grid* rettangolare):

- **Vicinato-4** (von Neumann): include i pixel nelle posizioni $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, $(x, y+1)$.
- **Vicinato-8** (Moore): include tutti gli otto pixel adiacenti (aggiunge i quattro diagonali).

La scelta del vicinato influenza operazioni come il rilevamento dei bordi, il calcolo dei gradienti e la connettività.

```
# # Creazione di una matrice 3x3 per esempio topologico
# viz = np.zeros((3, 3), dtype='uint8')

# # Definizione della Vicinanza-4 (N4) con valore diverso per evidenziazione
# viz[0, 1] = viz[2, 1] = viz[1, 0] = viz[1, 2] = viz[1, 1] = 255

# ovvero semplicemente (prova con numeri come argomenti):
viz = mm.secross()

# Visualizzazione della matrice per l'analisi delle coordinate
mm.drawImgPlt(viz, scale=40)
```

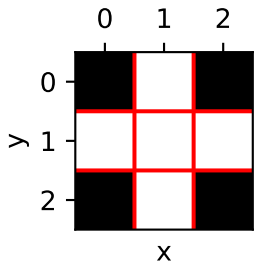


Figura 2.6: Illustrazione delle vicinanze 4 in una matrice 3x3. Al centro (1,1), il pixel di interesse.

```
import importlib
import morph
importlib.reload(morph)
from morph import mm
```

2.6.2 Adiacenza, connettività e percorsi

Due pixel sono **adiacenti** se sono in contatto secondo un intorno definito **e** soddisfano un criterio di valore (es.: stesso livello di intensità). Una **connettività** definisce una relazione di equivalenza tra pixel che formano una regione connessa. Un **percorso** è una sequenza di pixel adiacenti.

La **connettività-4** (N4) e la **connettività-8** (N8) possono produrre risultati differenti nella segmentazione e nel calcolo delle componenti connesse (etichettatura). Ad esempio, un motivo a scacchiera può essere completamente sconnesso in N4, ma totalmente connesso in N8.

2.6.3 Distanze tra pixel

Le metriche di distanza sono fondamentali per quantificare la prossimità fisica e la connettività tra gli elementi che compongono la griglia digitale. Come dimostrato nella Tabella 2.3, la scelta della metrica definisce il costo di spostamento tra pixel e altera il comportamento degli algoritmi di segmentazione e di analisi morfologica.

Per misurare la distanza tra due pixel $p(x_1, y_1)$ e $q(x_2, y_2)$, si utilizzano diverse funzioni metriche che impongono vincoli di movimento distinti sulla *griglia*:

Tabella 2.3: Confronto tra metriche di distanza applicate alla maglia di pixel.

Metrica	Definizione	Interpretazione
Euclidea	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$	Distanza **esatta** in linea retta (continua)
Manhattan (*City block*)	$\lvert x_1 - x_2\rvert + \lvert y_1 - y_2\rvert$	Movimenti orizzontali + verticali
Chebyshev (*Scacchiera*)	$\max(\lvert x_1 - x_2\rvert, \lvert y_1 - y_2\rvert)$	Maggiore spostamento tra gli assi

Queste distanze sono applicate in diversi contesti di PDI, inclusi algoritmi di interpolazione geometrica, trasformate della distanza, crescita di regioni e analisi delle forme.

> **Esempio pratico**
>
> Considerando due pixel con spostamenti relativi $\Delta x = 3$ e $\Delta y = 4$:
> - **Euclidea**: $\sqrt{3^2 + 4^2} = 5$ (ipotenusa del triangolo rettangolo).
> - ***Manhattan***: $3 + 4 = 7$ (somma dei cateti).
> - ***Chebyshev***: $\max(3, 4) = 4$ (predominio dello spostamento maggiore).

2.7 Archiviazione delle Immagini

La scelta del formato di file rappresenta un passo decisivo nel flusso di elaborazione, poiché determina come i dati di campionamento e quantizzazione verranno preservati o scartati. Come presentato in Tabella 2.4, ciascuna estensione bilancia in modo distinto la fedeltà dei dati e l'efficienza di archiviazione.

Tabella 2.4: Principali formati di archiviazione delle immagini digitali e le loro applicazioni nell'elaborazione digitale delle immagini (PDI).

Formato	Caratteristiche	Uso tipico
PGM	Formato semplice di mappa dei grigi (testo o binario).	Ricerca accademica e strumenti Unix.
BMP	Non compresso (o compressione semplice).	Windows, applicazioni legacy.
PNG	Compressione **senza perdita** (*lossless*).	Web, immagini con trasparenza.
JPEG	Compressione **con perdita** (*lossy*), ideale per fotografie.	Foto, fotocamere digitali.
TIFF	Supporta più livelli e compressione variabile.	Editoria, archiviazione.
RAW	Dati grezzi del sensore, senza elaborazione.	Fotografia professionale.

Formato	Caratteristiche	Uso tipico
DICOM	Standard medico con metadati clinici incorporati (paziente, apparecchiatura, protocollo).	Radiologia, tomografia, risonanza magnetica.

I metadati di un'immagine includono parametri quali larghezza, altezza, profondità di bit e codifica del colore. Nei formati scientifici vengono preservate anche le informazioni di calibrazione e i dettagli dell'acquisizione. Utilizzando la funzione `mm.read()`, la libreria `morph` preserva automaticamente questi dati affinché vengano rispettate le proprietà originali dell'immagine.

In contesti scientifici e ospedalieri, si privilegia lo standard **DICOM** (*Digital Imaging and Communications in Medicine*) per garantire l'assenza di perdita di precisione diagnostica. Repository pubblici come **The Cancer Imaging Archive (TCIA)**, la **Alzheimer's Disease Neuroimaging Initiative (ADNI)** e **PhysioNet** mettono a disposizione vasti set di dati in questo formato, inclusi metadati clinici anonimizzati essenziali per la ricerca scientifica.

2.7.1 Esempio: Estrazione di Metadati e Localizzazione GPS

A differenza della matrice di pixel pura ottenuta dalla lettura convenzionale — come nell'immagine dell'uccello presentata all'inizio di questo capitolo —, l'uso dell'argomento `pil=True` nel metodo `mm.read()` altera la natura dell'oggetto restituito (vedere Figura 2.7). Mentre il comportamento predefinito (`pil=False`) restituisce un `numpy.ndarray` RGB, la lettura con `pil=True` restituisce un oggetto specializzato della libreria Pillow, in grado di interpretare l'intestazione **EXIF**.

L'intestazione **EXIF** (*Exchangeable Image File Format*) funziona come un archivio tecnico della cattura, consentendo all'oggetto Pillow di interpretare un'ampia gamma di informazioni che vanno ben oltre le coordinate GPS. Utilizzando `pil=True`, il sistema acquisisce accesso al "DNA" dell'immagine, inclusi metadati hardware (marca e modello della fotocamera), impostazioni ottiche (apertura, lunghezza focale e tempo di esposizione) e parametri di illuminazione (uso del flash e bilanciamento del bianco).

Questa distinzione è importante nell'elaborazione digitale delle immagini (PDI), poiché trasforma la matrice di campionamento in un insieme di dati contestualizzato, dove le caratteristiche fisiche del sensore e dell'obiettivo possono essere utilizzate per normalizzare le luminosità o correggere distorsioni geometriche.

```
from PIL.ExifTags import TAGS
import os, numpy as np

base    = "https://upload.wikimedia.org/wikipedia/commons"
arquivo = (
    "Area_de_Prote%C3%A7%C3%A3o_Ambiental_"
    "Quilombos_do_M%C3%A9dio_Ribeira_-_Thomas-"
    "Fuhrmann_%282023-_02%29_Malacoptila_striata.jpg"
)
url     = f"{base}/c/c5/{arquivo}"
caminho = "imagens/barbudo-rajado.jpg"

# 1. Lettura - preserva oggetto PIL con EXIF
if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    img_obj = mm.read(url, pil=True)
    mm.write(img_obj, caminho)      # salva preservando EXIF
else:
    img_obj = mm.read(caminho, pil=True)

img_numpy = np.array(img_obj)      # conversione in NumPy

# 2. Estrazione e conversione GPS (Tag 34853)
```

```python
exif = img_obj._getexif()
if exif and (gps := exif.get(34853)):
    to_dec = lambda dms, ref: float(-(dms[0]+dms[1]/60+dms[2]/3600) if ref in 'SW'
                                    else (dms[0]+dms[1]/60+dms[2]/3600))
    lat, lon = to_dec(gps[2], gps[1]), to_dec(gps[4], gps[3])
    print(f"GPS Decimale: {lat:.6f}, {lon:.6f}")
    print(f"Mappe: https://www.google.com/maps/search/?api=1&query={lat},{lon}")

# 3. Diagnostica di tipi, dimensioni e accesso ai pixel
print(f"\nTipo PIL  : {type(img_obj)}   | Dimensioni (x,y): {img_obj.size}")
print(f"Pillow (0,0): {img_obj.getpixel((0, 0))}")
print(f"Tipo NumPy: {type(img_numpy)} | Dimensioni [y,x,c]: {img_numpy.shape}")
print(f"NumPy [0,0]: {img_numpy[0, 0]}")

# 4. Visualizzazione
mm.show(img_numpy, scale=30)
```

```
GPS Decimale: -24.587955, -48.629758
Mappe: https://www.google.com/maps/search/?api=1&query=-24.587955,-48.629758333333335

Tipo PIL  : <class 'PIL.JpegImagePlugin.JpegImageFile'>   | Dimensioni (x,y): (2047, 3067)
Pillow (0,0): (58, 96, 0)
Tipo NumPy: <class 'numpy.ndarray'> | Dimensioni [y,x,c]: (3067, 2047, 3)
NumPy [0,0]: [58 96  0]
```

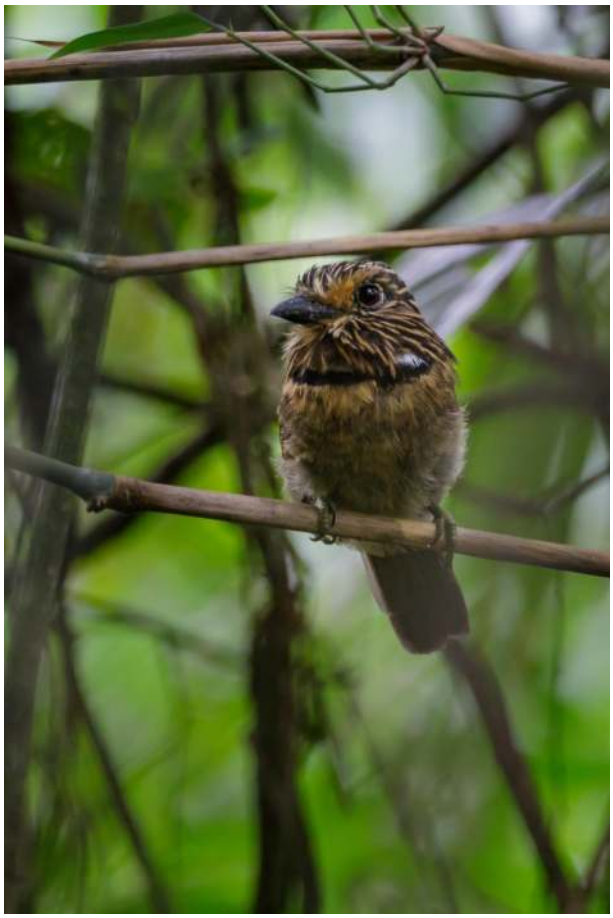


Figura 2.7: Area de Protezione Ambientale Quilombos do Médio Ribeira - Barbudo-rajado (Malacoptila striata). Credito: Thomas Fuhrmann (CC BY-SA 4.0).

Nota Pedagogica: La sottile differenza delle dimensioni

Si noti che la rappresentazione delle dimensioni cambia a seconda della struttura dati utilizzata:

- **In Pillow (`.size`):** Restituisce `(Larghezza, Altezza)` — nell'esempio: `(2047, 3067)`. È una prospettiva orientata al file immagine.
- **In NumPy (`.shape`):** Segue la convenzione matematica delle matrici: `(Righe/Altezza, Colonne/Larghezza, Canali)` — nell'esempio: `(3067, 2047, 3)`.

Questa distinzione è fondamentale per evitare errori di indicizzazione nell'implementazione di filtri manuali. Mentre l'oggetto Pillow trasporta il **"dove"** e il **"quando"** (contesto), l'array NumPy trasporta il **"quanto"** di luce (intensità) presente in ciascun punto dell'immagine.

2.7.2 Perché questa separazione è importante?

Caricando un'immagine tramite il percorso convenzionale (`pil=False`), il risultato è un `numpy.ndarray`, che contiene strettamente i valori numerici risultanti dalla **quantizzazione** e dal **campionamento**. Tuttavia, utilizzando `pil=True`, `mm.read()` restituisce un oggetto della classe `PIL.JpegImagePlugin.JpegImageFile`.

Questa classe mantiene il file "aperto" per consentire l'accesso al contesto dell'acquisizione prima che i dati vengano convertiti in una matrice grezza. Si noti che il pixel `(0, 0)` è identico nelle due rappresentazioni — `(58, 96, 0)` in Pillow e `[58, 96, 0]` in NumPy —, confermando che entrambe descrivono gli stessi dati, solo con interfacce diverse. Questa separazione è fondamentale: i pixel servono agli algoritmi; i metadati servono per la georeferenziazione, la catalogazione scientifica e le correzioni basate sull'hardware di acquisizione.

Per ispezionare tutti i metadati EXIF di un oggetto Pillow:

```
from PIL import ExifTags

exif_raw = img_obj._getexif()
if exif_raw:
    for tag_id, valor in sorted(exif_raw.items()):
        tag_nome = ExifTags.TAGS.get(tag_id, f"TAG_{tag_id}")
        print(f"  {tag_nome:40s} : {valor}")
```

2.8 Trasformazioni Geometriche Fondamentali

Le trasformazioni geometriche modificano la posizione dei pixel, mantenendo i valori di intensità. Sono fondamentali per l'allineamento, la correzione delle distorsioni e l'aumento dei dati (*data augmentation*) nell'apprendimento automatico.

Una **trasformazione affine** è qualsiasi mappatura che preservi la collinearità (punti su una retta rimangono su una retta) e i rapporti di distanza tra punti collineari. In 2D, ogni trasformazione affine può essere espressa in **coordinate omogenee** tramite una matrice 3×3:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{T}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.2}$$

La sottomatrice 2×2 in alto a sinistra $\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ controlla rotazione, scala e taglio; il vettore $(t_x, t_y)^\top$ controlla la traslazione. Le trasformazioni geometriche più utilizzate nell'elaborazione digitale delle immagini — traslazione, rotazione e scala — sono casi particolari di $\mathbf{T}$, e possono essere **composte** tramite moltiplicazione di matrici, nell'ordine $\mathbf{T} = \mathbf{T}_n \cdots \mathbf{T}_2 \mathbf{T}_1$.

Trasformazione inversa e interpolazione

Nell'implementazione pratica (`cv2.warpAffine`), si applica la **trasformazione inversa**: per ogni pixel (x', y') dell'immagine di destinazione, si calcola la posizione di origine $(x, y) = \mathbf{T}^{-1}(x', y')$ e si interpola il valore. Ciò evita i buchi nell'immagine risultante causati dalla mappatura diretta di pixel interi su posizioni non intere.

2.8.1 Traslazione

La traslazione è la trasformazione affine più semplice: sposta tutti i pixel di un vettore (t_x, t_y). In coordinate omogenee, è espressa dalla matrice:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \Rightarrow \quad \begin{cases} x' = x + t_x \\ y' = y + t_y \end{cases} \tag{2.3}$$

La terza riga della matrice garantisce che l'operazione rimanga nello spazio affine, consentendo di combinare traslazione, rotazione e scala tramite semplice moltiplicazione di matrici. In pratica, `cv2.warpAffine` utilizza solo le prime due righe (matrice 2×3), poiché la terza è sempre $[0, 0, 1]$.

I pixel spostati oltre l'area originale vengono scartati; le aree scoperte vengono riempite con 0 (nero). Si veda un esempio nella Figura 2.8.

```
img_tx1 = mm.translate(img_gray, 50, 50)
img_tx2 = mm.translate(img_gray, 100, 50)
mm.show([img_gray, img_tx1, img_tx2],
        titles=["Originale", "Traslazione (50,50)", "Traslazione (100,50)"],
        cols=3, figsize=(16, 12))
```

Figura 2.8: Esempio di traslazione dell'immagine dell'uccello con spostamenti (50,50) e (100,50).

2.8.2 Rotazione

La rotazione di un angolo θ attorno a un punto centrale (c_x, c_y) è composta da tre trasformazioni affini: traslazione verso l'origine, rotazione pura e traslazione di ritorno. La matrice risultante è:

$$\mathbf{T}_{\text{rot}} = \begin{bmatrix} \cos\theta & -\sin\theta & c_x(1-\cos\theta) + c_y \sin\theta \\ \sin\theta & \cos\theta & c_y(1-\cos\theta) - c_x \sin\theta \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Nell'implementazione, `cv2.getRotationMatrix2D` genera direttamente le prime due righe di $\mathbf{T}_{\text{rot}}$ (matrice 2×3 per `warpAffine`), accettando anche un fattore di scala s che moltiplica $\cos\theta$ e $\sin\theta$. Si veda l'esempio nella Figura 2.9.

Poiché la rotazione sposta i pixel in nuove posizioni non intere, `cv2.warpAffine` deve stimare il colore di ogni pixel di destinazione a partire dai vicini — processo chiamato **interpolazione**. Il parametro `interp` controlla questo comportamento:

- **`nearest`** (`INTER_NEAREST`): assegna il valore del pixel più vicino. Veloce, ma produce un effetto a scalini (*aliasing*) sui bordi diagonali.
- **`bilinear`** (`INTER_LINEAR`, predefinito): media ponderata dei 4 vicini più prossimi. Bilancia qualità e prestazioni — adatto alla maggior parte dei casi.
- **`bicubic`** (`INTER_CUBIC`): considera i 16 vicini su una superficie cubica. Produce bordi più morbidi, al costo di un maggiore carico di elaborazione.

```
img_rot30 = mm.rotate(img_gray, 30, interp='bilinear')
img_rot45 = mm.rotate(img_gray, 45, interp='bilinear')

mm.show([img_gray, img_rot30, img_rot45],
        titles=["Originale", "Rotazione 30°", "Rotazione 45°"],
        cols=3, figsize=(16, 12))
```

Figura 2.9: Esempio di rotazione dell'immagine dell'uccello a 30° e 45° utilizzando l'interpolazione bilineare.

2.8.3 Scala (Ridimensionamento)

Il ridimensionamento mediante fattori (s_x, s_y) è una trasformazione affine con matrice:

$$\mathbf{T}_{\text{scala}} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Quando $s > 1$ (ingrandimento), i pixel dell'immagine di destinazione mappano su posizioni non intere nell'origine — richiedendo interpolazione per stimare il valore. Quando $s < 1$ (riduzione), più pixel di origine contribuiscono a un singolo pixel di destinazione — richiedendo **decimazione**. Gli stessi tre metodi descritti nella rotazione sono disponibili in `mm.resize`, con la differenza che qui l'impatto visivo è più percettibile: nell'ingrandimento, `nearest` produce un effetto a blocchi (*pixelation*), mentre `bicubic` preserva meglio la nitidezza dei bordi, come indicato nella Tabella 2.5:

Tabella 2.5: Metodi di interpolazione disponibili in mm.resize e i rispettivi numeri di vicini utilizzati nel calcolo.

Metodo	Vicini utilizzati	Caratteristica
`'nearest'`	1	Veloce; produce effetto a blocchi (*pixelation*)
`'bilinear'`	4	Buon compromesso qualità/costo; bordi morbidi
`'bicubic'`	16	Maggiore nitidezza; preferito nei software professionali

Il parametro `size_or_factor` accetta sia uno scalare (fattore uniforme, es. `0.5` per ridurre della metà) sia una tupla `(larghezza, altezza)` per dimensioni assolute.

```
# 1. Ritaglio della regione del becco
y, x, offset = 210, 40, 40
crop = img_gray[y-offset:y+offset, x-offset:x+offset]
print(f"Immagine: {img_gray.shape} | Ritaglio: {crop.shape}")

# 2. Ingrandire 4× con mm.resize
crop_nearest  = mm.resize(crop, size_or_factor=4, method='nearest')
crop_bilinear = mm.resize(crop, size_or_factor=4, method='bilinear')

# 3. Visualizzazione comparativa
mm.show(
    [crop, crop_nearest, crop_bilinear],
    titles=["Originale (ritaglio)", "Vicino più prossimo (4×)", "Bilineare (4×)"],
    cols=3, figsize=(16, 12), dpi=200
)
```

```
Immagine: (1030, 660) | Ritaglio: (80, 80)
```

Figura 2.10: Confronto tra interpolazione con zoom sul dettaglio dell'occhio (ritaglio 60×60, ingrandito 4×). Si noti l'effetto a blocchi nel vicino più prossimo rispetto all'ammorbidimento nella bilineare.

2.8.4 Taglio (*Shear*)

Il taglio è una trasformazione affine che distorce l'immagine spostando ogni pixel proporzionalmente alla sua posizione su un asse. La matrice generale combina taglio orizzontale (sh_x) e verticale (sh_y):

$$\mathbf{T}_{\text{taglio}} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Per $sh_x \neq 0$ e $sh_y = 0$, ogni riga viene spostata orizzontalmente in modo proporzionale alla sua posizione verticale — producendo l'effetto di "inclinazione" caratteristico. Si veda l'esempio nella Figura 2.11.

```
img_shx = mm.shear(img_gray, shx=0.3)
img_shy = mm.shear(img_gray, shy=0.3)
img_shc = mm.shear(img_gray, shx=0.2, shy=0.2)

mm.show(
    [img_gray, img_shx, img_shy, img_shc],
    titles=["Originale", "Oriz. (shx=0.3)", "Vert. (shy=0.3)", "Combinato (0.2, 0.2)"],
    cols=4, figsize=(16, 12)
)
```

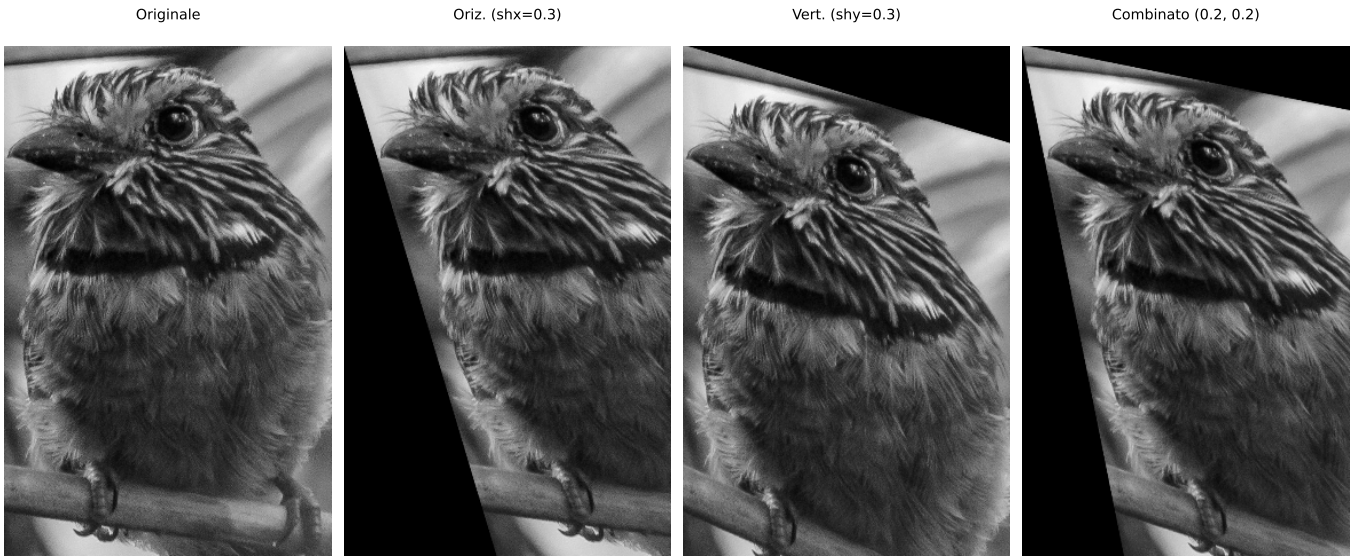


Figura 2.11: Esempio di taglio dell'immagine dell'uccello: orizzontale (shx=0.3), verticale (shy=0.3) e combinato (shx=0.2, shy=0.2).

2.9 Riassunto

In questo capitolo sono stati presentati i fondamenti della digitalizzazione e della topologia delle immagini:

- **Formazione dell'immagine:** $f(x, y) = i(x, y) \cdot r(x, y)$.
- **Campionamento:** discretizzazione dello spazio $\rightarrow$ risoluzione spaziale $M \times N$.
- **Quantizzazione:** discretizzazione dell'intensità $\rightarrow$ profondità di bit b.
- **Relazioni topologiche:** vicinato-4, vicinato-8, connettività, distanze (Euclidea, Manhattan, Chebyshev).
- **Trasformazioni geometriche:** traslazione, rotazione, scala (con interpolazione bilineare o vicino più prossimo).
- **Formati di file:** BMP, PNG, JPEG, TIFF, RAW; ciascuno con diversi compromessi tra qualità e dimensione.

Il Capitolo 3 tratterà **operazioni spaziali** come convoluzione, filtraggio e morfologia matematica (erosione, dilatazione).

2.10 🤖 Uso del Gemini Notebook come Tutor Complementare

In questa edizione, incoraggiamo l'uso del **Gemini Notebook** come strumento complementare di apprendimento. Questo strumento di IA utilizza esclusivamente i documenti forniti dall'autore come base di conoscenza, garantendo risposte coerenti con il contenuto del libro.

Per ogni capitolo, abbiamo preparato un progetto specifico sulla piattaforma. Per un'esperienza di studio ampliata, utilizza il seguente accesso:

🎓 Studia con il Tutor Intelligente

Per interagire con il contenuto di questo capitolo, accedi al seguente link. L'ambiente contiene materiali didattici in diversi formati, generati a partire dal **PDF** del capitolo. Sulla piattaforma, esplora in particolare le opzioni **Guida allo Studio** e **Conversazione** per approfondire la tua comprensione.
🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 02

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato costruito esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, oppure seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ **Avviso sul Contenuto Generato dall'IA**

L'IA è una potente alleata nello studio, ma il contenuto generato può contenere **errori o imprecisioni**. Consulta sempre **libri, articoli scientifici e altre fonti accademiche affidabili** per validare le informazioni. Quando possibile, esegui gli esempi pratici forniti in questo capitolo per verificarne i risultati.

2.11 Elenco di Esercizi

1. **(15%)** Spiega, con le tue parole, la differenza tra **campionamento** e **quantizzazione**. Fornisci un esempio concreto di ciascuno nel contesto di un'immagine digitale.
2. **(15%)** Considera un'immagine con risoluzione spaziale di 1024×768 pixel e profondità di 24 bit (8 bit per canale RGB). Calcola la dimensione totale non compressa dell'immagine in byte e in megabyte.
3. **(20%)** Utilizzando il codice del laboratorio, modifica il fattore di sottocampionamento a 3 e a 6. Descrivi visivamente cosa accade ai bordi degli oggetti. Che cos'è l'effetto di *aliasing*?
4. **(20%)** Per l'immagine in toni di grigio, applica la quantizzazione con 3 bit (8 livelli) e 5 bit (32 livelli). Confronta i risultati e spiega perché 5 bit possono già essere considerati sufficienti per molte applicazioni.
5. **(15%)** Dati due pixel $A = (10, 20)$ e $B = (15, 25)$, calcola le distanze Euclidea, di Manhattan e di Chebyshev tra di essi.
6. **(15%)** Usando la funzione `mm.rotate`, ruota l'immagine dell'uccello ad angoli di 90°, 180° e 270° con interpolazione bilineare. Confronta con la rotazione usando `method='nearest'`. In quali situazioni l'interpolazione del vicino più prossimo è ancora utile?

Riferimenti del Capitolo

La base teorica di questo capitolo si fonda sulle seguenti opere:

- Gonzalez (2018) per i concetti di campionamento, quantizzazione e relazioni tra pixel.
- Szeliski (2022) per trasformazioni geometriche e connettività.
- Bradski (2008) per l'implementazione pratica con OpenCV e `morph.py`.

Executar Colab | Abrir GitHub

2.12 💻 Parte pratica con esercizi di programmazione

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EP)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, eseguire `TestSuite("EP04_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare direttamente codice Python, senza salvare un file, utilizzare `run_code(codigo)` passando il codice come *stringa* in una variabile `codigo`:

```
codigo = """
from morph import mm
# ... tuo codice qui ...
"""
TestSuite("EP04_01").run_code(codigo)
```

2.12.1 EP02_01 ⊙ Regolazione di Luminosità e Contrasto

In questa attività, l'obiettivo è implementare un operatore puntuale per la **trasformazione lineare di intensità**, applicando la regolazione dinamica di luminosità e contrasto su un'immagine digitale.

2.12.1.1 📋 Linee Guida di Implementazione

L'algoritmo deve seguire il flusso di esecuzione di seguito:

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne) della matrice.
2. **Parametri:** Leggere il valore reale α (fattore di contrasto) e l'intero β (fattore di luminosità).
3. **Dati:** Leggere i valori interi della matrice originale.
4. **Mappatura:** Per ogni pixel p, calcolare il nuovo valore p' tramite l'equazione:

$$p' = \text{clip}(\text{round}(\alpha \cdot p + \beta))$$

5. **Output:** Visualizzare la matrice risultante con le dimensioni $L \times C$.

2.12.1.2 📌 Vincoli Computazionali

- **Arrotondamento (*Round*):** Si applica l'arrotondamento matematico all'intero più vicino prima della conversione di tipo.
- **Saturazione (*Clipping*):** I valori devono essere limitati all'intervallo $[0, 255]$ per preservare lo standard a 8 bit:

$$\text{clip}(x) = \max(0, \min(255, x))$$

- **Simulazione:** L'effetto dei parametri α e β sulla correzione istogrammatica può essere osservato nella Figura 2.12.

2.12.1.3 🧠 Fondamenti Teorici

Le modifiche alterano l'istogramma dell'immagine per regolare il profilo di illuminazione e la distinzione tonale.

Parametro	Funzione	Impatto Visivo
α (***Alpha***)	Scalare	Modula il **Contrasto**. Se $\alpha > 1$, espande l'istogramma; se $0 \leq \alpha < 1$, lo comprime.
β (***Beta***)	Additiva	Modula la **Luminosità**. Se positivo, trasla l'istogramma verso destra; se negativo, verso sinistra.
clip	Limitatore	Restringe la gamma dinamica, prevenendo errori di *underflow* e *overflow*.

2.12.1.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Valori di `alpha` (α) e `beta` (β).
- Righe successive: Elementi numerici della matrice originale.

Output:

- Matrice trasformata strutturata in L righe e C colonne.

2.12.1.5 📌 Esempi

La tabella seguente presenta un esempio pratico del comportamento atteso dell'algoritmo, evidenziando l'azione degli operatori di arrotondamento e saturazione.

Input	Output	Osservazione
1 4 1.5 -30 0 100 180 255	0 120 240 255	Si noti l'effetto di saturazione sull'ultimo pixel

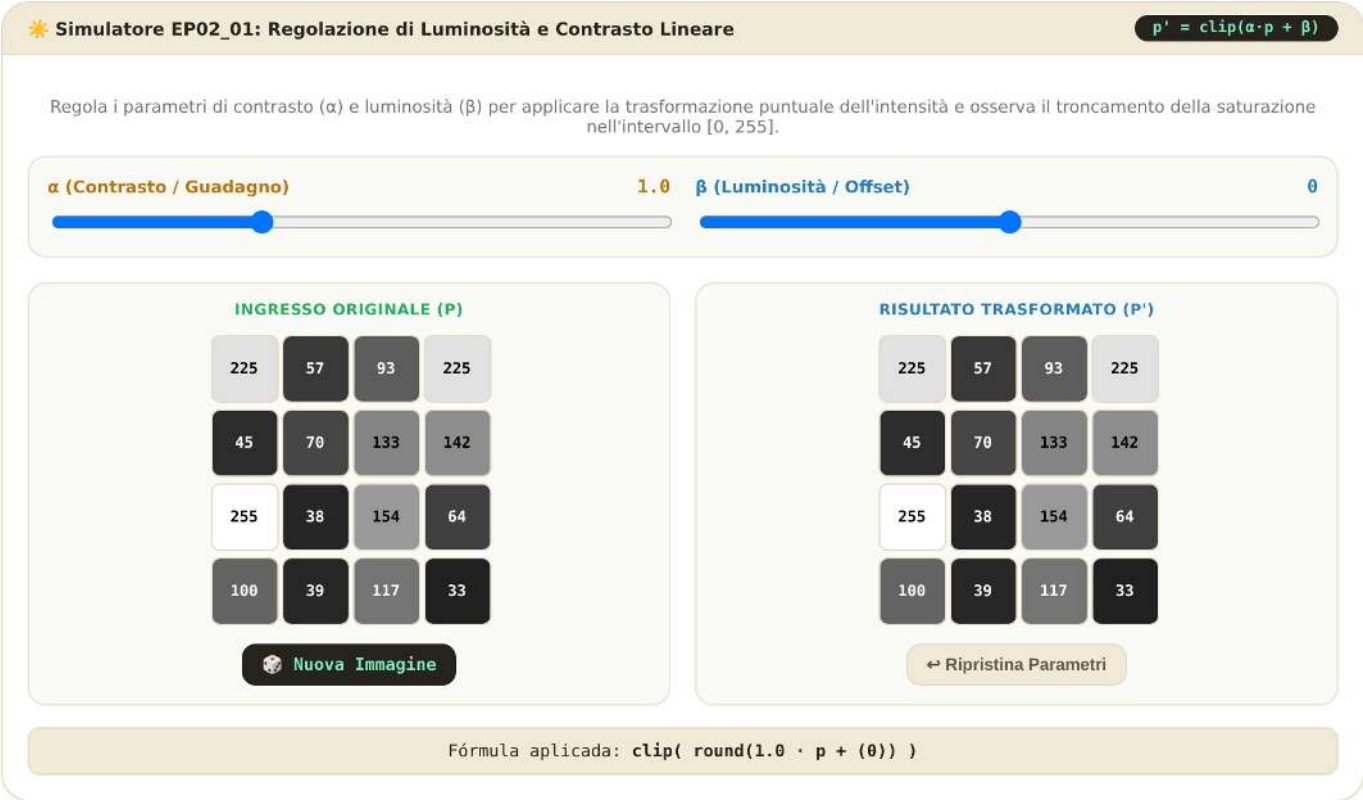
Esecuzione dei Test

Per valutare i test, eseguire `TestSuite("EP02_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

```
%%writefile EP02_01.py
# la tua soluzione
```

```
Overwriting EP02_01.py
```

```
TestSuite("EP02_01.py").run()
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0201-brilho-contraste.html

Figura 2.12: Simulatore EP02_01: Regolazione di Luminosità e Contrasto Lineare (p' = p +)

```
<IPython.core.display.HTML object>
```

2.12.2 EP02_02 🔬 Sottocampionamento Spaziale

In questa attività, devi implementare la riduzione della risoluzione spaziale di un'immagine attraverso il processo di sottocampionamento.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice originale.
- Leggi un valore intero f ($f \geq 1$), che rappresenta il fattore di campionamento.
- Leggi i valori interi della matrice originale.
- La nuova immagine deve essere costruita selezionando il pixel nella posizione $(f \cdot i, f \cdot j)$ dell'immagine originale.
- Stampa la matrice risultante con le nuove dimensioni.
- Vedi una simulazione di questo EP in Figura 2.13.

📌 **Importante**:

- **Dimensioni Finali:** L'immagine campionata avrà dimensioni $\lceil L/f \rceil \times \lceil C/f \rceil$. Nel contesto della programmazione, ciò equivale alla dimensione risultante da uno slicing con passo f.
- **Implementazione:** Non utilizzare funzioni pronte di librerie di elaborazione delle immagini (come OpenCV o PIL) per il ridimensionamento. Implementa la logica di selezione dei pixel manualmente o tramite slicing di matrici.
- **Aliasing:** Nota che questo processo può causare l'effetto di *aliasing* (effetto a scaletta), dove i dettagli fini vengono persi o compaiono pattern indesiderati.

2.12.2.1 🧠 Discretizzazione dello Spazio

Il sottocampionamento riduce la risoluzione spaziale di un'immagine, selezionando solo un pixel ogni f pixel in ciascuna direzione. È il processo inverso dell'interpolazione:

Parametro	Funzione	Effetto
Fattore f	Passo di campionamento	Definisce l'intervallo di selezione. Un fattore 2 riduce larghezza e altezza della metà.
Risoluzione	Densità dei pixel	Riduce la quantità totale di informazione spaziale dell'immagine.
Aliasing	Effetto collaterale	Comparsa di pattern a scaletta o a blocchi dovuti alla perdita di dettagli fini.

2.12.2.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene il fattore `f`.

Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice ridotta con le dimensioni corrispondenti allo slicing per `f`.

2.12.2.3 📌 Esempi

Input	Output	Osservazione
2 4 2 10 20 30 40 50 60 70 80	10 30	Il fattore 2 seleziona i pixel (0,0) e (0,2) della prima riga. La seconda riga viene ignorata.

```
%%writefile EP02_02.py
# Codice Python
```

```
Overwriting EP02_02.py
```

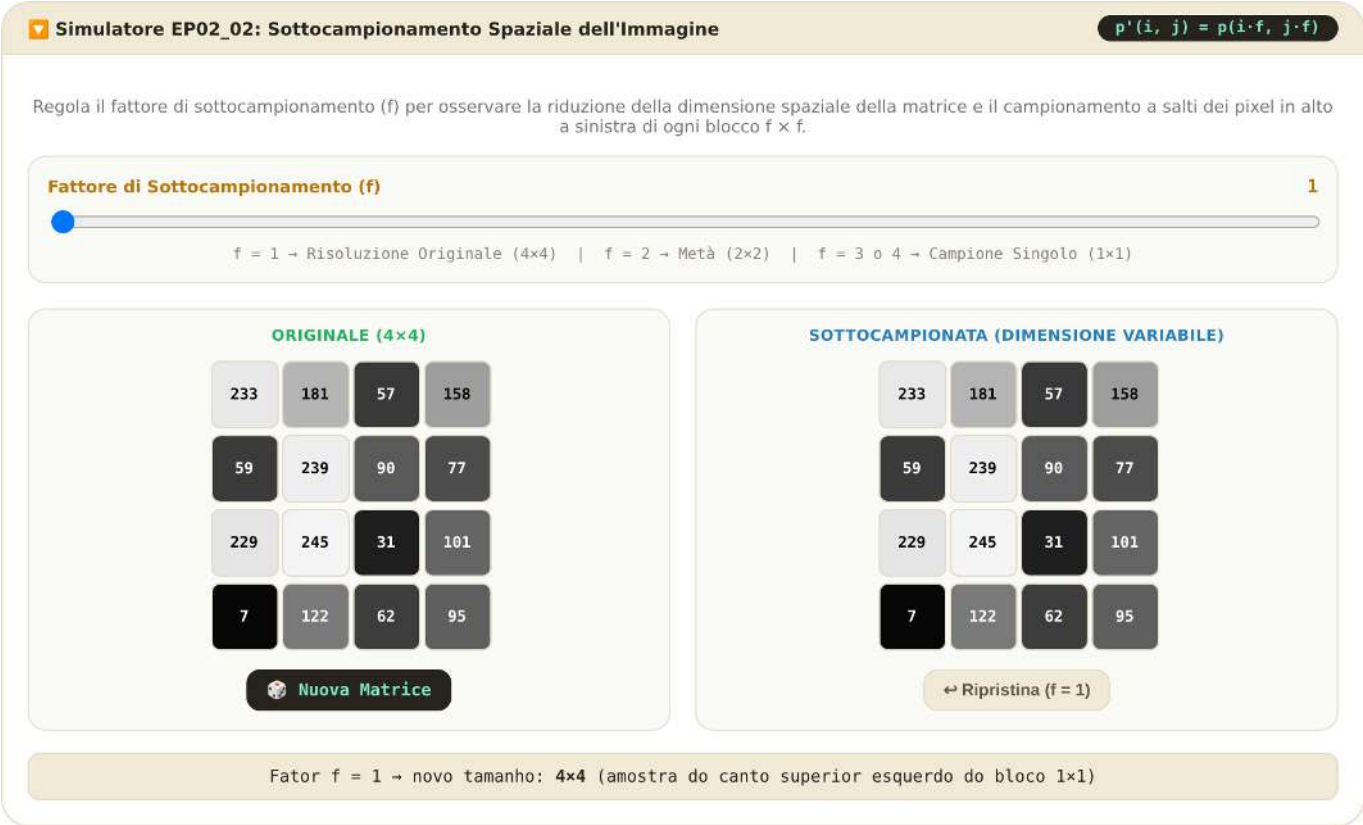
```
TestSuite("EP02_02.py").run()
```

```
<IPython.core.display.HTML object>
```

2.12.3 EP02_03 🎨 Quantizzazione dei Livelli di Grigio

In questa attività, devi implementare la quantizzazione uniforme di un'immagine, riducendo il numero di livelli di intensità di grigio originali a una nuova scala basata su un numero inferiore di bit.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi un intero k ($1 \leq k \leq 8$), che rappresenta il nuovo numero di bit dell'immagine.
- Calcola il numero di livelli ($N = 2^k$) e la dimensione dell'intervallo (passo).
- Per ogni pixel p, calcola il nuovo valore p' mappandolo all'indice del livello discretizzato corrispondente (variabile da 0 a $2^k - 1$).



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0202-subamostragem.html

Figura 2.13: Simulatore EP02_02: Sottocampionamento Spaziale (Riduzione di Risoluzione per Salto f)

- Stampa la matrice risultante con gli stessi valori delle dimensioni originali.
- Vedi in Figura 2.14 una simulazione di questo EP.

📌 **Importante**:

- **Posterizzazione:** Riducendo drasticamente i livelli (es: $k = 2$), noterai che le sfumature morbide si trasformano in bande cromatiche nette a causa della perdita di risoluzione dell'ampiezza.
- **Calcolo del Passo:** L'intervallo tra ogni livello è definito da $passo = 256/2^k$.
- **Mappatura:** Il metodo di quantizzazione uniforme per troncamento che mappa il pixel all'indice del rispettivo livello discretizzato è dato da:

$$p' = \left\lfloor \frac{p}{passo} \right\rfloor$$

In termini di implementazione (come in Python), ciò equivale alla divisione intera: `p' = p // passo`.

2.12.3.1 🧠 Discretizzazione dell'Ampiezza

Mentre il sottocampionamento gestisce la risoluzione spaziale, la quantizzazione si concentra sulla precisione del colore (ampiezza). Ridurre i bit significa semplificare l'informazione cromatica:

Parametro	Funzione	Effetto
Bit (k)	Profondità di colore	Definisce quanti toni diversi può avere l'immagine (2^k).
Passo	Intervallo di tono	Spaziatura tra i livelli di grigio consentiti.

Parametro	Funzione	Effetto
Posterizzazione	Fenomeno visivo	Trasformazione di variazioni continue in blocchi di colore solido.

2.12.3.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene il numero di bit `k`.

Le righe seguenti contengono gli elementi della matrice $L \times C$.

Output:

La matrice trasformata con gli indici dei livelli quantizzati, mantenendo la dimensione originale $L \times C$.

2.12.3.3 📌 Esempi

Input	Output	Osservazione
1 4 2 0 80 170 255	0 1 2 3	Con $k = 2$, abbiamo $2^2 = 4$ livelli discreti disponibili $(0, 1, 2, 3)$. Il passo è $256/4 = 64$. Applicando la divisione intera per elemento: $0//64 = 0$, $80//64 = 1$, $170//64 = 2$, $255//64 = 3$.
1 5 1 10 50 120 200 250	0 0 0 1 1	Con $k = 1$, abbiamo $2^1 = 2$ livelli (0 e 1). Passo $= 256/2 = 128$. I pixel minori di 128 risultano in 0, mentre i pixel maggiori o uguali a 128 risultano in 1.

```
%%writefile EP02_03.py
# Codice Python
```

```
Overwriting EP02_03.py
```

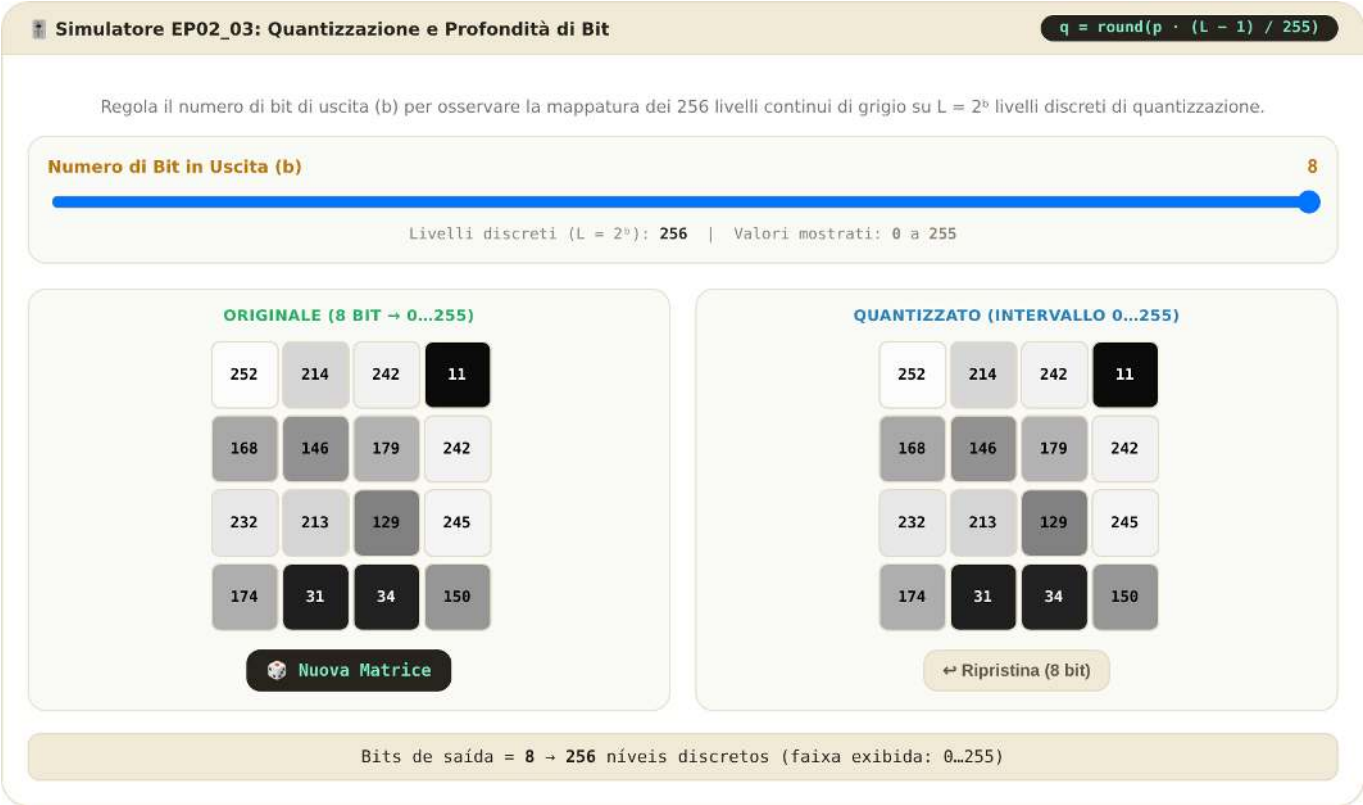
```
TestSuite("EP02_03.py").run()
```

```
<IPython.core.display.HTML object>
```

2.12.4 EP02_04 📐 Trasformata di Distanza in Immagine Binaria

Data un'immagine binaria in cui i pixel di valore **1** rappresentano l'*oggetto* e i pixel **0** rappresentano lo *sfondo*, la distanza di un pixel di sfondo riceve la **minore distanza al pixel dell'oggetto più vicino**. I pixel dell'oggetto ricevono distanza **0**. Per semplicità, si consideri che l'immagine ha **un solo oggetto con un singolo pixel di valore 1**.

Problema: Leggere un'immagine binaria $L \times C$ e una metrica, e calcolare questa distanza semplificata applicando una delle tre formule:



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0203-quantizacao.html

Figura 2.14: Simulatore EP02_03: Quantizzazione e Profondità di Bit (Riduzione del Numero di Livelli di Grigio)

$$d_{\text{Euclidea}} = \sqrt{(\Delta r)^2 + (\Delta c)^2}$$

$$d_{\text{City-block}} = |\Delta r| + |\Delta c|$$

$$d_{\text{Scacchiera}} = \max(|\Delta r|,\ |\Delta c|)$$

dove Δr è la differenza di righe e Δc la differenza di colonne tra due pixel.

2.12.4.1 🖼️ Perché è importante? - Applicazioni della DT

La Trasformata di Distanza (DT) compare in decine di pipeline di visione artificiale:

Metrica	Complessità	Applicazione tipica
Euclidea	🔴 $O(n^2)$ ingenuo	Scheletrizzazione, matching di forme
City-block	🟡 $O(n)$ con 2 passaggi	Morfologia, dilatazione/erosione
Scacchiera	🟢 $O(n)$ con 2 passaggi	Morfologia, dilatazione/erosione

2.12.4.2 📌 Requisiti Tecnici

- **Input:** * Prima riga: L e C (interi).
  - Seconda riga: nome della metrica (`euclidean`, `cityblock` o `chessboard`).
  - Successivamente, la matrice binaria $L \times C$ (valori 0 o 1).

- **Pixel dell'oggetto (1):** distanza = 0 (o 0.00 per euclidea).
- **Pixel di sfondo (0):** distanza all'unico pixel dell'oggetto nell'immagine.
- **Arrotondamento (euclidea):** stampare con **2 cifre decimali** (formato `:.2f`). City-block e Scacchiera producono interi — stampare senza decimali.
- **Output:** valori separati da spazio, una riga per ogni riga della matrice.
- Vedere in Figura 2.15 una simulazione di questo EP.

2.12.4.3 📌 Esempi

Input	Output	Osservazione
4 4 chessboard 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0	2 2 2 2 2 1 1 1 2 1 0 1 2 1 1 1	La distanza Scacchiera è $\max(\|dx\|, \|dy\|)$. L'unico pixel oggetto è $(2, 2) = 0$; gli altri memorizzano la loro distanza minima fino ad esso.

2.12.4.4 📌 Osservazioni finali

- Poiché l'immagine ha **un solo oggetto di un pixel**, la distanza di ogni pixel di sfondo è semplicemente la distanza di quel pixel all'unico punto dell'oggetto.
- L'implementazione può usare la forza bruta (percorrere tutti i pixel dell'immagine e calcolare la distanza direttamente), poiché L e C sono piccoli nei casi di test.
- Questo problema è un riscaldamento per la Trasformata di Distanza generale, che verrà affrontata nei capitoli successivi con più oggetti e algoritmi ottimizzati.

```
%%writefile EP02_04.py
# Codice Python
```

```
Overwriting EP02_04.py
```

```
TestSuite("EP02_04.py").run()
```

```
<IPython.core.display.HTML object>
```

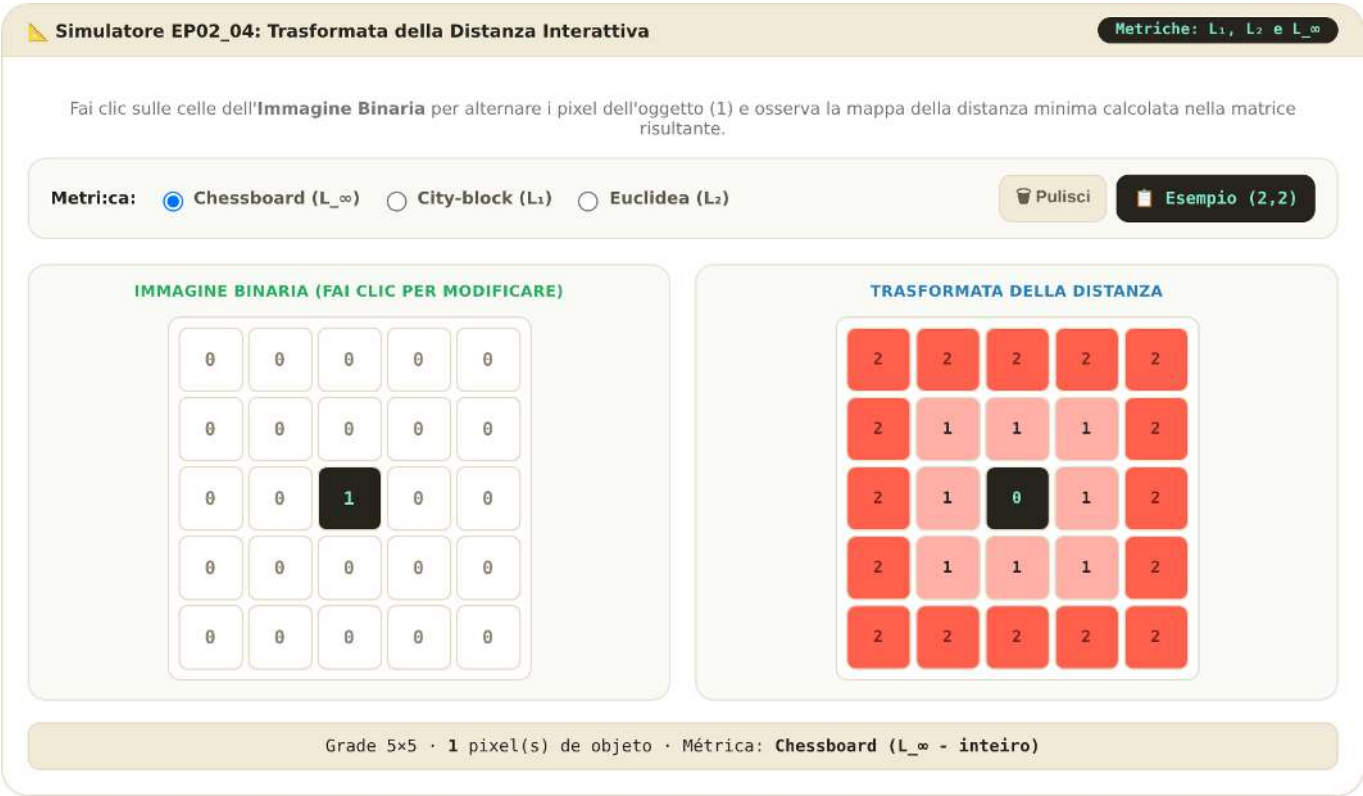
2.12.5 EP02_05 Traslazione dell'Immagine

In questa attività, devi implementare lo spostamento spaziale di un'immagine. La traslazione sposta ogni pixel dell'immagine originale in una nuova posizione basata su un vettore di spostamento.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi due interi t_x (spostamento orizzontale) e t_y (spostamento verticale).
- Leggi i valori interi della matrice originale.
- Calcola la nuova posizione (x', y') per ogni pixel originale (x, y).
- Stampa la matrice risultante con le stesse dimensioni dell'originale.
- Vedi una simulazione di questo EP in Figura 2.16.

📌 **Importante**:

- **Riempimento:** I pixel che "entrano" nell'immagine a causa dello spostamento e non hanno un corrispondente nell'originale devono essere riempiti con **0** (nero).



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0204-distancia.html

Figura 2.15: Simulatore EP02_04: Trasformata della Distanza in Immagine Binaria (Scacchiera, City-block ed Euclidea)

- **Scarto:** I pixel che, dopo la traslazione, escono dai limiti della matrice ($0 \dots L-1$ o $0 \dots C-1$) devono essere ignorati.
- **Coordinate:** Considera x come indice di riga e y come indice di colonna.

2.12.5.1 🧠 Spostamento Spaziale

Traslare un'immagine significa spostare tutti i suoi punti di una distanza fissa in direzioni specificate. Matematicamente, usando coordinate omogenee, l'operazione è descritta come:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Che risulta nelle semplici equazioni:

- $x' = x + t_x$
- $y' = y + t_y$

2.12.5.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene gli interi `tx` e `ty`.

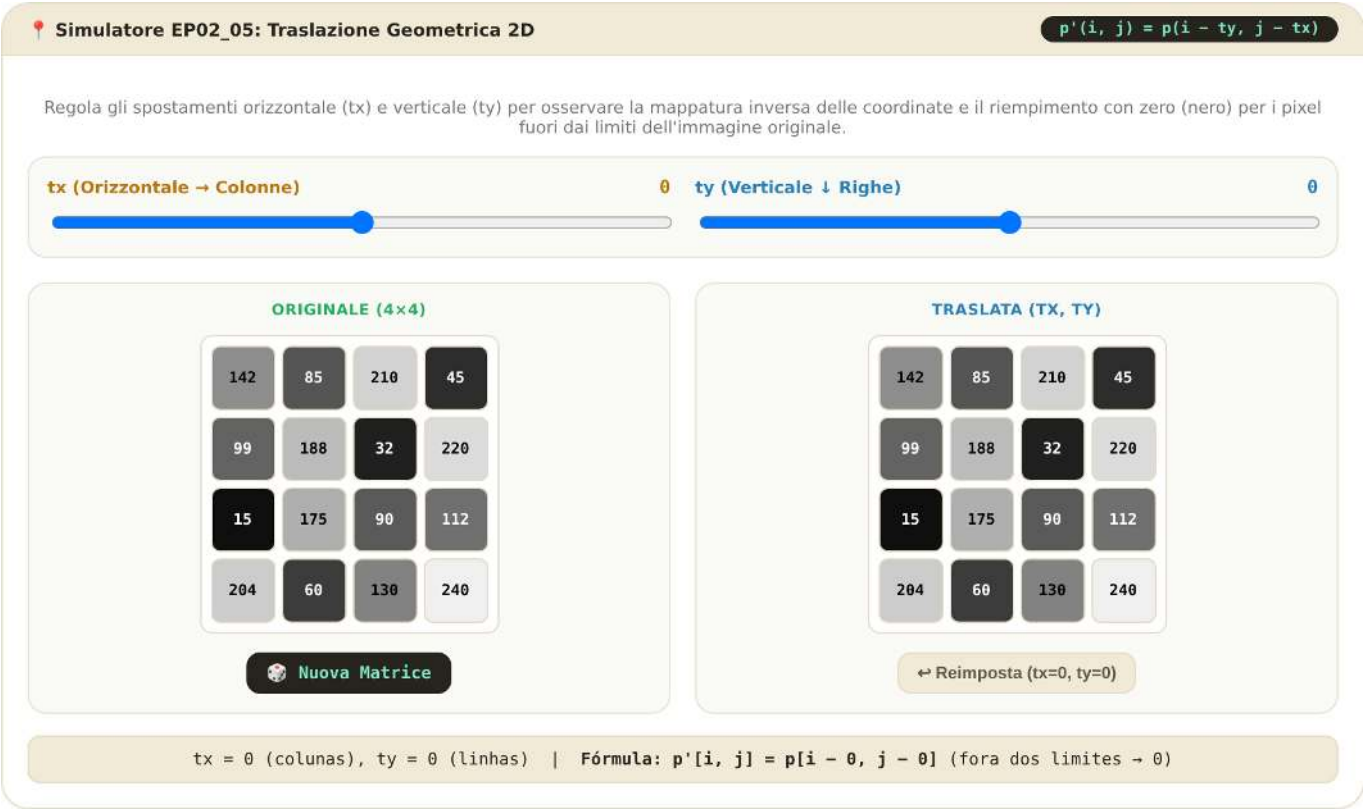
Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice risultante con le stesse dimensioni $L \times C$ dopo lo spostamento.

2.12.5.3 📌 Esempi

Input	Output	Osservazione
2 2 1 1 10 20 30 40	0 0 0 10	**Spostamento ($t_x = 1, t_y = 1$):** Ogni pixel si sposta di una posizione a destra (orizzontale) e una in basso (verticale). Il pixel $(0, 0) = 10$ va alla destinazione $(1, 1)$ (angolo inferiore destro). Le posizioni vuote sono riempite con 0.
3 3 -1 0 1 2 3 4 5 6 7 8 9	2 3 0 5 6 0 8 9 0	**Spostamento ($t_x = -1, t_y = 0$):** Ogni pixel si sposta di una posizione a sinistra (orizzontale). La prima colonna originale (`1`, `4`, `7`) viene scartata, le altre colonne si spostano a sinistra, e l'ultima colonna risultante è riempita con zeri (0).



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0205-translacao.html

Figura 2.16: Simulatore EP02_05: Traslazione Geometrica dell'Immagine (Spostamento tx e ty con Riempimento del Bordo)

```
%%writefile EP02_05.py
# Codice Python
```

```
Overwriting EP02_05.py
```

```
TestSuite("EP02_05.py").run()
```

```
<IPython.core.display.HTML object>
```

2.12.6 EP02_06 🔄 Rotazione dell'Immagine

In questa attività, devi implementare la rotazione di un'immagine attorno al suo centro geometrico. Questa operazione richiede il mapping delle coordinate e l'uso di tecniche di interpolazione per determinare i nuovi valori dei pixel.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi un valore reale θ (angolo in gradi) e una *stringa* che rappresenta il **metodo** di interpolazione (`nearest` o `bilinear`).
- Leggi i valori interi della matrice originale.
- Esegui la rotazione attorno al centro dell'immagine $(L/2, C/2)$.
- Stampa la matrice risultante con le stesse dimensioni dell'originale.
- Vedi in Figura 2.17 una simulazione di questo EP.

📌 **Importante**:

- **Mapping Inverso:** Per evitare "buchi" nell'immagine finale, percorri ogni pixel (x', y') dell'immagine di destinazione e calcola la sua posizione corrispondente (x, y) nell'immagine originale usando la matrice di rotazione inversa.
- **Interpolazione:**
- `nearest`: Assegna il valore del pixel più vicino alla coordinata calcolata.
- `bilinear`: Calcola una media ponderata basata sui 4 vicini più prossimi.
- **Bordi:** I pixel la cui origine (x, y) cade fuori dai limiti dell'immagine originale devono essere riempiti con **0**.

2.12.6.1 🧠 Trasformazione per Angolo

La rotazione di un punto (x, y) rispetto all'origine di un angolo θ è data dalla matrice di trasformazione. Per ruotare attorno a un centro (x_c, y_c), prima trasliamo il centro nell'origine, ruotiamo e trasliamo di nuovo:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_c \\ \sin\theta & \cos\theta & y_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_c \\ y - y_c \\ 1 \end{bmatrix}$$

Suggerimento: Usa il mapping inverso per garantire che tutti i pixel dell'immagine di uscita siano riempiti correttamente.

2.12.6.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene l'angolo `theta` (in gradi) e il metodo `interp` (`nearest` o `bilinear`).

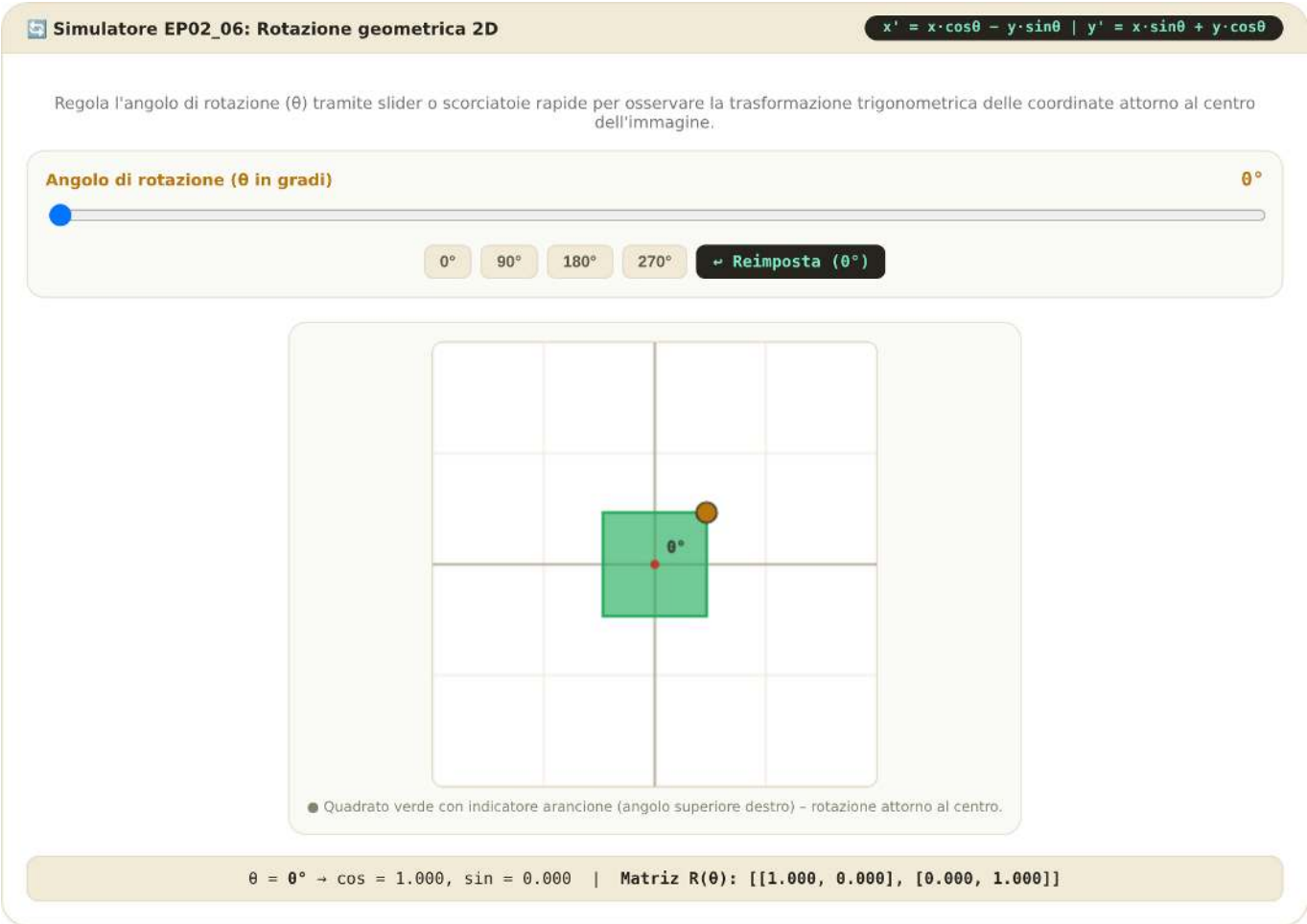
Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice ruotata con `L` righe e `C` colonne.

2.12.6.3 📌 Esempi

Input	Output	Osservazione
2 2 90 nearest 1 2 3 4	3 1 4 2	Rotazione di 90° in senso orario: la colonna 0 diventa la riga 0 (dal basso verso l'alto). $(0,0) = 1 \to (1,0)$, $(1,0) = 3 \to (0,0)$, $(0,1) = 2 \to (1,1)$, $(1,1) = 4 \to (0,1)$.
3 3 45 bilinear 0 0 0 0 255 0 0 0 0	0 180 0 180 255 180 0 180 0	Rotazione di 45°: il pixel centrale rimane 255; i vicini diretti ricevono un valore interpolato ≈ 180 tramite bilineare; gli angoli rimangono 0.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0206-rotacao.html

Figura 2.17: Simulatore EP02_06: Rotazione dell'immagine attorno all'origine di un angolo

```
%%writefile EP02_06.py
# Codice Python
```

```
Overwriting EP02_06.py
```

```
TestSuite("EP02_06.py").run()
```

```
<IPython.core.display.HTML object>
```

2.12.7 EP02_07 🔍 Ridimensionamento (Scala)

In questa attività, devi implementare il ridimensionamento di un'immagine utilizzando fattori di scala. Diversamente dal semplice sottocampionamento, qui useremo tecniche di interpolazione per consentire sia l'ingrandimento che la riduzione dell'immagine.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice originale.
- Leggi due valori reali s_x (scala sulle righe) e s_y (scala sulle colonne).
- Leggi una *stringa* che rappresenta il **metodo** di interpolazione (`nearest` o `bilinear`).
- Leggi i valori interi della matrice originale.
- Calcola le nuove dimensioni: $L' = \text{round}(L \times s_x)$ e $C' = \text{round}(C \times s_y)$.
- Stampa la matrice risultante con le nuove dimensioni.
- Vedi in Figura 2.18 una simulazione di questo EP.

📌 **Importante**:

- **Mappatura inversa:** Per ogni pixel (x', y') dell'immagine di destinazione, trova la posizione corrispondente nell'origine usando $(x, y) = (x'/s_x, y'/s_y)$.
- **Interpolazione:**
- `nearest`: Seleziona il valore del pixel più vicino (arrotondamento delle coordinate).
- `bilinear`: Esegue un'interpolazione lineare doppia tra i quattro pixel vicini più prossimi nell'immagine originale.
- **Bordi:** Assicurati che la mappatura non tenti di accedere a indici al di fuori dell'intervallo $[0, L-1]$ e $[0, C-1]$.

2.12.7.1 🧠 Interpolazione per Ingrandimento/Riduzione

Ridimensionare un'immagine mediante fattori (s_x, s_y) richiede il riempimento dei vuoti (nell'ingrandimento) o la fusione delle informazioni (nella riduzione). Il metodo di interpolazione definisce la qualità visiva del risultato:

Metodo	Funzionamento	Effetto Visivo
Nearest	Prende il valore del vicino più prossimo.	Veloce, ma genera un effetto "pixelato" o a blocchi.
Bilineare	Media ponderata dei 4 vicini (2×2).	Ammorbidisce l'immagine, riducendo l'effetto scalettato (aliasing).

2.12.7.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene i fattori `sx` e `sy`.

La quarta riga contiene il metodo `interp` (`nearest` o `bilinear`).

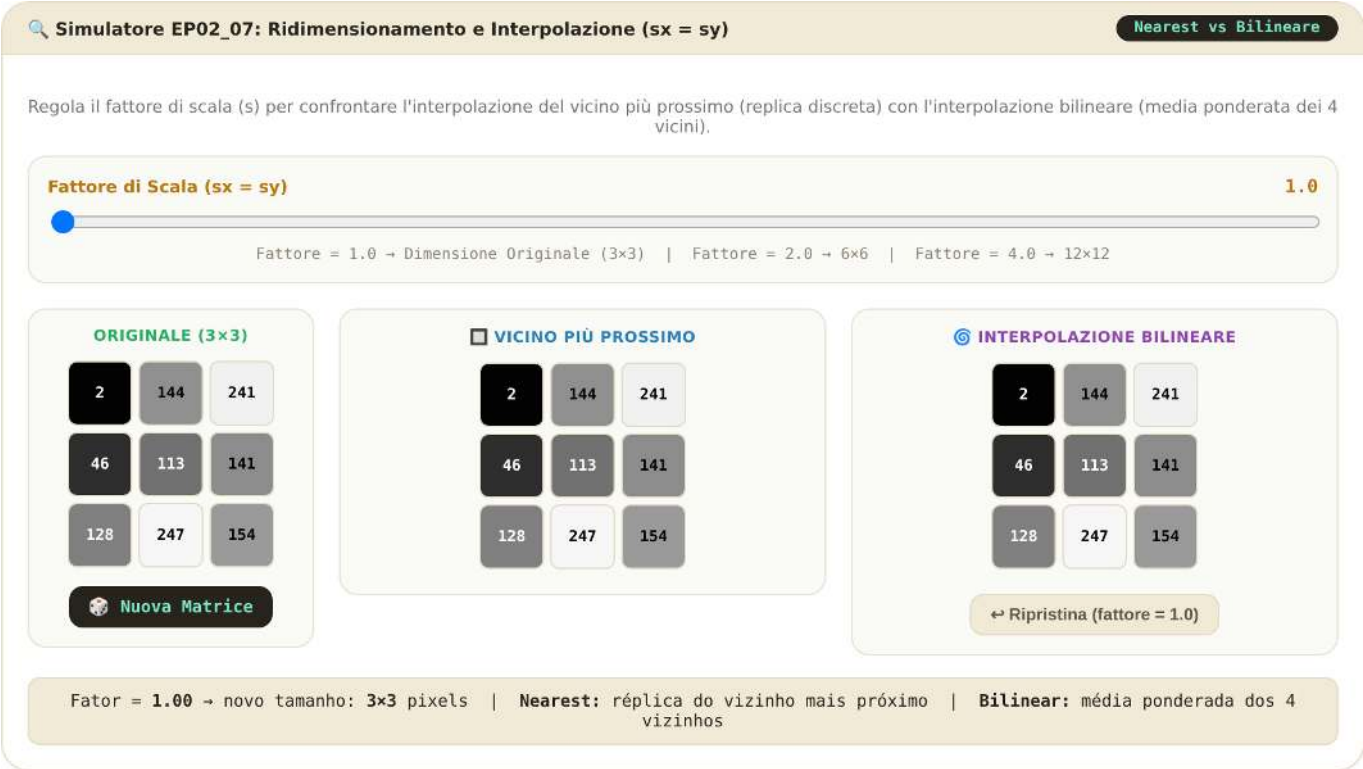
Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice ridimensionata con dimensioni $L' \times C'$.

2.12.7.3 📌 Esempi

Input	Output	Osservazione
2 2 2.0 2.0 nearest 1 2 3 4	1 1 2 2 1 1 2 2 3 3 4 4 3 3 4 4	Ingrandimento 2×: ogni pixel originale viene replicato in un blocco 2×2. L'immagine 2×2 diventa 4×4.
2 2 0.5 0.5 nearest 10 20 30 40	10	Riduzione 0.5×: l'immagine 2×2 diventa 1×1. Con nearest, l'unico pixel di output campiona la posizione $(0, 0) = 10$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0207-escala.html

Figura 2.18: Simulatore EP02_07: Ridimensionamento Spaziale e Interpolazione (Nearest Neighbor vs Bilineare)

```
%%writefile EP02_07.py
# Codice Python
import numpy as np
from morph import mm

# 1. Lettura delle dimensioni, dei fattori e del metodo
```

```
l = int(input())
c = int(input())
sx, sy = map(float, input().split())
interp = input().strip()

# 2. Lettura dell'immagine originale
img = mm.readImg(l, c)

# 3. Nuove dimensioni
l_new = round(l * sx)
c_new = round(c * sy)

# 4. Ridimensionamento usando mm.resize
# cv2.resize usa (larghezza, altezza) = (colonne, righe)
resultado = mm.resize(img, (c_new, l_new), method=interp)

# 5. Visualizzazione
print(mm.drawImg(resultado))
```

```
Overwriting EP02_07.py
```

```
TestSuite("EP02_07.py").run()
```

```
<IPython.core.display.HTML object>
```

2.12.8 EP02_08 🔀 Taglio (Shear)

In questa attività, devi implementare la trasformazione di taglio su un'immagine. Il taglio è una trasformazione affine che sposta ogni punto in una direzione fissa, di un valore proporzionale alla sua distanza da una retta parallela a tale direzione, producendo un effetto di inclinazione.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi due valori reali sh_x (taglio orizzontale) e sh_y (taglio verticale).
- Leggi una *stringa* che rappresenta il **metodo** di interpolazione (`nearest` o `bilinear`).
- Leggi i valori interi della matrice originale.
- Applica la trasformazione mantenendo le dimensioni originali dell'immagine (tagliando ciò che supera i limiti).
- Stampa la matrice risultante con le dimensioni $L \times C$.
- Vedi in Figura 2.19 una simulazione di questo EP.

📌 **Importante**:

- **Mapping inverso:** Per ogni pixel (x', y') dell'immagine di destinazione, calcola la posizione corrispondente nell'origine (x, y) utilizzando la matrice di taglio inversa.
- **Riempimento:** Le coordinate che risultano in posizioni fuori dalla matrice originale devono essere riempite con **0**.
- **Coordinate:** Ai fini di questa implementazione, considera x come indice di riga e y come indice di colonna.

2.12.8.1 🧠 Distorsione affine

Il taglio altera la geometria dell'immagine inclinandone gli assi. La relazione tra le coordinate originali (x, y) e quelle trasformate (x', y') è data da:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & sh_x & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Questo produce le equazioni:

- $x' = x + sh_x \cdot y$
- $y' = y + sh_y \cdot x$

2.12.8.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene i fattori `shx` e `shy`.

La quarta riga contiene il metodo `interp` (`nearest` o `bilinear`).

Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice trasformata con le stesse dimensioni $L \times C$.

2.12.8.3 📌 Esempi

Input	Output	Osservazione
3 3 0.5 0.0 nearest 10 20 30 40 50 60 70 80 90	10 20 30 0 40 50 0 0 70	Taglio orizzontale: la riga i si sposta di $\lfloor i \cdot 0.5 \rfloor$ pixel. Riga $0 \to 0$px, riga $1 \to 0$px, riga $2 \to 1$px. I pixel spostati fuori dai limiti vengono scartati e le posizioni vuote vengono riempite con 0.
2 2 0.0 1.0 nearest 10 20 30 40	10 0 30 20	Taglio verticale: la colonna j si sposta di $\lfloor j \cdot 1.0 \rfloor$ pixel verso il basso. Colonna $0 \to 0$px (invariata), colonna $1 \to 1$px: 20 scende a $(1, 1)$ e $(0, 1)$ diventa 0.

```
%%writefile EP02_08.py
# Codice Python
```

```
Overwriting EP02_08.py
```

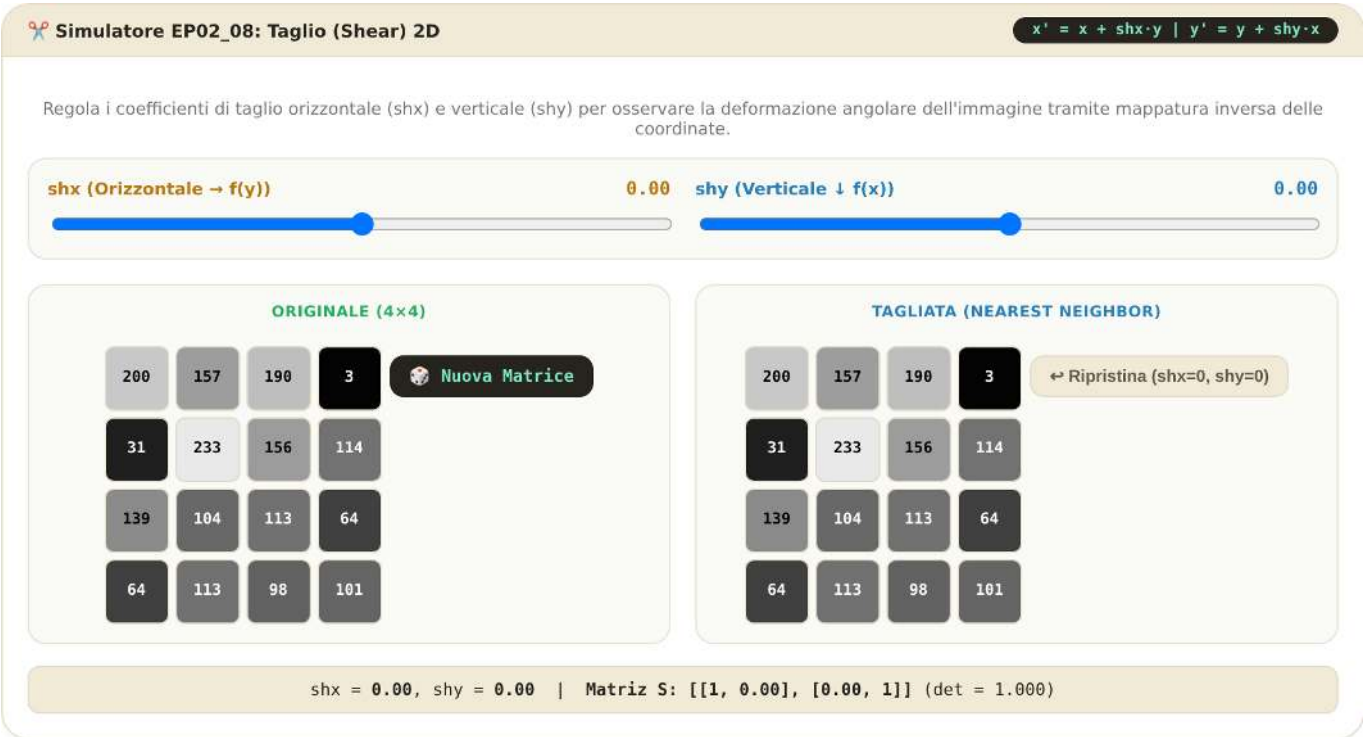
```
TestSuite("EP02_08.py").run()
```

```
<IPython.core.display.HTML object>
```

2.12.9 EP02_09 🧩 Trasformazione Affine Generica

In questa attività, devi implementare una trasformazione affine arbitraria su un'immagine. Questa operazione è la generalizzazione di tutte le trasformazioni lineari (scala, rotazione, taglio) combinate con la traslazione, consentendo manipolazioni geometriche complesse attraverso un'unica matrice.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice.
- Leggi **sei valori reali** (a, b, t_x, c, d, t_y) che compongono la matrice di trasformazione affine 2×3.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0208-cisalhamento.html

Figura 2.19: Simulatore EP02_08: Trasformazione Geometrica di Taglio 2D (Shear Orizzontale e Verticale)

- Leggi una *stringa* che rappresenta il **metodo** di interpolazione (`nearest` o `bilinear`).
- Leggi i valori interi della matrice originale.
- Applica la trasformazione mantenendo la dimensione originale $L \times C$.
- Stampa la matrice risultante.
- Vedi in Figura 2.20 una simulazione di questo EP.

📌 **Importante**:

- **Mappatura inversa:** Per calcolare il valore di ogni pixel nell'immagine di destinazione, devi utilizzare l'**inversa** della matrice di trasformazione affine fornita per trovare la coordinata corrispondente nell'immagine originale.
- **Riempimento:** Le coordinate calcolate che cadono al di fuori dei limiti $[0, L-1]$ e $[0, C-1]$ dell'immagine originale devono risultare in un pixel di valore **0**.
- **Flessibilità:** Questa implementazione deve essere in grado di eseguire uno qualsiasi dei compiti precedenti (traslazione, rotazione, ecc.) semplicemente modificando i parametri della matrice.

Suggerimento:

```
flags = cv2.INTER_NEAREST if interp == 'nearest' else \
        cv2.INTER_CUBIC   if interp == 'bicubic'  else \
        cv2.INTER_LANCZOS4 if interp == 'lanczos' else \
        cv2.INTER_LINEAR

r = cv2.warpAffine(img, M, (C, L), flags=flags)
```

2.12.9.1 🧠 Combinazione di Operazioni

La trasformazione affine preserva punti, rette e piani. Nell'elaborazione delle immagini, mappa la posizione (x, y) in (x', y') seguendo il sistema:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Oppure, in forma compatta in coordinate omogenee:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2.12.9.2 📋 Compito (specifica per VPL)

Input:

La prima riga contiene `L`.

La seconda riga contiene `C`.

La terza riga contiene sei float: `a b tx c d ty`.

La quarta riga contiene il metodo `interp` (`nearest` o `bilinear`).

Le righe successive contengono gli elementi della matrice $L \times C$.

Output:

La matrice trasformata con le dimensioni originali $L \times C$.

2.12.9.3 📌 Esempi

Input	Output	Osservazione
2 2 1.0 0.0 0.5 0.0 1.0 0.5 bilinear 10 20 30 40	15 20 25 30	Traslazione frazionaria ($t_x = 0.5, t_y = 0.5$): ogni pixel di output (i, j) campiona la posizione $(i + 0.5,\ j + 0.5)$ dell'input tramite bilineare. Es: $(0, 0)$ interpola i quattro vicini $\rightarrow$ 15.
3 3 2.0 0.0 0.0 0.0 2.0 0.0 nearest 1 2 3 4 5 6 7 8 9	1 1 2 1 1 2 4 4 5	Scala 2× tramite matrice affine ($a = 2, d = 2$): ogni pixel di output (i, j) campiona la posizione $(2i, 2j)$ dell'input con nearest. Es: $(0, 2) \rightarrow (0, 4)$ fuori dall'immagine $\rightarrow$ nearest blocca a $(0, 2) = 3$… in attesa di conferma della logica di bordo.

```
%%writefile EP02_09.py
# Codice Python
```

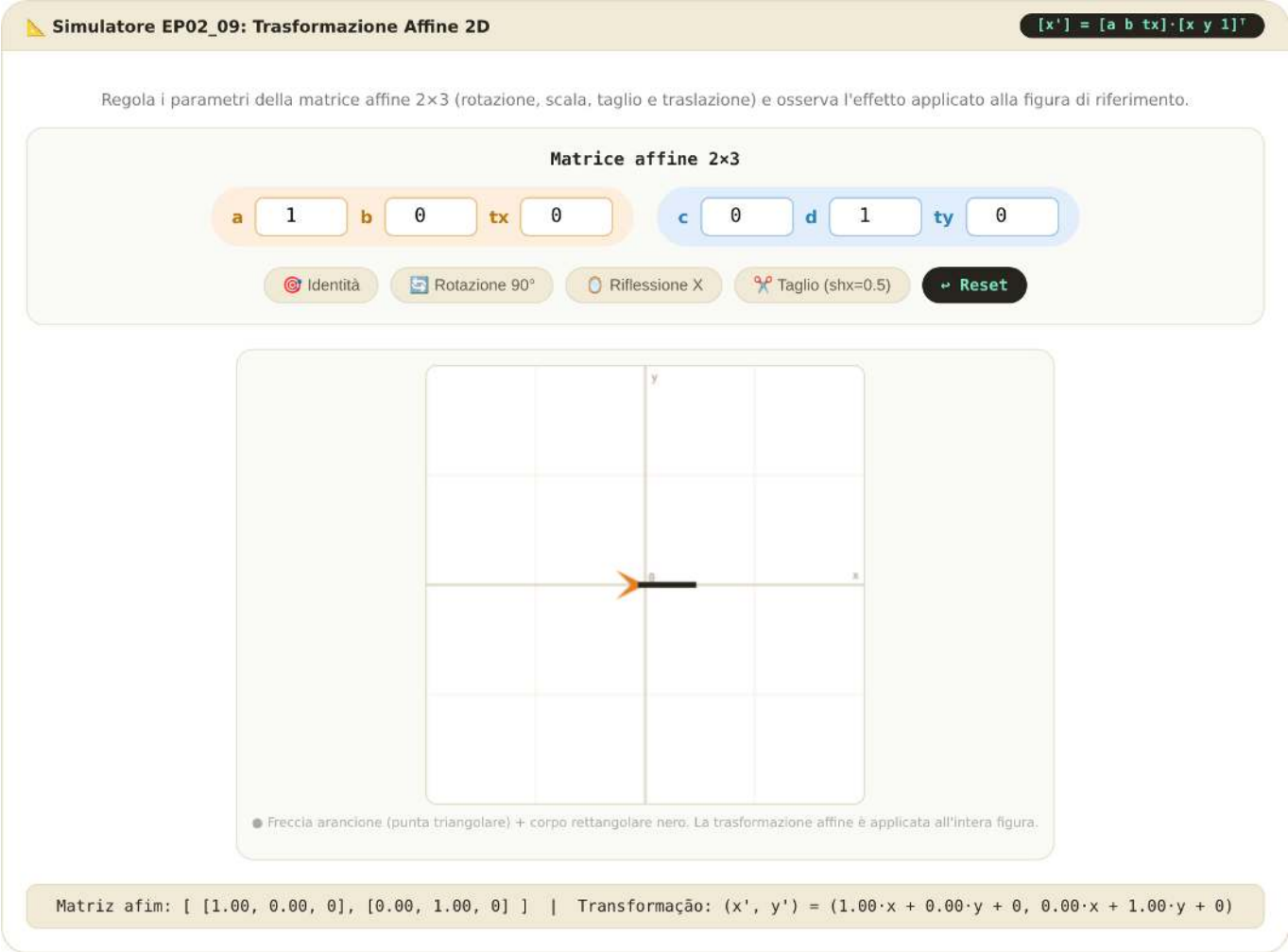
```
Overwriting EP02_09.py
```

```
TestSuite("EP02_09.py").run()
```

```
<IPython.core.display.HTML object>
```

2.12.10 EP02_10 🎯 Correzione della Prospettiva (Omografia)

In questa attività, devi implementare la trasformazione di prospettiva, nota anche come omografia. A differenza delle trasformazioni affini, la prospettiva non preserva il parallelismo, consentendo di "rettificare" oggetti inclinati, come documenti o targhe catturati da angolazioni oblique.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0209-afim.html

Figura 2.20: Simulatore EP02_09: Trasformazione Affine 2D (Matrice 2×3)

- Leggi due interi **L** e **C**, che rappresentano le dimensioni della matrice originale.
- Leggi **quattro coppie di coordinate** (x, y) che rappresentano i vertici del quadrilatero di origine (oggetto distorto).
- Leggi **quattro coppie di coordinate** (x, y) che rappresentano i vertici del quadrilatero di destinazione (dove l'oggetto deve essere mappato).
- Leggi i valori della matrice originale.
- Calcola la matrice di omografia 3×3 e applica la trasformazione.
- Stampa la matrice risultante con le dimensioni di uscita specificate.
- Vedi in Figura 2.21 una simulazione di questo EP.

📌 **Importante**:

- **Gradi di libertà:** L'omografia possiede 8 gradi di libertà (il nono elemento della matrice 3×3 è una costante di normalizzazione, generalmente 1), richiedendo almeno 4 punti corrispondenti per essere calcolata.
- **Proiezione:** Dopo aver moltiplicato le coordinate per la matrice, è necessario dividere i risultati x' e y' per la componente omogenea w per tornare al piano 2D.
- **Uso di librerie:** Per questo compito, puoi utilizzare le funzioni `cv2.getPerspectiveTransform` per ottenere la matrice e `cv2.warpPerspective` per applicare la trasformazione, oppure implementare manualmente il sistema lineare e la mappatura inversa per una sfida extra.

```
# Dimensioni di uscita: bounding box dei punti di destinazione + 1
w = int(max(pts2[:, 0])) + 1; h = int(max(pts2[:, 1])) + 1
# M = cv2.getPerspectiveTransform(pts1, pts2)
# dst = cv2.warpPerspective(img, M, (w, h))
# oppure
dst = mm.perspective_transform(img, pts1, pts2, size=(w, h))
```

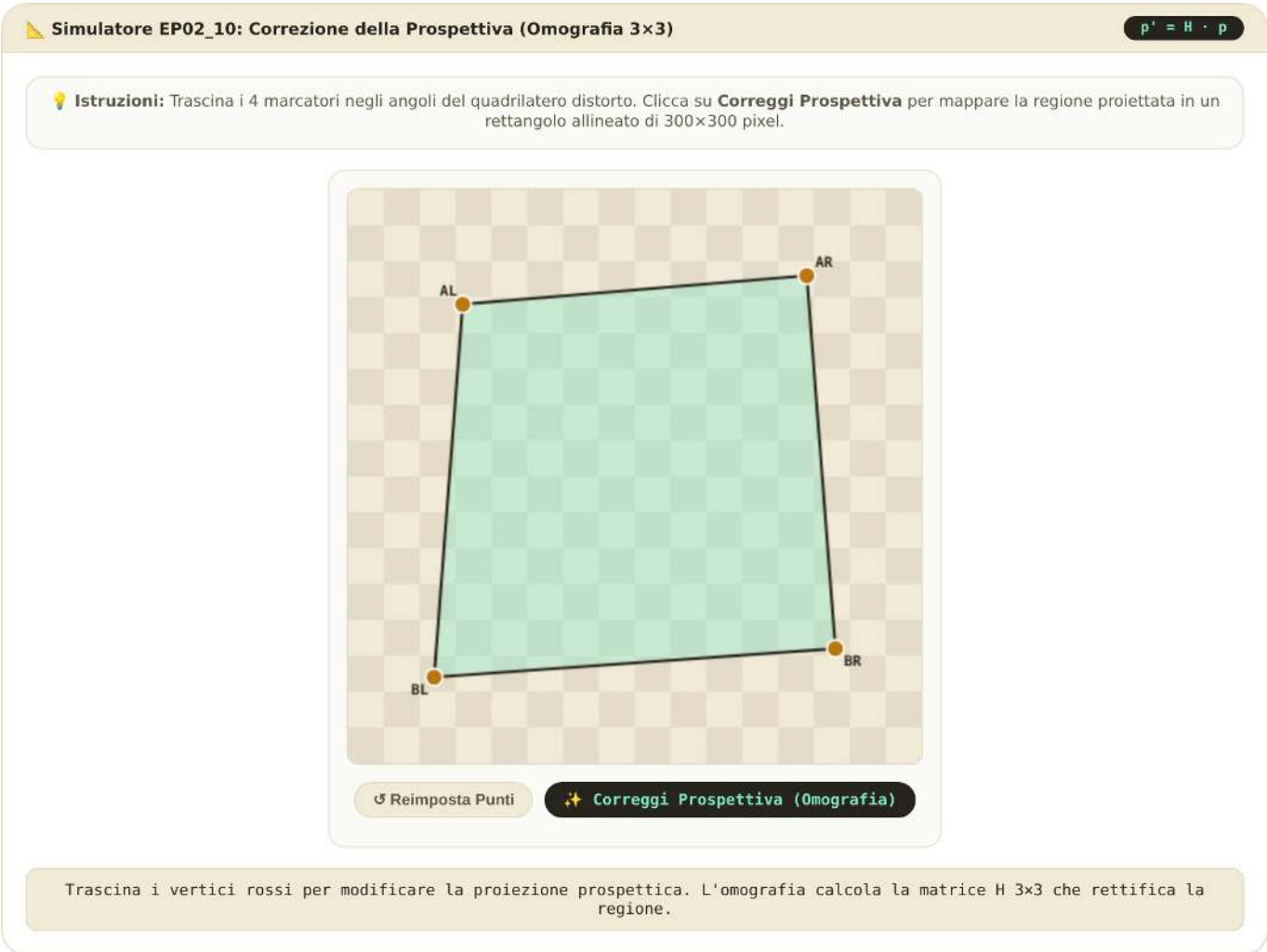
2.12.10.1 🧠 Deformazione non affine

Mentre le trasformazioni affini mappano parallelogrammi in parallelogrammi, l'omografia mappa qualsiasi quadrilatero in un altro quadrilatero. Questo è essenziale per la visione artificiale:

Operazione	Caratteristica	Applicazione Tipica
Omografia	Proiezione su piano	Correzione di documenti, scansione di targhe.
Punto di fuga	Convergenza di linee	Ricostruzione 3D da immagini 2D.
Warping	Deformazione di mesh	Stabilizzazione video e panoramiche (stitching).

2.12.10.2 📌 Esempi

Input	Output	Osservazione
4 4 0 0 3 0 0 3 3 3 0 0 3 0 0 3 3 3 10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160	10 20 30 40 50 60 70 80 90 100 110 120 130 140 150 160	Le prime 4 righe dopo le dimensioni sono i punti di origine; le 4 successive sono le destinazioni. Con punti identici, la trasformazione di prospettiva è l'identità e l'immagine viene preservata.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0210-perspectiva.html

Figura 2.21: Simulatore EP02_10: Correzione della Prospettiva (Trasformazione di Omografia 3×3)

```
%%writefile EP02_10.py
# Codice Python
```

```
Overwriting EP02_10.py
```

```
TestSuite("EP02_10.py").run()
```

```
<IPython.core.display.HTML object>
```

2.12.11 EP02_11 🏆 Correzione della Prospettiva (Omografia) su Immagine Reale

In questa attività, l'obiettivo è applicare la trasformazione prospettica (omografia) per "raddrizzare" un oggetto inclinato in una fotografia reale. Lavorerai con l'immagine di un giornale, dove la griglia di un gioco di Sudoku è distorta a causa dell'angolo con cui è stata scattata la foto.

Il tuo programma deve leggere i parametri di input dal terminale, caricare l'immagine, calcolare la matrice di omografia 3×3, applicare la trasformazione geometrica e visualizzare un indicatore globale di validazione.

- Leggi due interi **L** e **C**, che rappresentano le dimensioni di righe e colonne (altezza e larghezza) che l'immagine rettificata di uscita deve avere.
- Leggi **quattro coppie di coordinate** (x, y) dal terminale, che rappresentano i quattro angoli del quadrilatero di origine (il Sudoku distorto nell'immagine originale).
- Calcola automaticamente le **quattro coppie di coordinate di destinazione** utilizzando le dimensioni L e C fornite, mappando gli angoli agli estremi della nuova immagine: $(0, 0)$, $(C - 1, 0)$, $(0, L - 1)$ e $(C - 1, L - 1)$.
- Carica l'immagine locale `sudoku.png` e convertila in scala di grigi (grayscale).
- Calcola la matrice di omografia e applica la trasformazione spaziale all'immagine.
- **Output:** Calcola e stampa la **somma di tutti i pixel** dell'immagine risultante.

📌 **Importante**:

- **File di input:** L'immagine `sudoku.png` deve trovarsi nella stessa cartella dello script. Il programma deve leggerla direttamente dal disco (es: usando `mm.read("sudoku.png")` o `cv2.imread`).
- **Ordine dei Punti:** Assicurati che la lettura dei 4 punti di origine e la generazione dei 4 punti di destinazione seguano rigorosamente lo stesso ordine degli angoli: Superiore-Sinistro (TL), Superiore-Destro (TR), Inferiore-Sinistro (BL) e Inferiore-Destro (BR).
- **Dimensioni in OpenCV:** Ricorda che funzioni come `cv2.warpPerspective` si aspettano la dimensione dell'immagine di uscita nel formato `(larghezza, altezza)`, che equivale a `(C, L)`.
- **Interpolazione:** Per garantire la coerenza matematica della somma dei pixel con il correttore automatico, utilizza l'interpolazione bilineare standard (`flags=cv2.INTER_LINEAR`).
- **Crediti:** L'immagine utilizzata è **"Sudoku en periódico"** di Héctor Rodríguez, sotto licenza **CC BY 2.0**.

2.12.11.1 🧠 Contesto del Problema

L'omografia ha 8 gradi di libertà, richiedendo almeno 4 corrispondenze di punti per essere calcolata. A differenza delle trasformazioni affini, mappa qualsiasi quadrilatero in un altro quadrilatero, permettendo che le linee che convergono verso punti di fuga tornino ad essere parallele:

Operazione	Caratteristica	Applicazione Tipica
Omografia	Proiezione tra piani	Rettifica di documenti, scansione di targhe e codici QR.
Mappatura Inversa	Scansione dalla destinazione all'origine	Evita "buchi" o pixel vuoti nell'immagine finale rettificata.
Warping	Ricampionamento spaziale	Correzione della distorsione delle lenti e montaggio di panorami (*stitching*).

2.12.11.2 📌 Esempi

Input	Output	Osservazione
500 500 100 120 420 95 80 440 450 460	32982820	I primi due input sono le dimensioni di uscita (L e C). Le 4 righe successive sono le coordinate (x, y) degli angoli del Sudoku nell'immagine originale + PAD. L'output è la somma totale dei pixel dell'immagine rettificata.
200 200 100 120 420 95 80 440 450 460	5277150	Stessi punti di origine dell'esempio precedente, ma generando un'immagine di uscita più piccola (200×200). La somma dei pixel si riduce proporzionalmente a causa della scala.

2.12.11.3 Acquisizione dell'immagine del sudoku e conversione in livelli di grigio

La Figura 2.22 mostra la lettura dell'immagine originale seguita dalla conversione in tonalità di grigio e dal ridimensionamento a una matrice di 500×500 pixel, preparando i dati per la fase successiva.

La correzione di prospettiva, applicata nella Figura 2.23 tramite matrice di omografia, elimina le deformazioni causate dall'angolo della telecamera e produce una visione frontale e regolare della griglia del Sudoku.

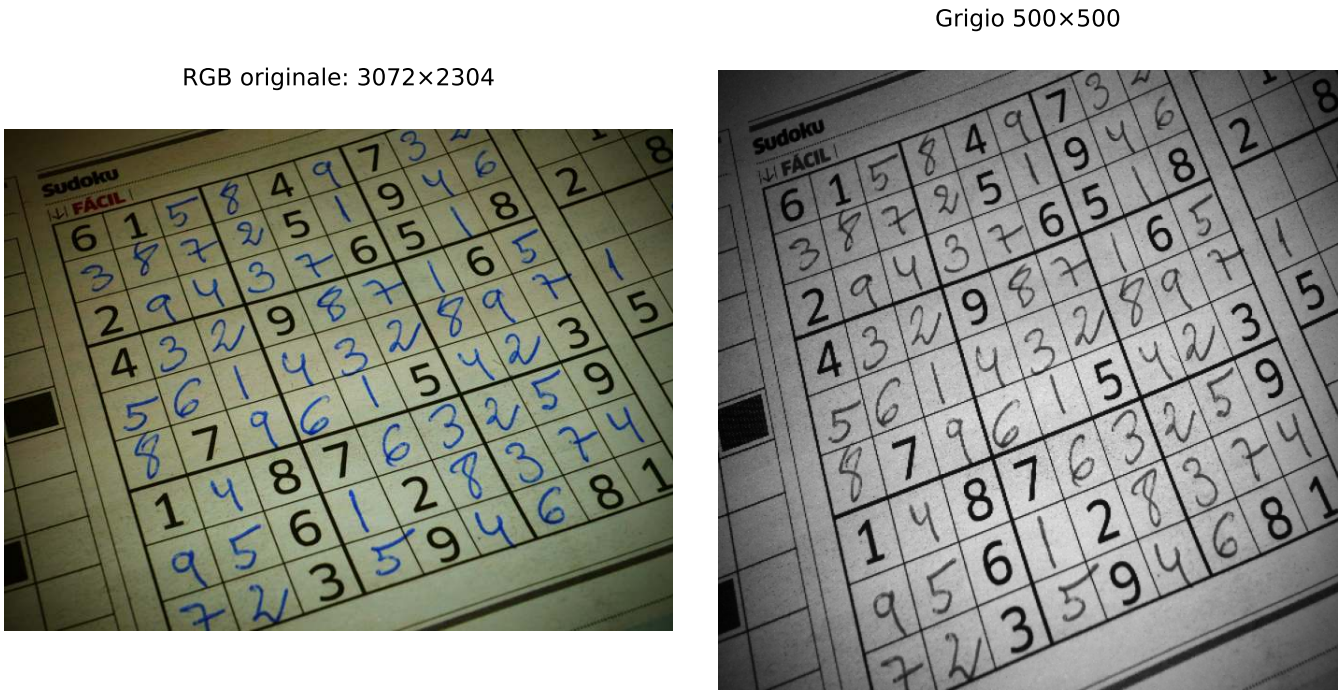


Figura 2.22: Acquisizione dell'immagine di un Sudoku a sinistra. A destra, conversione in scala di grigi e ridimensionamento. Credito: Héctor Rodríguez de Guardamar, Spagna (CC BY 2.0).

```
import cv2
import numpy as np

# --- 1. Carica l'immagine salvata (sudoku.png) ---
img = mm.read("sudoku.png")                # BGR, 500×500

# --- 2. Padding per non tagliare i vertici ---
PAD = 60
```

```
img_pad = cv2.copyMakeBorder(
    img, PAD, PAD, PAD, PAD,
    cv2.BORDER_CONSTANT, value=[255, 255, 255]
)

# --- 3. Punti di origine (angoli della griglia nell'immagine espansa) ---
pts1 = np.float32([
    [100, 160],    # TL
    [390, 45],    # TR
    [200, 580],   # BL
    [570, 420],   # BR
])
#     W    H

# --- 4. Punti di destinazione (vista frontale 500×500) ---
SIZE = 500
pts2 = np.float32([
    [0,    0],
    [SIZE, 0],
    [0,    SIZE],
    [SIZE, SIZE],
])

# --- 5. Omografia e raddrizzamento ---
img_rect = mm.perspective_transform(img_pad, pts1, pts2, size=(SIZE, SIZE))

# --- 6. Visualizzazione ---
mm.show(
    [img_pad, img_rect],
    titles=["Originale (con padding)", "Vista frontale raddrizzata"],
    cols=2, figsize=(10, 6), axis=True
)
```

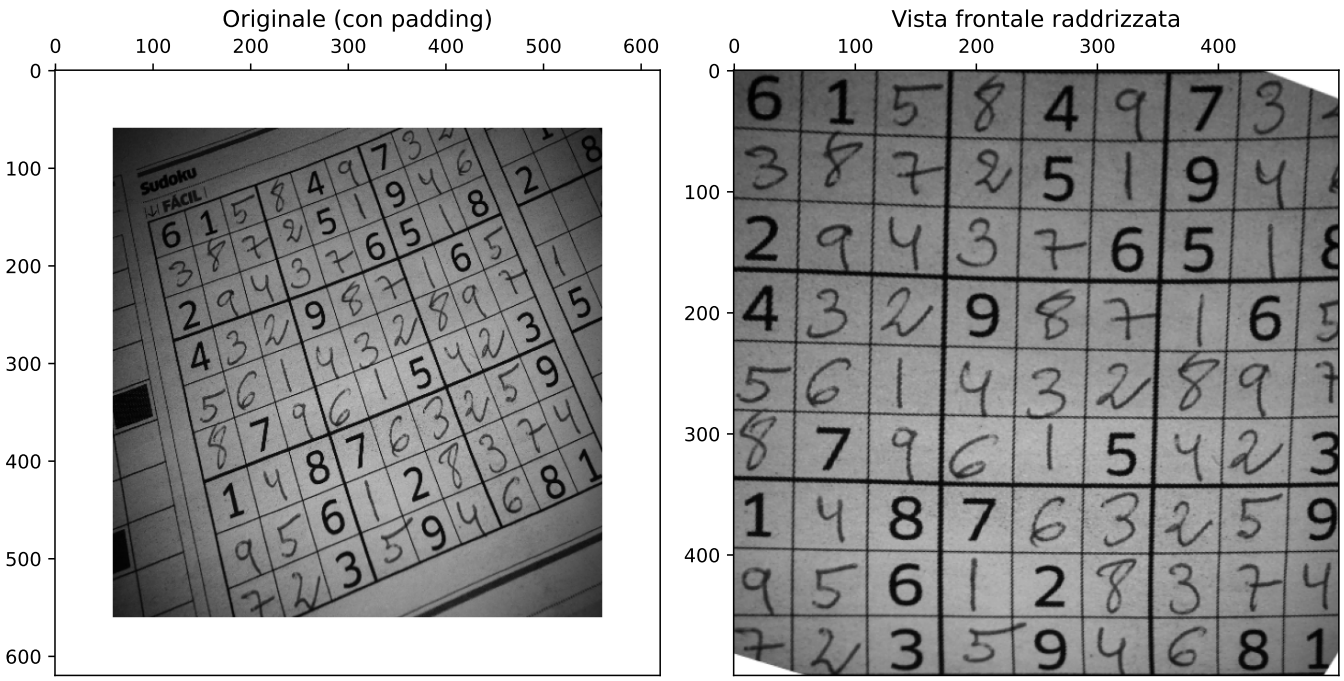
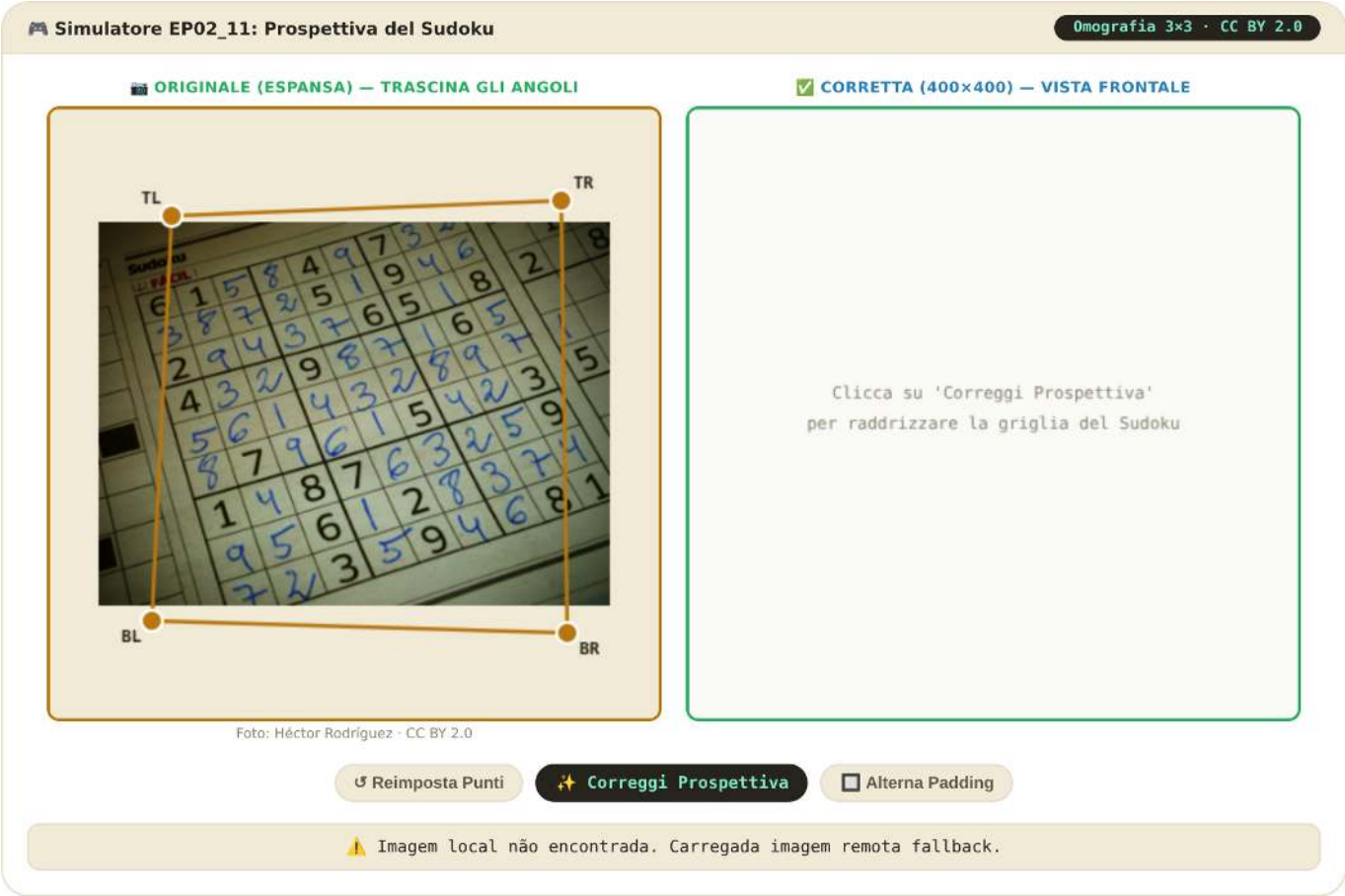


Figura 2.23: Correzione prospettica: originale e vista frontale raddrizzata.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap02/sim-ep0211-sudoku.html

Figura 2.24: Simulatore EP02_11: Correzione della Prospettiva del Sudoku (Omografia 3×3 con Ricampionamento Bilineare)

```
%%writefile EP02_11.py
# Codice Python
```

```
Overwriting EP02_11.py
```

```
TestSuite("EP02_11.py").run()
```

```
<IPython.core.display.HTML object>
```

Capitolo 3

Operazioni Spaziali: Intensità, Istogramma e Filtraggio

Executar Colab Abrir GitHub

Questo capitolo approfondisce l'elaborazione delle immagini nel dominio spaziale, partendo dalla manipolazione diretta dei pixel e degli istogrammi per l'enfatizzazione del contrasto, fino all'applicazione di filtri locali tramite convoluzione per la smussatura, la riduzione del rumore e il rilevamento dei bordi. L'obiettivo è sviluppare l'intuizione matematica e computazionale alla base di gran parte degli algoritmi moderni della Visione Artificiale.

3.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- **Gestire intensità e pixel:** Eseguire operazioni aritmetiche sature (`mm.addm`, `mm.subm`) e logiche bit a bit (`mm.band`, `mm.bor`, `mm.bnot`) per la combinazione e la selezione di regioni di interesse (ROI), e applicare l'*alpha blending* (`mm.blend`) per la fusione ponderata di immagini;
- **Elaborare istogrammi:** Interpretare l'istogramma come diagnosi tonale e applicare l'equalizzazione globale tramite CDF (`mm.equalize`); nel percorso Python, anche l'equalizzazione adattiva (CLAHE) e la specifica dell'istogramma per il trasferimento del profilo tonale tra immagini;
- **Comprendere i fondamenti spaziali:** Capire il concetto di vicinato, il *padding* dei bordi (`mm.pad`) e la differenza tra correlazione incrociata (`mm.conv`) e convoluzione — inclusa la ragione per cui *kernel* asimmetrici come quello di Sobel producono risultati diversi nelle due operazioni;
- **Applicare il filtraggio di smoothing:** Usare il filtro della media (`mm.blur`, oppure `mm.conv` con *kernel* uniforme) e il filtro Gaussiano (`mm.gaussian`) per la riduzione del rumore, comprendendo il vantaggio della ponderazione radiale e della separabilità Gaussiana;
- **Applicare il filtraggio di enhancement:** Usare il Laplaciano w_4 e w_8 (`mm.laplacian`) per l'accentuazione isotropica dei bordi, l'operatore di Sobel (`mm.sobel`) per la magnitudine del gradiente — e, nel percorso Python, la decomposizione direzionale G_x, G_y e l'angolo —, e l'*Unsharp Masking* (`mm.usm`) per l'amplificazione delle alte frequenze controllata dal parametro k;
- **Utilizzare filtri d'ordine:** Applicare il filtro mediano (`mm.median`) per la rimozione del rumore sale e pepe, comprendendo perché la sua natura non lineare e la robustezza agli *outlier* lo rendano superiore ai filtri lineari in questo scenario;
- **Risolvere problemi pratici:** Combinare tecniche in *pipeline* di pre-processing (equalizzazione → Gaussiano → Canny; con CLAHE al posto dell'equalizzazione nel percorso Python) e utilizzare le funzioni di `morph` (`mm.conv`, `mm.histImg`, `mm.equalize`, `mm.drawImgKernel`) per l'analisi e la visualizzazione didattica di ogni fase.

3.2 Operazioni a Livello di Intensità

Il livello più elementare di elaborazione delle immagini agisce direttamente sui valori dei pixel, senza considerare il vicinato. Queste operazioni — chiamate **trasformazioni puntuali** (*point operations*) — sono

le più veloci dal punto di vista computazionale e costituiscono la base per tecniche più complesse.

Formalmente, una trasformazione puntuale può essere descritta come:

$$g(x, y) = T[f(x, y)] \tag{3.1}$$

dove $f(x, y)$ è l'immagine di ingresso, $g(x, y)$ è l'uscita e T è una funzione applicata a ciascun pixel individualmente.

3.2.1 Preparazione dell'Ambiente Pratico

Il blocco seguente carica la libreria `morph` dal repository (il modulo `morph.py` e, nel percorso C++, anche la `morph.hpp` utilizzata nel `#include` delle celle compilate).

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup()
from morph import mm
import numpy as np
import cv2
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0
```

Come oggetto di studio lungo questo capitolo, utilizzeremo le immagini di vita selvatica presentate nelle Figura 3.1 e Figura 3.3. A partire da esse, esploreremo operazioni spaziali sull'intensità, istogrammi e filtraggio, analizzando i loro effetti sull'amplificazione, sulla levigatura, sulla riduzione del rumore e sulla rilevazione dei bordi, in modo da comprendere i fondamenti matematici e computazionali dell'elaborazione digitale delle immagini (PDI).

```
import os

base    = "https://upload.wikimedia.org/wikipedia/commons"
arquivo = "Carlos_Guilherme_Rodrigues_%2876515283%29.jpeg"
url     = f"{base}/9/9b/{arquivo}"
caminho = "imagens/mandrill-exif.jpg"

if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    mm.write(mm.read(url), caminho)

img_color = mm.read(caminho)
img_gray  = mm.gray(img_color)

mm.show(img_color)
```

Figura 3.1: *Mandrill* (*Mandrillus sphinx*) fotografato in ambiente naturale in Sudafrica. Credito: Carlos Guilherme Rodrigues (CC BY-SA 3.0).

3.2.2 Operazioni Aritmetiche

Le operazioni aritmetiche tra immagini sono ampiamente utilizzate nell'elaborazione digitale delle immagini (PDI) per combinare, confrontare o migliorare le informazioni. La **sottrazione di immagini** è particolarmente efficace per rilevare differenze tra due fotogrammi — ad esempio, nella rimozione dello sfondo statico nelle telecamere di sorveglianza:

$$g(x, y) = f_1(x, y) - f_2(x, y) \tag{3.2}$$

L'**addizione saturata** limita il risultato all'intervallo $[0, 255]$: i valori superiori a 255 vengono fissati a 255, evitando l'*overflow* silenzioso del tipo `uint8` (es.: $200 + 100 = 44$ anziché 300). La **sottrazione saturata** applica lo stesso principio sul lato inferiore: i valori negativi vengono fissati a 0.

> ⚠ Saturazione e *overflow*
>
> Le operazioni aritmetiche su `uint8` subiscono un *overflow* silenzioso: $200+100 = 44$ (non 300). `mm.addm` e `mm.subm` eseguono la **saturazione automatica**, fissando il risultato in $[0, 255]$. Il *blending* utilizza pesi frazionari: `mm.blend` opera internamente in virgola mobile e solo successivamente arrotonda e satura a `uint8`.

La Figura 3.2 dimostra l'addizione di una costante (schiarimento) e la sottrazione di una costante (scurimento con saturazione a 0).

```
fundo = 60

img_add = mm.addm(img_gray, fundo)
img_sub = mm.subm(img_gray, fundo)

mm.show(
    [img_gray, img_add, img_sub],
    titles=["Originale", "addm (+60)", "subm (-60)"],
    cols=3
)
```

Figura 3.2: Operazioni aritmetiche sature: addizione di costante (schiarimento) e sottrazione di costante (scurimento con saturazione a 0).

3.2.3 Miscelazione Ponderata (*Alpha Blending*)

La **miscelazione ponderata** (*alpha blending*) combina due immagini utilizzando pesi complementari α e $(1-\alpha)$:

$$g(x,y) = \alpha\, f_1(x,y) + (1-\alpha)\, f_2(x,y), \quad \alpha \in [0,1] \tag{3.3}$$

Quando $\alpha = 1$, si ottiene soltanto l'immagine f_1; quando $\alpha = 0$, soltanto f_2. I valori intermedi producono una transizione fluida tra le due, essendo ampiamente utilizzati nella composizione di immagini, sovrapposizione di livelli, filigrane ed effetti di fusione visiva.

Affinché la combinazione produca un risultato coerente, è necessario allineare preventivamente le regioni di interesse. Nella Figura 3.4, si ritaglia il volto del leopardo con `mm.crop(img_leop_gray, 250, H-300, 100, W-200)` e la regione facciale del mandrillo con `mm.crop(img_gray, 100, 400, 380, 530)`, in modo che occhi e struttura facciale risultino approssimativamente allineati. Il ritaglio del leopardo viene quindi ridimensionato (`mm.resize`) alle dimensioni del mandrillo prima della miscelazione.

`mm.blend` esegue l'operazione in virgola mobile — evitando *overflow* nei calcoli con pesi frazionari — e solo successivamente arrotonda e satura il risultato in `uint8`.

```
import os

base    = "https://upload.wikimedia.org/wikipedia/commons"
arquivo = "Leopard_%28Panthera_pardus%29_portrait.jpg"
url     = f"{base}/9/92/{arquivo}"
caminho = "imagens/leopardo.jpg"

if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    mm.write(mm.read(url), caminho)

img_leop      = mm.read(caminho)
img_leop_gray = mm.gray(img_leop)

mm.show(img_leop)
```

Figura 3.3: Ritratto di un leopardo (*Panthera pardus*) in ambiente naturale. Credito: C. Brück (CC BY-SA 4.0).

```
# ritagli allineati: volto del leopardo e regione facciale del mandrillo
leo      = mm.crop(img_leop_gray, 250, img_leop_gray.shape[0] - 300, 100,
img_leop_gray.shape[1] - 200)
mandrill = mm.crop(img_gray, 100, 400, 380, 530)
leo_r    = mm.resize(leo, (mandrill.shape[1], mandrill.shape[0]), method="bilinear")

mm.show(
    [mm.blend(mandrill, leo_r, alpha=1.0), mm.blend(mandrill, leo_r, alpha=0.8),
     mm.blend(mandrill, leo_r, alpha=0.6), mm.blend(mandrill, leo_r, alpha=0.4),
     mm.blend(mandrill, leo_r, alpha=0.2), mm.blend(mandrill, leo_r, alpha=0.0)],
    titles=[" =1.0", " =0.8", " =0.6", " =0.4", " =0.2", " =0.0"],
    cols=6
)
```

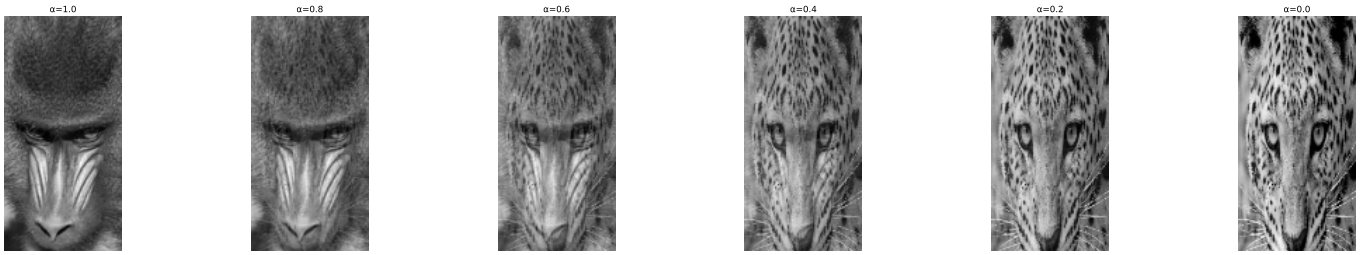


Figura 3.4: *Alpha blending* tra ritagli allineati del mandrillo e del leopardo (Figura 3.3) per diversi valori di . Con =1 si vede solo il mandrillo; con =0, solo il leopardo; valori intermedi fondono proporzionalmente gli sguardi delle due immagini.

3.2.4 Operazioni Logiche e Maschere Bit a Bit

Le operazioni logiche bit a bit (AND, OR e NOT) agiscono direttamente sui bit di ciascun pixel e costituiscono la base per la creazione e l'applicazione di **maschere** (*masks*) — immagini binarie con solo 0 (nero) e 255 (bianco) utilizzate per isolare **Regioni di Interesse** (**ROI**).

Il comportamento di ciascuna operazione deriva dalla rappresentazione binaria del 255 (`11111111`) e dello 0 (`00000000`):

- **AND** con la maschera: dove $m = 255$, i bit originali vengono preservati; dove $m = 0$, il pixel viene azzerato. Risultato: ritaglio della ROI.

$$g(x, y) = f(x, y) \text{ AND } m(x, y) \tag{3.4}$$

- **OR** con la maschera: dove $m = 255$, il pixel viene forzato al bianco; dove $m = 0$, il valore originale viene mantenuto. Risultato: illuminazione della ROI.
- **NOT** (senza maschera): inverte tutti i bit ($g = 255 - f$), producendo il negativo fotografico dell'immagine.

La Figura 3.5 illustra le tre operazioni applicate all'immagine del mandrillo con una maschera circolare.

```
h, w = img_gray.shape

# Maschera circolare piena, centrata sull'immagine (mm.circle disegna il
# disco; la versione didattica, test del raggio pixel per pixel, è mm.circle0).
mask_circ = mm.circle(np.zeros((h, w), dtype=np.uint8),
                      (w // 2, h // 2), min(h, w) // 3 - 10, 255, -1)

# Operazioni tramite morph
img_not = mm.bnot(img_gray)            # NOT: negativo fotografico
img_and = mm.band(img_gray, mask_circ) # preserva solo la ROI circolare
img_or  = mm.bor(img_gray, mask_circ)  # illumina la regione della maschera

mm.show(
    [img_gray, img_and, img_or, img_not],
    titles=["Originale", "AND (ROI circolare)", "OR (illumina ROI)", "NOT (negativo)"],
    cols=4
)
```

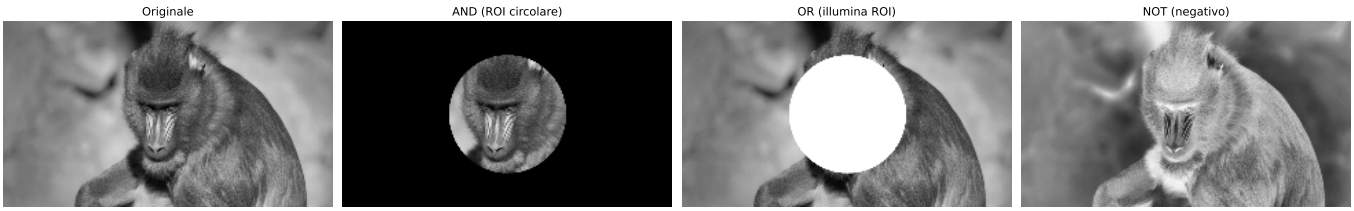


Figura 3.5: Operazioni logiche bit a bit con maschera circolare: AND (isolamento della ROI), OR (illuminazione della ROI) e NOT (negativo).

3.3 Istogramma delle Immagini

L'**istogramma** di un'immagine in scala di grigi è una funzione discreta che descrive la distribuzione delle frequenze delle intensità:

$$h(r_k) = n_k, \quad k = 0, 1, \ldots, L - 1 \tag{3.5}$$

dove r_k è il k-esimo livello di intensità, n_k è il numero di pixel con tale intensità e L è il totale dei livelli (tipicamente 256 per 8 bit). L'istogramma normalizzato stima la probabilità di ciascun livello:

$$p(r_k) = \frac{n_k}{MN} \tag{3.6}$$

dove MN è il totale dei pixel. Essendo una **statistica globale**, l'istogramma non contiene informazioni posizionali, ma rivela caratteristiche essenziali come luminosità media, contrasto e distribuzione tonale. In

pratica, `mm.hist(img)` restituisce il vettore delle occorrenze $h(r_k)$, che serve sia per la visualizzazione (tramite `mm.histImg`) sia per calcoli come la **funzione di distribuzione cumulativa (CDF)** e l'equalizzazione.

> **Interpretazione dell'Istogramma**
>
> - **Stretto a sinistra:** immagine sottoesposta (scura).
> - **Stretto a destra:** immagine sovraesposta (chiara).
> - **Concentrato al centro:** basso contrasto.
> - **Distribuito su tutta la gamma:** alto contrasto, buon utilizzo dei toni disponibili.

La Figura 3.6 presenta l'istogramma dell'immagine del mandrillo, nonché versioni scurita (`mm.subm`) e schiarita (`mm.addm`). Si osserva lo spostamento della distribuzione delle intensità rispettivamente verso sinistra e verso destra. Si noti che l'intervallo rappresentato sull'asse x non corrisponde necessariamente all'intera gamma da 0 a 255.

```
img_dark = mm.subm(img_gray, 80)
img_high = mm.addm(img_gray, 80)

mm.show(
    [img_gray, img_dark, img_high,
     mm.histImg(img_gray), mm.histImg(img_dark), mm.histImg(img_high)],
    titles=["Originale", "Scurita (-80)", "Schiarita (+80)",
            "Istogramma - originale", "Istogramma - scurita", "Istogramma - schiarita"],
    cols=3
)
```

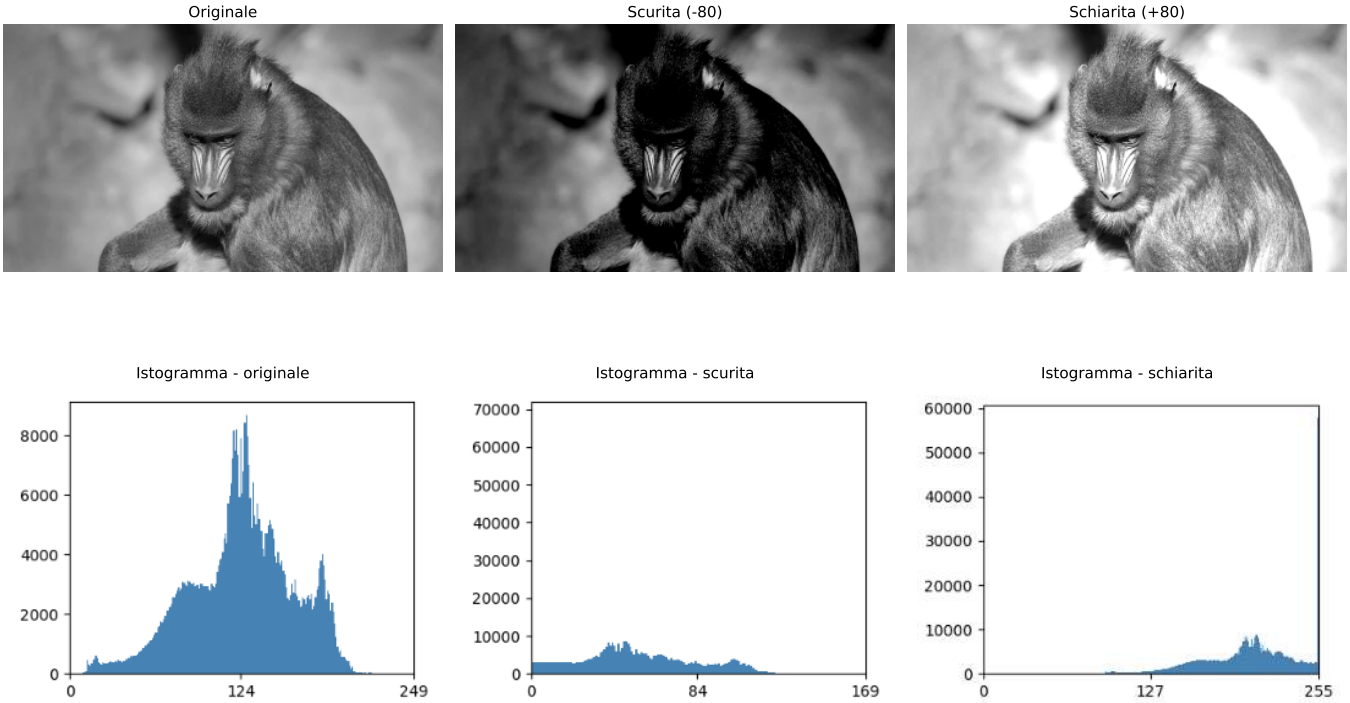


Figura 3.6: Istogrammi dell'immagine originale, di una versione scurita (-80) e di una schiarita ($+80$). La sottrazione/addizione satura a 0 e 255.

3.3.1 Equalizzazione dell'Istogramma

L'**equalizzazione dell'istogramma** ridistribuisce le intensità affinché l'istogramma risultante sia il più uniforme possibile. La mappatura è data dalla **funzione di distribuzione cumulativa (CDF)**:

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p(r_j) = \frac{L-1}{MN}\sum_{j=0}^{k} n_j \tag{3.7}$$

La trasformazione è monotòna: i livelli frequenti ricevono intervalli più ampi nel dominio di uscita (maggiore separazione → maggiore contrasto), mentre i livelli rari vengono compressi.

L'algoritmo completo, in cinque fasi, è presentato nella Tabella 3.1.

Tabella 3.1: Algoritmo di equalizzazione dell'istogramma.

Fase	Operazione	Formula
1	**Istogramma**	$h[k] \leftarrow$ numero di pixel con intensità k, $k = 0 \dots L-1$
2	**Probabilità**	$p[k] \leftarrow h[k]/MN$
3	**CDF**	$\text{cdf}[k] \leftarrow \sum_{j=0}^{k} p[j]$ (somma cumulativa)
4	***Look-Up Table* (mappatura)**	$\text{lut}[k] \leftarrow \text{round}(\text{cdf}[k] \times (L-1))$
5	**Applicazione**	$g[i,j] \leftarrow \text{lut}[f[i,j]]$ (per ogni pixel)

Si noti nella Figura 3.7 che l'equalizzazione **ridistribuisce** i toni esistenti in posizioni più distanziate lungo l'intervallo $[0, L-1]$, ma non crea nuovi toni — l'immagine equalizzata continua ad avere esattamente 3 toni distinti, ora in $\{1, 5, 7\}$ invece di $\{2, 3, 4\}$.

```
img5 = np.array([[3, 4, 2, 3, 4],
                 [4, 3, 3, 4, 3],
                 [2, 3, 4, 3, 2],
                 [3, 4, 3, 2, 3],
                 [4, 3, 2, 3, 4]], dtype=np.uint8)
L = 8  # 3 bit: livelli 0..7

# 1. Istogramma con dimensione garantita fino a L
h = mm.hist(img5, 3)  # B=3 → 2³=8 livelli, dimensione garantita

p   = h / h.sum()  # 2. Probabilità di ogni livello p[k] = h[k] / MN

cdf = np.cumsum(p) # 3. CDF normalizzata o
# cdf = np.zeros(len(p))
# cdf[0] = p[0]
# for k in range(1, len(p)):
#     cdf[k] = cdf[k-1] + p[k] # cdf[k] = Σ p[j], j=0..k

lut = np.round(cdf * (L - 1)).astype(np.uint8)  # 4. LUT: mappatura a [0, L-1]

img5_eq = lut[img5] # 5. Applica LUT pixel per pixel o, equivalentemente a:
# l, c = img5.shape
# img5_eq = np.zeros((l, c), dtype=np.uint8)
# for i in range(l):
#     for j in range(c):
#         img5_eq[i, j] = lut[img5[i, j]]      # g[i,j] = lut[f[i,j]]

print("Immagine originale (5×5, 3 bit):")
print(img5)
print()
header = f"{'k':>3} {'h[k]':>6} {'p[k]':>7} {'CDF[k]':>8} {'lut[k]':>7}"
print(header)
print("-" * len(header))
for k in range(L):
```

```
    print(f"{k:>3} {h[k]:>6} {p[k]:>7.4f} {cdf[k]:>8.4f} {lut[k]:>7}")
print()
print("Immagine equalizzata (5×5):")
print(img5_eq)

# Conta toni distinti
tons_orig = len(np.unique(img5))
tons_eq   = len(np.unique(img5_eq))

mm.show(
    [img5, img5_eq, mm.histImg(img5, L-1), mm.histImg(img5_eq, L-1)],
    titles=[f"Originale ({tons_orig} toni: {sorted(np.unique(img5).tolist())})",
            f"Equalizzata ({tons_eq} toni: {sorted(np.unique(img5_eq).tolist())})",
            "Istogramma - Originale",
            "Istogramma - Equalizzata"],
    rows=2, cols=2, figsize=(5, 4), dpi=100
)
```

```
Immagine originale (5×5, 3 bit):
[[3 4 2 3 4]
 [4 3 3 4 3]
 [2 3 4 3 2]
 [3 4 3 2 3]
 [4 3 2 3 4]]

  k   h[k]    p[k]   CDF[k]  lut[k]
-----------------------------------
  0      0  0.0000   0.0000       0
  1      0  0.0000   0.0000       0
  2      5  0.2000   0.2000       1
  3     12  0.4800   0.6800       5
  4      8  0.3200   1.0000       7
  5      0  0.0000   1.0000       7
  6      0  0.0000   1.0000       7
  7      0  0.0000   1.0000       7

Immagine equalizzata (5×5):
[[5 7 1 5 7]
 [7 5 5 7 5]
 [1 5 7 5 1]
 [5 7 5 1 5]
 [7 5 1 5 7]]
```

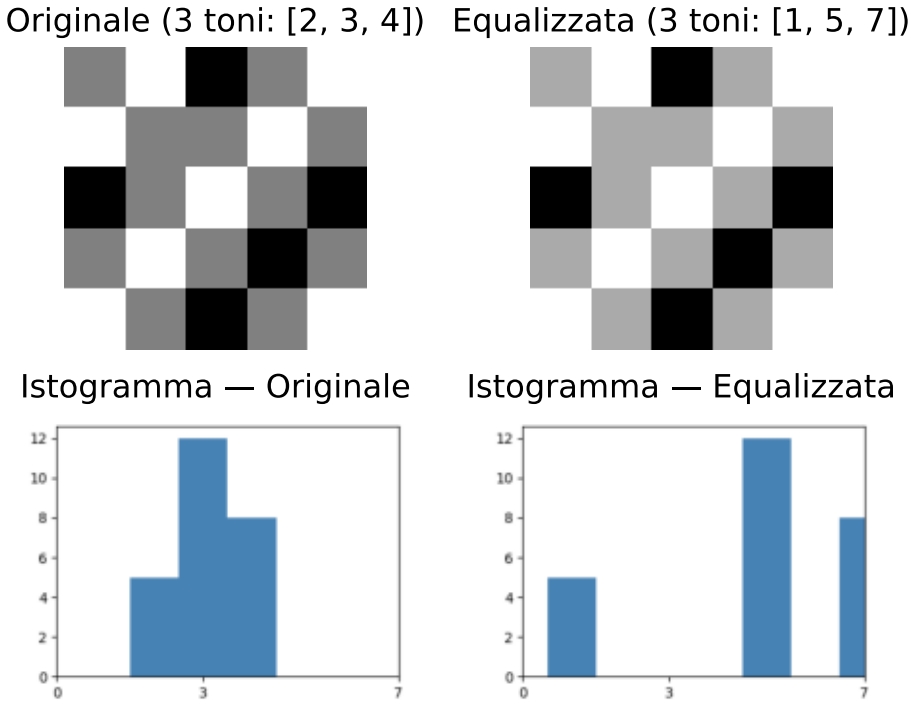


Figura 3.7: Equalizzazione dell'istogramma in un'immagine 5×5 a 3 bit (L=8): immagine originale con toni concentrati in {2,3,4}, immagine equalizzata con toni redistribuiti in {1,5,7}, e i rispettivi istogrammi che evidenziano la dispersione delle frequenze.

Limitazione: l'equalizzazione globale può enfatizzare eccessivamente il rumore e produrre contrasto eccessivo nelle regioni omogenee. Il **CLAHE** (*Contrast Limited Adaptive Histogram Equalization*) riduce questo problema applicando l'equalizzazione a blocchi locali (*tiles*) e limitando l'altezza dei picchi dell'istogramma prima dell'equalizzazione.

La Figura 3.8 confronta l'immagine originale, l'equalizzazione globale tramite `mm.equalize` e il CLAHE di OpenCV, mostrando anche gli istogrammi risultanti. Diversamente dall'equalizzazione globale, che utilizza una singola trasformazione basata sulla CDF dell'intera immagine, il CLAHE adatta il contrasto a ciascuna regione, essendo particolarmente utile in immagini con illuminazione non uniforme.

Nell'esempio sono stati utilizzati `clipLimit=2.0` e `tileGridSize=(32,32)`. Il parametro `clipLimit` definisce quanto i picchi dell'istogramma locale possono crescere prima di essere tagliati (*clipped*). In OpenCV, questo valore è un fattore relativo: il limite effettivo è approssimativamente calcolato come `clipLimit × (numero di pixel del blocco / numero di livelli di grigio)`. Ad esempio, in un blocco con 4096 pixel e un'immagine a 8 bit (256 livelli di grigio), la frequenza media per livello è $4096/256 = 16$. Pertanto, `clipLimit=2.0` consente picchi di circa $2 \times 16 = 32$ occorrenze prima del taglio. Le occorrenze eccedenti non vengono scartate: vengono ridistribuite tra gli altri livelli di grigio dell'istogramma, riducendo la concentrazione eccessiva in pochi livelli ed evitando un'amplificazione esagerata del contrasto locale. Valori minori limitano maggiormente il contrasto e riducono l'amplificazione del rumore, mentre valori maggiori consentono un miglioramento più intenso ma possono introdurre artefatti.

- `clipLimit=1.0`: miglioramento morbido e conservativo;
- `clipLimit=2.0`: buon equilibrio tra contrasto e naturalezza;
- `clipLimit=4.0`: maggiore evidenziazione dei dettagli locali;
- `clipLimit=8.0`: contrasto aggressivo, con possibile amplificazione del rumore.

Pertanto, il CLAHE tende a produrre risultati più naturali rispetto all'equalizzazione globale, specialmente in immagini con ombre, riflessi o illuminazione disomogenea.

```
img_eq    = mm.equalize(img_gray)
clahe     = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(32, 32))
img_clahe = clahe.apply(img_gray)

images = [img_gray, img_eq, img_clahe]
titles = ["Original", "mm.equalize (CDF)", "CLAHE"]

mm.show(
    images + [mm.histImg(img) for img in images],
    titles=titles + [f"Histograma - {t}" for t in titles],
    rows=2, cols=3,
    figsize=(14, 8)
)
```

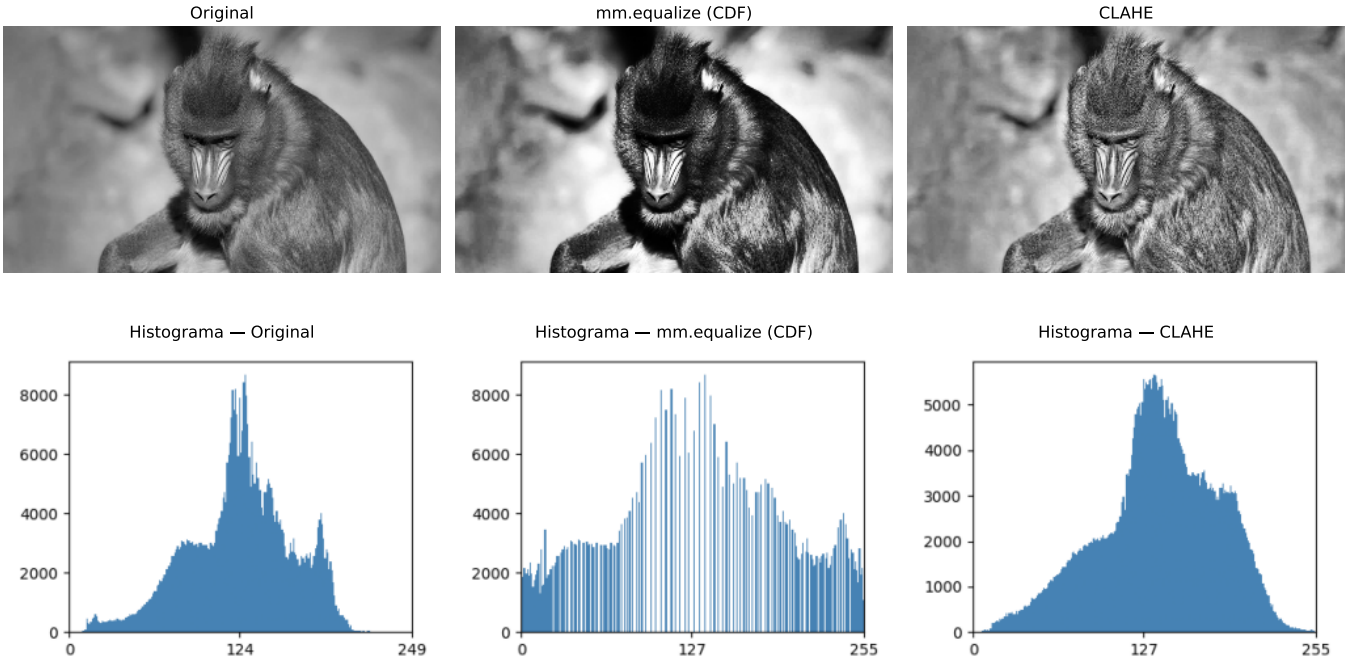


Figura 3.8: Equalizzazione dell'istogramma: mm.equalize (globale tramite CDF) vs. CLAHE (adattativa con limitazione del contrasto). Gli istogrammi rivelano la dispersione progressiva delle intensità.

3.3.2 Specifica dell'istogramma

Mentre l'equalizzazione impone una distribuzione uniforme, la **specifica dell'istogramma** (*histogram matching*) consente che l'istogramma dell'immagine di uscita segua una distribuzione **arbitraria** — ad esempio, l'istogramma di un'altra immagine di riferimento.

La procedura prevede tre fasi:

1. Calcolare la CDF dell'immagine di ingresso: $P_r(r_k)$.
2. Calcolare la CDF dell'immagine di riferimento: $P_z(z_k)$.
3. Per ogni livello r_k, trovare il livello z che minimizza $|P_z(z) - P_r(r_k)|$.

$$T(r_k) = \arg\min_z |P_z(z) - P_r(r_k)| \tag{3.8}$$

Nella Figura 3.9, trasferiamo il profilo tonale del leopardo (Figura 3.3) all'immagine del mandrillo — un'applicazione diretta del concetto visto nel *blending*: invece di fondere i pixel, qui fondiamo le distribuzioni tonali.

```
# mappa src sul profilo tonale di ref tramite CDF
def hist_specify(src, ref):
    cdf_src = np.cumsum(mm.hist(src) / src.size)
    cdf_ref = np.cumsum(mm.hist(ref) / ref.size)
    lut = np.array([np.argmin(np.abs(cdf_ref - v)) for v in cdf_src], dtype=np.uint8)
    return lut[src]

img_spec = hist_specify(img_gray, img_leop_gray)

mm.show(
    [img_gray, img_leop_gray, img_spec,
     mm.histImg(img_gray), mm.histImg(img_leop_gray), mm.histImg(img_spec)],
    titles=["Mandrillo (originale)", "Leopardo (riferimento)", "Mandrillo → profilo del
leopardo",
            "Istogramma - Mandrillo", "Istogramma - Leopardo", "Istogramma - risultato"],
    rows=2, cols=3, figsize=(12, 9)
)
```

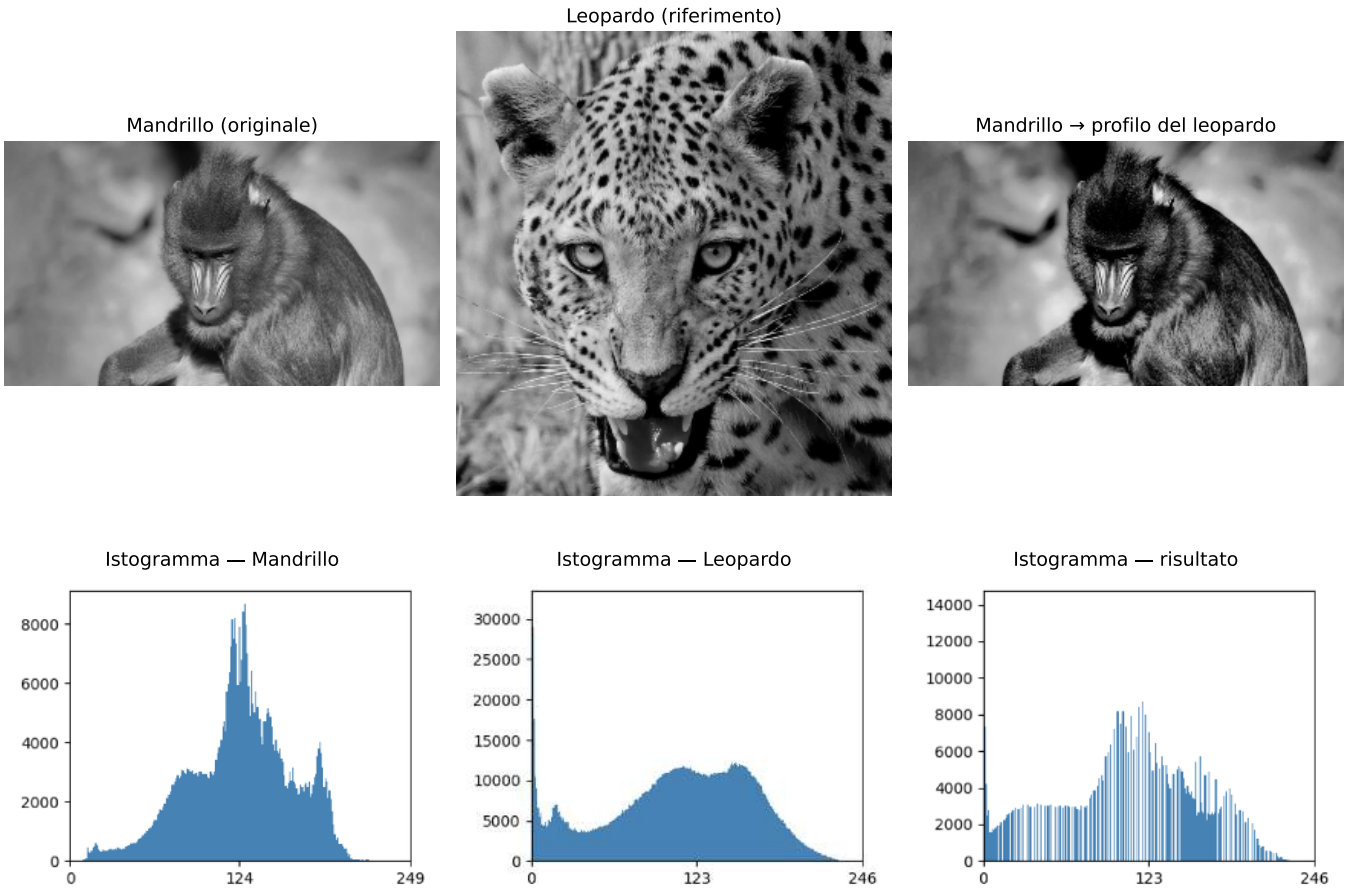


Figura 3.9: Specifica dell'istogramma: mandrillo mappato sul profilo tonale del leopardo. La CDF dell'uscita approssima la CDF di riferimento.

3.4 Fondamenti Spaziali: Intorno, Convoluzione e *Kernel*

Le operazioni di filtraggio spaziale non agiscono su un singolo pixel isolato, ma su un **intorno** che lo circonda. A tale scopo, si utilizza una piccola matrice di coefficienti denominata **kernel** (o maschera), che percorre l'intera immagine attraverso una **finestra scorrevole** (*sliding window*).

Le finestre più comuni sono 3×3, 5×5 e 7×7. In una finestra 3×3, ad esempio, il pixel centrale viene elaborato insieme ai suoi otto vicini immediati. In ogni posizione della finestra, i valori dei pixel vengono combinati

con i coefficienti del *kernel*, producendo un nuovo valore per il pixel centrale.

3.4.1 Intorno

Si consideri una finestra 3×3 centrata sul pixel (x, y):

$$\begin{bmatrix} (x-1, y-1) & (x, y-1) & (x+1, y-1) \\ (x-1, y) & (x, y) & (x+1, y) \\ (x-1, y+1) & (x, y+1) & (x+1, y+1) \end{bmatrix} \tag{3.9}$$

In generale, una finestra di dimensioni $(2a+1) \times (2b+1)$ comprende tutti i pixel situati fino a a posizioni in orizzontale e fino a b posizioni in verticale rispetto al pixel centrale. Pertanto, una finestra 3×3 corrisponde a $a = b = 1$, una finestra 5×5 a $a = b = 2$, e così via.

Matematicamente, l'intorno è definito da

$$\mathcal{V}(x, y) = \{(x+s,\, y+t) : -a \leq s \leq a,\, -b \leq t \leq b\} \tag{3.10}$$

3.4.2 Trattamento dei Bordi

I pixel vicini ai bordi hanno parte del loro intorno al di fuori dell'immagine. Per applicare filtri in queste regioni, è necessario definire come verranno ottenuti i valori esterni. Le tre strategie più comuni (con la costante equivalente di OpenCV tra parentesi) sono:

- **Zero-padding** (`BORDER_CONSTANT`): completa la regione esterna con zeri.
- **Replicazione** (`BORDER_REPLICATE`): ripete il valore del pixel del bordo.
- **Riflessione** (`BORDER_REFLECT_101`): rispecchia i pixel vicini, senza ripetere quello del bordo.

`mm.conv` utilizza la riflessione per impostazione predefinita, poiché preserva meglio la continuità dei livelli di grigio e riduce gli artefatti nel trattamento dei bordi.

L'esempio seguente confronta le tre strategie con `mm.pad` su una matrice 3×3. Osserva come ciascuna riempie i pixel esterni necessari per applicare un filtro 3×3 anche agli angoli.

```
img = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
], dtype=np.uint8)

bordas = ["constant", "replicate", "reflect101"]

print("Immagine originale:")
print(mm.drawImg(img))

for k in [3, 5]:
    b = k // 2
    print(f"=== Kernel {k}x{k} (padding b={b}) ===")
    for nome in bordas:
        print(nome)
        print(mm.drawImg(mm.pad(img, b, border=nome)))
```

```
Immagine originale:
1 2 3
4 5 6
7 8 9

=== Kernel 3x3 (padding b=1) ===
```

```
constant
0 0 0 0 0
0 1 2 3 0
0 4 5 6 0
0 7 8 9 0
0 0 0 0 0

replicate
1 1 2 3 3
1 1 2 3 3
4 4 5 6 6
7 7 8 9 9
7 7 8 9 9

reflect101
5 4 5 6 5
2 1 2 3 2
5 4 5 6 5
8 7 8 9 8
5 4 5 6 5

=== Kernel 5x5 (padding b=2) ===
constant
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 1 2 3 0 0
0 0 4 5 6 0 0
0 0 7 8 9 0 0
0 0 0 0 0 0 0
0 0 0 0 0 0 0

replicate
1 1 1 2 3 3 3
1 1 1 2 3 3 3
1 1 1 2 3 3 3
4 4 4 5 6 6 6
7 7 7 8 9 9 9
7 7 7 8 9 9 9
7 7 7 8 9 9 9

reflect101
9 8 7 8 9 8 7
6 5 4 5 6 5 4
3 2 1 2 3 2 1
6 5 4 5 6 5 4
9 8 7 8 9 8 7
6 5 4 5 6 5 4
3 2 1 2 3 2 1
```

Osserva che il risultato di un filtro può variare significativamente in base al trattamento adottato per i bordi dell'immagine.

Nella libreria didattica `morph.py`, gli algoritmi implementati direttamente in Python (senza OpenCV o *scikit-image*) — le varianti con suffisso `0`, come `mm.conv0` — non usano *padding*. Per ogni pixel, gli elementi del vicinato vengono esaminati singolarmente, e solo i vicini le cui coordinate appartengono al dominio dell'immagine partecipano al calcolo. Pertanto, nelle regioni vicine ai bordi, il vicinato effettivo può contenere meno pixel rispetto alla finestra specificata; negli operatori basati su `mm._viz` (ad esempio `mm.laplacian`), i pixel di bordo con finestra incompleta vengono lasciati invariati.

3.4.3 Correlazione vs. Convoluzione

Esistono due meccanismi matematicamente correlati.

Correlazione incrociata (*cross-correlation*) — il *kernel* viene applicato direttamente:

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t)\, f(x + s,\, y + t) \tag{3.11}$$

Convoluzione bidimensionale — il *kernel* viene ruotato di 180° prima dell'applicazione:

$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t)\, f(x - s,\, y - t) \tag{3.12}$$

Per *kernel* simmetrici (Gaussiano, Laplaciano, media) le due operazioni producono risultati identici. Per *kernel* asimmetrici (Sobel, Prewitt) la differenza è significativa, come mostrano gli esempi seguenti.

3.4.3.1 Correlazione (`mm.conv`)

```
# immagine 4x4 (uint8) e kernel asimmetrico 3x3
img = np.array([[ 1,  2,  3,  4],
                [ 5,  6,  7,  8],
                [ 9, 10, 11, 12],
                [13, 14, 15, 16]], dtype=np.uint8)

w = np.array([[0, 1, 2],
              [0, 0, 0],
              [0, 0, 0]], dtype=np.float32)

corr = mm.conv(img, w, border="constant")  # zero fuori dall'immagine

print("Immagine originale:");         print(mm.drawImg(img))
print("Kernel:");                  print(mm.drawImg(w))
print("Risultato della correlazione:"); print(mm.drawImg(corr))
```

```
Immagine originale:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

Kernel:
  0   1   2
  0   0   0
  0   0   0

Risultato della correlazione:
 0  0  0  0
 5  8 11  4
17 20 23  8
29 32 35 12
```

`mm.conv` esegue la correlazione, cioè applica il *kernel* esattamente nell'orientamento fornito.

3.4.3.2 Convoluzione

```
# ripetuti qui affinché la cella sia indipendente
img = np.array([[ 1,  2,  3,  4],
                [ 5,  6,  7,  8],
                [ 9, 10, 11, 12],
                [13, 14, 15, 16]], dtype=np.uint8)
w   = np.array([[0, 1, 2],
                [0, 0, 0],
                [0, 0, 0]], dtype=np.float32)

# kernel ruotato di 180° (equivale a np.rot90(w, 2))
w_conv = np.array([[0, 0, 0],
                   [0, 0, 0],
                   [2, 1, 0]], dtype=np.float32)

conv = mm.conv(img, w_conv, border="constant")

print("Immagine originale:");            print(mm.drawImg(img))
print("Kernel originale:");            print(mm.drawImg(w))
print("Kernel ruotato di 180°:");    print(mm.drawImg(w_conv))
print("Risultato della convoluzione:");    print(mm.drawImg(conv))
```

```
Immagine originale:
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

Kernel originale:
  0   1   2
  0   0   0
  0   0   0

Kernel ruotato di 180°:
  0   0   0
  0   0   0
  2   1   0

Risultato della convoluzione:
 5 16 19 22
 9 28 31 34
13 40 43 46
 0  0  0  0
```

La convoluzione utilizza il *kernel* ruotato di 180°. Per riprodurre la definizione matematica di convoluzione, si ruota il *kernel* (qui, `[[0,1,2],[0,0,0],[0,0,0]]` → `[[0,0,0],[0,0,0],[2,1,0]]`) prima di applicare `mm.conv`.

3.4.4 Il Ruolo del *Kernel*

I coefficienti del *kernel* determinano completamente l'effetto prodotto dal filtro, come riassunto nella Tabella 3.2.

Tabella 3.2: Interpretazione tipica dei coefficienti del *kernel*.

Caratteristica	Effetto tipico
Coefficienti positivi con somma 1	Attenuazione (*passa-basso*)
Somma pari a 0, con valori positivi e negativi	Rilevamento dei bordi (*passa-alto*)

Caratteristica	Effetto tipico
Coefficiente centrale positivo dominante e vicini negativi	Enfasi della nitidezza
Coefficienti asimmetrici	Gradiente direzionale

Esempi:

Attenuazione:

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Rilevamento dei bordi:

$$\begin{bmatrix}-1 & -1 & -1\\ -1 & 8 & -1\\ -1 & -1 & -1\end{bmatrix}$$

Enfasi della nitidezza:

$$\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix}$$

Gradiente direzionale (Sobel):

$$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$$

La Figura 3.10 dimostra il meccanismo passo dopo passo: per ogni posizione della finestra, si moltiplica ciascun coefficiente del *kernel* per il pixel corrispondente dell'intorno e si sommano i prodotti ottenuti. Il risultato è esattamente il valore definito dalla Equazione 3.11 per quella posizione dell'immagine. Sebbene le immagini prodotte da `mm.conv0` e `cv2.filter2D` (o `mm.conv`) siano visivamente molto simili, l'implementazione basata su OpenCV è migliaia di volte più rapida, come mostrato di seguito.

⚠ Prestazioni: cicli Python vs. operazioni vettorializzate

La funzione `mm.conv0` implementa la correlazione direttamente in Python tramite cicli annidati. Sebbene questo approccio sia adeguato a fini didattici, esegue un numero elevato di operazioni e diventa lento per immagini di dimensioni maggiori.
Invece, `mm.conv` utilizza `cv2.filter2D`, implementato in C++ e ottimizzato per operazioni matriciali. Nell'esempio presentato, la versione vettorializzata è risultata oltre **3000 volte più rapida** rispetto all'implementazione didattica, producendo un risultato visivamente equivalente.
Le differenze numeriche osservate si concentrano principalmente sui bordi dell'immagine. In `mm.conv0`, i pixel del bordo rimangono invariati, mentre `mm.conv` adotta una strategia di riflessione dei bordi (`cv2.BORDER_REFLECT_101`, predefinita di `cv2.filter2D`).
Pertanto, `mm.conv0` dovrebbe essere utilizzata per comprendere l'algoritmo, mentre `mm.conv` è l'opzione consigliata per applicazioni pratiche.

```
import time

w_mean = np.ones((3,3), dtype=np.float32) / 9.0
img_gray = img_leop_gray
# Demonstração numérica em patch 5×5
patch = img_gray[250:255, 250:255].astype(np.float32)
roi   = patch[0:3, 0:3]
print("Patch 5×5 (intensidades):")
```

```python
print(patch.astype(np.int32))
print(f"\nKernel 3×3 de média:\n{w_mean}")
print(f"\nCorrelação no pixel central [1,1]: \
      {(w_mean * roi).sum():.1f}  (original: {patch[1,1]:.0f})")

# Comparação de tempo na imagem completa
t0 = time.perf_counter()
img_conv0 = mm.conv0(img_gray, w_mean)
t_conv0 = time.perf_counter() - t0

t0 = time.perf_counter()
img_conv = mm.conv(img_gray, w_mean)
t_conv = time.perf_counter() - t0

diff = np.abs(img_conv0.astype(int) - img_conv.astype(int)).max()
print(f"\nTempos na imagem {img_gray.shape[0]}x{img_gray.shape[1]}:")
print(f"  mm.conv0 (laços Python): {t_conv0:.3f} s")
print(f"  mm.conv  (cv2.filter2D): {t_conv:.5f} s")
print(f"  Aceleração:              {t_conv0/t_conv:.0f}× mais rápido")
print(f"  Diferença máxima:        {diff} (bordas com padding diferente)")

mm.show(
    [img_gray, img_conv0, img_conv],
    titles=["Original", f"conv0 ({t_conv0:.2f}s)", f"conv ({t_conv:.4f}s)"],
    cols=3
)
```

```
Patch 5×5 (intensidades):
[[ 91  95 108 116 114]
 [106 107 108 111 114]
 [102 103 107 112 116]
 [ 83  90 111 125 123]
 [ 79  88 110 126 124]]

Kernel 3×3 de média:
[[0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]
 [0.11111111 0.11111111 0.11111111]]

Correlação no pixel central [1,1]:       103.0  (original: 107)

Tempos na imagem 1324x1242:
  mm.conv0 (laços Python): 6.419 s
  mm.conv  (cv2.filter2D): 0.00250 s
  Aceleração:              2568× mais rápido
  Diferença máxima:        39 (bordas com padding diferente)
```

Figura 3.10: Correlação com *kernel* de média 3×3: versão didática (mm.conv0) vs. vetorizada (mm.conv via cv2.filter2D). A diferença máxima de 39 ocorre nas bordas.

3.4.5 Esempio Numerico: Correlazione Passo per Passo

Per rendere concreto il meccanismo della Equazione 3.11, si consideri il *kernel* di media 3×3 ($a = b = 1$, tutti i coefficienti $= 1/9 \approx 0{,}111$) applicato al *patch* 5×5 estratto dall'immagine del leopardo. La Figura 3.11 mostra il *patch* con griglia ed evidenzia in giallo la finestra 3×3 centrata sul pixel $[1, 1]$:

```
patch = img_gray[250:255, 250:255]
B     = np.ones((3, 3), dtype=np.uint8)

print("Patch 5×5 (intensità):")
print(mm.drawImg(patch))

mm.drawImgKernel(patch, B, 1, 1, 40)
```

```
Patch 5×5 (intensità):
 91  95 108 116 114
106 107 108 111 114
102 103 107 112 116
 83  90 111 125 123
 79  88 110 126 124
```

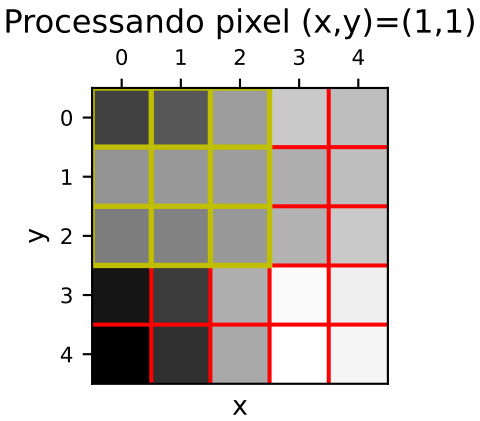


Figura 3.11: *Patch* 5×5 estratto dall'immagine del leopardo (posizione [250:255, 250:255]). La finestra gialla evidenzia il vicinato 3×3 centrato sul pixel [1,1] dove la correlazione verrà calcolata.

Per illustrare il calcolo della correlazione, si consideri il pixel nella posizione $[1, 1]$ del *patch* 5×5 mostrato

nella Figura 3.11.. Tale posizione è stata scelta solo per comodità didattica, poiché possiede un intorno 3×3 completo attorno a sé.

I valori di tale intorno corrispondono alla sottomatrice in alto a sinistra del *patch*:

$$\text{vicinato} = \begin{bmatrix} 91 & 95 & 108 \\ 106 & 107 & 108 \\ 102 & 103 & 107 \end{bmatrix}$$

Applicando la Equazione 3.11 con il kernel di media:

$$g[1,1] = \frac{91 + 95 + 108 + 106 + 107 + 108 + 102 + 103 + 107}{9} = \frac{927}{9} = 103$$

Il risultato (103) è leggermente inferiore al valore originale del pixel centrale (107), poiché la media incorpora vicini di minore intensità, producendo l'effetto di smussamento. In pratica, l'algoritmo avvia l'elaborazione in $[0, 0]$ e ripete lo stesso calcolo per ogni posizione dell'immagine, spostando la finestra fino a coprire l'intero dominio.

3.5 Filtraggio Spaziale di Levigatura

I filtri di levigatura (*smoothing filters*) attenuano le variazioni brusche di intensità, riducendo rumore e dettagli ad alta frequenza. Sono filtri **passa-basso** — preservano le componenti a bassa frequenza (strutture grandi) e attenuano quelle ad alta frequenza (rumore, bordi).

3.5.1 Filtro a Media (*Box Filter*)

Il filtro a media utilizza un *kernel* uniforme di dimensione $n \times n$, dove tutti i coefficienti valgono $1/n^2$:

$$w_{\text{media}} = \frac{1}{n^2} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}_{n \times n} \tag{3.13}$$

Ogni pixel di uscita è la media aritmetica degli n^2 pixel del suo intorno. Si noti che la somma dei coefficienti è sempre 1 — la luminosità media dell'immagine viene preservata. *Kernel* più grandi producono una levigatura più aggressiva, ma sfumano progressivamente i bordi.

La Figura 3.12 mostra l'effetto del filtro di media con *kernel* 3×3, 7×7 e 15×15 su un dettaglio dell'immagine del leopardo. I risultati sono stati ottenuti con `mm.blur`, che implementa il filtro di media tramite la funzione `cv2.blur`, equivalente alla convoluzione dell'immagine con un *kernel* uniforme i cui coefficienti sono $h(x, y) = 1/N^2$; in modo equivalente, lo stesso risultato può essere ottenuto con `mm.conv`, calcolando ($g = f * h$). Man mano che il *kernel* aumenta, più pixel contribuiscono a ciascun valore di uscita, intensificando la levigatura, riducendo il rumore e rendendo i dettagli fini e i bordi progressivamente più sfocati.

```
# Dettaglio della regione dell'occhio
y0, y1, x0, x1 = 580, 740, 680, 900
crop = lambda img: img[y0:y1, x0:x1]

img_gray_crop = crop(img_gray)

sizes = [3, 7, 15]
imgs  = [img_gray_crop] + \
    [mm.blur(img_gray_crop, k) for k in sizes] # o
    #[mm.conv(img_gray_crop, np.ones((k,k), dtype=np.float32)/(k*k)) for k in sizes]
titles = ["Original"] + [f"Média {k}×{k}" for k in sizes]
```

```
mm.show(imgs, titles=titles, cols=4)
```

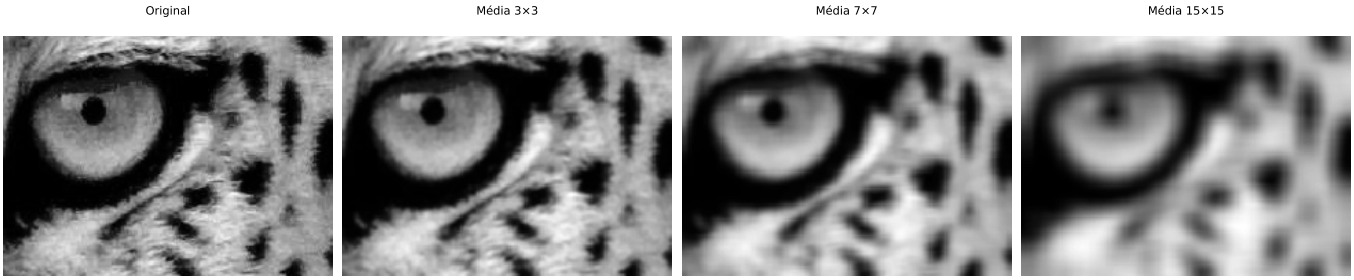


Figura 3.12: Filtro medio con *kernel* di dimensione crescente (3×3, 7×7, 15×15). La sfocatura dei bordi aumenta con la dimensione del *kernel*.

3.5.2 Filtro Gaussiano

Il filtro gaussiano pesa i pixel dell'intorno secondo una funzione gaussiana bidimensionale:

$$G(s,t) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{s^2+t^2}{2\sigma^2}} \tag{3.14}$$

dove σ è la deviazione standard e controlla il raggio di influenza. I pixel più vicini al centro hanno peso maggiore; i pixel distanti vengono progressivamente ignorati.

A Figura 3.13 apresenta o *kernel* Gaussiano 5×5 gerado para $\sigma = 1$. O *kernel* foi costruído a partir do produto externo de due vettori Gaussiani unidimensionali e successivamente normalizzato affinché la somma dei suoi coefficienti sia uguale a 1. Si osserva che i pesi maggiori si concentrano al centro della matrice, decrescendo radialmente verso i bordi. Questa distribuzione fa sì che i pixel centrali abbiano maggiore influenza sul risultato del filtraggio, contribuendo a una smussatura più naturale e con una migliore conservazione dei bordi rispetto al filtro della media.

```
# Genera kernel Gaussiano 5×5 tramite cv2
k_gauss = cv2.getGaussianKernel(5, 1)
w_gauss = k_gauss @ k_gauss.T          # @ => moltiplicazione matriciale: G(s,t) = G(s)·G(t)
w_gauss = w_gauss / w_gauss.sum()      # garantisce somma = 1

print("Kernel Gaussiano 5×5 ( =1), normalizzato:")
for row in w_gauss:
    print("  " + "  ".join(f"{v:.4f}" for v in row))
print(f"\nSomma dei coefficienti: {w_gauss.sum():.6f}")
print(f"Peso centrale vs. angolo: {w_gauss[2,2]:.4f} vs. {w_gauss[0,0]:.4f} "
      f"({w_gauss[2,2]/w_gauss[0,0]:.1f}× maggiore)")

mm.drawImgPlt((w_gauss * 1000).astype(np.uint8), scale=40)
```

```
Kernel Gaussiano 5×5 ( =1), normalizzato:
  0.0030  0.0133  0.0219  0.0133  0.0030
  0.0133  0.0596  0.0983  0.0596  0.0133
  0.0219  0.0983  0.1621  0.0983  0.0219
  0.0133  0.0596  0.0983  0.0596  0.0133
  0.0030  0.0133  0.0219  0.0133  0.0030

Somma dei coefficienti: 1.000000
Peso centrale vs. angolo: 0.1621 vs. 0.0030 (54.6× maggiore)
```

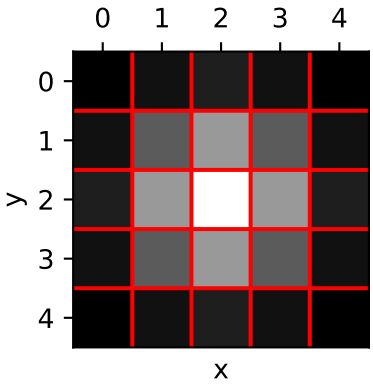


Figura 3.13: *Kernel* Gaussiano 5×5 (=1): pesi normalizzati, maggiori al centro e decrescenti radialmente. La griglia facilita la lettura di ciascun coefficiente.

Vantaggio computazionale della separabilità

Si consideri un *kernel* quadrato di dimensione $n \times n$. Se tale filtro è separabile (come quello gaussiano), la convoluzione 2D può essere decomposta in due convoluzioni 1D: una orizzontale e una verticale. In questo caso, il costo per pixel passa da circa $O(n^2)$ operazioni (convoluzione 2D diretta) a $O(2n)$ operazioni (due convoluzioni 1D). La complessità viene quindi ridotta in modo significativo, rendendo l'elaborazione più efficiente.

Rispetto al filtro medio, il filtro gaussiano:

- **Preserva meglio i bordi** — la ponderazione radiale smussa senza creare transizioni brusche;
- **Non introduce anelli** (*ringing*) nel dominio della frequenza, poiché la gaussiana è la propria trasformata di Fourier (Capitolo 5);
- **È controllato da** σ — aumentare σ equivale ad aumentare il raggio di smussamento in modo continuo e prevedibile.

La Figura 3.14 confronta i filtri media e gaussiano applicati all'immagine del leopardo utilizzando una finestra 9×9. Il filtro media è stato implementato tramite convoluzione con un *kernel* uniforme, in cui tutti gli 81 pixel dell'intorno possiedono lo stesso peso (1/81), mentre il filtro gaussiano è stato ottenuto con `cv2.GaussianBlur`, utilizzando pesi definiti da una distribuzione gaussiana. Entrambi riducono il rumore e levigano l'immagine, ma il filtro gaussiano preserva meglio i bordi e i dettagli locali, come si può osservare nella regione ingrandita dell'occhio.

La Figura 3.14 confronta i filtri media e gaussiano applicati a un dettaglio dell'immagine del leopardo con *kernel* 9×9. Il filtro media è stato ottenuto con `mm.blur`, equivalente alla convoluzione con un *kernel* uniforme i cui coefficienti valgono 1/81, mentre il filtro gaussiano è stato ottenuto con `mm.gaussian`, equivalente alla convoluzione con un *kernel* generato a partire da una distribuzione gaussiana. Entrambi promuovono levigatura e riduzione del rumore, ma il filtro gaussiano attribuisce un peso maggiore ai pixel centrali dell'intorno, preservando meglio i bordi e i dettagli locali, come si può osservare nella regione ingrandita dell'occhio.

```
img_media9 = mm.blur(img_gray, 9)
img_gauss9 = mm.gaussian(img_gray, 9, 0)

# Dettaglio della regione dell'occhio
img_gray_crop  = mm.crop(img_gray,   580, 740, 680, 900)
img_media9_crop = mm.crop(img_media9, 580, 740, 680, 900)
img_gauss9_crop = mm.crop(img_gauss9, 580, 740, 680, 900)

mm.show(
```

```
    [img_gray_crop, img_media9_crop, img_gauss9_crop],
    titles=["Dettaglio: Originale", "Media 9×9", "Gaussiano 9×9"],
    cols=3, figsize=(12, 8)
)
```

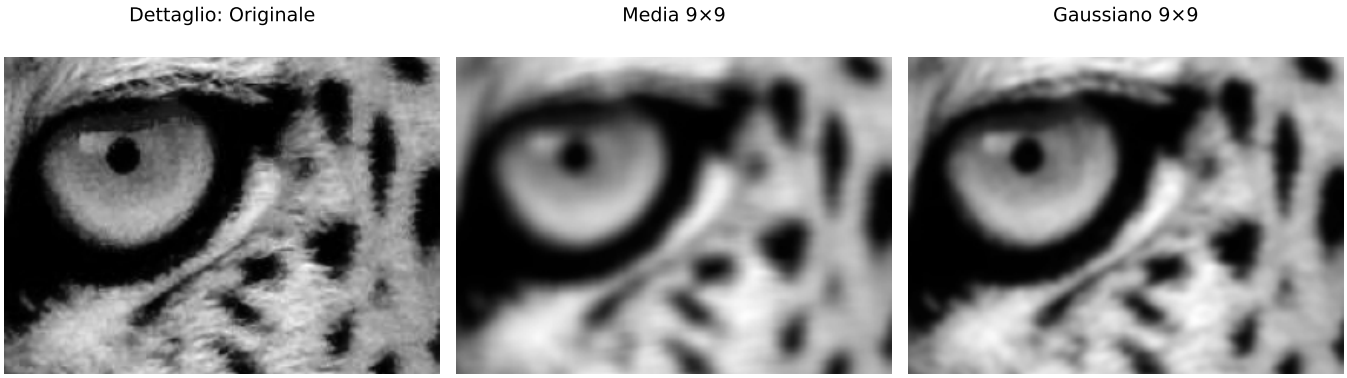


Figura 3.14: Confronto tra filtro medio e Gaussiano (*kernel* 9×9, =0). Il Gaussiano preserva meglio i bordi, visibile nel dettaglio del viso.

3.6 Filtraggio Spaziale di Enfasi

I filtri di enfasi (*sharpening filters*) enfatizzano le transizioni brusche di intensità, aumentando la nitidezza e la visibilità dei bordi. Sono filtri **passa-alto** — amplificano le componenti ad alta frequenza (bordi, texture) e sopprimono quelle a bassa frequenza (regioni uniformi).

L'intuizione è semplice: se sottraiamo da un'immagine la sua versione attenuata (che contiene solo le basse frequenze), ciò che resta sono le alte frequenze — bordi e dettagli. Sommando questo residuo all'immagine originale, il contrasto locale aumenta:

$$g = f + k\,(f - f_{\text{suave}}), \quad k > 0 \tag{3.15}$$

La Figura 3.15 illustra questo processo su un segnale 1D sintetico con tre strutture distinte: un gradino largo, un picco sottile e una rampa dolce. Nel pannello , il segnale originale $f(x)$; nel , la versione attenuata $f_{\text{suave}}(x)$ ottenuta mediante media mobile — si noti come il picco sottile venga attenuato. Il pannello mostra il residuo $f - f_{\text{suave}}$, che conserva solo le transizioni brusche. Infine, il pannello mostra $g(x)$: il picco, prima attenuato, viene ripristinato e amplificato rispetto all'originale. Regolate k e la dimensione della finestra per osservare il *trade-off* tra nitidezza e amplificazione del rumore.

I filtri di enfasi formalizzano questa idea direttamente nel *kernel*, senza necessità di due fasi separate.

3.6.1 Laplaciano

Il Laplaciano è un operatore di **derivata seconda** isotropico, cioè risponde in modo uguale alle variazioni in tutte le direzioni, a differenza degli operatori di derivata prima, come Sobel e Prewitt, che sono direzionali:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3.16}$$

Una proprietà importante della derivata seconda è che il suo valore è **prossimo allo zero nelle regioni uniformi** ed elevato nelle transizioni di intensità. Così, sottraendo il Laplaciano dall'immagine originale, si rafforzano bordi e dettagli, aumentando il contrasto locale:

$$g(x, y) = f(x, y) - \nabla^2 f(x, y) \tag{3.17}$$

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-03-filtragem1d.html

Figura 3.15: Simulatore: Filtraggio Spaziale di Enfasi 1D (Unsharp Masking e High-Boost)

Nella forma discreta, la derivata seconda in x è approssimata da $f(x+1,y) - 2f(x,y) + f(x-1,y)$, e analogamente in y. Sommando le due direzioni, si ottiene il *kernel* w_4 (4-vicini) o w_8 (8-vicini, includendo le diagonali):

$$w_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \qquad w_8 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.18}$$

Somma zero e centro negativo

Entrambi i *kernel* hanno **somma dei coefficienti pari a zero**: nelle regioni uniformi, l'uscita è 0 — il Laplaciano non modifica la luminosità media, ma rileva solo le variazioni. Il **centro negativo** indica che il pixel viene confrontato con i suoi vicini: quanto più esso si distingue (verso l'alto o verso il basso), tanto maggiore è il valore assoluto del Laplaciano in quel punto.

Nell'esempio seguente, il pixel centrale $[1,1] = 107$ possiede vicini $\{95, 106, 108, 103\}$. Poiché questi valori sono tra loro prossimi, la regione è quasi uniforme e il Laplaciano restituisce un valore basso, producendo poco miglioramento. Nelle regioni di bordo, dove vi sono differenze maggiori tra il pixel centrale e i suoi vicini, il Laplaciano assume valori più elevati (positivi o negativi), e l'operazione di Equazione 3.17 intensifica queste transizioni.

La Figura 3.16 illustra il calcolo del Laplaciano con il *kernel* w_4 in un intorno 3×3 evidenziato all'interno di un *patch* 5×5. L'esempio mostra il valore ottenuto dall'operatore e il corrispondente pixel migliorato nell'immagine di uscita, che passa da 107 a 123.

```
patch  = img_gray[250:255, 250:255]
w4     = np.array([[0, 1, 0],
                   [1,-4, 1],
                   [0, 1, 0]], dtype=np.float32)
B      = np.ones((3,3), dtype=np.uint8)

p      = patch.astype(np.float32)
lap_11 = int(p[0,1] + p[1,0] - 4*p[1,1] + p[1,2] + p[2,1])
real_11 = int(np.clip(p[1,1] - lap_11, 0, 255))

print("Patch originale:")
print(mm.drawImg(patch))
print(f"Laplac. w4 in [1,1]: ", end="")
print(f"{p[0,1]:.0f} + {p[1,0]:.0f} - 4×{p[1,1]:.0f} + {p[1,2]:.0f} + {p[2,1]:.0f} =
{lap_11}")
print(f"Pixel evidenziato [1,1]: {p[1,1]:.0f} - ({lap_11}) = {real_11}")

mm.drawImgKernel(patch, B, x=1, y=1, scale=40.0)
```

```
Patch originale:
 91  95 108 116 114
106 107 108 111 114
102 103 107 112 116
 83  90 111 125 123
 79  88 110 126 124

Laplac. w4 in [1,1]: 95 + 106 - 4×107 + 108 + 103 = -16
Pixel evidenziato [1,1]: 107 - (-16) = 123
```

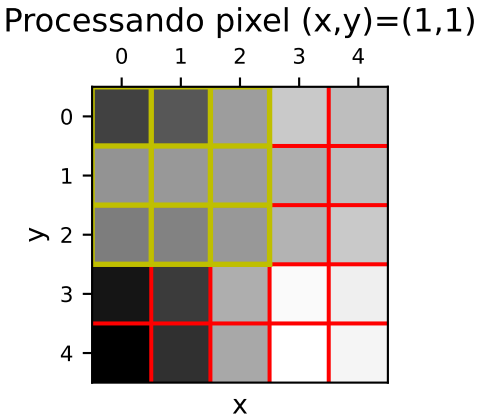


Figura 3.16: *Kernel* laplaciano w4 applicato al *patch* 5×5: la finestra gialla evidenzia l'intorno 3×3 dove viene calcolato l'operatore di derivata seconda.

Figura 3.17 confronta l'applicazione dei *kernel* laplaciani w_4 e w_8 su un ritaglio più ampio dell'immagine del leopardo. Per ciascun caso, vengono mostrate la risposta grezza dell'operatore, che evidenzia i bordi e le transizioni di intensità, e l'immagine ottenuta dopo l'enfatizzazione mediante sottrazione del laplaciano. Si osserva che il *kernel* w_8, considerando anche i vicini diagonali, produce una risposta più intensa e rileva variazioni in più direzioni, risultando in un'enfatizzazione lievemente più marcata.

```
w4 = np.array([[0, 1, 0],
               [1,-4, 1],
               [0, 1, 0]], dtype=np.float32)
w8 = np.array([[1, 1, 1],
               [1,-8, 1],
               [1, 1, 1]], dtype=np.float32)

mm.show(
    [img_gray_crop, mm.laplacian_viz(img_gray_crop, w4), mm.laplacian(img_gray_crop, w4),
     img_gray_crop, mm.laplacian_viz(img_gray_crop, w8), mm.laplacian(img_gray_crop, w8)],
    titles=["Originale", "Laplaciano w4", "Evidenziazione w4",
            "Originale", "Laplaciano w8", "Evidenziazione w8"],
    cols=3, figsize=(14, 8)
)
```

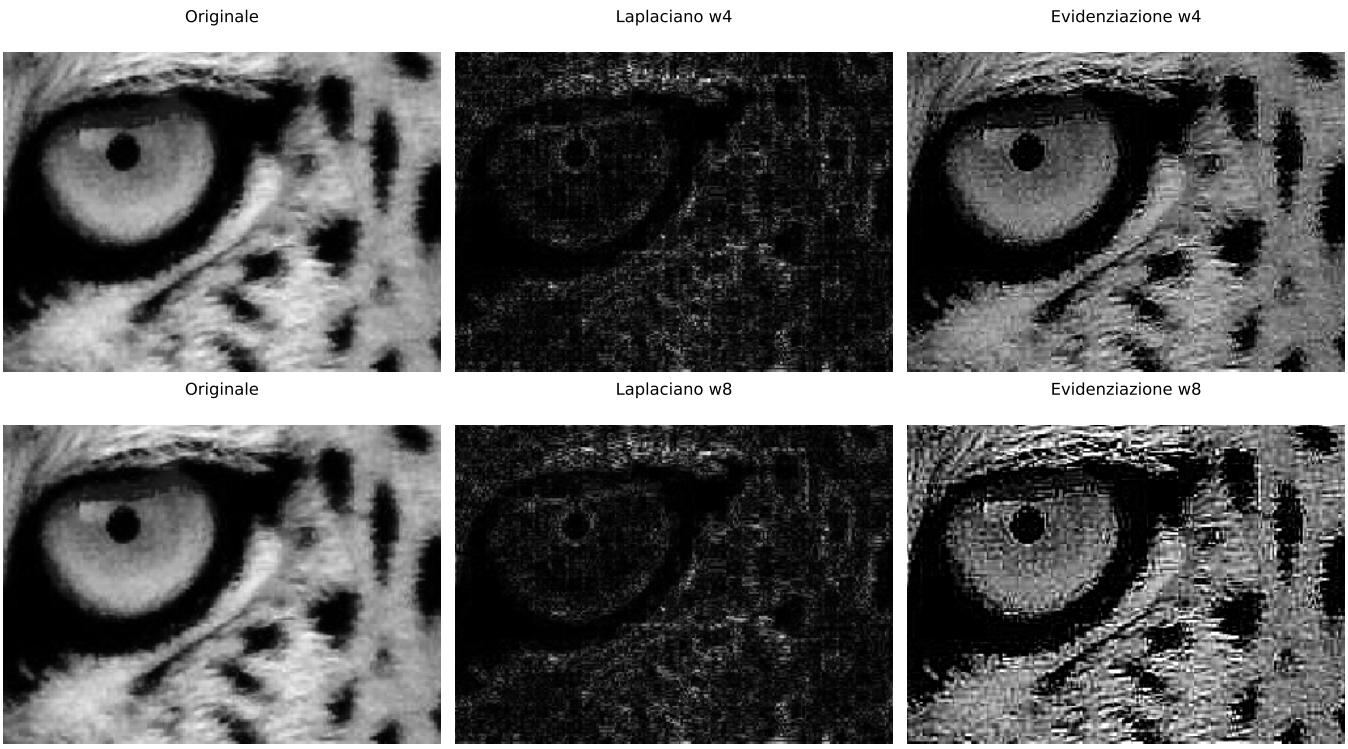


Figura 3.17: Laplaciano applicato all'immagine del leopardo: risposta grezza (bordi) con w4 e w8, e immagini evidenziate tramite sottrazione del Laplaciano. w8 è più sensibile alle diagonali.

3.6.2 Operatore di Sobel

L'operatore di Sobel stima le **derivate parziali del primo ordine** nelle direzioni orizzontale e verticale. A differenza del Laplaciano (derivata seconda), il Sobel è direzionale e più robusto al rumore, poiché ogni *kernel* combina una derivata con un smoothing gaussiano perpendicolare:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * f, \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * f \tag{3.19}$$

G_x rileva i bordi **verticali** (variazione nella direzione x); G_y rileva i bordi **orizzontali** (variazione nella direzione y). I pesi $\{1, 2, 1\}$ nella direzione perpendicolare corrispondono allo smoothing gaussiano 1D, che riduce la sensibilità al rumore.

> **Sobel è correlazione, non convoluzione**
>
> I *kernel* di Sobel sono **asimmetrici** — la rotazione di 180° modifica il risultato. `cv2.Sobel` implementa la correlazione incrociata (come `cv2.filter2D`). Per ottenere la derivata direzionale corretta, i segni sono già definiti per la correlazione: G_x restituisce valori positivi dove l'intensità cresce da sinistra a destra.

La magnitudine del **gradiente** combina le due componenti, rappresentando la forza del bordo indipendentemente dalla direzione:

$$|\nabla f| = \sqrt{G_x^2 + G_y^2} \tag{3.20}$$

E la **direzione** del gradiente (perpendicolare al bordo) è:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \tag{3.21}$$

Per illustrare numericamente, si calcolano G_x e G_y manualmente sul pixel centrale $[1,1]$ del *patch* 5×5:

```
patch = img_gray[250:255, 250:255]
p     = patch.astype(np.float32)

# Gx nel pixel [1,1]: colonna destra - colonna sinistra (ponderata)
gx = (p[0,2] + 2*p[1,2] + p[2,2]) - (p[0,0] + 2*p[1,0] + p[2,0])

# Gy nel pixel [1,1]: riga inferiore - riga superiore (ponderata)
gy = (p[2,0] + 2*p[2,1] + p[2,2]) - (p[0,0] + 2*p[0,1] + p[0,2])

mag   = np.sqrt(gx**2 + gy**2)
theta = np.degrees(np.arctan2(gy, gx))

print("Patch 5x5:")
print(mm.drawImg(patch))
print(f"Gx in [1,1]: ({p[0,2]:.0f} + 2*{p[1,2]:.0f} + {p[2,2]:.0f}) \
      - ({p[0,0]:.0f} + 2*{p[1,0]:.0f} + {p[2,0]:.0f}) = {gx:.1f}")
print(f"Gy in [1,1]: ({p[2,0]:.0f} + 2*{p[2,1]:.0f} + {p[2,2]:.0f}) \
      - ({p[0,0]:.0f} + 2*{p[0,1]:.0f} + {p[0,2]:.0f}) = {gy:.1f}")

# Sostituito il | con il prefisso mag per non rompere l'interprete di tabelle di Quarto
print(f"mag(Grad f) = sqrt({gx:.1f}^2 + {gy:.1f}^2) = {mag:.1f}")
print(f"Theta        = arctan({gy:.1f}/{gx:.1f}) = {theta:.1f} gradi")
```

```
Patch 5x5:
 91  95 108 116 114
106 107 108 111 114
102 103 107 112 116
 83  90 111 125 123
 79  88 110 126 124

Gx in [1,1]: (108 + 2*108 + 107)       - (91 + 2*106 + 102) = 26.0
Gy in [1,1]: (102 + 2*103 + 107)       - (91 + 2*95 + 108) = 26.0
mag(Grad f) = sqrt(26.0^2 + 26.0^2) = 36.8
Theta        = arctan(26.0/26.0) = 45.0 gradi
```

Il valore ridotto di $|\nabla f|$ in questo *patch* conferma che la regione è quasi uniforme, poiché il gradiente assume valori elevati solo dove vi sono cambiamenti significativi di intensità. La Figura 3.18 applica l'operatore di Sobel a un ritaglio più ampio dell'immagine del leopardo. Vengono presentate le risposte orizzontale (G_x) e verticale (G_y), ottenute per convoluzione con i rispettivi *kernel* di Sobel, oltre alla magnitudine $|\nabla f|$, calcolata dalla combinazione di entrambe. Mentre G_x evidenzia i bordi verticali e G_y quelli orizzontali, la magnitudine mette in risalto i bordi in qualsiasi direzione.

```
def norm255(img):
    mn, mx = img.min(), img.max() #+1e-9 per evitare la divisione per zero
    return ((img - mn) / (mx - mn + 1e-9) * 255).astype(np.uint8)

sobelx    = cv2.Sobel(img_gray_crop, cv2.CV_64F, 1, 0, ksize=3)
sobely    = cv2.Sobel(img_gray_crop, cv2.CV_64F, 0, 1, ksize=3)
magnitude = np.sqrt(sobelx**2 + sobely**2)
direcao   = np.degrees(np.arctan2(sobely, sobelx))

# magnitude = mm.sobel(img_gray_crop) # oppure, ma con clip invece di norm255

mm.show(
```

```
    [img_gray_crop, norm255(np.abs(sobelx)), norm255(np.abs(sobely)),
     norm255(magnitude), norm255(direcao % 180)],
    titles=["Originale", "Gx (bordi vert.)", "Gy (bordi orizz.)",
            "Magnitudine |f|", "Direzione  (mod 180°)"],
    cols=5
)
```

Figura 3.18: Operatore di Sobel sull'immagine del leopardo: Gx rileva i bordi verticali, Gy rileva i bordi orizzontali, e la magnitudine | f| combina entrambi, rivelando tutti i bordi indipendentemente dalla direzione.

3.6.3 Operatore di Prewitt

L'operatore di Prewitt è strutturalmente identico a Sobel, ma sostituisce la ponderazione gaussiana $\{1, 2, 1\}$ con pesi uniformi $\{1, 1, 1\}$:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * f, \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * f \tag{3.22}$$

La magnitudine e la direzione del gradiente seguono le stesse equazioni di Sobel (Equazione 3.20 e Equazione 3.21). La differenza pratica è che Prewitt è leggermente più sensibile al rumore — l'appianamento perpendicolare uniforme pondera meno il pixel centrale della linea — ma è computazionalmente più semplice. In immagini a basso rumore i risultati sono equivalenti.

```
Kx = np.array([[-1,0,1],[-1,0,1],[-1,0,1]], dtype=np.float32)
Ky = np.array([[-1,-1,-1],[0,0,0],[1,1,1]], dtype=np.float32)
f  = img_gray_crop.astype(np.float32)

mm.show(
    [img_gray_crop,
     norm255(np.abs(cv2.filter2D(f, -1, Kx))),
     norm255(np.abs(cv2.filter2D(f, -1, Ky))),
     mm.prewitt(img_gray_crop)],
    titles=["Originale", "Gx", "Gy", "Magnitudine | f|"],
    cols=4
)
```

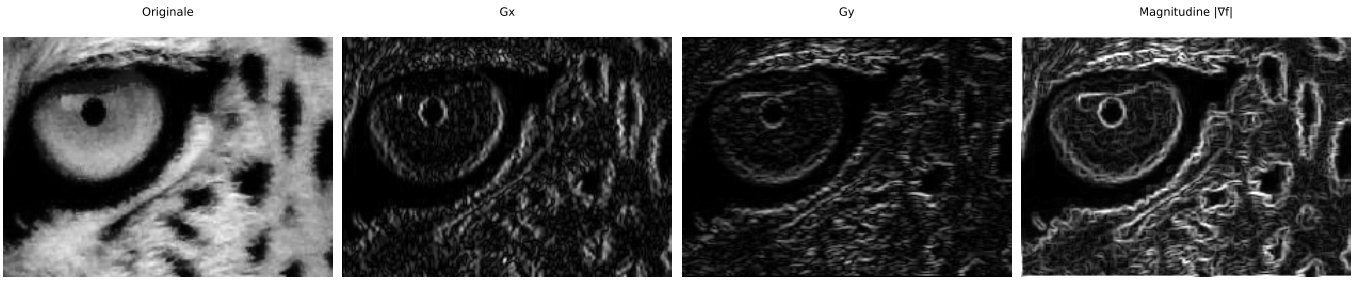


Figura 3.19: Operatore di Prewitt: Gx, Gy e magnitudine — paragonabile a Sobel, ma senza ponderazione Gaussiana perpendicolare.

3.6.4 *Unsharp Masking* (USM)

L'*Unsharp Masking* è una tecnica classica di miglioramento della nitidezza, originaria della fotografia analogica, oggi ampiamente utilizzata nei software di editing delle immagini. L'idea centrale è estrarre le **componenti ad alta frequenza** dell'immagine (bordi e dettagli) e sommarle nuovamente all'originale con un peso k:

Tabella 3.3: Fasi dell'*Unsharp Masking*.

Fase	Operazione	Descrizione
1	$\bar{f} = f * G_\sigma$	Smussa con una gaussiana — mantiene le basse frequenze
2	$m = f - \bar{f}$	Maschera: differenza = alte frequenze (bordi)
3	$g = f + k \cdot m$	Somma ponderata della maschera all'originale

Sostituendo la fase 2 nella fase 3, si ottiene l'espressione compatta:

$$g = f + k\,(f - f * G_\sigma) = (1 + k)\,f - k\,(f * G_\sigma) \tag{3.23}$$

Il parametro k controlla l'intensità del miglioramento:

- $k = 0$: nessun miglioramento ($g = f$);
- $k = 1$: USM classico — raddoppia il contributo delle alte frequenze;
- $k > 1$: *High Boost Filtering* — amplificazione oltre il doppio, utile per immagini molto sfocate.

Amplificazione del rumore

L'USM non distingue i bordi dal rumore — entrambi sono componenti ad alta frequenza. Per valori elevati di k, il rumore presente nell'immagine viene amplificato insieme ai bordi. Per questo motivo, è consigliabile applicare una leggera smussatura prima dell'USM su immagini rumorose, oppure utilizzare un σ piccolo nella gaussiana.

Per illustrare le fasi dell'USM, la Figura 3.20 applica il metodo a una *patch* 30×30 dell'immagine del leopardo, utilizzando $\sigma = 1$ e $k = 1$. Inizialmente, l'immagine viene smussata da un filtro gaussiano. Successivamente, la maschera ad alta frequenza si ottiene dalla differenza tra l'immagine originale e quella smussata. Infine, tale maschera viene sommata all'immagine originale, rafforzando bordi e dettagli. La figura presenta le tre fasi del processo e il risultato finale dell'enhancement.

```
patch  = img_gray_crop[35:65, 45:75]
k = 1.0
sigma = 1.0

p = patch.astype(np.float32)

# Fase 1: sfocatura gaussiana
ksize   = int(6 * sigma + 1) | 1  # l'operatore binario garantisce dimensione dispari
p_suave = mm.gaussian(patch, ksize, sigma).astype(np.float32)

# Fase 2: maschera ad alta frequenza
mascara = p - p_suave

# Fase 3: rafforzamento
p_usm = np.clip(p + k * mascara, 0, 255).astype(np.uint8)

# p_usm = mm.usm(patch, k)  # oppure, diretta
```

```
print(f"Pixel centrale [1,1]: originale={int(p[1,1])} sfocato={p_suave[1,1]:.1f}"
      f" maschera={mascara[1,1]:.1f} rafforzato={p_usm[1,1]}")

mm.show(
    [patch,
     p_suave.astype(np.uint8),
     np.clip(mascara + 128, 0, 255).astype(np.uint8),  # negativi visibili
     p_usm],
    titles=["Patch originale",
            f"Sfocato ( ={sigma})",
            "Maschera (alta freq., +128)",
            f"Rafforzato USM (k={k})"],
    cols=4, figsize=(12, 4)
)
```

```
Pixel centrale [1,1]: originale=16 sfocato=20.0 maschera=-4.0 rafforzato=12
```

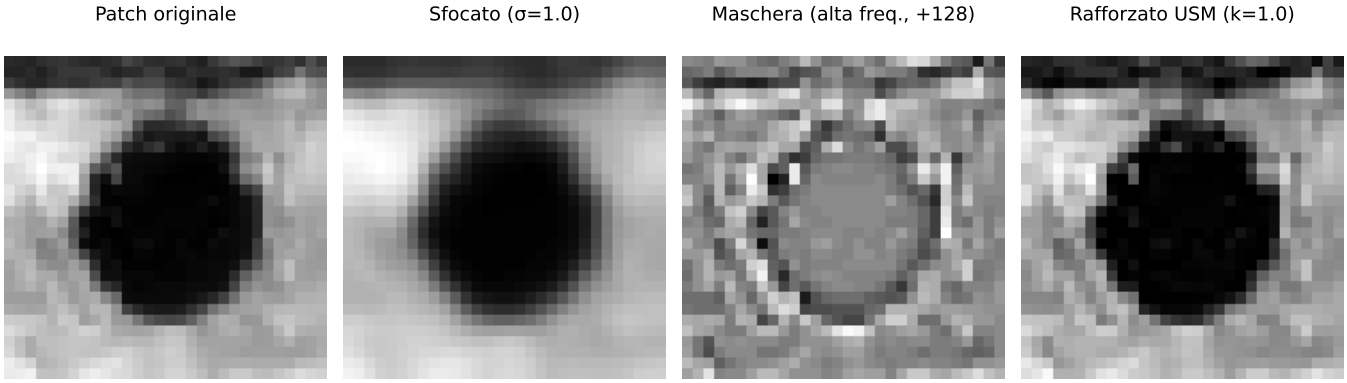


Figura 3.20: Rafforzamento della nitidezza tramite *Unsharp Masking* nell'immagine del leopardo: l'immagine sfocata viene sottratta da quella originale per generare la maschera ad alta frequenza, che viene poi reintrodotta per enfatizzare bordi e dettagli.

La Figura 3.21 applica il metodo USM a un ritaglio più ampio dell'immagine del leopardo utilizzando $\sigma = 1$ e diversi valori del fattore di guadagno k. In tutti i casi, la maschera ad alta frequenza è ottenuta dalla differenza tra l'immagine originale e la sua versione smussata tramite filtro gaussiano. Il parametro k controlla l'intensità del miglioramento: valori più piccoli producono un aumento sottile della nitidezza, mentre valori più grandi rafforzano progressivamente bordi e dettagli. Si osserva che, per valori elevati di k, compaiono aloni attorno ai bordi e il rumore presente nell'immagine viene amplificato.

```
mm.show(
    [img_gray_crop,
     mm.usm(img_gray_crop, 0.5), mm.usm(img_gray_crop, 1.0),
     mm.usm(img_gray_crop, 3.0), mm.usm(img_gray_crop, 5.0),
     mm.usm(img_gray_crop, 8.0)],
    titles=["Originale", "USM k=0.5", "USM k=1.0", "USM k=3.0", "USM k=5.0", "USM k=8.0"],
    cols=3
)
```

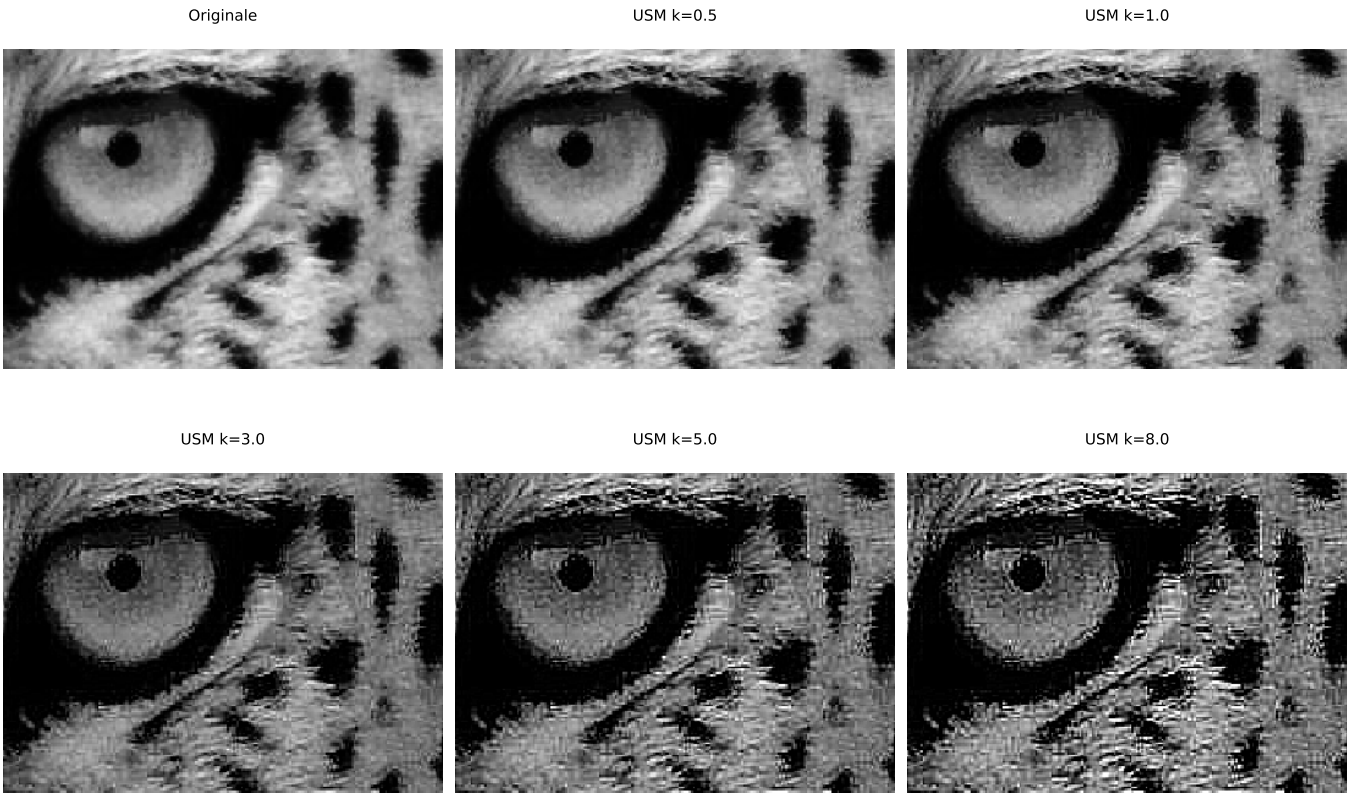


Figura 3.21: *Unsharp Masking* sull'immagine del leopardo con =1 e k da 0.5 a 8.0. Per k>2 compaiono aloni sui bordi e appare il rumore di sfondo.

3.6.5 Rilevatore di Canny

Il Canny combina quattro fasi in sequenza — smoothing gaussiano, gradiente di Sobel, soppressione dei non-massimi e isteresi con doppia soglia — per produrre bordi **sottili, binari e connessi**. A differenza di Sobel e Prewitt, il risultato non è una mappa di gradiente continua, ma una maschera in cui ogni pixel è bordi o non lo è.

Il parametro centrale è la coppia di soglie (T_{low}, T_{high}). I pixel con gradiente superiore a T_{high} sono bordi certi; al di sotto di T_{low}, vengono scartati. I pixel ambigui — tra le due soglie — vengono decisi mediante **isteresi**: diventano bordi se sono connessi a un bordo certo, e vengono scartati in caso contrario. Questo evita sia la perdita di tratti deboli di bordi reali, sia l'inclusione di rumore isolato. Un'euristica comune è $T_{high} = 3 \times T_{low}$.

```
mm.show(
    [img_gray_crop,
     mm.canny(img_gray_crop, 30,  90),
     mm.canny(img_gray_crop, 60,  180),
     mm.canny(img_gray_crop, 120, 240)],
    titles=["Originale", "Canny (30/90)", "Canny (60/180)", "Canny (120/240)"],
    cols=4
)
```

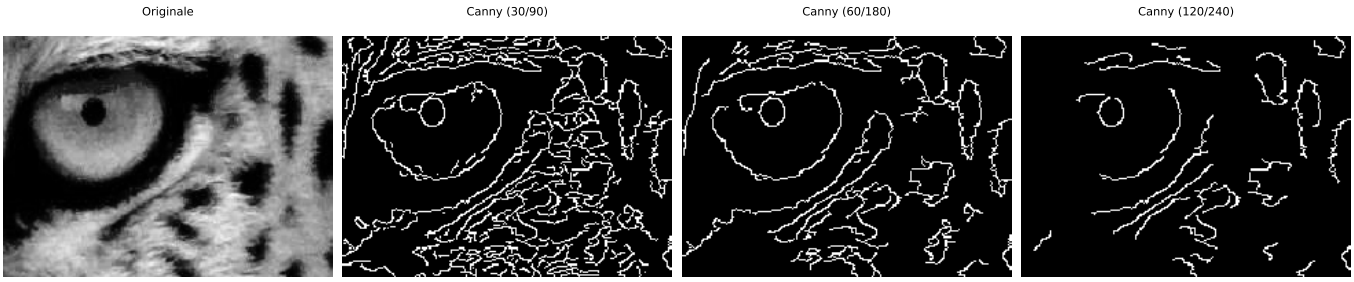


Figura 3.22: Rivelatore di Canny con diverse coppie di soglia: soglie basse catturano più bordi (incluso il rumore); soglie alte mantengono solo i bordi più forti.

Scelta delle soglie

Un'euristica comune è $T_{high} = 3 \times T_{low}$. I valori tipici dipendono dall'intervallo di gradiente dell'immagine — `cv2.Canny` accetta valori assoluti in $[0, 255]$. Per immagini con contrasto variabile, calcolare le soglie a partire dai percentili della magnitudine di Sobel è più robusto rispetto a valori fissi.

3.7 Filtri d'Ordine: Filtro Mediano

I filtri d'ordine (*order-statistic filters*) sostituiscono il pixel centrale con il valore di un **percentile** della distribuzione delle intensità del vicinato — a differenza dei filtri lineari, che calcolano combinazioni ponderate. Il più importante è il **filtro mediano**.

3.7.1 Rumore Impulsivo: Sale e Pepe

Il rumore **sale e pepe** (*salt-and-pepper noise*) sostituisce pixel casuali con valori estremi: 0 (pepe, nero) o 255 (sale, bianco). È comune nella trasmissione di immagini con errori di bit e in fotocamere con sensori difettosi.

Per capire perché i filtri lineari falliscono, si consideri un vicinato 3×3 in cui un singolo pixel è stato corrotto a 255:

$$\text{vizinhança} = \begin{bmatrix} 102 & 98 & 105 \\ 100 & \mathbf{255} & 97 \\ 103 & 99 & 101 \end{bmatrix}$$

Tabella 3.4: Media vs. mediana con un pixel corrotto. La mediana ignora il valore anomalo; la media è spostata di ~40 livelli.

Metodo	Calcolo	Risultato
Media	(102+98+…+255+…+101)/9	140
Mediana	{97,98,99,100,**101**,102,103,105,255}	101

Perché i filtri di media falliscono con il rumore impulsivo?

La media è sensibile ai ***valori anomali*** — un singolo pixel con valore 255 in un vicinato di valore 100 innalza l'uscita a 140, diffondendo il rumore nell'immagine. La mediana, essendo uno **stimatore robusto**, seleziona il valore centrale della distribuzione ordinata, scartando naturalmente gli estremi senza alcun aggiustamento speciale.

L'esempio seguente illustra il comportamento della media e della mediana in presenza di un pixel corrotto da rumore impulsivo. Si osserva che la media è fortemente influenzata dal valore estremo (255), producendo una stima lontana dai valori predominanti del vicinato. La mediana, invece, rimane vicina al valore originale della regione, evidenziando la sua maggiore robustezza ai *valori anomali* e giustificando il suo utilizzo nella rimozione del rumore sale e pepe.

```
# Esempio numerico: media vs. mediana con pixel corrotto
vizinhanca = np.array([102, 98, 105, 100, 255, 97, 103, 99, 101])
print(f"Vicinato: {sorted([int(x) for x in vizinhanca])}")
print(f"Media:      {vizinhanca.mean():.1f}  (spostata dall'outlier 255)")
print(f"Mediana:    {int(np.median(vizinhanca))}    (ignora l'outlier)")
print(f"Valore originale:  ~100")
```

```
Vicinato: [97, 98, 99, 100, 101, 102, 103, 105, 255]
Media:      117.8  (spostata dall'outlier 255)
Mediana:    101    (ignora l'outlier)
Valore originale:  ~100
```

La Figura 3.23 presenta l'effetto del rumore sale e pepe a diverse densità. Il rumore è stato generato sostituendo casualmente una frazione dei pixel con valori minimi (0, pepe) e massimi (255, sale). All'aumentare della densità dal 2% al 10%, cresce la quantità di pixel corrotti, rendendo la degradazione visiva più evidente e ostacolando la percezione dei dettagli dell'immagine.

```
def add_salt_pepper(img, prob=0.05, seed=42):
    """Aggiunge rumore sale e pepe con probabilità totale prob."""
    noisy = img.copy()
    rnd   = np.random.default_rng(seed).random(img.shape)
    noisy[rnd < prob / 2]      = 0    # pepe
    noisy[rnd > 1 - prob / 2] = 255  # sale
    n_corrompidos = int((rnd < prob/2).sum() + (rnd > 1-prob/2).sum())
    return noisy, n_corrompidos

probs = [0.02, 0.05, 0.10]
imgs_noise, titles_noise = [img_gray_crop], ["Original"]

for p in probs:
    noisy, n = add_salt_pepper(img_gray_crop, p)
    imgs_noise.append(noisy)
    titles_noise.append(f"Ruído {int(p*100)}%\n({n:,} pixels)")

mm.show(imgs_noise, titles=titles_noise, cols=4)
```

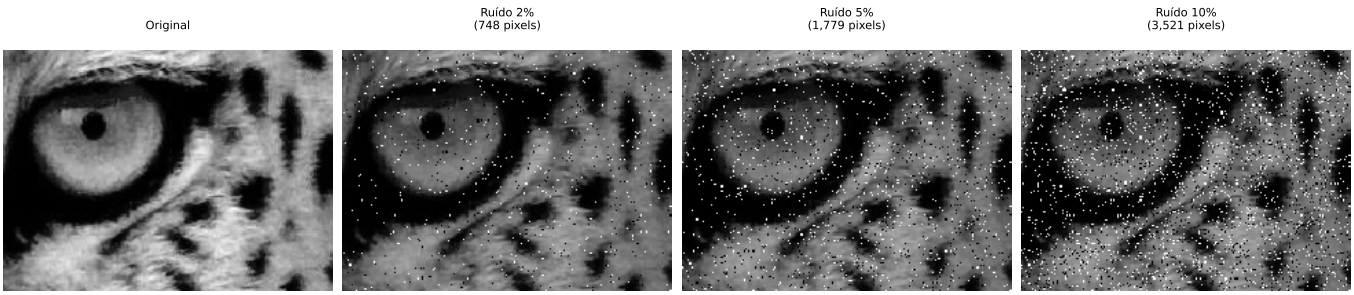


Figura 3.23: Rumore sale e pepe con densità crescenti (2%, 5%, 10%). Il parametro prob indica la frazione totale di pixel corrotti, metà sale (255) e metà pepe (0).

3.7.2 Filtro della Mediana

Il filtro della mediana sostituisce ogni pixel con il **valore mediano** dei pixel del suo intorno $n \times n$:

$$g(x, y) = \text{med}_{(s,t)\in\mathcal{V}_n}\{f(x + s, y + t)\} \tag{3.24}$$

Il valore mediano è quello che occupa la posizione centrale quando gli n^2 valori dell'intorno vengono ordinati. Per una finestra 3×3 ($n^2 = 9$ pixel), la mediana è il 5° valore della sequenza ordinata.

Per illustrare, si consideri lo stesso *patch* 5×5 con un pixel corrotto artificialmente in $[1, 1]$:

```
patch    = img_gray[250:255, 250:255].copy()
corrompido = patch.copy()
corrompido[1, 1] = 255   # injeta pixel sal no centro

# Vizinhança 3×3 centrada em [1,1]
viz_orig = sorted(patch[0:3, 0:3].ravel().tolist())
viz_corr = sorted(corrompido[0:3, 0:3].ravel().tolist())

print("Patch originale:")
print(mm.drawImg(patch))
print("\nPatch con pixel corrotto [1,1]=255:")
print(mm.drawImg(corrompido))
print(f"\nVicinato 3×3 originale (ordinato):   {viz_orig}")
print(f"Mediana originale:                       {viz_orig[4]}")
print(f"\nVicinato 3×3 corrotto (ordinato): {viz_corr}")
print(f"Mediana corrotta:                    {viz_corr[4]}  ← ignora il 255")
print(f"Media corrotta:                  {np.mean(viz_corr):.1f}  ← distorta dal 255")
```

```
Patch originale:
 91  95 108 116 114
106 107 108 111 114
102 103 107 112 116
 83  90 111 125 123
 79  88 110 126 124


Patch con pixel corrotto [1,1]=255:
 91  95 108 116 114
106 255 108 111 114
102 103 107 112 116
 83  90 111 125 123
 79  88 110 126 124


Vicinato 3×3 originale (ordinato):   [91, 95, 102, 103, 106, 107, 107, 108, 108]
Mediana originale:                       106

Vicinato 3×3 corrotto (ordinato): [91, 95, 102, 103, 106, 107, 108, 108, 255]
Mediana corrotta:                     106  ← ignora il 255
Media corrotta:                  119.4  ← distorta dal 255
```

L'esempio conferma: anche con il pixel corrotto a 255, la mediana restituisce il valore centrale corretto — il *valore anomalo* occupa l'ultima posizione nell'ordinamento e viene scartato naturalmente.

Poiché si basa sull'ordinamento e non sulla somma, la mediana possiede tre proprietà fondamentali che la differenziano dai filtri lineari:

- **Robusta** al rumore impulsivo — i valori anomali finiscono alle estremità della sequenza ordinata e non influenzano il valore centrale;
- **Preservatrice dei bordi** — le transizioni brusche di intensità vengono mantenute, poiché la mediana seleziona un valore che esiste già nell'intorno, senza creare nuovi livelli intermedi;
- **Non lineare** — non può essere espressa come convoluzione, quindi `mm.conv` non si applica; si usa `cv2.medianBlur`.

Figura 3.24 confronta diverse tecniche di rimozione del rumore sale e pepe applicate a un'immagine con il 10% dei pixel corrotti. Sono stati valutati i filtri Gaussiano, Media, Mediana, Bilaterale e Morfologico (prossimo capitolo), consentendo di osservare il compromesso tra rimozione del rumore e preservazione dei dettagli. In generale, i filtri di media e Gaussiano riducono il rumore, ma tendono a sfocare i bordi, mentre la mediana presenta prestazioni migliori per il rumore impulsivo. Il filtro bilaterale preserva meglio i bordi, e il filtro morfologico rimuove gran parte dei pixel corrotti senza degradare eccessivamente la struttura dell'immagine.

```
# 10% di rumore per evidenziare le differenze tra i filtri
noisy_5, _ = add_salt_pepper(img_gray_crop, prob=0.1)

#   Filtri
f_gauss   = cv2.GaussianBlur(noisy_5, (5, 5), 1.0)
f_media   = mm.conv(noisy_5, np.ones((5, 5), dtype=np.float32) / 20.0)
f_median3 = cv2.medianBlur(noisy_5, 3)
f_median5 = cv2.medianBlur(noisy_5, 5)
f_bilat   = cv2.bilateralFilter(noisy_5, d=9, sigmaColor=75, sigmaSpace=75)

# filtri morfologici nel prossimo capitolo, ma già anticipati qui per confronto
# B         = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
# f_morf    = cv2.morphologyEx(           # presentato nel prossimo capitolo
#                 cv2.morphologyEx(noisy_5, cv2.MORPH_OPEN,  B),
#                 cv2.MORPH_CLOSE, B)
B = mm.secross()  # elemento strutturante di scatola 3×3
f_morf     = mm.asf(noisy_5, 'OC', B)  # equivalente tramite morph

#   MSE
def mse(a, b):
    return float(np.mean((a.astype(np.float32) - b.astype(np.float32))**2))

print(f"{'Filtro':<22} {'MSE':>8}")
print("-" * 32)
pares = [("Gaussiano 5×5",    f_gauss),
         ("Média 5×5",        f_media),
         ("Mediana 3×3",      f_median3),
         ("Mediana 5×5",      f_median5),
         ("Bilateral",        f_bilat),
         ("Morf. open+close", f_morf)]
for nome, img in pares:
    print(f"{nome:<22} {mse(img_gray_crop, img):>8.2f}")

imgs   = [img_gray_crop, noisy_5,    f_gauss,        f_media,
          f_median3,     f_median5,  f_bilat,        f_morf]
titles = ["Original",    "Ruído 10%", "Gaussiano 5×5","Média 5×5",
          "Mediana 3×3", "Mediana 5×5", "Bilateral", "Morf. open+close"]

mm.show(
    imgs,
    titles=[f"Crop: {t}" for t in titles],
    cols=4, rows=2, figsize=(14, 8)
)
```

```
Filtro                     MSE
--------------------------------
Gaussiano 5×5            260.84
Média 5×5                872.57
Mediana 3×3               31.26
Mediana 5×5               65.86
Bilateral               1001.22
Morf. open+close          65.16
```

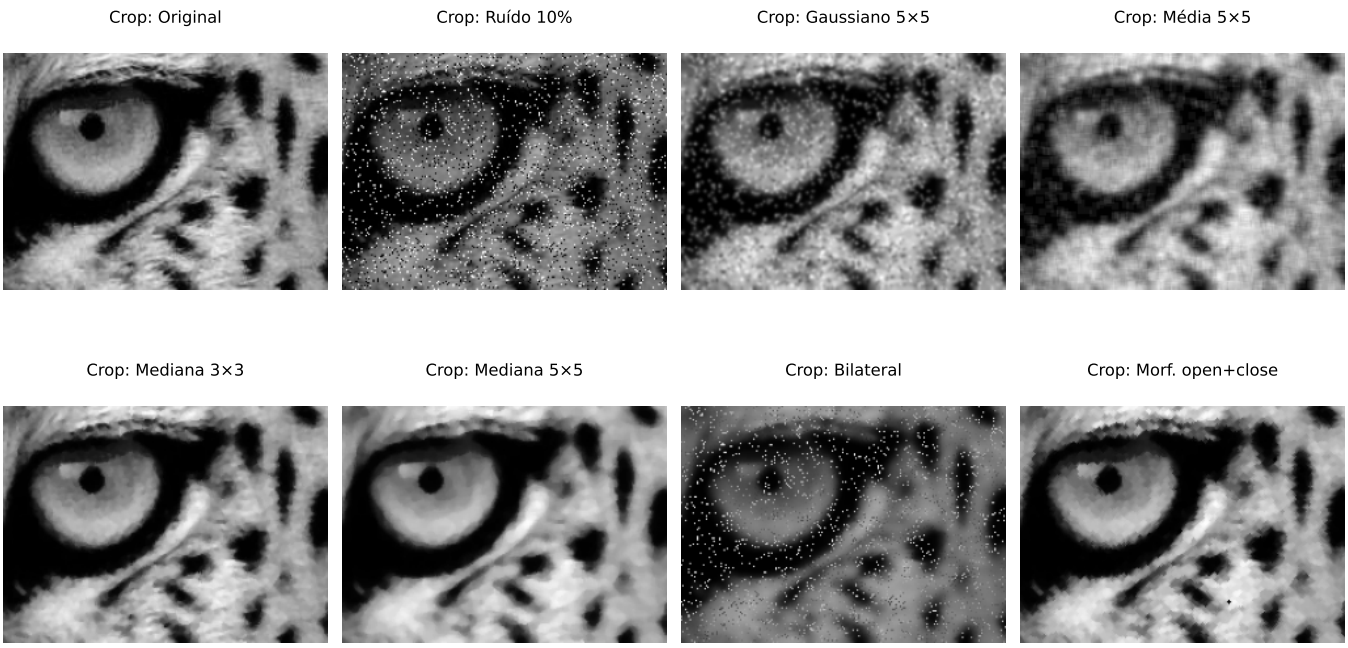


Figura 3.24: Comparazione dei filtri per la rimozione del rumore sale e pepe (10%): Gaussiano, Media, Mediana, Bilaterale e Morfologico. Riga superiore: immagini complete; riga inferiore: crop della regione di interesse.

3.8 Applicazione Pratica: Pre-elaborazione per la Segmentazione

Nella pratica, le tecniche di questo capitolo raramente vengono utilizzate singolarmente. Una ***pipeline* di pre-elaborazione** tipica combina più fasi in sequenza, adattandosi al tipo di immagine e all'applicazione. La Figura 3.25 illustra una *pipeline* completa:

1. **Equalizzazione dell'istogramma (CLAHE):** normalizza il contrasto indipendentemente dalle condizioni di illuminazione;
2. **Filtro Gaussiano:** attenua il rumore di acquisizione senza distruggere i bordi;
3. **Rilevamento dei bordi (Sobel/Canny):** estrae strutture rilevanti per la segmentazione.

> **L'ordine è importante**
>
> L'ordine delle operazioni influisce sul risultato finale. In generale: **(1) normalizzazione dell'intensità → (2) riduzione del rumore → (3) miglioramento/segmentazione**. Invertire l'ordine può amplificare il rumore o perdere i bordi prima di rilevarli.

```
# Fase 1: CLAHE
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
img_clahe = clahe.apply(img_gray_crop)

# Fase 2: Gaussiano
img_gauss = cv2.GaussianBlur(img_clahe, (5, 5), 0)

# Fase 3: Canny
edges = cv2.Canny(img_gauss, 50, 150)

# Confronto: pipeline vs. Canny diretto
edges_direct = cv2.Canny(img_gray_crop, 50, 150)

mm.show(
    [img_gray_crop, img_clahe, img_gauss, edges, edges_direct],
```

```
    titles=["Originale", "1. CLAHE", "2. Gaussiano", "3. Canny (pipeline)", "Canny
(diretto)"],
    cols=5
)
```

Figura 3.25: *Pipeline* di pre-processamento: CLAHE → Gaussiano → Canny. Ogni fase prepara l'immagine per la successiva, ottenendo bordi puliti e ben definiti.

3.9 Riassunto

In questo capitolo sono state presentate le principali tecniche di elaborazione nel dominio spaziale, dalla manipolazione diretta dei pixel fino al filtraggio per vicinato:

- **Operazioni puntuali:** aritmetiche sature (`mm.addm`, `mm.subm`) e logiche bit a bit (`mm.band`, `mm.bor`, `mm.bnot`) per il ritaglio di ROI e la combinazione di immagini; *alpha blending* (`mm.blend`) per la fusione pesata con peso $\alpha \in [0, 1]$.
- **Istogramma:** funzione discreta di distribuzione delle intensità; visualizzato con `mm.histImg` e calcolato con `mm.hist`; base per la diagnosi tonale e per le tecniche di equalizzazione e specificazione.
- **Equalizzazione:** redistribuzione automatica delle intensità tramite la CDF (`mm.equalize`), con variante adattativa CLAHE per il controllo locale del contrasto.
- **Specificazione dell'istogramma:** trasferimento del profilo tonale di un'immagine di riferimento mediante mappatura inversa della CDF — generalizzazione dell'equalizzazione per distribuzioni arbitrarie.
- **Correlazione e convoluzione:** meccanismo di finestra scorrevole implementato in `mm.conv` (`cv2.filter2D`); differenziati dalla rotazione di 180° del *kernel* — rilevante solo per kernel asimmetrici.
- **Filtri di smoothing:** media (*kernel* uniforme, sfuma i bordi proporzionalmente alla dimensione) e Gaussiano (ponderazione radiale, separabile, senza *ringing*, preserva meglio i bordi).
- **Filtri di enhancement:** Laplaciano (w_4/w_8, seconda derivata isotropica), Sobel (gradiente direzionale del primo ordine, con magnitudine $|\nabla f|$ e direzione θ) e *Unsharp Masking* (amplificazione delle alte frequenze con parametro k).
- **Filtro mediano:** non lineare, robusto agli outlier, preserva i bordi — superiore ai filtri lineari per il rumore sale e pepe.
- **Pipeline pratica:** concatenazione CLAHE → Gaussiano → Canny come strategia di pre-elaborazione; `mm.drawImgKernel` per la visualizzazione didattica della finestra scorrevole.

Il Capitolo 4 tratterà la **morfologia matematica** (erosione, dilatazione, apertura e chiusura), esplorando in profondità le funzioni `mm.ero` e `mm.dil` della libreria `morph.py`. Successivamente, il Capitolo 5 presenterà l'**elaborazione nel dominio della frequenza**, con focus sulla Trasformata di Fourier e sulle tecniche di filtraggio spettrale.

3.10 🤖 Uso di Gemini Notebook come Tutor Complementare

In questa edizione, incoraggiamo l'uso di **Gemini Notebook** come strumento complementare di apprendimento. Questo strumento di IA utilizza esclusivamente i documenti forniti dall'autore come base di conoscenza, garantendo risposte coerenti con il contenuto del libro — incluse le funzioni della libreria `morph.py` e gli esperimenti realizzati in questo capitolo.

Per ogni capitolo, abbiamo preparato un progetto specifico sulla piattaforma con il PDF del capitolo, i *notebook* e i materiali ausiliari. Suggeriamo di esplorare in particolare:

- **Guida allo Studio:** riassunto strutturato dei concetti, ideale per il ripasso prima delle verifiche;
- **Conversazione:** chiarisci dubbi su equalizzazione, convoluzione, filtri e pipeline direttamente con il tutor;
- **Domande frequenti:** quesiti tipici sulla differenza tra media e mediana, USM, Laplaciano vs. Sobel.

🎓 Studia con il Tutor Intelligente

Per interagire con il contenuto di questo capitolo, accedi al *link* seguente. L'ambiente contiene materiali didattici in diversi formati, generati dal **PDF** del capitolo. Sulla piattaforma, esplora in particolare le opzioni **Guida allo Studio** e **Conversazione** per approfondire la tua comprensione.
🚀 ACCEDI A Gemini Notebook: CAPITOLO 03

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo su Gemini Notebook è stato costruito solo con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, o seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ **Avviso sui Contenuti Generati dall'IA**

L'IA è una potente alleata nello studio, ma il contenuto generato può contenere **errori o imprecisioni**. Consulta sempre **libri, articoli scientifici e altre fonti accademiche affidabili** per validare le informazioni. Quando possibile, esegui gli esempi pratici forniti in questo capitolo per verificare i risultati.

3.11 Lista di Esercizi

1. **(10%)** Spiegare la differenza tra **convoluzione** e **correlazione incrociata**. Per quali tipi di *kernel* i risultati sono identici? Fornire un esempio di *kernel* asimmetrico (come Sobel G_x) e mostrare numericamente che i risultati differiscono applicandolo al patch 5×5 del capitolo in entrambi i modi.

2. **(15%)** Considerare un'immagine 5×5 con intensità concentrate tra i livelli 3 e 5 (basso contrasto, 3 bit). Applicare manualmente l'algoritmo di equalizzazione della Tabella 3.1, completando tutte le colonne della tabella (k, $h[k]$, $p[k]$, cdf$[k]$, lut$[k]$). Verificare il risultato con `mm.equalize`.

3. **(15%)** Utilizzando `mm.conv`, applicare il filtro media con kernel di dimensione 3×3, 9×9 e 21×21 all'immagine del mandrillo. Per ciascuna versione, calcolare il **PSNR** (*Peak Signal-to-Noise Ratio*) rispetto all'originale:

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{255^2}{\mathrm{MSE}}\right), \quad \mathrm{MSE} = \frac{1}{MN}\sum_{i,j}(f-g)^2$$

   Tracciare il PSNR in funzione della dimensione del kernel e spiegare cosa indica la diminuzione progressiva riguardo alla relazione tra smoothing e perdita di informazione.

4. **(15%)** Utilizzando `add_salt_pepper` con densità del 5%, applicare e confrontare: (a) `mm.conv` con media 3×3, (b) `cv2.GaussianBlur` con $\sigma = 1$, (c) `cv2.medianBlur` con finestra 3×3 e (d) `cv2.medianBlur` con finestra 5×5. Visualizzare le immagini con `mm.show` in una griglia 2×4 (riga 1: immagini, riga 2: istogrammi tramite `mm.histImg`). Spiegare perché la mediana supera i filtri lineari utilizzando l'argomento della Tabella 3.4..

5. **(15%)** Implementare `mm.conv0` utilizzando solo operazioni NumPy vettorizzate — senza cicli Python e senza `cv2.filter2D` — con l'operatore di *stride tricks* (`np.lib.stride_tricks.sliding_window_view`).

Confrontare il risultato e il tempo di esecuzione con `mm.conv0` (cicli) e `mm.conv` (cv2) per kernel 3×3 e 15×15 sull'immagine del mandrillo.

6. **(15%)** Applicare l'*Unsharp Masking* con $\sigma = 1$ e $k \in \{0.5, 1.0, 2.0, 4.0\}$ utilizzando la funzione `usm` del capitolo. Per ciascun valore di k: (a) calcolare la differenza assoluta $|g - f|$, (b) visualizzare le immagini e le differenze con `mm.show`, e (c) tracciare l'istogramma delle differenze con `mm.histImg`. Identificare a partire da quale k gli artefatti (aloni e amplificazione del rumore) diventano visivamente inaccettabili.
7. **(15%)** Scegliere un'immagine radiografica o tomografica disponibile pubblicamente (es.: tramite `mm.read` da URL) e progettare una pipeline di pre-elaborazione con almeno 4 fasi sequenziali, giustificando ogni scelta sulla base dei concetti del capitolo. Visualizzare con `mm.show` in griglia: l'immagine originale, ogni fase intermedia e il risultato finale con i rispettivi istogrammi (`mm.histImg`).

Riferimenti del Capitolo

La base teorica di questo capitolo si fonda sulle seguenti opere:

- Gonzalez (2018) per i concetti di operazioni di intensità, istogramma, convoluzione e filtraggio spaziale.
- Szeliski (2022) per la visione computazionale e le applicazioni pratiche del filtraggio.
- Bradski (2008) per l'implementazione pratica con OpenCV e `morph.py`.

Executar Colab | Abrir GitHub

3.12 💻 Parte pratica con esercizi di programmazione

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EP)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, eseguire `TestSuite("EP03_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare direttamente codice Python, senza salvare file, utilizzare `run_code(codigo)` passando il codice come *stringa* in una variabile `codigo`:

```
codigo = """
from morph import mm
# ... il tuo codice qui ...
"""
TestSuite("EP03_01").run_code(codigo)
```

3.12.1 EP03_01 ➕ Addizione Saturata di una Costante

Nei sistemi di videosorveglianza, le telecamere in ambienti con illuminazione variabile producono immagini sottoesposte. La regolazione della luminosità mediante **addizione saturata di una costante** è l'operazione più semplice per una correzione immediata, applicata in tempo reale sui *chip* delle telecamere integrate e nelle *pipeline* di pre-elaborazione dei robot mobili.

Vedi in Figura 3.26 una simulazione di questo EP.

3.12.1.1 📋 Linee Guida per l'Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Costante:** Leggere l'intero k (valore da sommare).
3. **Dati:** Leggere i valori interi della matrice originale riga per riga.
4. **Mappatura:** Per ogni pixel p, calcolare il nuovo valore tramite l'equazione:

$$p' = \text{clip}(p + k)$$

5. **Output:** Visualizzare la matrice risultante con dimensioni $L \times C$.

3.12.1.2 📌 Vincoli Computazionali

- **Saturazione (*Clipping*):** I valori devono essere confinati nell'intervallo $[0, 255]$:

$$\text{clip}(x) = \max(0, \min(255, x))$$

- **Tipo:** Il risultato finale deve essere intero (senza cifre decimali).
- k **può essere negativo:** i valori negativi scuriscono l'immagine; quelli positivi la schiariscono.

3.12.1.3 🧠 Fondamento Teorico

Parametro	Tipo	Impatto Visivo
$k > 0$	Intero	Schiarisce l'immagine; i pixel vicini a 255 saturano in bianco
$k < 0$	Intero	Scurisce l'immagine; i pixel vicini a 0 saturano in nero
$k = 0$	Intero	Immagine invariata

3.12.1.4 📦 Specifica di Input e Output (VPL)

Input:

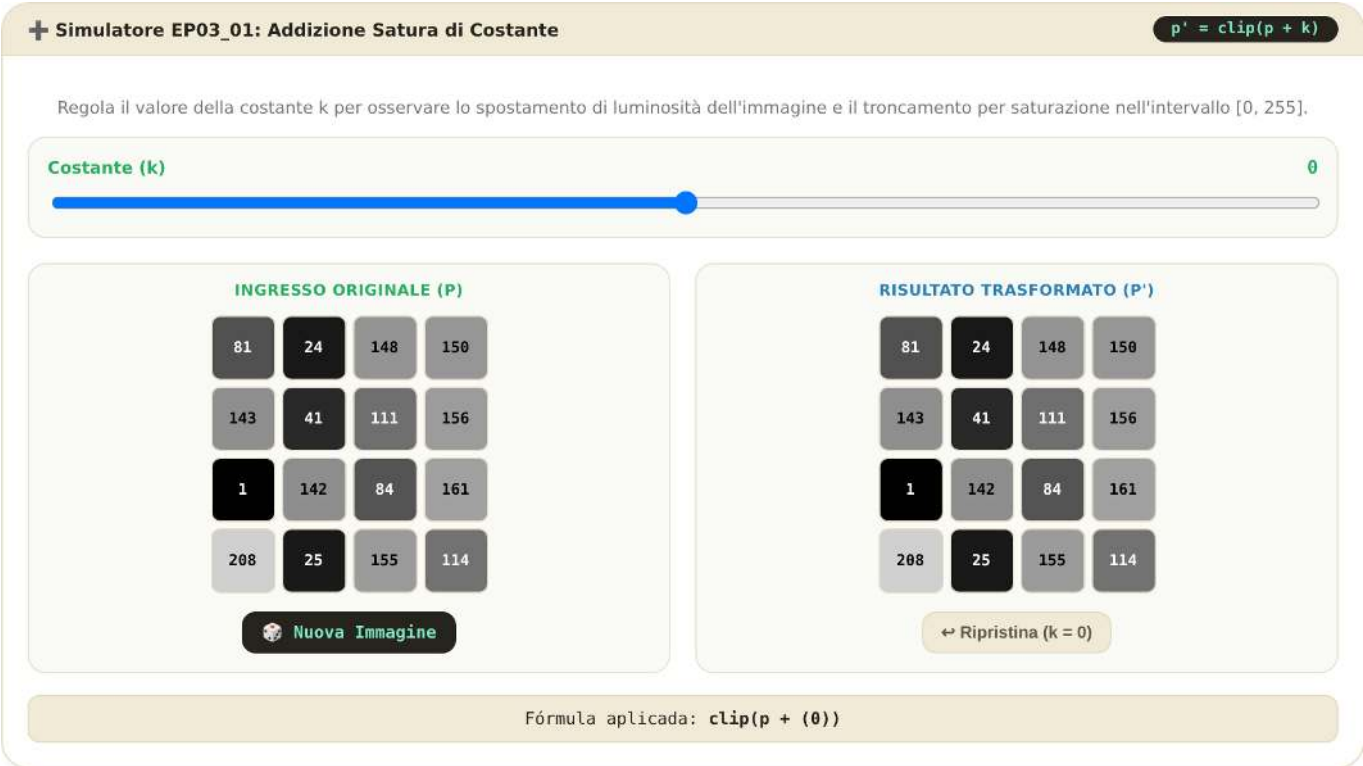
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero k.
- Righe successive: Elementi interi della matrice originale.

Output:

- Matrice trasformata in L righe e C colonne, valori interi separati da spazi.

3.12.1.5 📌 Esempi

Input	Output	Osservazione
2 3 50 0 100 200 210 240 255	50 150 250 255 255 255	Saturazione a 255 nei pixel alti
1 4 -30 0 20 200 255	0 0 170 225	Saturazione a 0 nei pixel bassi



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0301-adicao.html

Figura 3.26: Simulatore EP03_01: Addizione Saturata di Costante (p' = clip(p + k))

```
%%writefile EP03_01.py
# Codice Python
```

```
Overwriting EP03_01.py
```

```
TestSuite("EP03_01.py").run()
```

```
<IPython.core.display.HTML object>
```

3.12.2 EP03_02 🔀 Alpha *Blending* di Due Immagini

In medicina nucleare, immagini di diverse modalità (tomografia computerizzata e risonanza magnetica) vengono fuse per supportare la diagnosi. La **miscela ponderata** (*alpha blending*) è l'operazione fondamentale di questo processo, consentendo al radiologo di controllare interattivamente il peso di ciascuna modalità nell'immagine visualizzata.

Vedi in Figura 3.27 una simulazione di questo EP.

3.12.2.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Parametro:** Leggere il valore reale $\alpha \in [0, 1]$.
3. **Dati:** Leggere i valori interi della matrice f_1 (immagine 1) e successivamente della matrice f_2 (immagine 2).
4. **Mappatura:** Per ogni posizione (i, j), calcolare:

$$g(i, j) = \text{clip}\left(\text{round}\left(\alpha \cdot f_1(i, j) + (1 - \alpha) \cdot f_2(i, j)\right)\right)$$

5. **Uscita:** Visualizzare la matrice risultante $L \times C$.

3.12.2.2 📌 Vincoli Computazionali

- **Arrotondamento:** Applicare `round` prima della conversione a intero.
- **Saturazione:** Confinare all'intervallo $[0, 255]$ con $\text{clip}(x) = \max(0, \min(255, x))$.
- **Operazione in virgola mobile:** Eseguire l'operazione in virgola mobile prima di arrotondare.

3.12.2.3 🧠 Fondamenti Teorici

Valore di α	Risultato
$\alpha = 1.0$	Solo f_1
$\alpha = 0.5$	Media aritmetica di f_1 e f_2
$\alpha = 0.0$	Solo f_2

3.12.2.4 📦 Specifica di Input e Output (VPL)

Input:

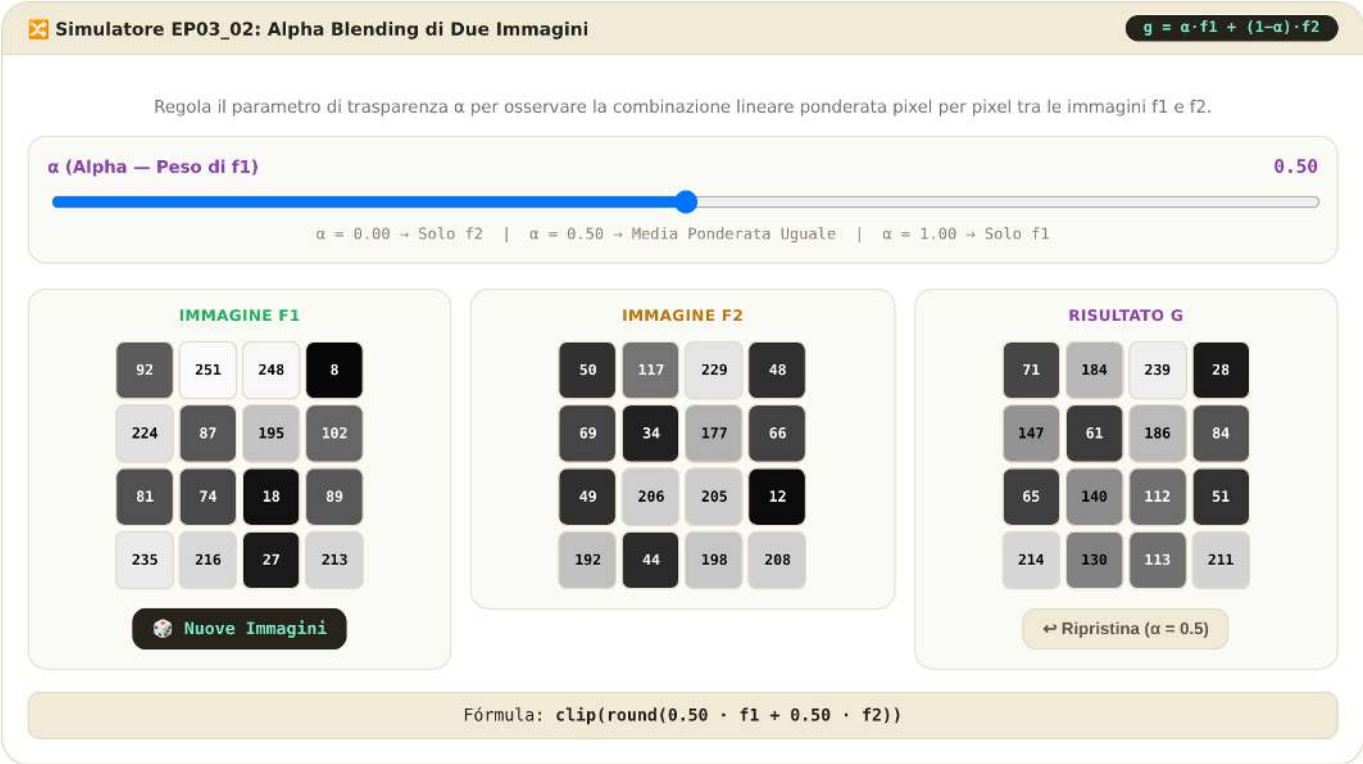
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Reale α.
- Righe successive: Elementi di f_1 (L righe con C valori ciascuna).
- Righe successive: Elementi di f_2 (L righe con C valori ciascuna).

Output:

- Matrice risultante $L \times C$.

3.12.2.5 📌 Esempi

Input	Output	Osservazione
1 3 0.5 0 100 200 100 200 50	50 150 125	Media tra le due immagini
1 3 1.0 10 20 30 90 80 70	10 20 30	Solo f_1 (alpha=1)



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0302-blending.html

Figura 3.27: Simulatore EP03_02: Alpha Blending di Due Immagini (g = · f1 + (1−) · f2)

```
%%writefile EP03_02.py
# Codice Python
```

```
Overwriting EP03_02.py
```

```
TestSuite("EP03_02.py").run()
```

```
<IPython.core.display.HTML object>
```

3.12.3 EP03_03 🎭 Inversione dell'Immagine (Negativo Fotografico)

In radiologia, le immagini ai raggi X sono tradizionalmente visualizzate in negativo: le ossa appaiono in nero su sfondo bianco. L'operazione di **negativo fotografico** viene applicata di routine nei PACS (*Picture Archiving and Communication Systems*) per facilitare la rilevazione di fratture e densità ossee.

Vedere nella Figura 3.28 una simulazione di questo EP.

3.12.3.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere i valori interi della matrice originale.
3. **Mappatura:** Per ogni pixel p, calcolare il negativo:

$$p' = 255 - p$$

4. **Output:** Visualizzare la matrice risultante $L \times C$.

3.12.3.2 📌 Vincoli Computazionali

- **Nessun *clipping* necessario:** Il risultato di $255 - p$ con $p \in [0, 255]$ è sempre $\in [0, 255]$.

- **Tipo intero:** L'output deve essere composto da valori interi.
- **Equivalenza logica:** L'operazione è identica al `NOT` bit a bit (`mm.bnot`) su immagini a 8 bit.

3.12.3.3 🧠 Fondamenti Teorici

Pixel Originale p	Pixel Negativo p'	Osservazione
0 (nero)	255 (bianco)	Inversione totale
128 (grigio medio)	127 (grigio medio)	Valore centrale
255 (bianco)	0 (nero)	Inversione totale

3.12.3.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi interi della matrice originale.

Output:

- Matrice negativa in L righe e C colonne.

3.12.3.5 📌 Esempi

Input	Output	Osservazione
1 4 0 128 200 255	255 127 55 0	Inversione di ogni pixel
2 2 10 20 30 40	245 235 225 215	Matrice 2x2 invertita

```
%%writefile EP03_03.py
# Codice Python
```

```
Overwriting EP03_03.py
```

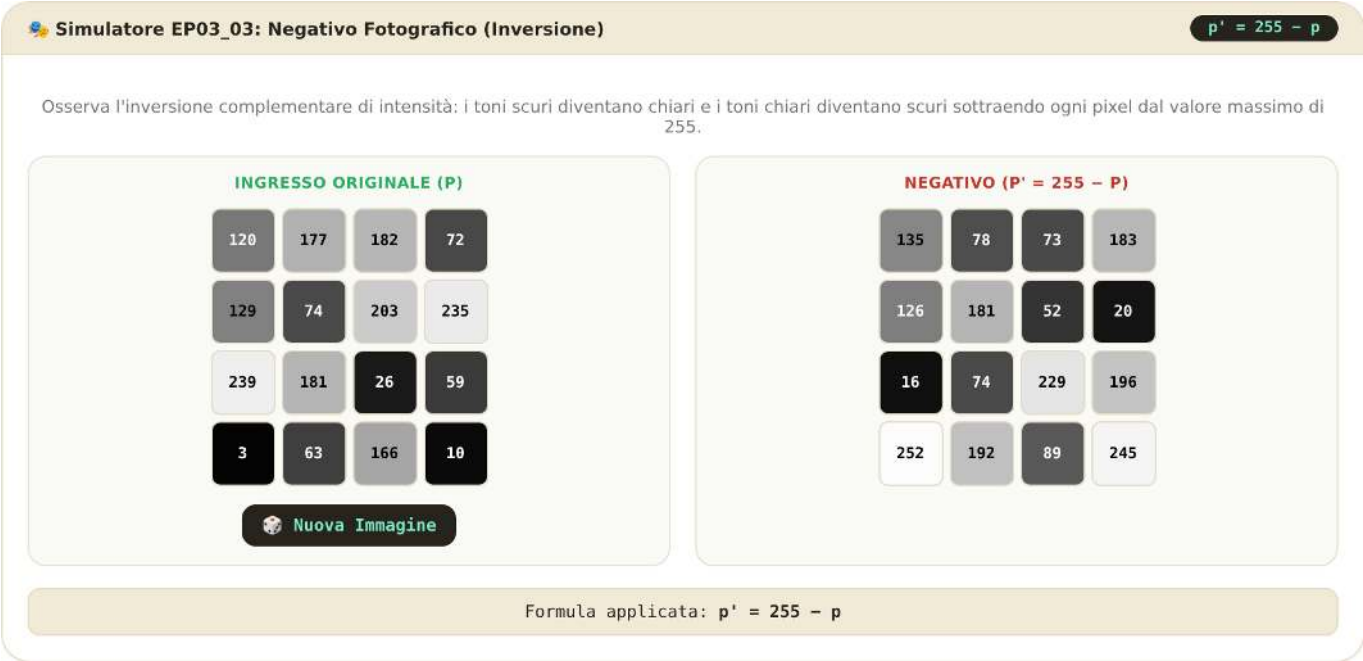
```
TestSuite("EP03_03.py").run()
```

```
<IPython.core.display.HTML object>
```

3.12.4 EP03_04 📊 Equalizzazione dell'istogramma (L bit)

Nelle immagini satellitari di telerilevamento, la variazione dell'illuminazione durante il giorno produce immagini a basso contrasto. L'**equalizzazione dell'istogramma** viene applicata automaticamente su satelliti come il Landsat per redistribuire i toni, rivelando dettagli di vegetazione, rilievo e aree urbane invisibili nell'immagine originale.

Vedere in Figura 3.29 una simulazione di questo EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0303-inversao.html

Figura 3.28: Simulatore EP03_03: Inversione dell'immagine — Negativo fotografico (p' = 255 − p)

3.12.4.1 📋 Linee guida di implementazione

1. **Dimensioni:** Leggere gli interi L (righe), C (colonne) e B (numero di bit, con $L_{\max} = 2^B$).
2. **Dati:** Leggere la matrice di pixel f con valori in $[0, 2^B - 1]$.
3. **Istogramma:** Calcolare $h[k]$ = numero di pixel con intensità k, per $k = 0 \ldots 2^B - 1$.
4. **Probabilità:** $p[k] = h[k]/(L \cdot C)$.
5. **CDF:** $\text{cdf}[k] = \sum_{j=0}^{k} p[j]$; funzione di distribuzione cumulativa.
6. **LUT:** $\text{lut}[k] = \text{round}\,(\text{cdf}[k] \cdot (2^B - 1))$; *Look-Up Table* (tabella di consultazione).
7. **Applicazione:** $g[i, j] = \text{lut}[f[i, j]]$.
8. **Uscita:** Visualizzare la matrice equalizzata $L \times C$.

3.12.4.2 📌 Vincoli computazionali

- **Arrotondamento:** Usare l'arrotondamento matematico (`round`) nella LUT.
- **Bit:** Il numero di livelli è 2^B (es.: $B = 3 \Rightarrow 8$ livelli, $B = 8 \Rightarrow 256$ livelli).
- **CDF cumulativa:** $\text{cdf}[k] = \sum_{j=0}^{k} p[j]$, con $\text{cdf}[2^B - 1] = 1.0$.

3.12.4.3 🧠 Fondamenti teorici

Fase	Operazione	Formula
1	Istogramma	$h[k] \leftarrow$ n. pixel con intensità k
2	Probabilità	$p[k] = h[k]/(L \cdot C)$
3	CDF	$\text{cdf}[k] = \sum_{j=0}^{k} p[j]$
4	LUT	$\text{lut}[k] = \text{round}(\text{cdf}[k] \cdot (2^B - 1))$
5	Applicazione	$g[i, j] = \text{lut}[f[i, j]]$

3.12.4.4 📦 Specifica di input e output (VPL)

Input:

- Riga 1: Intero L.

- Riga 2: Intero C.
- Riga 3: Intero B (numero di bit).
- Righe successive: Elementi interi della matrice.

Output:

- Matrice equalizzata in L righe e C colonne.

3.12.4.5 📌 Esempi

Input	Output	Osservazione
5 5 3 3 4 2 3 4 4 3 3 4 3 2 3 4 3 2 3 4 3 2 3 4 3 2 3 4	5 7 1 5 7 7 5 5 7 5 1 5 7 5 1 5 7 5 1 5 7 5 1 5 7	Esempio a 3 bit del capitolo
1 4 3 0 0 7 7	0 0 7 7	Istogramma bimodale estremo

```
%%writefile EP03_04.py
# Codice Python
```

```
Overwriting EP03_04.py
```

```
TestSuite("EP03_04.py").run()
```

```
<IPython.core.display.HTML object>
```

3.12.5 EP03_05 🔲 Applicazione della Maschera AND Binaria

Nei sistemi di ispezione industriale tramite visione artificiale, è necessario isolare le regioni di interesse (ROI) nelle immagini dei pezzi per verificare difetti di fabbricazione. L'operazione **AND bit a bit con una maschera binaria** è il meccanismo fondamentale per ritagliare esattamente l'area di ispezione, azzerando tutti i pixel al di fuori di essa.

Vedi in Figura 3.30 una simulazione di questo EP.

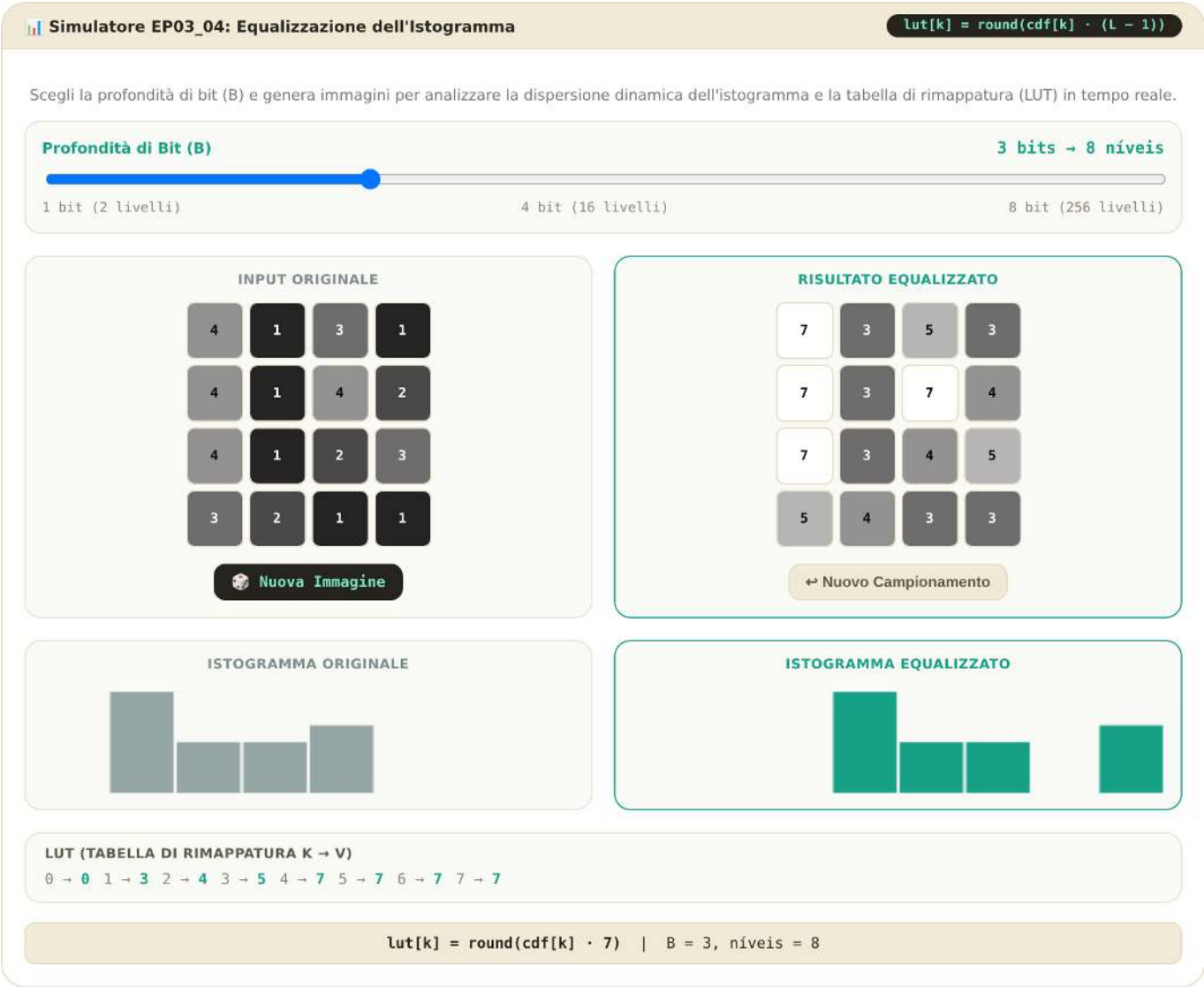
3.12.5.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere la matrice di pixel f (valori $\in [0, 255]$).
3. **Maschera:** Leggere la matrice binaria m (valori: solo 0 o 255).
4. **Mappatura:** Per ogni pixel (i, j), applicare l'AND bit a bit:

$$g(i, j) = f(i, j) \text{ AND } m(i, j)$$

dove $255 =$ `11111111` e $0 =$ `00000000` in binario.

5. **Output:** Visualizzare la matrice risultante $L \times C$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0304-equalizacao.html

Figura 3.29: Simulatore EP03_04: Equalizzazione dell'istogramma (Livelli L = 2^B)

3.12.5.2 📌 Vincoli Computazionali

- **AND con 255:** p AND $255 = p$ (tutti i bit preservati).
- **AND con 0:** p AND $0 = 0$ (tutti i bit azzerati).
- **Maschera:** Gli unici valori possibili nella maschera sono 0 e 255.
- **Implementazione:** In Python, l'AND bit a bit tra interi utilizza l'operatore `&`.

3.12.5.3 🧠 Fondamento Teorico

Pixel f	Maschera m	Risultato f AND m
qualsiasi v	255 (`11111111`)	v (preservato)
qualsiasi v	0 (`00000000`)	0 (azzerato)

3.12.5.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi di f (L righe).
- Righe successive: Elementi di m (L righe con valori 0 o 255).

Output:

- Matrice risultante $L \times C$.

3.12.5.5 📌 Esempi

Input	Output	Osservazione
2 3 100 150 200 50 80 120 255 255 0 0 255 255	100 150 0 0 80 120	La maschera seleziona la regione
1 4 10 20 30 40 255 0 255 0	10 0 30 0	Alternato preservato/azzerato

```
%%writefile EP03_05.py
# Codice Python
```

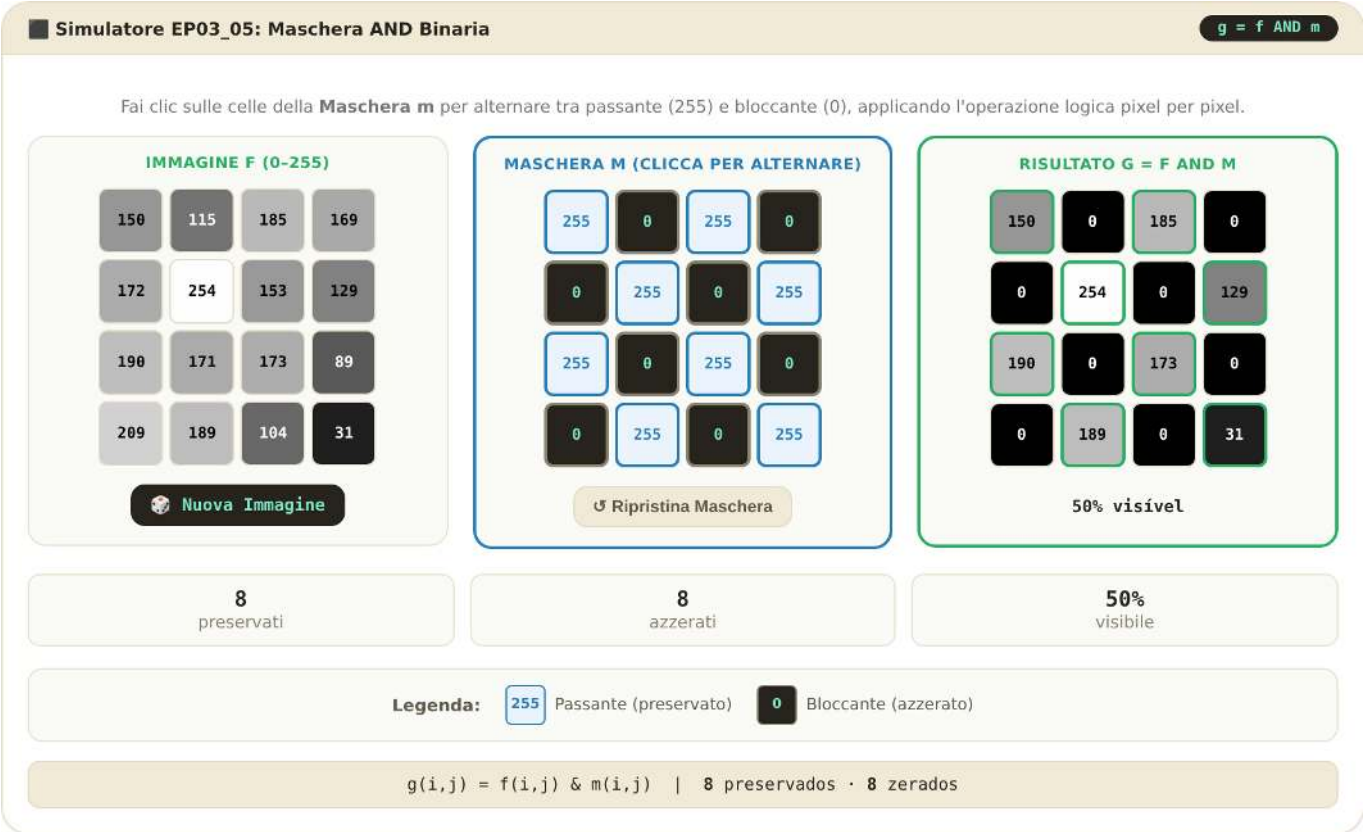
```
Overwriting EP03_05.py
```

```
TestSuite("EP03_05.py").run()
```

```
<IPython.core.display.HTML object>
```

3.12.6 EP03_06 Filtro di Media con *Kernel* N×N

Nelle telecamere dei veicoli autonomi, le immagini acquisite sotto pioggia o nebbia presentano rumore gaussiano. Il **filtro di media** è ampiamente utilizzato per la sua riduzione in tempo reale, essendo



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0305-mascara.html

Figura 3.30: Simulatore EP03_05: Applicazione della Maschera AND Binaria

implementato direttamente nell'**ISP** (*Image Signal Processor*) dei sensori **CMOS** (*Complementary Metal-Oxide-Semiconductor*).

I sensori CMOS sono i sensori di immagine utilizzati nella maggior parte delle fotocamere moderne (*smartphone*, *webcam*, fotocamere automobilistiche, ecc.). Essi convertono la luce in segnali elettrici, e l'ISP elabora questi segnali in tempo reale — applicando operazioni come la riduzione del rumore, il bilanciamento del bianco e altre regolazioni dell'immagine.

Vedere in Figura 3.31 una simulazione di questo EP.

3.12.6.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe), C (colonne) e N (dimensione del *kernel*, sempre dispari).
2. **Dati:** Leggere la matrice di pixel f.
3. **Filtro di Media:** Per ogni pixel (i, j) **interno** (senza bordi), calcolare:

$$g(i,j) = \text{round}\left(\frac{1}{N^2}\sum_{s=-(r)}^{r}\sum_{t=-(r)}^{r} f(i+s,\, j+t)\right), \quad r = \lfloor N/2 \rfloor$$

4. **Trattamento dei Bordi:** I pixel sul bordo (dove la finestra $N \times N$ supera i limiti) devono essere **copiati direttamente** dall'originale senza modifiche.
5. **Uscita:** Visualizzare la matrice risultante $L \times C$.

3.12.6.2 📌 Vincoli Computazionali

- **Raggio:** $r = \lfloor N/2 \rfloor$ (metà del *kernel*, intero).
- **Pixel interni:** (i, j) con $r \leq i < L - r$ e $r \leq j < C - r$.
- **Arrotondamento:** Usare l'arrotondamento matematico prima di convertire in intero.

- **Senza *clipping*:** La media dei valori $\in [0, 255]$ rimane in $[0, 255]$.

3.12.6.3 🧠 Fondamenti Teorici

Dimensione N	Coefficiente	Pixel nella finestra	Effetto
3	$1/9 \approx 0.111$	9	Morbido
5	$1/25 = 0.04$	25	Medio
7	$1/49 \approx 0.020$	49	Forte

3.12.6.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero N (dispari, $N \geq 3$).
- Righe successive: Elementi della matrice originale.

Output:

- Matrice filtrata $L \times C$.

3.12.6.5 📌 Esempi

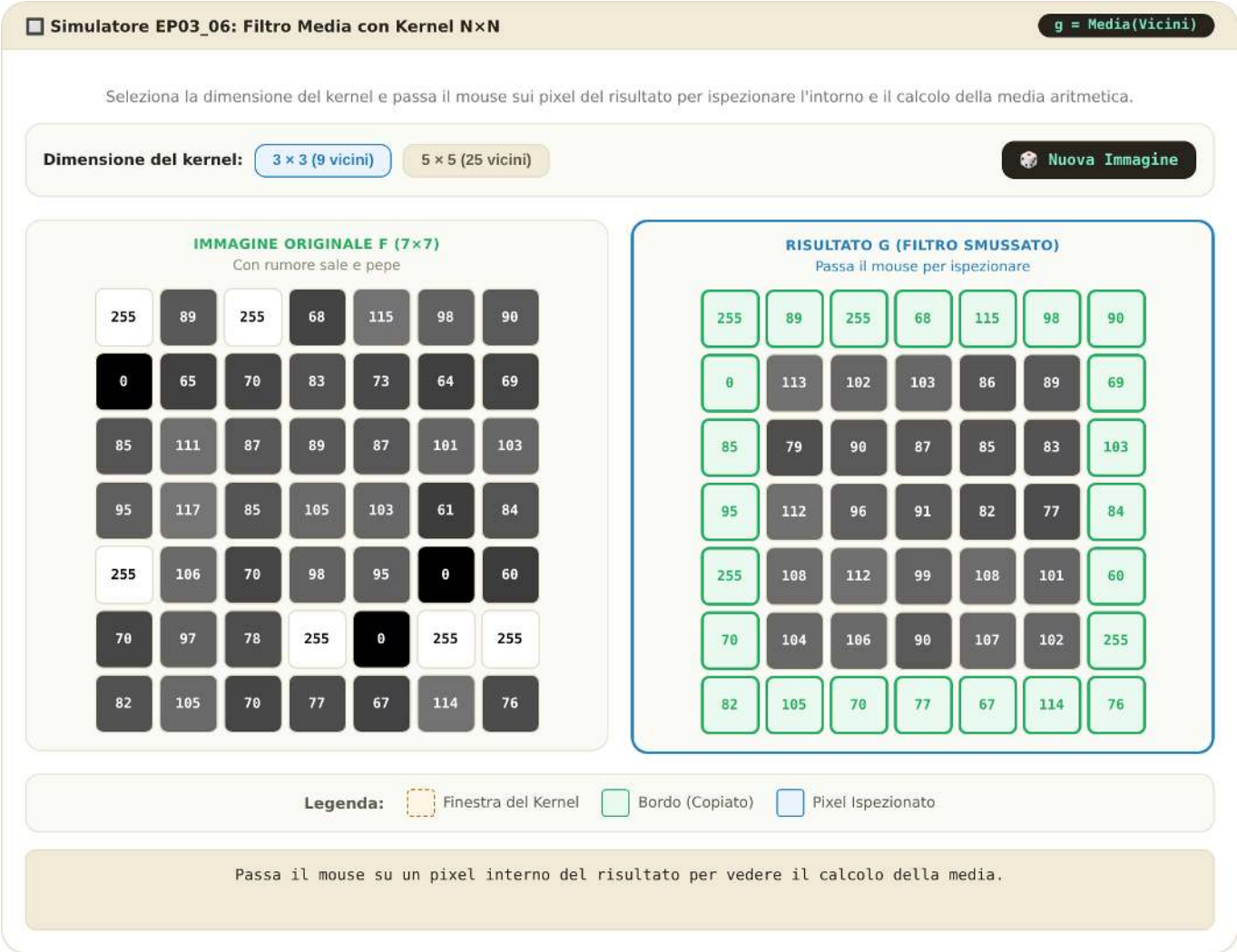
Input	Output	Osservazione
3 3 3 10 20 30 40 50 60 70 80 90	10 20 30 40 50 60 70 80 90	Solo bordo (3×3 = bordo totale)
5 5 3 0 0 0 0 0 0 0 0 0 0 0 0 100 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 11 11 11 0 0 11 11 11 0 0 11 11 11 0 0 0 0 0 0	Pixel isolato: tutti i 9 pixel interni la cui finestra 3×3 include il valore 100 ricevono round(100/9)=11

```
%%writefile EP03_06.py
# Codice Python
```

```
Overwriting EP03_06.py
```

```
TestSuite("EP03_06.py").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0306-media.html

Figura 3.31: Simulatore EP03_06: Filtro Medio con Kernel N×N

3.12.7 EP03_07 🔍 Operatore Laplaciano (w4) per l'Enfatizzazione dei Bordi

Nelle tomografie ad alta risoluzione, la nitidezza dei bordi tra i tessuti è critica per la diagnosi. L'**operatore Laplaciano** è ampiamente utilizzato nei *pipeline* di pre-elaborazione delle immagini mediche per enfatizzare automaticamente i contorni anatomici prima della segmentazione, evitando l'intervento manuale del radiologo.

Vedi in Figura 3.32 una simulazione di questo EP.

3.12.7.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere la matrice di pixel f.
3. **Laplaciano (w4):** Per ogni pixel **interno** (i, j) con $1 \leq i < L-1$, $1 \leq j < C-1$, calcolare:

$$\nabla^2 f(i,j) = f(i-1,j) + f(i+1,j) + f(i,j-1) + f(i,j+1) - 4 \cdot f(i,j)$$

4. **Enfatizzazione:** Calcolare l'immagine enfatizzata:

$$g(i,j) = \text{clip}(f(i,j) - \nabla^2 f(i,j))$$

5. **Bordo:** I pixel sul bordo vengono copiati direttamente: $g(i,j) = f(i,j)$.
6. **Uscita:** Visualizzare la matrice enfatizzata $L \times C$.

3.12.7.2 📌 Vincoli Computazionali

- ***Kernel* w4:** $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ — solo 4-vicini.
- **Saturazione:** $\text{clip}(x) = \max(0, \min(255, x))$ applicato al risultato dell'enfatizzazione.
- **Senza arrotondamento:** Il Laplaciano usa solo somme/sottrazioni di interi.

3.12.7.3 🧠 Fondamenti Teorici

Regione	$\nabla^2 f$	Effetto dell'Enfatizzazione
Uniforme	≈ 0	Nessuna modifica
Bordo crescente	< 0	Pixel schiarito
Bordo decrescente	> 0	Pixel scurito

3.12.7.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

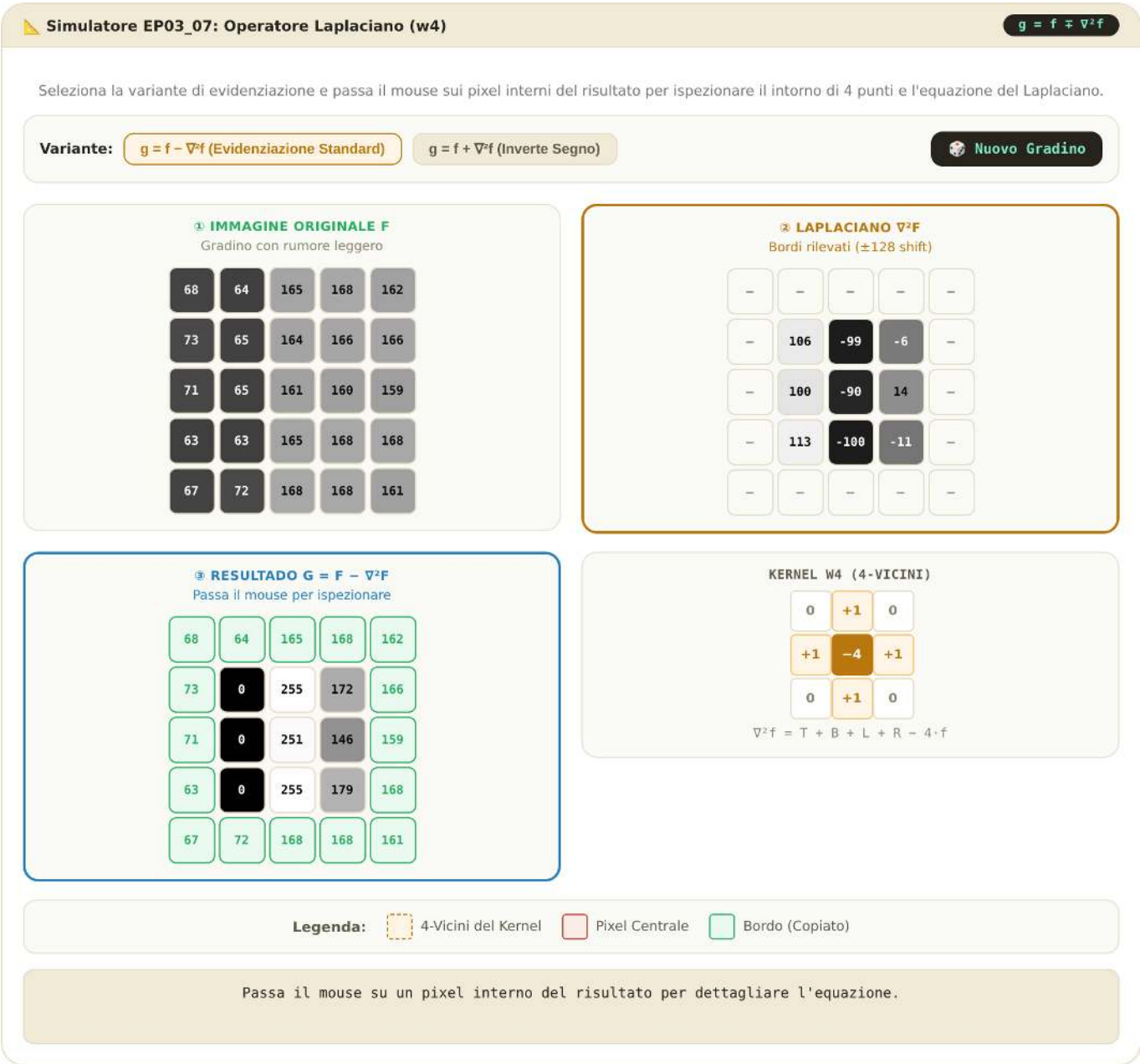
- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi della matrice originale.

Uscita:

- Matrice enfatizzata $L \times C$.

3.12.7.5 📌 Esempi

Ingresso	Uscita	Osservazione
3 3 0 0 0 0 100 0 0 0 0	0 0 0 0 255 0 0 0 0	Picco isolato: lap=−400, g=100−(−400)=500 → clip=255
3 3 50 50 50 50 50 50 50 50 50	50 50 50 50 50 50 50 50 50	Regione uniforme: Laplaciano=0, nessuna modifica



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0307-laplaciano.html

Figura 3.32: Simulador EP03_07: Operador Laplaciano (w4) para Realce de Bordas

```
%%writefile EP03_07.py
# Codice Python
```

```
Overwriting EP03_07.py
```

```
TestSuite("EP03_07.py").run()
```

```
<IPython.core.display.HTML object>
```

3.12.8 EP03_08 🧭 Gradiente di Sobel: Gx e Gy

Nei robot esploratori di Marte (come il Perseverance), il rilevamento degli ostacoli viene eseguito in tempo reale da telecamere stereoscopiche. L'**operatore di Sobel** calcola il gradiente direzionale della scena e viene utilizzato nell'algoritmo di rilevamento dei bordi per identificare rocce, crepe e dislivelli del terreno che potrebbero compromettere la navigazione.

Vedi in Figura 3.33 una simulazione di questo EP.

3.12.8.1 📋 Linee guida di implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere la matrice f.
3. **Gx e Gy:** Per ogni pixel **interno** (i, j) con $1 \leq i < L-1$, $1 \leq j < C-1$:

$$G_x(i,j) = [f(i-1,j+1) + 2f(i,j+1) + f(i+1,j+1)] - [f(i-1,j-1) + 2f(i,j-1) + f(i+1,j-1)]$$

$$G_y(i,j) = [f(i+1,j-1) + 2f(i+1,j) + f(i+1,j+1)] - [f(i-1,j-1) + 2f(i-1,j) + f(i-1,j+1)]$$

4. **Magnitudo:** $|\nabla f(i,j)| = \text{clip}(\text{round}(\sqrt{G_x^2 + G_y^2}))$.
5. **Bordo:** I pixel di bordo ricevono magnitudo 0.
6. **Output:** Visualizzare il magnitudo $L \times C$.

3.12.8.2 📌 Vincoli computazionali

- **Arrotondamento:** Applicare `round` prima di convertire in intero.
- **Saturazione:** $\text{clip}(x) = \max(0, \min(255, x))$.
- **Radice quadrata:** Usare $\sqrt{G_x^2 + G_y^2}$ (non l'approssimazione $|G_x| + |G_y|$).

3.12.8.3 🧠 Fondamenti teorici

Operatore	Rileva	Coefficienti diagonali
G_x	Bordi verticali	± 1
G_y	Bordi orizzontali	± 1
$\|\nabla f\|$	Tutti i bordi	Combinato

3.12.8.4 📦 Specifica di input e output (VPL)

Input:

- Riga 1: Intero L.

- Riga 2: Intero C.
- Righe successive: Elementi della matrice.

Output:

- Magnitudo del gradiente, matrice $L \times C$.

3.12.8.5 📌 Esempi

Input	Output	Osservazione
3 3 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0	Immagine nulla: gradiente zero
3 3 0 0 255 0 0 255 0 0 255	0 0 0 0 255 0 0 0 0	Bordo verticale centrale: Gx alto

```
%%writefile EP03_08.py
# Codice Python
```

```
Overwriting EP03_08.py
```

```
TestSuite("EP03_08.py").run()
```

```
<IPython.core.display.HTML object>
```

3.12.9 EP03_09 📡 Filtro della Mediana 3×3

Le immagini radar ad apertura sintetica (SAR) utilizzate nel monitoraggio ambientale e militare sono soggette a un tipo specifico di rumore chiamato *speckle*, che presenta caratteristiche simili al rumore sale e pepe. Il **filtro della mediana** è il metodo standard per la rimozione di tale rumore perché preserva i bordi delle strutture eliminando al contempo i punti spuri.

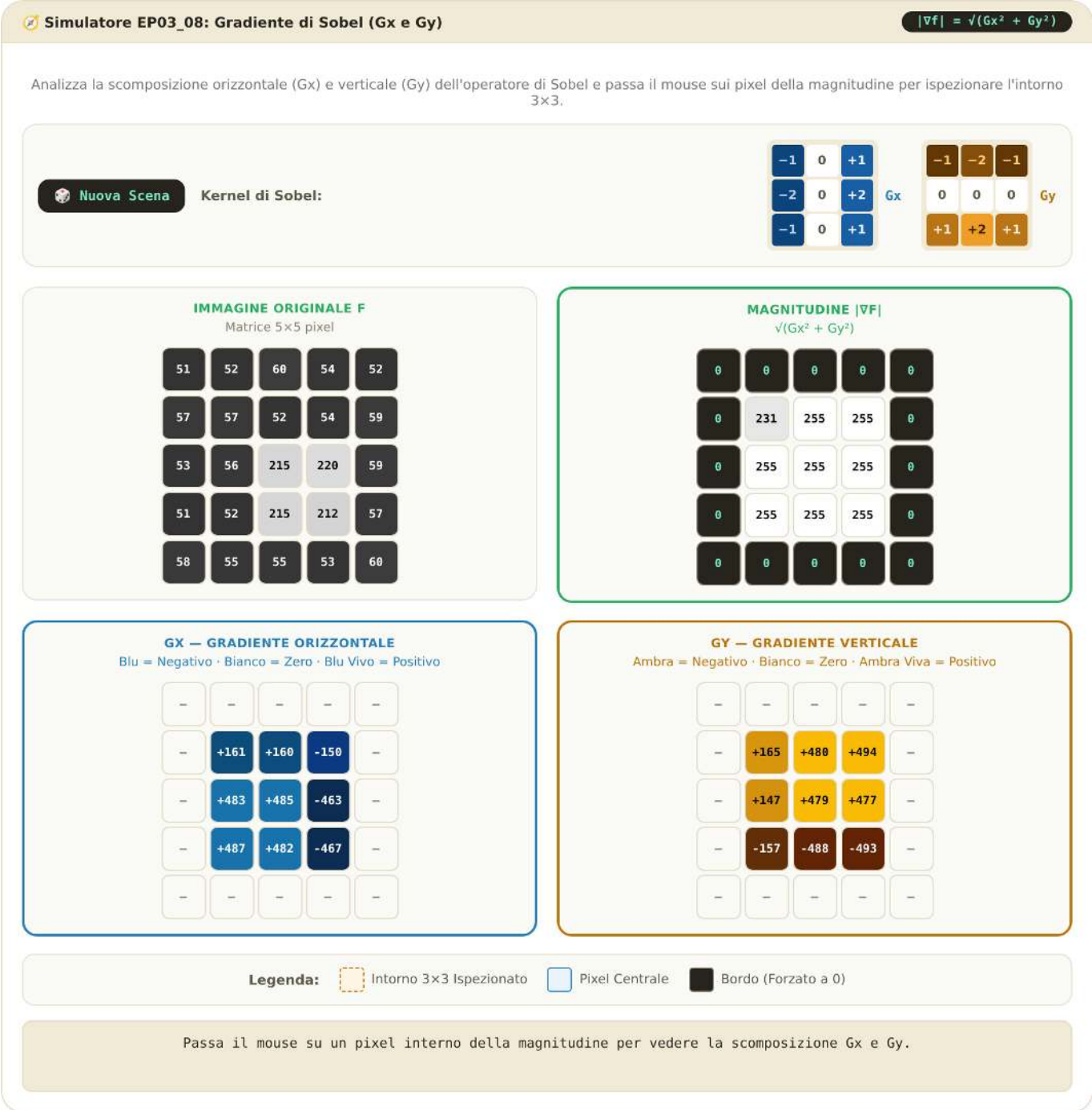
Vedere in Figura 3.34 una simulazione di questo EP.

3.12.9.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere la matrice dei pixel f.
3. **Filtro della Mediana 3×3:** Per ogni pixel **interno** (i, j) con $1 \leq i < L - 1$, $1 \leq j < C - 1$:
   - Raccogliere i 9 pixel del vicinato 3×3: $\{f(i + s, j + t) : s, t \in \{-1, 0, 1\}\}$.
   - Ordinare i 9 valori in ordine crescente.
   - Assegnare a $g(i, j)$ il valore centrale (posizione indice 4, considerando indice 0).

$$g(i, j) = \text{mediana}\{f(i + s, j + t) : s, t \in \{-1, 0, 1\}\}$$

4. **Bordo:** Copiare direttamente: $g(i, j) = f(i, j)$.
5. **Uscita:** Visualizzare la matrice filtrata $L \times C$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0308-sobel.html

Figura 3.33: Simulatore EP03_08: Gradiente di Sobel (Gx e Gy)

3.12.9.2 📌 Vincoli Computazionali

- **Finestra:** Sempre $3 \times 3 = 9$ elementi.
- **Mediana:** L'elemento centrale della sequenza ordinata (indice 4 da 0 a 8).
- **Nessun clipping:** La mediana di valori in $[0, 255]$ rimane in $[0, 255]$.
- **Non lineare:** Il filtro mediana non può essere espresso come convoluzione lineare.

3.12.9.3 🧠 Fondamenti Teorici

Rumore	Filtro della Media	Filtro della Mediana
Sale e pepe (0 o 255)	Diffonde il rumore	Rimuove senza distorcere i bordi
Gaussiano	Riduce efficacemente	Riduce parzialmente

3.12.9.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi della matrice.

Uscita:

- Matrice filtrata $L \times C$.

3.12.9.5 📌 Esempi

Ingresso	Uscita	Osservazione
3 3 100 100 100 100 0 100 100 100 100	100 100 100 100 100 100 100 100 100	Punto nero eliminato: mediana di $8{\times}100{+}1{\times}0 = 100$
3 3 50 50 50 50 255 50 50 50 50	50 50 50 50 50 50 50 50 50	Punto bianco (sale) eliminato

```
%%writefile EP03_09.py
# Codice Python
```

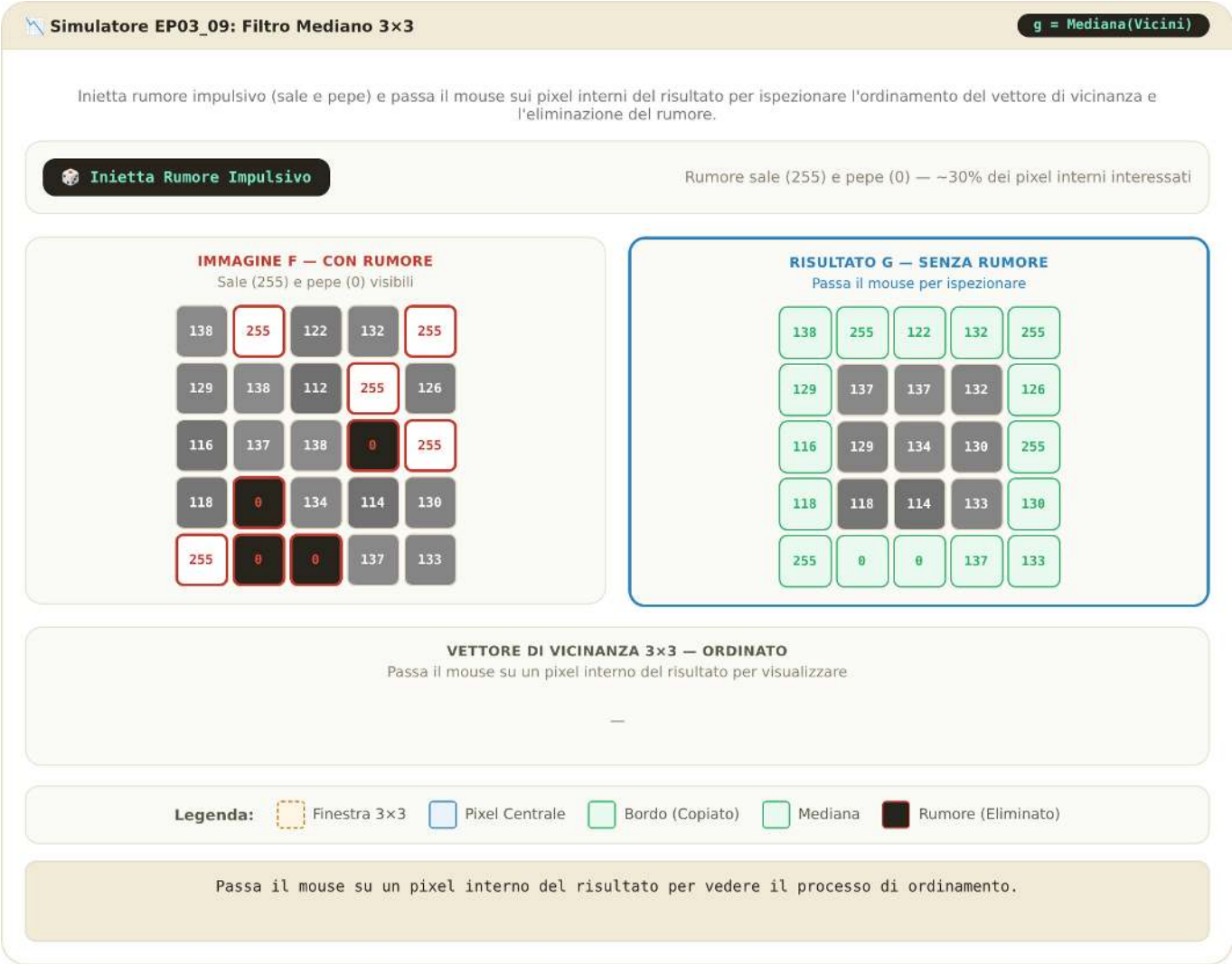
```
Overwriting EP03_09.py
```

```
TestSuite("EP03_09.py").run()
```

```
<IPython.core.display.HTML object>
```

3.12.10 EP03_10 ✨ *Unsharp Masking* (USM)

Nei sistemi di digitalizzazione di documenti storici e opere d'arte, la nitidezza delle immagini è fondamentale per la lettura di testi manoscritti e dettagli ornamentali. Lo ***Unsharp Masking*** **(USM)** è l'algoritmo



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0309-mediana.html

Figura 3.34: Simulatore EP03_09: Filtro della Mediana 3×3

di miglioramento della nitidezza standard utilizzato negli *scanner* professionali e in software come Adobe Photoshop, controllato dal parametro k che determina l'intensità del miglioramento.

Vedi in Figura 3.35 una simulazione di questo EP.

3.12.10.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Parametro:** Leggere il valore reale k (intensità del miglioramento, $k \geq 0$).
3. **Dati:** Leggere la matrice di pixel f.
4. **Smussamento:** Calcolare $\bar{f}$ con filtro di media 3×3 (solo pixel interni; bordi mantenuti):

$$\bar{f}(i,j) = \frac{1}{9} \sum_{s=-1}^{1} \sum_{t=-1}^{1} f(i+s, j+t)$$

5. **Maschera ad alta frequenza:** $m(i,j) = f(i,j) - \bar{f}(i,j)$.
6. **Miglioramento USM:** Per ogni pixel interno:

$$g(i,j) = \text{clip}\left(\text{round}\left(f(i,j) + k \cdot m(i,j)\right)\right)$$

7. **Bordo:** $g(i,j) = f(i,j)$ (copia diretta).
8. **Output:** Visualizzare la matrice migliorata $L \times C$.

3.12.10.2 📌 Vincoli Computazionali

- **Arrotondamento:** Applicare `round` prima del clipping.
- **Saturazione:** $\text{clip}(x) = \max(0, \min(255, x))$.
- **Operazioni in virgola mobile:** Calcolare $\bar{f}$ e m in virgola mobile prima di arrotondare il risultato finale.
- $k = 0$**:** Nessun miglioramento — l'output è identico all'input (eccetto per i bordi).

3.12.10.3 🧠 Fondamento Teorico

Fase	Operazione	Descrizione
1	$\bar{f} = f * \frac{1}{9}\mathbf{1}_{3\times3}$	Smussamento (basse frequenze)
2	$m = f - \bar{f}$	Maschera (alte frequenze)
3	$g = \text{clip}(\text{round}(f + k \cdot m))$	Miglioramento ponderato

3.12.10.4 📦 Specifica di Input e Output (VPL)

Input:

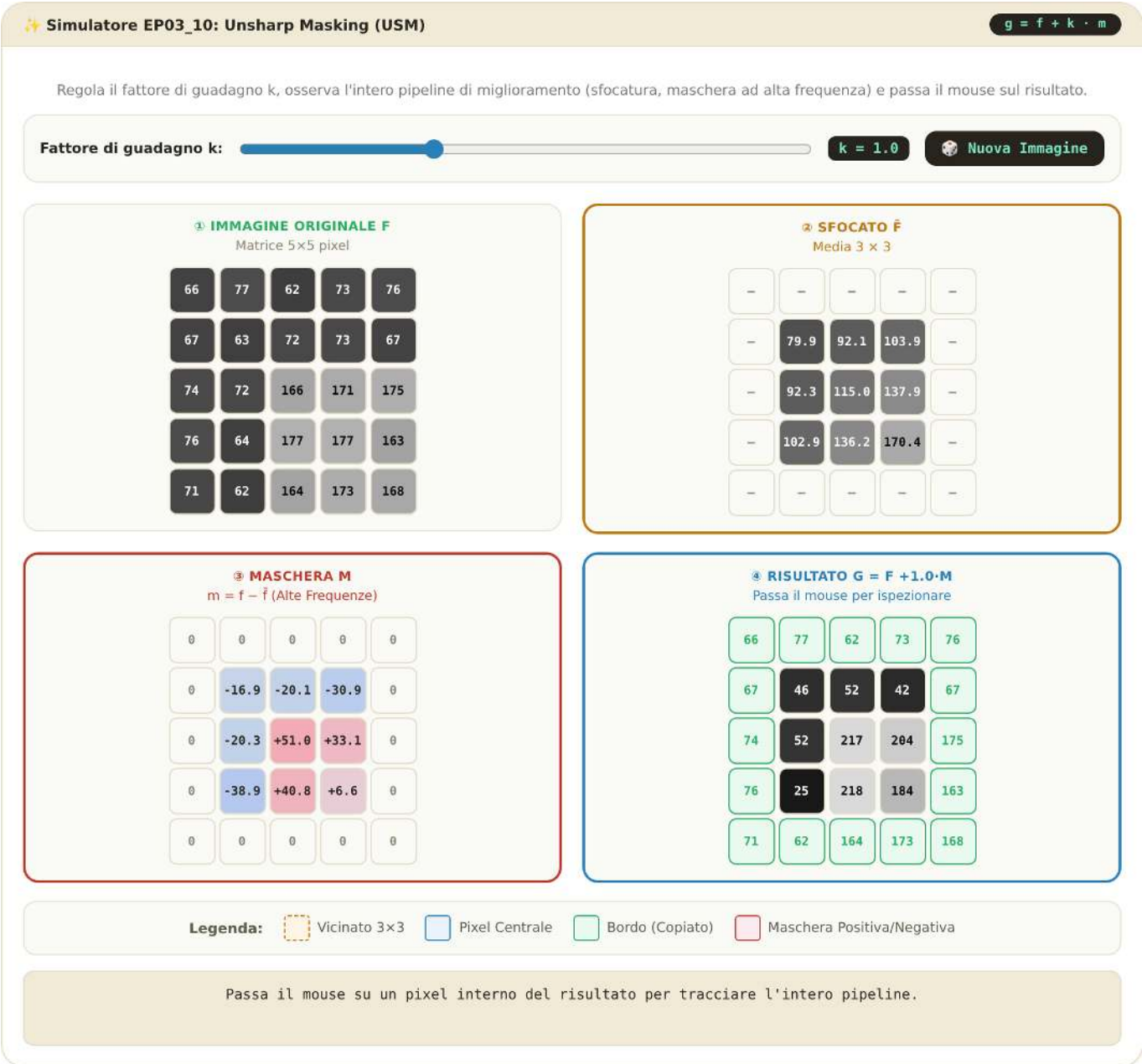
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Reale k.
- Righe successive: Elementi della matrice originale.

Output:

- Matrice migliorata $L \times C$.

3.12.10.5 📌 Esempi

Input	Output	Osservazione
3 3 0.0 100 100 100 100 100 100 100 100 100	100 100 100 100 100 100 100 100 100	k=0: nessun miglioramento
3 3 1.0 50 50 50 50 200 50 50 50 50	50 50 50 50 255 50 50 50 50	k=1: pixel centrale migliorato e saturato



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap03/sim-ep0310-unsharp.html

Figura 3.35: Simulatore EP03_10: *Unsharp Masking* (USM)

```
%%writefile EP03_10.py
# Codice Python
```

```
Overwriting EP03_10.py
```

```
TestSuite("EP03_10.py").run()
```

```
<IPython.core.display.HTML object>
```

Capitolo 4

Morfologia Matematica e Segmentazione delle Immagini

Executar Colab Abrir GitHub

Questo capitolo presenta due temi fondamentali dell'Elaborazione Digitale delle Immagini (EDI): la **morfologia matematica** e la **segmentazione delle immagini**. La morfologia matematica fornisce un quadro teorico basato sulla teoria degli insiemi per analizzare, raffinare e quantificare la forma degli oggetti in immagini binarie e in scala di grigi, mediante operatori fondamentali come l'**erosione** e la **dilatazione**. La segmentazione, a sua volta, ha l'obiettivo di partizionare l'immagine in regioni di interesse, separando gli oggetti dallo sfondo e producendo rappresentazioni adeguate per l'analisi e l'interpretazione.

Il capitolo inizia con la **sogliatura**, una delle tecniche più importanti di segmentazione, introducendo il metodo automatico di **Otsu** e riprendendo l'analisi degli istogrammi tramite la varianza interclasse, presentata nel Capitolo 1. Successivamente, vengono studiati i principali operatori della morfologia matematica, inclusi erosione, dilatazione, apertura, chiusura e ricostruzione morfologica, che consentono di raffinare le maschere binarie e preservare le strutture rilevanti degli oggetti. Infine, vengono presentate tecniche di segmentazione basata su regioni, come l'**etichettatura delle componenti connesse**, la **trasformata della distanza** e l'algoritmo ***watershed*** basato su marcatori, culminando nell'estrazione di descrittori geometrici e nella generazione di *bounding boxes* compatibili con i moderni sistemi di rilevamento degli oggetti.

4.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- **Applicare la sogliatura:** Comprendere il criterio automatico di Otsu basato sulla massimizzazione della varianza tra le classi (σ_B^2) e selezionare strategie adeguate di pre-elaborazione per facilitare la segmentazione;
- **Padroneggiare la morfologia binaria:** Comprendere e applicare l'erosione ($A \ominus B$) e la dilatazione ($A \oplus B$) come operatori fondamentali, derivando l'apertura ($A \circ B$), la chiusura ($A \bullet B$) e le operazioni basate sulla ricostruzione morfologica, come `mm.clohole` e `mm.edgeoff`;
- **Applicare la morfologia in toni di grigio:** Utilizzare il gradiente morfologico e i filtri *top-hat* per l'enhancement e l'analisi di strutture locali;
- **Etichettare i componenti connessi:** Identificare e separare regioni connesse in immagini binarie mediante algoritmi di etichettatura;
- **Applicare la trasformata della distanza:** Interpretare e calcolare le distanze dallo sfondo utilizzando approcci morfologici e metriche geometriche;
- **Segmentare per regioni:** Costruire *pipeline* di segmentazione basate su marcatori utilizzando la Trasformata della Distanza e l'algoritmo ***watershed***;
- **Estrarre descrittori geometrici:** Calcolare proprietà come area, perimetro, centroide, circolarità e *bounding box* tramite `mm.label0` ed estrazione dei contorni;
- **Mettere in relazione l'elaborazione delle immagini digitali e la visione artificiale:** Comprendere come i descrittori estratti tramite segmentazione possano essere convertiti in formati utilizzati da rilevatori moderni, come YOLO.

```python
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup()
from morph import mm
import numpy as np
import cv2
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0
```

4.2 Soglie

La **sogliatura** (*thresholding*) è una delle forme più semplici ed efficaci di segmentazione delle immagini. Il suo obiettivo è classificare ogni pixel in due classi di intensità, normalmente associate a *oggetto* e *sfondo*:

$$g(x,y) = \begin{cases} 255, & \text{se } f(x,y) > T \\ 0, & \text{caso contrário} \end{cases} \tag{4.1}$$

dove $f(x,y)$ rappresenta l'intensità del pixel nell'immagine originale e $g(x,y)$ l'immagine binaria risultante.

La scelta della soglia T è importante per la qualità della segmentazione. Il metodo di **Otsu** determina automaticamente la soglia ottimale massimizzando la **varianza tra le classi** σ_B^2, definita come:

$$\sigma_B^2(T) = w_0(T)\, w_1(T)\, \left[\mu_0(T) - \mu_1(T)\right]^2 \tag{4.2}$$

dove:

- $w_0(T)$ e $w_1(T)$ sono le probabilità cumulative delle classi sfondo e oggetto;
- $\mu_0(T)$ e $\mu_1(T)$ sono le medie di intensità di queste classi;
- $\sigma_B^2(T)$ rappresenta la varianza tra le classi per una data soglia T.

Il metodo funziona meglio quando l'istogramma presenta due gruppi di intensità relativamente separati. A tal fine, l'algoritmo valuta tutte le possibili soglie dell'immagine — tipicamente nell'intervallo $[0, 255]$ per immagini a 8 bit — e seleziona il valore che massimizza la varianza tra le classi, denotata da σ_B^2:

$$T^* = \arg \max_{T \in [0,255]} \sigma_B^2(T)$$

ℹ Otsu presuppone istogrammi bimodali

Il metodo di Otsu produce risultati migliori quando l'istogramma presenta due picchi ben definiti (*bimodalità*), corrispondenti allo sfondo e all'oggetto. Quanto maggiore è la separazione tra questi picchi e quanto più pronunciato è il massimo di σ_B^2, tanto più affidabile tende a essere la soglia ottenuta.
In immagini con illuminazione non uniforme o con multiple regioni di intensità, le tecniche di **sogliatura adattiva** — nelle quali la soglia viene calcolata localmente — tendono a produrre segmentazioni più robuste.
L'indice B in σ_B^2 significa ***between classes*** (*tra le classi*). Pertanto, σ_B^2 rappresenta la **varianza tra le classi** (*between-class variance*).

4.2.1 Immagine di Monete

L'immagine utilizzata per esercitarsi nella segmentazione è una fotografia di una collezione di monete di diversi paesi ed epoche (Figura 4.1). Crediti: GAZI.MD.AHAD (CC BY-SA 4.0). Essa presenta oggetti circolari con bordi ben definiti, risultando ideale per dimostrare la sogliatura, gli operatori morfologici, la trasformata della distanza, il *watershed* e i descrittori di forma.

```
import os

url     = "https://upload.wikimedia.org/wikipedia/commons/2/25/GAZI.MD.AHAD_11.jpg"
caminho = "imagens/coins.png"

if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    mm.write(mm.read(url), caminho)

img_coins_color = mm.read(caminho)
img_coins_gray  = mm.gray(img_coins_color)

print(f"Dimensões [y,x,c]: {img_coins_color.shape}")
mm.show(img_coins_color, scale=30)
```

```
Dimensões [y,x,c]: (2560, 1920, 3)
```

Figura 4.1: Imagem com moedas de vários tipos. Crédito: GAZI.MD.AHAD (CC BY-SA 4.0).

4.2.2 Pre-processamento per l'Otsu

La qualità del metodo di Otsu dipende direttamente da quanto **bimodale** sia l'istogramma dell'immagine di ingresso. L'immagine originale delle monete presenta un'illuminazione non uniforme e monete scure (rame ossidato) con intensità vicine a quelle dello sfondo scuro, rendendo l'istogramma poco bimodale.

Per minimizzare queste limitazioni, verranno valutate due tecniche classiche di pre-processamento, presentate nel capitolo precedente, applicate prima della fase di sogliatura. La Tabella 4.1 riassume le caratteristiche

di ciascun approccio. Queste tecniche mirano ad aumentare la separazione tra oggetto e sfondo, rendendo l'istogramma più vicino a una distribuzione bimodale.

Tabella 4.1: Tecniche di pre-processamento valutate per migliorare la separazione tra monete e sfondo prima dell'applicazione del metodo di Otsu.

Tecnica	Cosa fa	Quando usarla
CLAHE	Equalizzazione adattiva locale dell'istogramma	Basso contrasto globale o regionale
Gaussiano	Smussamento tramite convoluzione con gaussiana	Rumore ad alta frequenza (trama dello sfondo)

La funzione `cv2.createCLAHE(clipLimit, tileGridSize)` divide l'immagine in blocchi e applica l'equalizzazione dell'istogramma a ciascuno di essi, limitando l'amplificazione del rumore tramite il parametro `clipLimit`. Il filtro gaussiano (`cv2.GaussianBlur`), invece, smussa le texture fini che potrebbero creare falsi picchi nell'istogramma.

Per confrontare oggettivamente quale pre-processamento produca il miglior input per il metodo di Otsu, verranno presentate, per ciascuna versione dell'immagine, l'immagine stessa, l'istogramma con la soglia ottimale T^* evidenziata, la curva $\sigma_B^2(T)$ e il risultato della binarizzazione.

Criterio di confronto

La versione che presenterà il valore massimo di σ_B^2 nel suo picco fornirà la migliore separazione tra le classi di sfondo e oggetto e, di conseguenza, il miglior input per il metodo di Otsu (Figura 4.2).

```
import cv2, io
import matplotlib.pyplot as plt

def otsu_criterio(img):
    h=mm.hist(img); p=h/h.sum();                  # istogramma e probabilità
    sigma2=np.zeros(len(p))
    for T in range(1,len(p)):                     # scorre le soglie
        w0,w1=p[:T].sum(),p[T:].sum()             # probabilità delle classi
        if w0*w1==0: continue                     # evita la divisione per zero
        mu0=(np.arange(T)*p[:T]).sum()/w0         # media sfondo
        mu1=(np.arange(T,len(p))*p[T:]).sum()/w1  # media oggetto
        sigma2[T]=w0*w1*(mu0-mu1)**2              #  ²B(T)
    return sigma2,np.argmax(sigma2)               # curva e T ottimale

def fig2img(fig):
    b=io.BytesIO(); fig.savefig(b,format='png',dpi=100)   # figura → buffer
    plt.close(fig); b.seek(0)
    return np.array(plt.imread(b))                        # buffer → array

def plot_curve(y,T,title,ylabel,color):
    fig,ax=plt.subplots(figsize=(4,3))
    ax.plot(y,color=color) if ylabel=="  ²B" else ax.bar(range(len(y)),y,color=color,width=1)
    ax.axvline(T,color='red',lw=2,label=f"T*={T}")       # soglia ottimale
    ax.set(xlabel="T" if ylabel=="  ²B" else "Intensidade",ylabel=ylabel)
    ax.legend(fontsize=8); plt.tight_layout()
    return fig2img(fig)

#   Pre-processamenti
img_clahe = mm.clahe(img_coins_gray, clipLimit=2.0, tiles=8)
img_gauss = cv2.GaussianBlur(img_clahe,(5,5),0)
imgs0=[("Original",img_coins_gray),
```

```
        ("CLAHE",img_clahe),
        ("CLAHE+Gauss",img_gauss)]

#   Tabella comparativa
print(f"{'Versão':<18}{'T*':>6}{' ²B pico':>14}")
print("-"*40)

imgs,titles=[],[]
for nome,img in imgs0:
    sigma2,T=otsu_criterio(img)                          # calcola  ²B(T)
    print(f"{nome:<18}{T:>6}{sigma2[T]:>14.4e}")
    imgs += [
        img,                                             # immagine
        plot_curve(mm.hist(img),T,f"Hist T*={T}","Freq.","steelblue"),
        plot_curve(sigma2,T," ²B(T)"," ²B","darkorange"),
        mm.threshold(img)                               # Otsu finale
    ]
    titles += [nome,f"Hist T*={T}"," ²B(T)",f"Otsu T*={T}"]

#   Visualizzazione finale
mm.show(imgs,titles=titles,cols=4,figsize=(12,12),dpi=200)
```

```
Versão                T*       ²B pico
----------------------------------------
Original             105    1.9437e+03
CLAHE                122    2.4695e+03
CLAHE+Gauss          123    2.4283e+03
```

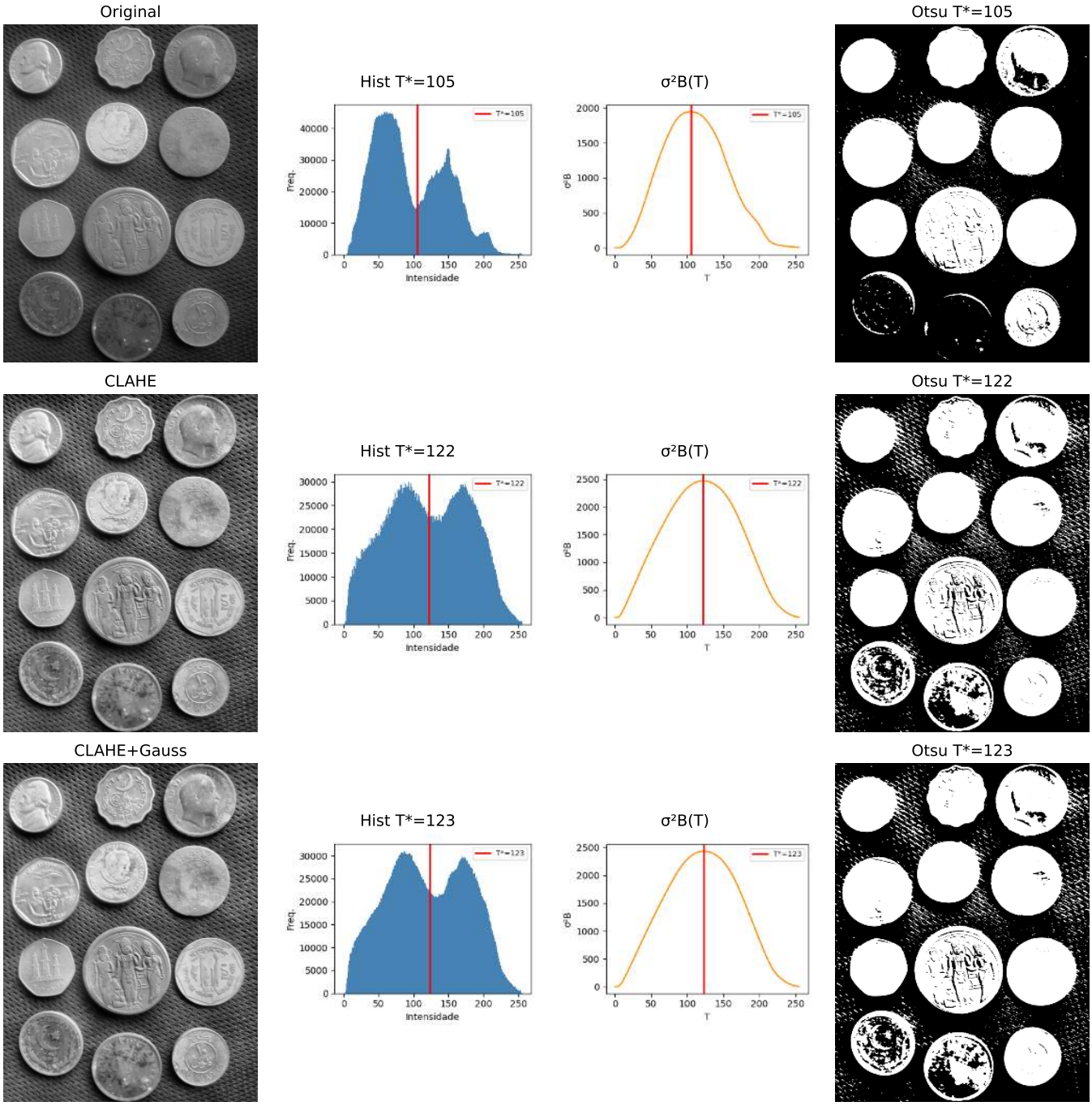


Figura 4.2: Confronto dei pre-processamenti: immagine | istogramma+T* | ²B(T) | Otsu. La migliore separazione bimodale indica la soglia più affidabile.

4.2.3 Risultato: CLAHE come Miglior Preprocessamento

L'analisi della Figura 4.2 indica che il **CLAHE** ha ottenuto il valore più alto della varianza inter-classi ($\sigma_B^2 \approx 2{,}47 \times 10^3$), con soglia ottimale $T^* = 122$. Sebbene la combinazione **CLAHE+Gaussiano** abbia prodotto un risultato molto simile ($\sigma_B^2 \approx 2{,}43 \times 10^3$, $T^* = 123$), il criterio quantitativo del metodo di Otsu favorisce leggermente l'uso del CLAHE da solo.

In termini visivi, le immagini binarizzate ottenute con CLAHE e CLAHE+Gaussiano sono praticamente equivalenti. La differenza tra i due approcci diventa più evidente nell'analisi degli istogrammi e dei valori di $\sigma_B^2(T)$ che nell'ispezione diretta delle segmentazioni risultanti. Pertanto, la scelta del CLAHE si basa principalmente sulla massimizzazione della separazione statistica tra le classi di sfondo e oggetto.

Interpretazione dei risultati

Si noti che i preprocessamenti con CLAHE e CLAHE+Gaussiano producono istogrammi e soglie ottimali molto vicini ($T^* = 122$ e $T^* = 123$). Di conseguenza, le immagini binarizzate risultanti sono anch'esse piuttosto simili. In questo caso, la decisione non si basa su differenze visive marcate, ma sul criterio oggettivo del metodo di Otsu: il valore più alto di σ_B^2 indica la migliore separazione tra le classi.

4.3 Morfologia Matematica

La **morfologia matematica** è una teoria basata su insiemi utilizzata per analizzare la forma e la struttura degli oggetti nelle immagini. A differenza dei filtri lineari presentati nel Capitolo 3, gli operatori morfologici sono **non lineari**, poiché si basano su operazioni di minimo, massimo e inclusione spaziale, anziché su combinazioni lineari di intensità. Questi operatori agiscono sul vicinato di ciascun pixel tramite un **elemento strutturante** $\mathbb{B}$, responsabile di definire la forma e la dimensione della regione analizzata.

Nelle immagini binarie e in scala di grigi con elementi piatti, l'elemento strutturante traslato nella posizione x è definito spazialmente come:

$$\mathbb{B}_x = \{x + b \mid b \in \mathbb{B}\}$$

Nelle regioni di bordo dell'immagine, parte dell'insieme $\mathbb{B}_x$ può estendersi oltre il dominio fisico della scena ($\mathbb{E}$). Per garantire la coerenza matematica degli operatori primitivi in questi confini, si assume teoricamente che lo spazio esterno al dominio dell'immagine sia riempito con l'**elemento neutro** dell'operazione corrispondente (infinito positivo per l'erosione e infinito negativo per la dilatazione), impedendo che l'ambiente esterno corrompa le strutture interne dell'oggetto.

Quando l'elemento strutturante associa pesi ai suoi elementi — cioè $b : \mathbb{B} \to \mathbb{Z}$ — esso è denominato **funzione strutturante** o **elemento strutturante non piano**.

Sviluppata da Matheron e Serra negli anni Sessanta per immagini binarie e successivamente estesa alla scala di grigi, la morfologia matematica fonda operatori come il gradiente morfologico, il *top-hat*, il *watershed* e la trasformata della distanza, tutti derivati da due primitivi: **erosione** e **dilatazione** [Matheron (1975); Serra (1982)].

4.3.1 Erosione e Dilatazione

I due operatori primitivi sono definiti in modo unificato per immagini in toni di grigio ($f : \mathbb{E} \to \mathbb{Z}$) e, per restrizione al dominio $\{0, 1\}$, anche per immagini binarie.

4.3.1.1 Erosione

L'**Erosione** di un'immagine f mediante una funzione strutturante $b : \mathbb{B} \to \mathbb{Z}$ è definita formalmente da:

$$\varepsilon_b(f)(x) = (f \ominus b)(x) = \min_{z \in \mathbb{B}} \{\, f(x + z) - b(z) \,\}, \quad \forall\, x \in \mathbb{E} \tag{4.3}$$

Nella pratica, l'erosione sostituisce l'intensità del pixel x con il valore minimo risultante dalla differenza tra l'immagine e l'elemento strutturante nell'intorno definito dal dominio $\mathbb{B}$. Valori positivi nei pesi di $b(z)$ spingono il risultato locale verso il basso, "scavando" il rilievo dell'immagine più profondamente e intensificando l'erosione.

Nel caso **piano** (dove i pesi sono nulli all'interno del dominio, cioè $b \equiv 0$), l'espressione si semplifica al minimo locale puro:

$$\varepsilon_B(f)(x) = \min\{\, f(y) : y \in \mathbb{B}_x \,\}$$

Nelle immagini binarie, questa operazione equivale a richiedere che l'insieme $\mathbb{B}$, traslato alla coordinata x, sia **completamente contenuto** nell'oggetto A:

$$A \ominus \mathbb{B} = \{\, z \in \mathbb{E} \mid \mathbb{B}_z \subseteq A \,\}$$

Effetto Visivo: *Rimpicciolisce* oggetti e strutture chiare, eliminando protuberanze, picchi luminosi o rumori che siano geometricamente più piccoli del dominio $\mathbb{B}$.

4.3.1.2 Implementazione dell'erosione

La versione didattica `mm.ero0` implementa il caso particolare di erosione con **elemento strutturante piano**. Per ogni pixel (y, x), la funzione esamina i vicini spaziali consentiti da B e memorizza il valore minimo trovato nell'immagine di ingresso f, riproducendo direttamente l'operazione di minimo locale descritta nella Equazione 4.3 per $b \equiv 0$.

Si noti che i valori dei vicini vengono sempre letti in modo statico dall'immagine originale f; la matrice di uscita g viene utilizzata esclusivamente per registrare il minimo accumulato dell'intorno corrente. In questo modo, il risultato finale è invariante rispetto all'ordine di scansione dei pixel (sia per righe che per colonne).

La funzione ausiliaria `_viz` calcola le coordinate dei vicini validi entro i limiti fisici dell'immagine. Ai bordi, l'inizializzazione dell'accumulatore a `255` emula esattamente il riempimento con elemento neutro richiesto dalla teoria. La funzione di interfaccia `mm.ero` ricorre invece all'implementazione nativa e ottimizzata di OpenCV (`mm.ero`) quando l'elemento strutturante è piano, passando alla routine generale `mm.ero1` qualora l'elemento presenti pesi topografici.

L'esempio computazionale seguente illustra l'applicazione di un elemento strutturante a croce (`mm.secross()`) evidenziato nella Figura 4.3, confrontando l'esecuzione della variante didattica a cicli (`mm.ero0`) con il motore computazionale di OpenCV (`mm.ero`).

```
B_cruz = mm.secross()
mm.drawImgPlt(B_cruz, scale=40)
```

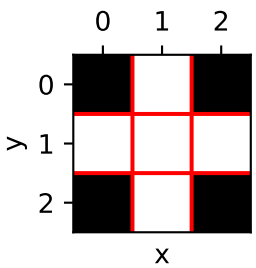


Figura 4.3: Elemento strutturante a forma di croce (B_{cruz}) utilizzato per la connettività-4.

```
def _viz(f, B, y, x):
    """Genera (vy, vx, b_val) per ogni vicino valido di (y,x)."""
    H, W = f.shape
    Bh, Bw = B.shape
    oh, ow = -Bh/2 + 0.5, -Bw/2 + 0.5  # offset fissi
    for by, bx in np.ndindex(Bh, Bw):
        vy, vx = int(y + by + oh), int(x + bx + ow)
        if 0 <= vy < H and 0 <= vx < W:
            yield vy, vx, B[by, bx]

def ero(f, Bc=np.zeros((3,3),dtype='uint8')):
    """Erosione (OpenCV o con pesi)."""
    try:    return cv2.erode(f, Bc)
```

```
    except: return mm.ero1(f, Bc)

def ero0(f, Bc=np.ones((3,3),dtype='uint8')):
    """Erosione classica senza pesi."""
    g = np.empty_like(f)
    for y in range(f.shape[0]):
        for x in range(f.shape[1]):
            g[y,x] = 255
            for vy,vx,bv in mm._viz(f,Bc,y,x):
                if bv and g[y,x] > f[vy,vx]: g[y,x] = f[vy,vx]
    return g

# Script di Test e Validazione
B = mm.secross()
print("Elemento strutturante B:")
print(mm.drawImage(B))

f = mm.randomImage(5,5)
print("Immagine originale f:")
print(mm.drawImage(f))

print("Erosione Piana Didattica (ero0):")
print(mm.drawImage(ero0(f, B)))

print("Erosione Ottimizzata OpenCV (ero):")
print(mm.drawImage(ero(f, B)))
```

```
Elemento strutturante B:
0 1 0
1 1 1
0 1 0

Immagine originale f:
2 5 2 7 5
3 6 2 6 6
9 1 8 2 7
0 6 9 2 7
3 8 5 7 3

Erosione Piana Didattica (ero0):
2 2 2 2 5
2 1 2 2 5
0 1 1 2 2
0 0 2 2 2
0 3 5 2 3

Erosione Ottimizzata OpenCV (ero):
2 2 2 2 5
2 1 2 2 5
0 1 1 2 2
0 0 2 2 2
0 3 5 2 3
```

La funzione `_viz` utilizza `yield` per generare ogni vicino su richiesta, senza memorizzare tutti i risultati in memoria in una lista. Nell'esperimento riportato di seguito, una finestra di 3000×3000 produce 9 milioni di vicini. L'implementazione basata su lista ha consumato più di 1 GB di RAM e ha richiesto circa 46 s, mentre la versione con `yield` ha consumato una memoria trascurabile e ha impiegato 38 s. Nell'elaborazione delle immagini, i generatori sono particolarmente utili per attraversare grandi vicinanze in modo efficiente.

```python
import tracemalloc

def lista(n): return [(i, i) for i in range(n)]
def gera(n):
    for i in range(n): yield i, i

for nome, f in [("LISTA", lista), ("YIELD", gera)]:
    tracemalloc.start()
    sum(x+y for x,y in f(9_000_000))
    _, pico = tracemalloc.get_traced_memory()
    tracemalloc.stop()
    print(f"{nome}: {pico/1024/1024:.1f} MB")
```

```
LISTA: 830.8 MB
YIELD: 0.0 MB
```

4.3.1.3 Dilatazione

La **Dilatazione** di un'immagine f mediante una funzione strutturante $b : \mathbb{B} \to \mathbb{Z}$ è definita formalmente da:

$$\delta_b(f)(x) = (f \oplus b)(x) = \max_{z \in \mathbb{B}} \{ f(x - z) + b(z) \}, \quad \forall\, x \in \mathbb{E} \tag{4.4}$$

In pratica, la dilatazione sostituisce l'intensità del pixel x con il valore più elevato risultante dalla somma tra l'immagine e l'elemento strutturante nell'intorno definito. L'argomento di inversione spaziale $(x - z)$ indica che la dilatazione valuta implicitamente l'elemento trasposto (riflesso) $\hat{b}$, proprietà fondamentale per garantire la dualità matematica rispetto all'erosione.

Nel caso **piano** (dove i pesi sono nulli all'interno del dominio, ovvero $b \equiv 0$), l'espressione si riduce al massimo locale puro:

$$\delta_B(f)(x) = \max\{ f(y) : y \in \mathbb{B}_x \}$$

Nelle immagini binarie, tale operazione equivale a richiedere che l'insieme riflesso $\hat{\mathbb{B}}$, traslato alla coordinata x, possieda **intersezione non vuota** con l'oggetto A:

$$A \oplus \mathbb{B} = \{ z \in \mathbb{E} \mid \hat{\mathbb{B}}_z \cap A \neq \emptyset \}$$

Effetto visivo: *Espande* le strutture chiare dell'immagine, aumentando il riempimento degli oggetti, collegando componenti vicine ed eliminando canali, cavità scure o valli che siano geometricamente più piccoli del dominio $\mathbb{B}$.

4.3.1.4 Implementazione della dilatazione

La versione didattica `mm.dil0` implementa il caso particolare di dilatazione con **elemento strutturante piatto**. Per ogni pixel (y, x), la funzione esamina i vicini spaziali consentiti da B e memorizza il valore massimo trovato nell'immagine di ingresso f, riproducendo direttamente l'operazione di massimo locale per $b \equiv 0$.

Prima di iniziare la scansione spaziale, l'elemento strutturante subisce una riflessione geometrica tramite l'istruzione `np.flip(Bc)` per costruire esplicitamente la matrice trasposta $\hat{B}$ richiesta dalla teoria. In maschere perfettamente simmetriche (come croci, quadrati e dischi centrati nell'origine), tale riflessione non altera la disposizione dei pixel; tuttavia, per elementi asimmetrici, questo passaggio è strettamente necessario per garantire l'equivalenza con le definizioni formali e salvaguardare le leggi di dualità.

Come verificato nell'operatore di erosione, i valori dei vicini vengono sempre letti in modo statico dalla matrice originale f, mentre la matrice di uscita g agisce puramente come registratore del massimo accumulato

dell'intorno. Ai bordi dell'immagine, l'inizializzazione dell'accumulatore a `0` emula esattamente il riempimento esterno con elemento neutro ($-\infty$, o zero in rappresentazioni a 8 bit), garantendo che i bordi fisici della scena vengano dilatati in perfetta conformità con lo standard adottato da OpenCV.

L'esempio computazionale seguente illustra l'applicazione pratica di un elemento a croce (`mm.secross()`), validando la coerenza tra la logica a cicli (`mm.dil0`) e il metodo nativo industriale (`mm.dil`).

```
def dil(f, Bc=np.zeros((3,3),dtype='uint8')):
    """Dilatazione (OpenCV o con pesi)."""
    try:    return cv2.dilate(f, Bc)
    except: return mm.dil1(f, Bc)

def dil0(f, Bc=np.zeros((3,3),dtype='uint8')):
    """Dilatazione piana seguendo rigorosamente la teoria."""
    g = np.empty_like(f)
    Bc = np.flip(Bc)     # riflessione esplicita: B
    for y in range(f.shape[0]):
        for x in range(f.shape[1]):
            g[y,x] = 0 # Inizializza con il valore minimo per cercare il massimo
            for vy,vx,bv in mm._viz(f,Bc,y,x):
                if bv and g[y,x] < f[vy,vx]:
                    g[y,x] = f[vy,vx]
    return g

# Script di Test e Validazione
B = mm.secross()
print("Elemento strutturante B:")
print(mm.drawImage(B))

f = mm.randomImage(5,5)
print("Immagine originale f:")
print(mm.drawImage(f))

print("Dilatazione Piana Didattica (dil0):")
print(mm.drawImage(dil0(f, B)))

print("Dilatazione Ottimizzata OpenCV (dil):")
print(mm.drawImage(dil(f, B)))
```

```
Elemento strutturante B:
0 1 0
1 1 1
0 1 0

Immagine originale f:
3 2 3 1 9
3 3 5 9 2
2 1 9 7 8
0 3 2 9 7
6 6 1 7 3

Dilatazione Piana Didattica (dil0):
3 3 5 9 9
3 5 9 9 9
3 9 9 9 8
6 6 9 9 9
6 6 7 9 7

Dilatazione Ottimizzata OpenCV (dil):
3 3 5 9 9
3 5 9 9 9
```

```
3 9 9 9 8
6 6 9 9 9
6 6 7 9 7
```

Nota: Il Confronto dei Segni ($f(x+z)$ vs $f(x-z)$)

Confronta le definizioni formali dell'erosione (Equazione 4.3) e della dilatazione (Equazione 4.4). Considera un elemento strutturante asimmetrico a destra $\mathbb{B} = \{0, 1\}$ (origine e un pixel a destra) applicato alla posizione $x = 10$.

1. **Nell'Erosione** (Equazione 4.3):

$$\min\{f(x+z) - b(z)\}$$

   - $z = 0 \Rightarrow f(10+0) = \mathbf{f(10)}$
   - $z = 1 \Rightarrow f(10+1) = \mathbf{f(11)}$

   L'operatore interroga il pixel corrente (10) e il pixel a destra (11), preservando l'orientamento originale di $\mathbb{B}$.

2. **Nella Dilatazione** (Equazione 4.4):

$$\max\{f(x-z) + b(z)\}$$

   - $z = 0 \Rightarrow f(10-0) = \mathbf{f(10)}$
   - $z = 1 \Rightarrow f(10-1) = \mathbf{f(9)}$

   A causa del segno negativo ($-z$), avanzare nell'elemento strutturante corrisponde a retrocedere nell'immagine, facendo sì che la dilatazione interroghi il pixel a sinistra (9).

La funzione `_viz`, utilizzata in `morph.py`, genera i vicini tramite spostamenti additivi della forma $x+z$. Per questo motivo, l'implementazione di `mm.dil0` riflette preventivamente l'elemento strutturante tramite `np.flip(B)`. Dopo la riflessione, la scansione basata su $x+z$ accede esattamente agli stessi punti definiti dall'espressione teorica $f(x-z)$ della dilatazione in Equazione 4.4.
Per elementi strutturanti simmetrici (come dischi, quadrati e croci centrate), la riflessione non altera la maschera. Per elementi asimmetrici, invece, questo passaggio è indispensabile affinché l'implementazione riproduca correttamente la definizione matematica della dilatazione e preservi la dualità erosione–dilatazione.

Dualità erosione–dilatazione

L'erosione e la dilatazione sono **duali rispetto al complemento**. Ciò significa che un operatore può essere completamente ottenuto dall'altro, purché si agisca sul complemento dell'immagine utilizzando l'elemento strutturante riflesso $\hat{B}$:

$$(A \ominus B)^c = A^c \oplus \hat{B} \quad \Longleftrightarrow \quad A \ominus B = (A^c \oplus \hat{B})^c$$

Analogamente, anche la **dilatazione può essere ottenuta dall'erosione**:

$$(A \oplus B)^c = A^c \ominus \hat{B} \quad \Longleftrightarrow \quad A \oplus B = (A^c \ominus \hat{B})^c$$

In termini pratici, l'erosione di un oggetto può essere ottenuta tramite la dilatazione del suo complemento, seguita dalla complementazione del risultato (e viceversa). Nell'implementazione del pacchetto `morph.py`, le versioni didattiche `mm.ero0` e `mm.dil0` rendono esplicita questa struttura mediante cicli (*loop*), mentre `mm.ero` e `mm.dil` delegano le operazioni a OpenCV per una maggiore efficienza computazionale.

Condizioni al contorno e immagini finite

Nella morfologia matematica classica, definita su un dominio infinito (tipicamente $\mathbb{Z}^2$), questa dualità è esatta. Nelle immagini digitali, tuttavia, si lavora con matrici finite, e il risultato dipende dal modo in cui vengono trattati i pixel situati al di fuori dell'immagine.
Affinché le identità di dualità rimangano valide, il complemento deve essere definito rispetto allo stesso universo e le condizioni al contorno adottate per l'erosione e per la dilatazione devono essere complementari tra loro. Ad esempio, se l'erosione assume che i pixel esterni appartengano all'oggetto (255), allora la dilatazione applicata al complemento deve assumere che questi stessi pixel esterni appartengano allo sfondo (0).
Quando vengono utilizzate diverse strategie di riempimento (replicazione, riflessione, valore costante, ecc.), la dualità teorica può cessare di essere soddisfatta esattamente nelle regioni prossime ai bordi dell'immagine.

Per illustrare numericamente gli operatori morfologici e la dualità erosione–dilatazione, la Figura 4.4 presenta un'immagine binaria 10×10 elaborata con un elemento strutturante a forma di "L". Nell'implementazione di `morph.py`, l'origine di B è fissata nel centro geometrico della maschera — posizione $(1, 1)$ per un *kernel* 3×3 — e deve corrispondere a un elemento attivo affinché l'erosione si comporti correttamente (come discusso in precedenza). L'elemento strutturante B_L definito di seguito soddisfa questa condizione. La Figura 4.5 completa l'analisi con un simulatore interattivo dell'erosione, consentendo di visualizzare lo spostamento dell'elemento strutturante sull'immagine e di identificare le posizioni in cui esso rimane completamente contenuto nell'oggetto.

```
A = np.array([
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,0,1,1,1,0,0,0,0],
    [0,0,1,1,1,1,1,0,0,0],
    [0,1,1,1,1,1,1,1,0,0],
    [0,1,1,1,1,1,1,1,0,0],
    [0,1,1,1,1,1,1,0,0,0],
    [0,0,1,1,1,1,1,1,0,0],
    [0,0,0,1,1,1,1,0,0,0],
    [0,0,0,0,1,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0]], dtype=np.uint8) * 255

B_L   = np.array([[1,0,0],[1,1,0],[1,1,0]], dtype=np.uint8)   # 'L', origine al centro
B_hat = np.array([[0,1,1],[0,1,1],[0,0,1]], dtype=np.uint8)   # B_L ruotato di 180° (B)

print("Elemento strutturante B_L:"); print(mm.drawImg(B_L))
print("Elemento strutturante riflesso B:"); print(mm.drawImg(B_hat))

img_ero0 = mm.ero0(A, B_L)

# Dualità: (A   B)  == A   B
A_c    = mm.neg(A)
ero_c  = mm.neg(img_ero0)
dil_Ac = mm.dil0(A_c, B_hat)
print(f"Dualità (A   B)  == A   B : {np.array_equal(ero_c, dil_Ac)}")

mm.show(
    [A, A_c, img_ero0, ero_c, dil_Ac],
    titles=["A", "A ", "A   B", "(A   B) ", "A   B"],
    cols=5, figsize=(15, 3)
)
```

```
Elemento strutturante B_L:
1 0 0
```

```
1 1 0
1 1 0

Elemento strutturante riflesso B:
0 1 1
0 1 1
0 0 1

Dualità (A   B) == A   B : True
```

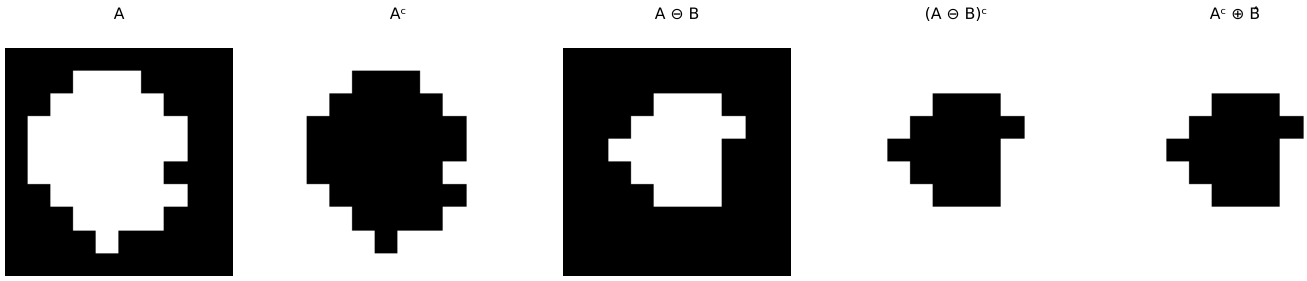
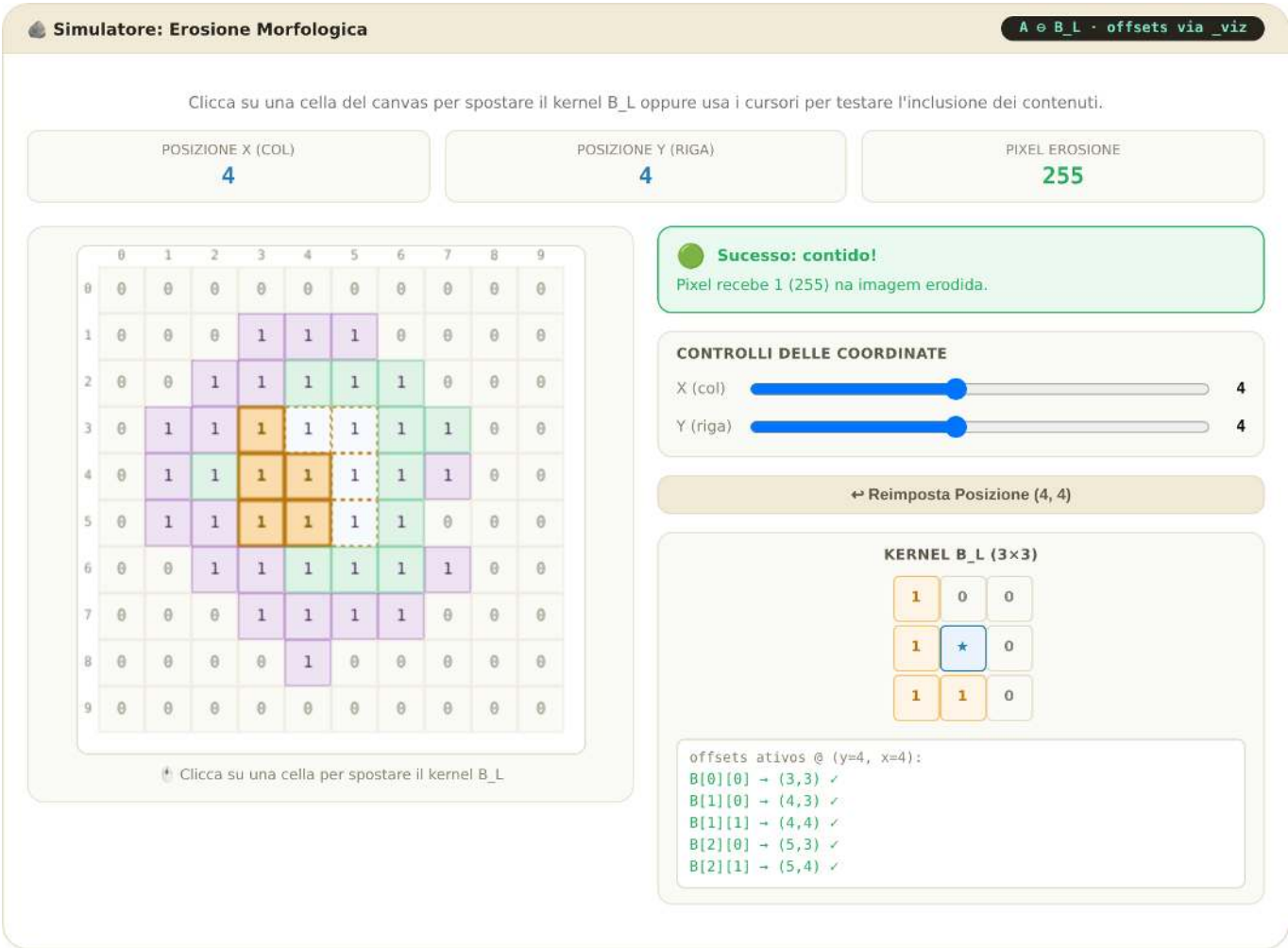


Figura 4.4: Erosione e dilatazione su immagine binaria 10×10 con elemento strutturante 'L' asimmetrico 3×3. Validazione della dualità erosione–dilatazione.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-04-erosao.html

Figura 4.5: Simulatore: Erosione Morfologica (A B_L)

4.3.2 Apertura e Chiusura

Combinando erosione e dilatazione si ottengono due operatori di grande utilità pratica: l'**apertura** e la **chiusura**, definiti dalle Equazioni Equazione 4.5 e Equazione 4.6. I loro principali effetti sono riassunti nella Tabella 4.2.

Apertura (*opening*) — erosione seguita da dilatazione con lo stesso B:

$$A \circ B = (A \ominus B) \oplus B \tag{4.5}$$

Chiusura (*closing*) — dilatazione seguita da erosione con lo stesso B:

$$A \bullet B = (A \oplus B) \ominus B \tag{4.6}$$

In pratica, `mm.open` e `mm.close` applicano lo stesso elemento strutturante nelle due fasi. Per elementi strutturanti simmetrici (i più comuni), questa implementazione coincide con la definizione matematica presentata sopra.

Tabella 4.2: Proprietà di apertura e chiusura.

Operatore	Sequenza	Effetto principale
Apertura $A \circ B$	erosione $\rightarrow$ dilatazione	Rimuove strutture incapaci di contenere l'elemento strutturante; smussa i contorni esterni
Chiusura $A \bullet B$	dilatazione $\rightarrow$ erosione	Riempie buchi più piccoli di B; smussa i contorni interni

Proprietà importante: entrambi sono **idempotenti**. Per esempio,

$$(A \circ B) \circ B = A \circ B,$$

cioè, dopo la prima applicazione, nuove applicazioni dello stesso operatore non modificano più il risultato.

4.3.2.1 Implementazione dell'apertura e della chiusura

A differenza dell'erosione e della dilatazione, l'apertura e la chiusura non introducono nuovi meccanismi computazionali. Entrambe sono ottenute tramite la composizione sequenziale degli operatori primitivi già presentati:

```
def open0(f, B):
    return mm.dil0(mm.ero0(f, B), B)

def close0(f, B):
    return mm.ero0(mm.dil0(f, B), B)
```

La funzione `mm.open` delega l'operazione a `mm.open(f, B)`, mentre `mm.close` utilizza `mm.close(f, B)`, producendo lo stesso risultato in modo più efficiente.

L'apertura eredita dall'erosione la capacità di rimuovere strutture più piccole dell' elemento strutturante e dalla dilatazione il ripristino parziale delle regioni conservate. La chiusura realizza il processo inverso: prima espande gli oggetti e poi ripristina le loro dimensioni originali, colmando lacune e buchi più piccoli dell'elemento strutturante.

4.3.2.2 Filtro Sequenziale Alternato

Nella pratica, l'apertura e la chiusura vengono spesso applicate in sequenza per rimuovere simultaneamente il rumore esterno e riempire i fori interni. La funzione `mm.asf` (*Alternating Sequential Filter*) generalizza questa strategia applicando aperture e chiusure alternativamente con elementi strutturanti progressivamente più grandi. Le sequenze disponibili sono presentate nella Tabella 4.3.

Tabella 4.3: Sequenze del filtro sequenziale alternato mm.asf.

Sequenza	Ordine	Uso tipico
`'OC'`	apertura → chiusura	rimuove il rumore esterno prima di riempire piccoli fori
`'CO'`	chiusura → apertura	riempie piccoli fori prima di rimuovere il rumore esterno
`'OCO'`	apertura → chiusura → apertura	enfatizza la rimozione del rumore esterno
`'COC'`	chiusura → apertura → chiusura	enfatizza il riempimento di fori e lacune

Il parametro `n` controlla il numero di scale utilizzate dal filtro. In ogni iterazione i, l'elemento strutturante viene ampliato mediante somma di Minkowski (`mm.sesum(b, i)`), producendo una sequenza di filtri morfologici sempre più comprensivi. A differenza di una singola apertura o chiusura con un elemento strutturante grande, l'ASF realizza una levigatura progressiva su più scale, preservando meglio la geometria degli oggetti rilevanti mentre elimina le strutture più piccole. La Figura 4.6 presenta un esempio di applicazione dell'apertura, della chiusura e dell'ASF sull'immagine delle monete.

```
img_bin    = mm.threshold(img_clahe)
B_disk     = mm.sedisk(19)

img_open  = mm.open(img_bin, B_disk)             # erosione → dilatazione
img_close = mm.close(img_bin, B_disk)            # dilatazione → erosione
img_oc    = mm.close(img_open, B_disk)           # apertura seguita da chiusura
img_asf   = mm.asf(img_bin, 'OC', mm.sedisk(3), n=7)
# ASF: disco base 3×3, cresce a ogni iterazione

mm.show(
    [img_bin, img_open, img_close, img_oc, img_asf],
    titles=["Binarizzazione Otsu", "Apertura (A B)", "Chiusura (A B)",
            "Apertura→Chiusura", "ASF-OC (n=7)"],
    cols=5, figsize=(15, 12)
)
```

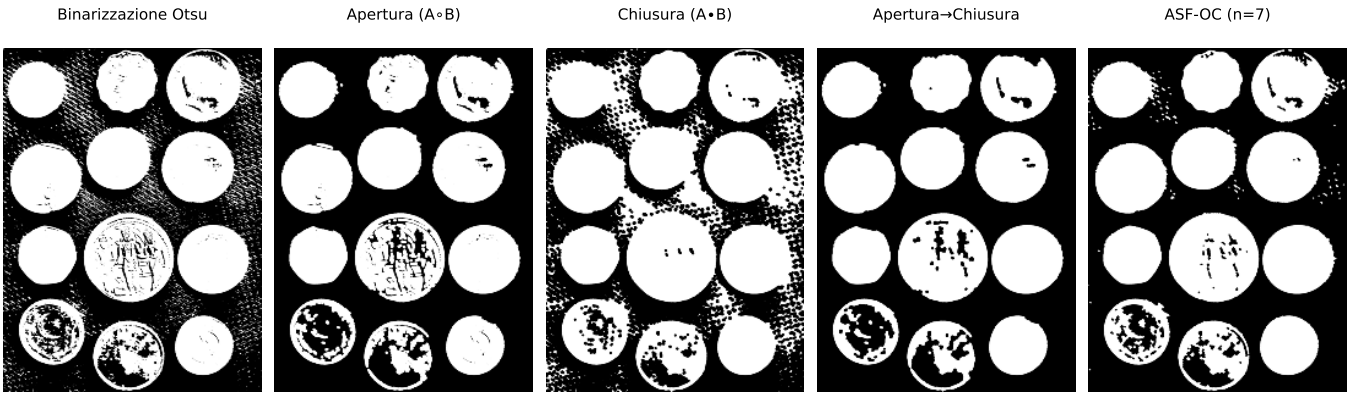


Figura 4.6: Apertura, chiusura, composizione e filtro sequenziale alternato applicati alla binarizzazione Otsu delle monete (preprocessate con miglioramento del contrasto). Elemento strutturante: disco 13×13.

4.3.3 Operatori Geodetici

Gli operatori geodetici introducono un vincolo aggiuntivo agli operatori morfologici classici tramite un'immagine di controllo denominata **maschera** g. Invece di consentire che l'erosione o la dilatazione si propaghino liberamente attraverso l'immagine, il risultato di ciascuna iterazione è limitato punto per punto dai valori della maschera, restringendo l'evoluzione dell'operazione alle regioni consentite.

4.3.3.1 Dilatazione Geodetica

La **dilatazione geodetica** di un'immagine marcatore f sotto un'immagine maschera g, utilizzando un elemento strutturante piano b, è definita da:

$$f \oplus_g b = (f \oplus b) \wedge g, \tag{4.7}$$

dove $\wedge$ rappresenta il minimo punto a punto.

In altre parole, si esegue inizialmente una dilatazione convenzionale sul marcatore e, successivamente, il risultato viene limitato dalla maschera g. In questo modo, la propagazione non può mai superare le regioni consentite dalla maschera.

La formulazione classica della dilatazione geodetica presuppone che il marcatore sia contenuto nella maschera, cioè $f \leq g$, garantendo che l'evoluzione dell'operazione rimanga sempre limitata dalla maschera.

4.3.3.2 Implementazione della dilatazione geodetica

La funzione `mm.cdil` implementa direttamente questo operatore e consente di eseguire più iterazioni consecutive:

```
def cdil(f, g, b=np.zeros((3,3),dtype='uint8'), n=1):
    """Dilatazione geodetica del marcatore f sotto la maschera g."""
    y = f.copy()
    for _ in range(n):
        y = np.minimum(mm.dil(y, b), g)
    return y
```

L'istruzione `np.minimum(mm.dil(y, b), g)` implementa esattamente la definizione matematica della dilatazione geodetica, cioè $(y \oplus b) \wedge g$.

Quando $n = 1$, la funzione esegue una singola dilatazione geodetica. Per $n > 1$, il risultato di ogni fase diventa il marcatore della fase successiva, producendo una propagazione progressiva controllata dalla maschera.

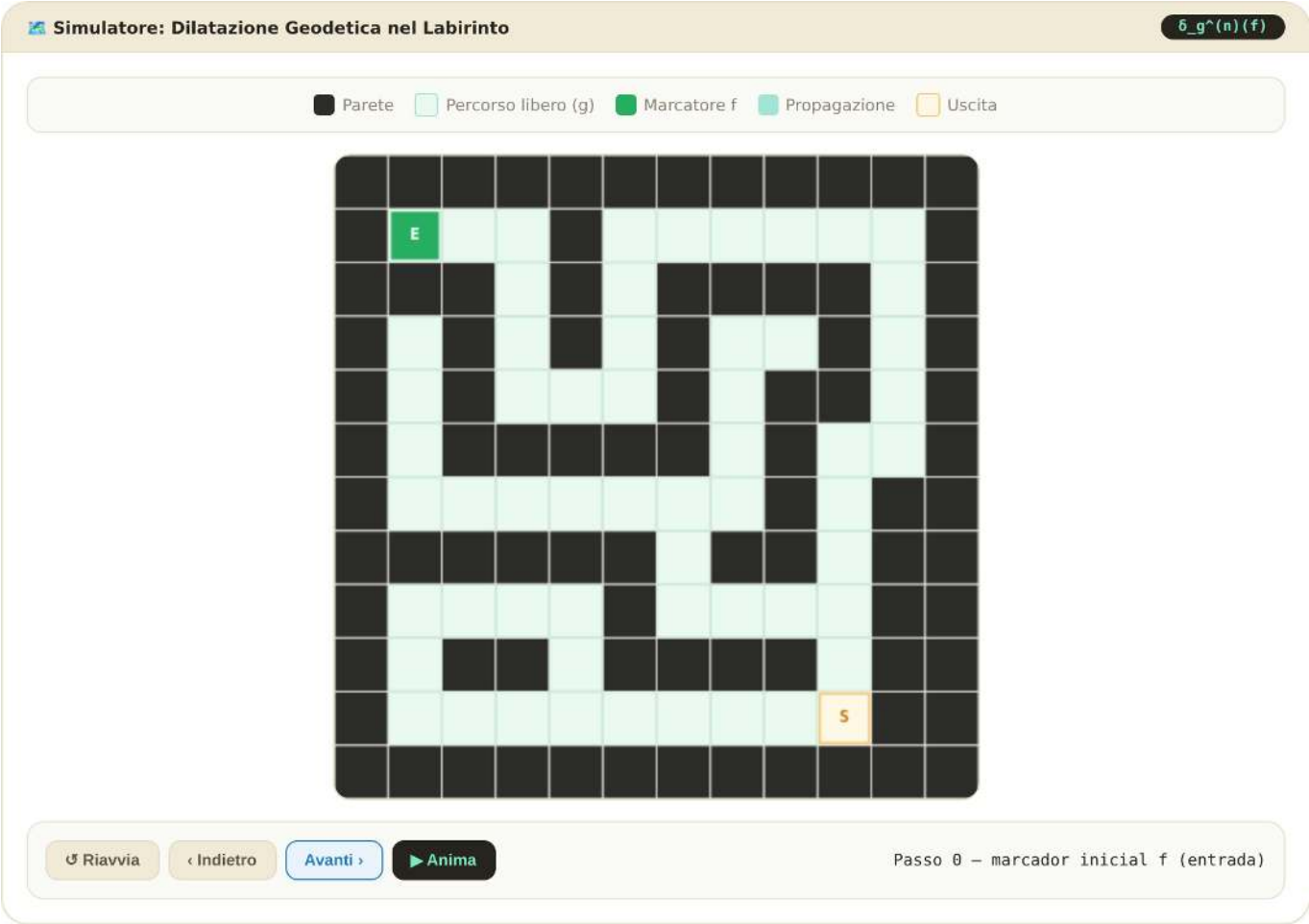
Il simulatore interattivo Figura 4.7 permette di seguire, passo dopo passo, la propagazione del marcatore f lungo i corridoi del labirinto. A ogni iterazione di `mm.cdil`, il fronte di dilatazione avanza verso le celle vicine libere — quelle in cui $g = 1$ —, mentre le pareti ($g = 0$) rimangono invalicabili. Il numero di passi necessari affinché il marcatore raggiunga l'uscita corrisponde esattamente alla lunghezza geodetica del percorso più breve all'interno della maschera, evidenziando la connessione diretta tra la dilatazione geodetica iterata e la nozione di distanza nei grafi.

4.3.3.3 Erosione Geodetica

In modo duale, l'**erosione geodetica** di un'immagine marcatore f sotto un'immagine maschera g è definita da:

$$f \ominus_g b = (f \ominus b) \vee g, \tag{4.8}$$

dove $\vee$ rappresenta l'operatore di massimo punto a punto.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-04-cdil.html

Figura 4.7: Simulatore interattivo di dilatazione geodetica: il marker f (verde) si propaga passo dopo passo attraverso i percorsi liberi della maschera g, senza attraversare pareti.

In questo caso, l'erosione convenzionale del marcatore è seguita da una restrizione inferiore imposta dalla maschera. Così, nessun pixel del risultato può assumere un valore inferiore al corrispondente pixel della maschera.

La formulazione classica dell'erosione geodetica presuppone la condizione duale

$$f \geq g,$$

cosicché la maschera agisca come limite inferiore durante tutto il processo.

4.3.3.4 Implementazione dell'erosione geodetica

La funzione statica `mm.cero` materializza questo operatore:

```
staticmethod
def cero(f, g, b=np.zeros((3,3),dtype='uint8'), n=1):
    """Erosione geodetica del marcatore f sotto la maschera g."""
    y = f.copy()
    for _ in range(n):
        y = np.maximum(mm.ero(y, b), g)
    return y
```

L'istruzione `np.maximum(mm.ero(y, b), g)` implementa direttamente l'espressione $(y \ominus b) \vee g$.

Così come per la dilatazione geodetica, il parametro n definisce quante erosioni geodetiche successive saranno calcolate prima di restituire l'immagine finale.

> **Relazione con la ricostruzione morfologica**
>
> La ricostruzione morfologica presentata nella prossima sezione è ottenuta mediante l'applicazione iterativa della dilatazione geodetica (`mm.cdil`) fino a raggiungere un punto fisso, cioè fino a quando nessuna cella cambia valore tra due iterazioni consecutive. In altre parole, la ricostruzione consiste in una sequenza di dilatazioni geodetiche successive che si propagano all'interno della maschera fino a quando non ci sono più modifiche.
> In modo duale, è anche possibile definire ricostruzioni basate sull'erosione geodetica tramite applicazioni successive di `mm.cero`.

4.3.3.5 Esempio: propagazione in un labirinto tramite dualità

La Figura 4.8 illustra la risoluzione del problema di connettività di un labirinto utilizzando la dualità morfologica tramite le funzioni `mm.cero` e `mm.suprec`.

Invece di propagare un marcatore attraverso i corridoi liberi mediante dilatazioni geodetiche, il problema viene formulato nel dominio complementare. Inizialmente, la maschera originale viene invertita,

$$g = 1 - g_{orig},$$

così che le pareti assumano valore 1 e i corridoi valore 0. In modo analogo, il marcatore viene costruito in questo stesso dominio complementare, contenente un unico valore 0 nella posizione di ingresso del labirinto e valore 1 nei restanti pixel.

Utilizzando l'elemento strutturante a croce (`mm.secross()`), l'erosione geodetica agisce sul marcatore complementato. A ogni iterazione di `mm.cero`, la regione connessa al marcatore iniziale subisce erosioni successive, mentre la maschera impone un limite inferiore che impedisce la propagazione attraverso le pareti del labirinto.

Le immagini intermedie mostrano stati dell'evoluzione dopo diversi numeri di iterazioni (`n=5`, `n=12` e `n=22`). Il risultato finale si ottiene mediante la ricostruzione geodetica per erosione (`mm.suprec`), che applica erosioni geodetiche successive fino a raggiungere un punto fisso, cioè una situazione in cui non si verifica alcuna ulteriore modifica tra due iterazioni consecutive.

Nel dominio complementare, la regione ricostruita corrisponde esattamente all'insieme di corridoi connessi all'ingresso del labirinto. Pertanto, la connettività tra ingresso e uscita può essere determinata direttamente a partire dall'immagine ricostruita.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

# 1. Maschera originale g (0 = Parete, 1 = Corridoio)
g_orig = np.array([
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 1, 1, 0, 1, 1, 1],
    [0, 0, 0, 0, 0, 1, 0, 1, 0, 1],
    [0, 1, 1, 1, 0, 1, 1, 1, 0, 1],
    [0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
    [0, 1, 0, 1, 1, 1, 1, 1, 1, 1],
    [0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 0, 1, 0],
    [0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]], dtype=np.uint8)

# Inversione globale all'inizio (Dualità Morfologica)
g = 1 - g_orig  # Ora: 1 = Parete, 0 = Corridoio
f = np.ones((10, 10), dtype=np.uint8)
f[0, 1] = 0     # Seme iniettato come 0 nel corridoio

# Elemento strutturante a croce
B_cruz = mm.secross()

# Elaborazione diretta usando mm.cero e mm.suprec nel dominio invertito
passo_5    = mm.cero(f, g, B_cruz, n=5)
passo_12   = mm.cero(f, g, B_cruz, n=12)
passo_22   = mm.cero(f, g, B_cruz, n=22)
ponto_fixo = mm.suprec(f, g, B_cruz)

# --- CONFIGURAZIONE DELLA VISUALIZZAZIONE GRAFICA ---

titles = [
    "Máscara (~g)", "Marcador (~f)",
    "Avanço (n=5)", "Avanço (n=12)",
    "Avanço (n=22)", "~mm.suprec"
]
images = [g, f, passo_5, passo_12, passo_22, ponto_fixo]

fig, axes = plt.subplots(1, 6, figsize=(16, 4), facecolor='#fcfcfc')

# Mappa di colori adattata al dominio complementare:
# Nel dominio invertito: 1 = Parete (Blu Scuro)
# Dove g == 0 e immagine == 1 = Corridoio Libero (Grigio Chiaro)
# Dove g == 0 e immagine == 0 = Onda Geodetica Attiva (Oro)
cmap_pipeline = mcolors.ListedColormap(['#ffc13b', '#e0e0e0', '#1e3d59'])

for i, ax in enumerate(axes):
    if i == 0 or i == 1:
        # Per le condizioni iniziali (~g e ~f)
        cmap_init = mcolors.ListedColormap(['#e0e0e0', '#1e3d59'])
```

```
        ax.imshow(images[i], cmap=cmap_init, vmin=0, vmax=1)
    else:
        # Rendering basato sugli stati complementari
        render_step = np.zeros_like(g, dtype=np.uint8)
        render_step[g == 1] = 2           # Parete (1 originale della mappa di colori)
        render_step[g == 0] = 1           # Corridoio standard
        render_step[images[i] == 0] = 0   # Onda attiva (0 originale della mappa di colori)
        ax.imshow(render_step, cmap=cmap_pipeline, vmin=0, vmax=2)

    ax.set_title(titles[i], fontsize=10, fontweight='bold', color='#1e3d59', pad=12)
    ax.set_xticks(np.arange(-0.5, 10, 1), minor=True)
    ax.set_yticks(np.arange(-0.5, 10, 1), minor=True)
    ax.grid(which='minor', color='#ffffff', linestyle='-', linewidth=1)
    ax.tick_params(which='both', bottom=False, left=False, labelbottom=False, labelleft=False)

    # Indicatori grafici di Ingresso e Uscita
    ax.plot(1, 0, marker='v', color='#2ecc71', markersize=7, markeredgewidth=1.5)
    ax.plot(8, 9, marker='o', color='#e74c3c', markersize=6, fillstyle='none',
markeredgewidth=2)

plt.tight_layout()
plt.show()
```

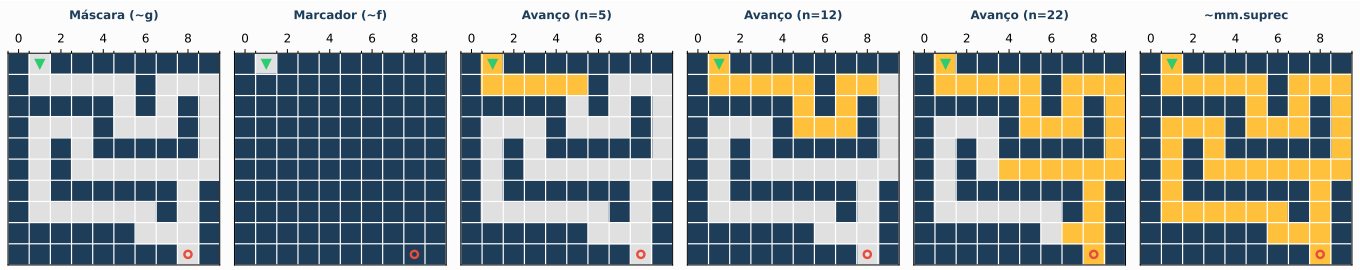


Figura 4.8: Risoluzione del labirinto tramite Operazioni Complementari: invertendo la maschera e il marker all'inizio del *pipeline*, il flusso viene risolto direttamente nel dominio complementare tramite mm.cero e mm.suprec, eliminando le re-inversioni ridondanti.

4.3.4 Ricostruzione Morfologica

La **ricostruzione morfologica** propaga un'immagine **marcatore** f all'interno di un'immagine **maschera** g, garantendo che il risultato non superi mai i valori di intensità imposti dalla maschera. L'operatore fondamentale che rende possibile questa propagazione contenuta è la **dilatazione geodetica**, definita dalla Equazione 4.7.

La ricostruzione si ottiene applicando in modo iterativo questa dilatazione condizionata. Inizialmente, il marcatore è limitato dalla maschera per stabilire lo stato iniziale:

$$X^{(0)} = f \wedge g,$$

e le iterazioni successive sono definite in modo ricorsivo da:

$$X^{(k)} = (X^{(k-1)} \oplus b) \wedge g.$$

La sequenza cresce monotonamente fino a raggiungere un punto fisso, producendo la **ricostruzione morfologica per dilatazione** (nota anche in letteratura come *inf-ricostruzione*):

$$R_g^{\delta}(f) = \lim_{k\to\infty} X^{(k)} = X^{(k)} \quad \text{quando} \quad X^{(k)} = X^{(k-1)}. \tag{4.9}$$

La salita iterativa viene interrotta non appena si raggiunge la stabilità, cioè quando due iterazioni consecutive producono matrici con valori assolutamente identici.

4.3.4.1 Implementazione della ricostruzione morfologica

La routine didattica `mm.infrec` implementa direttamente l'algoritmo iterativo a punto fisso. Inizialmente, il marker effettivo iniziale $X^{(0)}$ è determinato tramite l'operazione `np.minimum(f, g)`. Per garantire che il ciclo di verifica esegua la prima iterazione senza innescare false convergenze premature, la variabile di controllo dell'iterazione precedente (`y1`) è inizializzata con un valore sentinella al di fuori del dominio dei dati (o semplicemente con una matrice che forzi la prima esecuzione).

```
def infrec(f, g, b=np.zeros((3,3), dtype='uint8')):
    """Inf-ricostruzione: dilata il marker (f   g) fino a convergenza sotto la maschera g."""
    y = np.minimum(f, g)
    # Inizializza y1 con valori impossibili per forzare l'ingresso nel ciclo
    y1 = np.full_like(f, 256, dtype=np.int16)
    while not np.array_equal(y, y1):
        y1 = y.copy()
        # Applica la dilatazione geodetica: (y   b)   g
        y = np.minimum(mm.dil(y, b), g)
    return y.astype('uint8')
```

All'interno del ciclo `while`, la variabile `y` memorizza la stima corrente della ricostruzione $X^{(k)}$, mentre `y1` preserva l'immagine dello stadio immediatamente precedente $X^{(k-1)}$. L'istruzione di controllo condizionale `np.minimum(mm.dil(y, b), g)` traduce fedelmente la dilatazione geodetica teorica, in cui l'espansione morfologica convenzionale comandata da OpenCV viene immediatamente "potata" e limitata dalle barriere di intensità della maschera g. Il ciclo termina quando nessuna modifica dei pixel viene registrata tra i passi.

4.3.4.2 Vantaggi della ricostruzione morfologica

La ricostruzione morfologica è significativamente più robusta dell'apertura convenzionale perché è in grado di rimuovere strutture indesiderate senza distorcere o alterare la morfologia degli oggetti che devono essere preservati.

Mentre l'apertura classica smussa gli angoli, elimina le punte e deforma i contorni a causa dell'imposizione geometrica rigida dell'elemento strutturante, la ricostruzione geodesica utilizza la maschera per recuperare con precisione i limiti e le forme originali degli oggetti che presentano connettività con il marcatore iniziale.

In termini intuitivi, il marcatore agisce come un seme di contagio che si espande progressivamente, ma solo percorrendo le regioni consentite dalla maschera. I componenti che non presentano alcuna intersezione con il marcatore non verranno mai ricostruiti (venendo eliminati), mentre i componenti toccati dal seme si espandono fino a ripristinare integralmente la loro geometria originale.

Questo comportamento discriminatorio e conservativo è illustrato nella Figura 4.9, utilizzando l'elemento strutturante a croce presentato nella Figura 4.3..

```
g = np.array([
    [0,0,0,0,0,0,0,0,0,0],
    [0,1,1,1,0,0,0,1,1,0],
    [0,1,1,1,1,0,0,1,1,0],
    [0,1,1,1,1,0,0,0,0,0],
    [0,1,1,1,1,0,0,0,0,0],
    [0,1,1,1,0,0,0,0,0,0],
    [0,0,1,1,1,0,0,1,0,0],
    [0,0,1,1,1,0,0,1,1,0],
    [0,0,0,1,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,0]], dtype=np.uint8) * 255

B_cruz = mm.secross()
```

```
f = mm.ero(g, mm.sebox())          # marcatore: nucleo dell'oggetto principale

iteracoes, titulos = [], []
img_atual = f
for i in range(1, 6):
    img_atual = mm.cdil(img_atual, g, B_cruz)
    iteracoes.append(img_atual)
    titulos.append(f"Dilatação Cond. (n={i})")

img_reconstruida = mm.infrec(f, g, B_cruz)
print(f"  Stabilità all'iterazione 5: {np.array_equal(img_reconstruida, iteracoes[-1])}")

mm.show(
    [g, f] + iteracoes + [img_reconstruida],
    titles=["Máscara (g)", "Marcador (f)"] + titulos + ["Reconstrução R_g(f)"],
    cols=8, figsize=(18, 3)
)
```

```
Stabilità all'iterazione 5: True
```

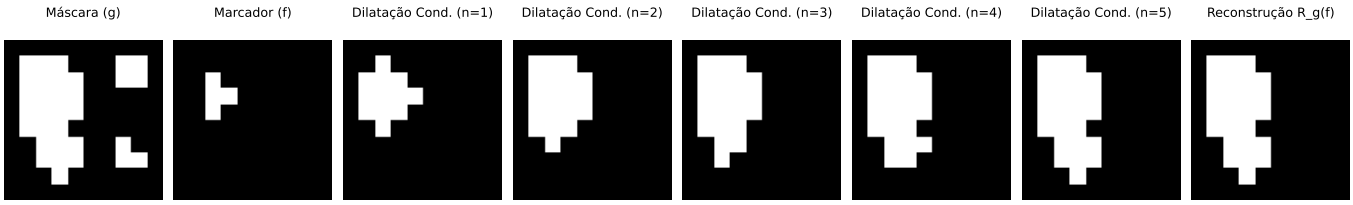


Figura 4.9: *Pipeline* di Ricostruzione Morfologica per Dilatazione Condizionata: la maschera contiene due oggetti, il marcatore isola solo il nucleo dell'oggetto principale, e le iterazioni ricostruiscono la sua forma esatta fino alla convergenza.

4.3.5 Riempimento di Buchi e Rimozione dei Bordi

Due operatori basati sulla ricostruzione morfologica completano la *pipeline* di pulizia binaria. Le loro caratteristiche principali sono riassunte nella Tabella 4.4.

Riempimento di buchi (`mm.clohole`) rimuove cavità completamente circondate dall'oggetto, indipendentemente dalla dimensione, senza alterare i contorni esterni. La procedura agisce sul complemento dell'immagine, utilizzando come marcatore una restrizione della cornice (*frame*) allo sfondo:

$$\text{clohole}(f) = \left(R^{\delta}_{f^c}(\text{frame}(f) \wedge f^c)\right)^c \tag{4.10}$$

In termini operativi, si ricostruisce prima lo sfondo esterno e, successivamente, si applica la complementazione per recuperare gli oggetti con i buchi riempiti.

Rimozione degli oggetti di bordo (`mm.edgeoff`) elimina tutti gli oggetti che toccano il bordo dell'immagine, preservando solo i componenti completamente interni. Il marcatore è ottenuto dall'intersezione tra la cornice (*frame*) e gli oggetti dell'immagine:

$$\text{edgeoff}(f) = f \setminus R^{\delta}_{f}(\text{frame}(f) \wedge f) \tag{4.11}$$

Tabella 4.4: Confronto tra gli operatori *clohole* e *edgeoff*.

Operatore	Marcatore	Maschera	Effetto
`mm.clohole`	*frame* ristretto allo sfondo (f^c)	f^c	Riempie i buchi interni
`mm.edgeoff`	*frame* ristretto all'oggetto (f)	f	Rimuove gli oggetti connessi al bordo

L'evoluzione passo passo di queste trasformazioni geodetiche può essere seguita nelle figure successive. La Figura 4.10 illustra il meccanismo di inondazione controllata dell'operatore `mm.clohole`, nel quale la ricostruzione avviene a partire dallo sfondo esterno e impedisce la propagazione verso regioni interne non connesse all'esterno, risultando nel riempimento consistente delle cavità interne. Al contrario, la Figura 4.11 dettaglia la dinamica dell'operatore `mm.edgeoff`, in cui solo i componenti connessi al bordo vengono ricostruiti e successivamente rimossi, preservando esclusivamente gli oggetti completamente contenuti all'interno dell'immagine.

La connettività della propagazione geodetica è controllata dall'elemento strutturante: `mm.sebox()` (vicinanza di 8) include connessioni diagonali, mentre `mm.secross()` (vicinanza di 4) le esclude. Di conseguenza, la scelta dell'elemento strutturante influisce su quali componenti vengono raggiunti dalla ricostruzione e, quindi, su quali verranno preservati o rimossi.

4.3.5.1 Conformità con l'implementazione

Le definizioni sopra sono direttamente allineate con l'implementazione in `morph.py`, riprodotta di seguito:

```
staticmethod
def clohole(f, b=np.ones((3,3),dtype='uint8')):
    # marcador restrito ao fundo da imagem
    marcador = mm.frame(f, border=1) & mm.neg(f)
    return mm.neg(mm.infrec(marcador, mm.neg(f), b))

staticmethod
def edgeoff(f, b=np.ones((3,3),dtype='uint8')):
    # marcador restrito aos objetos da imagem
    marcador = mm.frame(f, border=1) & f
    return mm.subm(f, mm.infrec(marcador, f, b))
```

Queste implementazioni rendono esplicito che entrambi gli operatori sono istanze dirette di ricostruzione morfologica per dilatazione geodetica con `mm.infrec`, differendo solo nella scelta del marcatore e della maschera: `clohole` opera sul complemento dell'immagine, mentre `edgeoff` opera direttamente sul dominio degli oggetti.

```
f = np.array([
    [0,0,0,0,0,0,0,0,0,0],
    [0,1,1,1,1,0,0,0,0,1],
    [0,1,0,0,1,0,0,1,0,1],
    [0,1,0,0,1,0,0,1,0,1],
    [0,1,1,1,1,0,0,1,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,1,1,1,0,1,1,1,0],
    [0,0,1,1,1,0,1,0,1,0],
    [0,0,1,1,1,0,1,1,1,0],
    [0,0,0,0,0,0,0,0,0,1]], dtype=np.uint8) * 255

B_cruz = mm.secross()
f_c = mm.neg(f)
marcador_ch = mm.frame(f, border=1)       # bordo esterno
```

```
iteracoes, titulos = [], []
img_atual = marcador_ch
for i in range(1, 6):
    img_atual = mm.cdil(img_atual, f_c, B_cruz)
    iteracoes.append(img_atual)
    titulos.append(f"Iter. (n={i})")

img_clohole = mm.neg(mm.infrec(marcador_ch, f_c, B_cruz))
print(f"  Validazione clohole: {np.array_equal(img_clohole, mm.clohole(f))}")

mm.show(
    [f, marcador_ch] + iteracoes + [img_clohole],
    titles=["f original", "Marcador (borda)"] + titulos + ["clohole(f)"],
    cols=8, figsize=(18, 3), axis=True
)
```

```
Validazione clohole: True
```

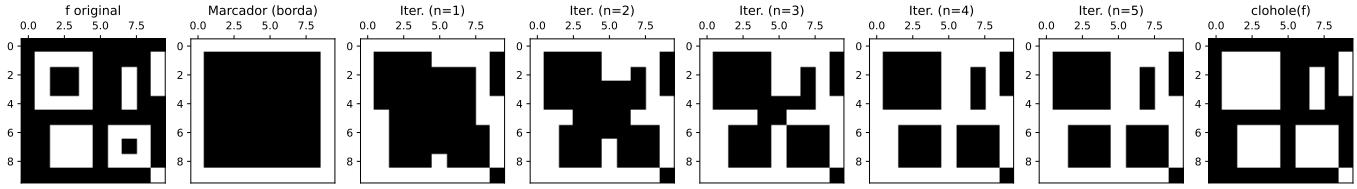


Figura 4.10: *Pipeline* di riempimento dei buchi (*clohole*): il marker proviene dal bordo dell'immagine, ristretto al complemento f^c. La dilatazione geodesica ricostruisce lo sfondo esterno; dopo la complementazione, i buchi interni vengono riempiti.

```
f = np.array([
    [0,0,0,0,0,0,0,0,0,0],
    [0,1,1,1,1,0,0,0,0,1],
    [0,1,0,0,1,0,0,1,0,1],
    [0,1,0,0,1,0,0,1,0,1],
    [0,1,1,1,1,0,0,1,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,0,1,1,1,0,1,1,1,0],
    [0,0,1,1,1,0,1,0,1,0],
    [0,0,1,1,1,0,1,1,1,0],
    [0,0,0,0,0,0,0,0,0,1]], dtype=np.uint8) * 255

B_box = mm.sebox()
marcador_eo = mm.band(mm.frame(f, border=1), f)   # borda   f

iteracoes, titulos = [], []
img_atual = marcador_eo
for i in range(1, 6):
    img_atual = mm.cdil(img_atual, f, B_box)
    iteracoes.append(img_atual)
    titulos.append(f"Iter. (n={i})")

img_edgeoff = mm.edgeoff(f, B_box)

mm.show(
    [f, marcador_eo] + iteracoes + [img_edgeoff],
    titles=["f original", "Marcador (borda)"] + titulos + ["edgeoff(f)"],
    cols=8, figsize=(18, 3)
)
```

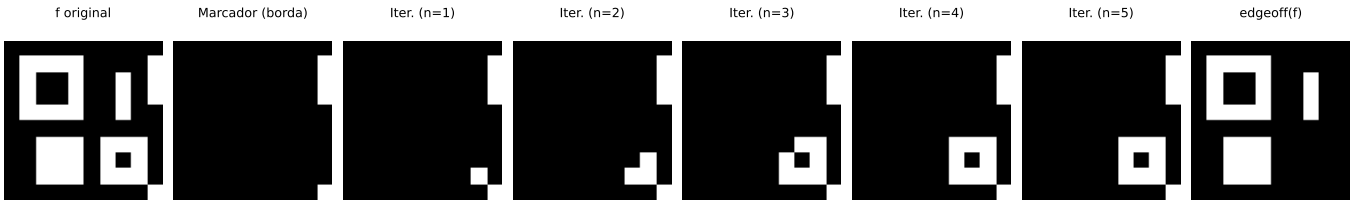


Figura 4.11: *Pipeline* di eliminazione delle strutture di bordo (*edgeoff*): il marcatore cattura le radici connesse alle estremità, la ricostruzione delimita questi elementi e la sottrazione preserva solo gli oggetti completamente interni.

4.3.6 *Pipeline* di Pulizia Binaria con CLAHE

Sulla base dell'analisi precedente — in cui il CLAHE ha prodotto il valore più alto della varianza interclassi ($\sigma_B^2 \approx 2{,}47 \times 10^3$), vedi Figura 4.2, e l'operatore `mm.clohole` si è mostrato efficace nel riempimento delle cavità interne —, il *pipeline* finale di segmentazione, illustrato in Figura 4.12, è strutturato dal seguente flusso computazionale:

$$\text{gray} \xrightarrow{\text{CLAHE}} \xrightarrow{\text{Otsu}} \xrightarrow{\text{open}} \xrightarrow{\text{clohole}} \xrightarrow{\text{open}} \xrightarrow{\text{edgeoff}} \text{segmentazione}$$

Dopo la fase di `mm.clohole`, si applica una seconda apertura morfologica con un elemento strutturante più grande (`mm.sedisk(33)`, disco di diametro 33). Questa operazione rimuove piccole regioni residue e artefatti che potrebbero essere rimasti dopo la segmentazione. In particolare, il riempimento geodetico può trasformare piccole cavità isolate in componenti connesse all'oggetto, rendendo conveniente un'ulteriore fase di filtraggio basata sulla dimensione. Il diametro è stato scelto in modo che le monete rimangano in grado di contenere l'elemento strutturante, mentre componenti significativamente più piccole vengano eliminate.

In questa immagine, nessuna moneta è connessa al bordo della matrice. Di conseguenza, l'applicazione di `mm.edgeoff` non altera il risultato ottenuto dopo la seconda apertura. Tuttavia, questa fase è mantenuta nel *pipeline* per robustezza, poiché in altre immagini possono esistere oggetti parzialmente visibili o connessi ai bordi, che devono essere rimossi prima della fase di analisi.

Perché l'apertura dopo il clohole?

L'operatore `mm.clohole` riempie tutte le cavità chiuse presenti negli oggetti segmentati. In alcune situazioni, piccole regioni indesiderate possono rimanere dopo questa fase o diventare connesse agli oggetti principali. L'apertura morfologica successiva rimuove componenti più piccole dell'elemento strutturante, preservando le monete grazie alla loro dimensione significativamente maggiore.

```
#   Pre-processamento: solo CLAHE
img_clahe0 = mm.clahe(img_coins_gray, 2.0, 8)

#   Fase 1: Binarizzazione Otsu
img_bin = mm.threshold(img_clahe0)

#   Fase 2: Apertura - rimuove i rumori bianchi di sfondo
img_open = mm.open(img_bin, mm.sedisk(9))

#   Fase 3: clohole - chiude tutti i buchi interni
img_hole = mm.clohole(img_open)

#   Fase 4: Apertura (kernel grande) - rimuove gli artefatti del clohole
img_limpo = mm.open(img_hole, mm.sedisk(33))

#   Fase 5: edgeoff - rimuove gli oggetti che toccano il bordo
img_final = mm.edgeoff(img_limpo, border=1)
```

```
mm.show(
    [img_clahe0,  img_bin,              img_open,
     img_hole,    img_limpo,            img_final],
    titles=["CLAHE",            "Otsu",                 "Apertura (r=9)",
            "clohole",          "Apertura (r=33)",      "Finale (edgeoff)"],
    cols=6, figsize=(18, 6)
)
```

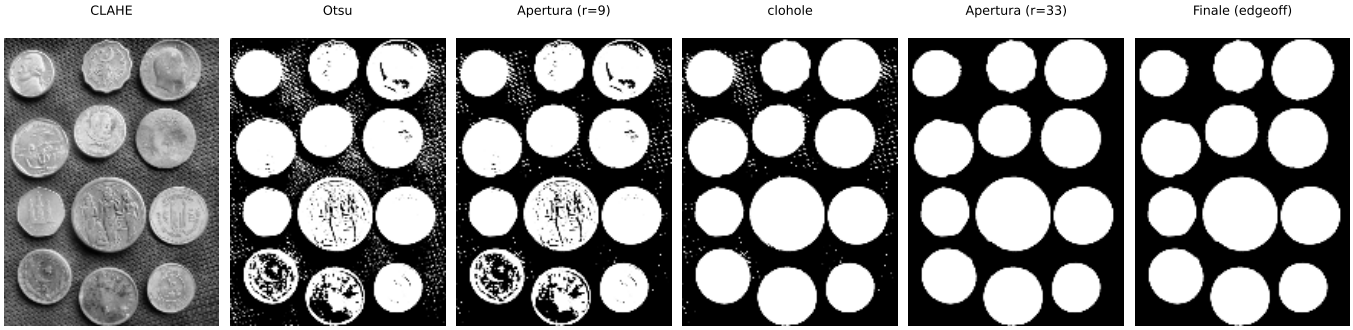


Figura 4.12: *Pipeline* completo di segmentazione con CLAHE: Otsu → apertura (r=9) → clohole → apertura (r=33) → edgeoff.

4.3.7 Morfologia in Toni di Grigio

Gli operatori morfologici si estendono naturalmente alle immagini in toni di grigio. In questa formulazione, l'erosione e la dilatazione agiscono direttamente sui livelli di intensità dell'immagine. Per elementi strutturanti piatti ($b \equiv 0$), l'erosione corrisponde al **minimo locale** e la dilatazione al **massimo locale** all'interno del vicinato definito dall'elemento strutturante.

L'interpretazione intuitiva è semplice: l'erosione **scurisce** le regioni sostituendo ogni pixel con il valore più piccolo presente nel suo vicinato, mentre la dilatazione **schiarisce** le regioni utilizzando il valore più grande disponibile. La combinazione di questi operatori consente di costruire trasformazioni in grado di evidenziare i bordi, rimuovere le tendenze di illuminazione e mettere in risalto le strutture locali.

Tre operatori derivati sono particolarmente utili:

Gradiente morfologico — evidenzia i bordi come differenza tra dilatazione ed erosione:

$$\text{grad}_B(f) = (f \oplus B) - (f \ominus B) \tag{4.12}$$

Top-hat — evidenzia le strutture brillanti più piccole dell'elemento strutturante (differenza tra l'immagine originale e la sua apertura):

$$\text{top-hat}_B(f) = f - (f \circ B) \tag{4.13}$$

Black-hat — evidenzia le strutture scure più piccole dell'elemento strutturante (differenza tra la chiusura e l'immagine originale):

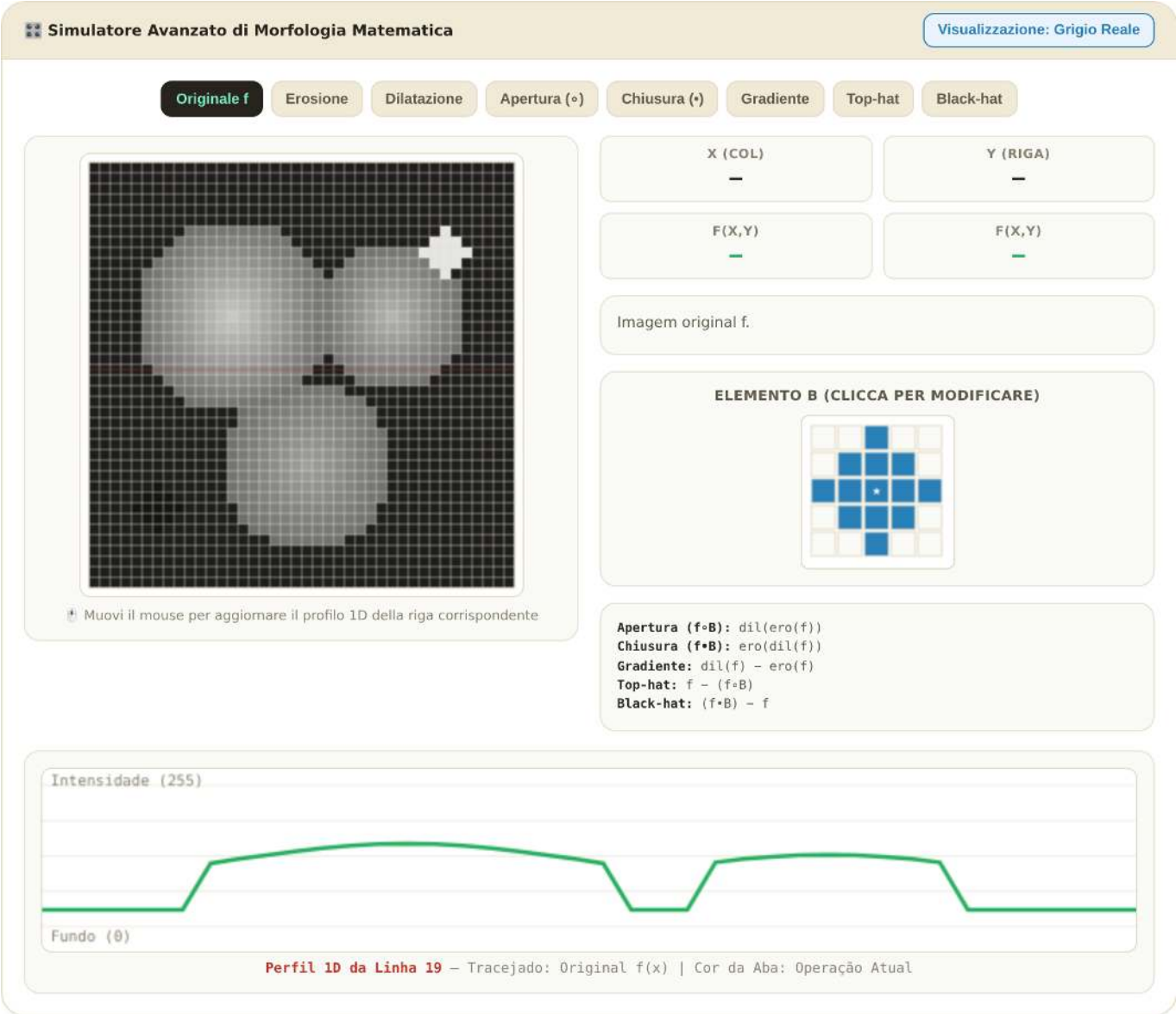
$$\text{black-hat}_B(f) = (f \bullet B) - f \tag{4.14}$$

Il *Top-hat* estrae i dettagli brillanti che non sopravvivono all'apertura, mentre il *Black-hat* evidenzia i dettagli scuri rimossi dalla chiusura. Il gradiente morfologico, invece, mette in risalto le transizioni brusche di intensità, producendo una rappresentazione simile a quella di un rivelatore di bordi.

Per comprendere il meccanismo di questi operatori a livello locale, la Figura 4.13 presenta un simulatore interattivo di morfologia in toni di grigio. Il simulatore consente di modificare liberamente l'elemento strutturante, visualizzare il suo spostamento sull'immagine e seguire simultaneamente il profilo unidimensionale delle intensità. In questo modo, diventa possibile osservare direttamente come l'erosione seleziona i minimi locali, come la dilatazione seleziona i massimi locali e come il gradiente morfologico emerge dalla differenza tra questi due operatori.

Il pulsante situato nell'angolo superiore destro consente di alternare tra la visualizzazione originale in toni di grigio e una rappresentazione pseudocolorata (*colormap*) esclusivamente per i tre tipi di gradiente. La versione colorata facilita la percezione visiva delle variazioni di intensità, rendendo più evidente l'azione degli operatori morfologici su massimi, minimi e transizioni locali dell'immagine.

Gli operatori presentati sono disponibili in `morph.py` tramite le funzioni `mm.gradm`, `mm.tophat` e `mm.blackhat`.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-04-operadores-av.html

Figura 4.13: Simulatore interattivo avanzato di morfologia con elemento strutturante modificabile e profilo 1D.

La Figura 4.14 illustra gli effetti di questi operatori sull'immagine delle monete e sui rispettivi istogrammi. Si osservi che l'erosione sposta la distribuzione verso intensità più basse, mentre la dilatazione la sposta verso intensità più alte. Il gradiente concentra i valori nelle regioni di contorno, e gli operatori *Top-hat* e *Black-hat* producono istogrammi fortemente concentrati su bassi livelli di intensità, poiché vengono evidenziate solo

piccole strutture locali.

```
import io
import matplotlib.pyplot as plt
import numpy as np

def fig2img(fig):
    b = io.BytesIO(); fig.savefig(b, format='png', dpi=100); plt.close(fig); b.seek(0)
    return (plt.imread(b)[:, :, :3] * 255).astype(np.uint8)

def plot_hist(img, title):
    fig, ax = plt.subplots(figsize=(4, 3))
    h = mm.hist(img)
    # CORREZIONE: range usa len(h) dinamicamente per combaciare con il ritorno della libreria
    ax.bar(range(len(h)), h, color='steelblue', width=1, edgecolor='steelblue')
    ax.set(xlim=(0, 255)); plt.tight_layout()
    return fig2img(fig)

# 1. Elaborazione morfologica di base
B = mm.sedisk(19)
operadores = [
    ("Original", img_coins_gray),
    ("Erosão", mm.ero(img_coins_gray, B)),
    ("Dilatação", mm.dil(img_coins_gray, B)),
    ("Gradiente Morf.", mm.gradm(img_coins_gray, B)),
    ("Top-hat", mm.tophat(img_coins_gray, B)),
    ("Black-hat", mm.blackhat(img_coins_gray, B))
]

# 2. Montaggio dinamico della coppia: [Immagine, Istogramma]
imgs, titles = [], []
for nome, img in operadores:
    imgs += [img, plot_hist(img, f"Hist. {nome}")]
    titles += [nome, f"Hist. {nome}"]

# 3. Visualizzazione finale in griglia a due colonne (Immagine | Istogramma)
mm.show(imgs, titles=titles, cols=4, figsize=(10, 8))
```

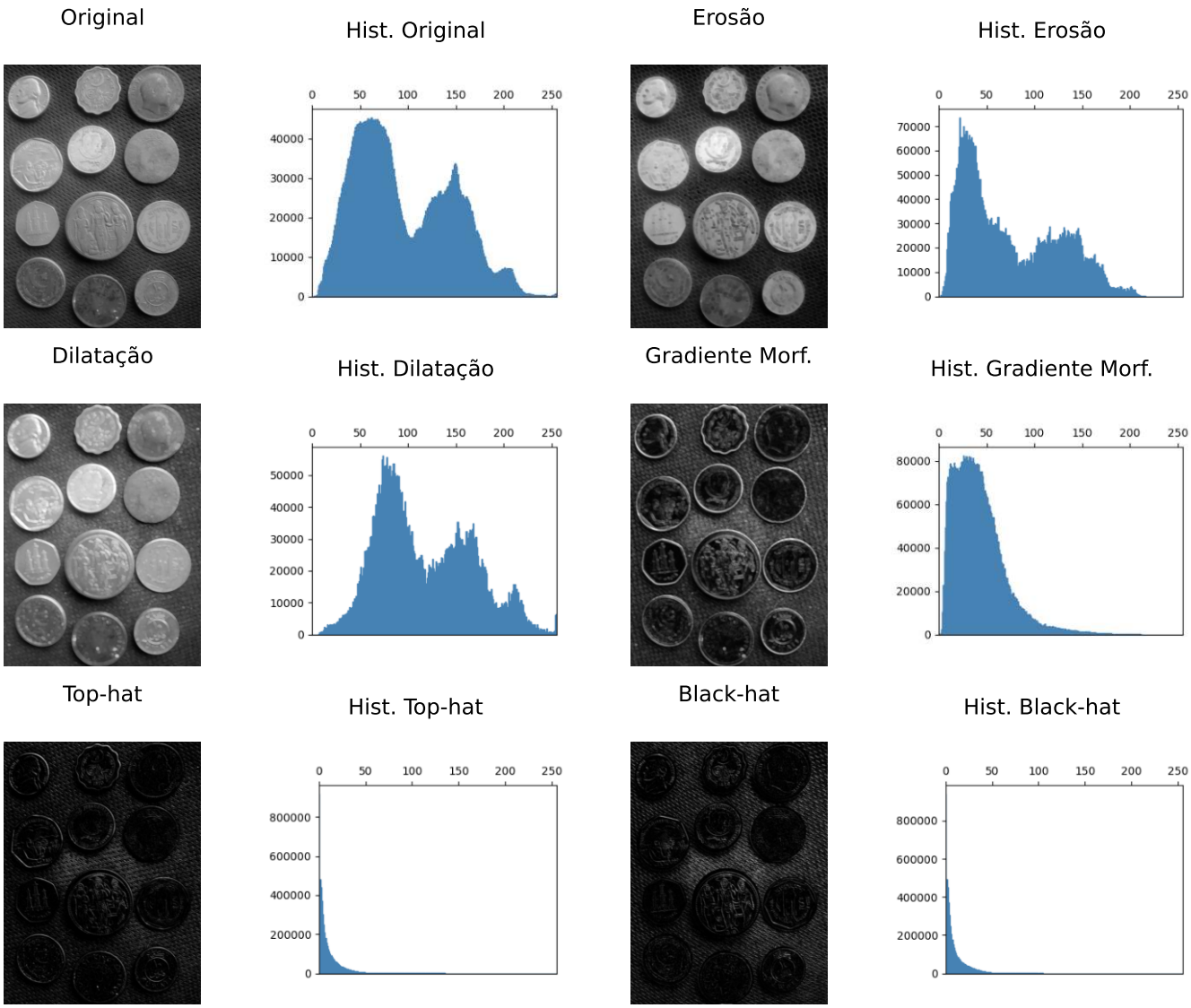


Figura 4.14: Morfologia in toni di grigio e i rispettivi istogrammi. L'erosione riduce le intensità locali, la dilatazione le amplifica, il gradiente evidenzia i bordi, e gli operatori Top-hat e Black-hat evidenziano dettagli locali luminosi e scuri. Elemento strutturante: disco di diametro 19.

Gli operatori morfologici presentati in precedenza saranno ora utilizzati come strumenti di raffinamento e generazione di marcatori per metodi di segmentazione più avanzati, presentati di seguito.

4.4 Segmentazione delle Immagini: Fondamenti e Tassonomia

La **segmentazione delle immagini** consiste nel dividere l'immagine in regioni associate a oggetti o strutture di interesse. In elaborazione digitale delle immagini (EDI), essa rappresenta la transizione tra l'elaborazione di basso livello — come filtraggio e miglioramento — e fasi di analisi più avanzate, come l'estrazione di caratteristiche, il riconoscimento e l'interpretazione della scena.

Formalmente, l'obiettivo della segmentazione consiste nel decomporre il dominio spaziale completo di un'immagine, denotato da Ω, in una partizione di sottoinsiemi $\{R_1, R_2, \dots, R_n\}$ che soddisfi simultaneamente i criteri di **completezza** e **disgiunzione**:

$$\bigcup_{i=1}^{n} R_i = \Omega, \qquad R_i \cap R_j = \emptyset \quad \forall\, i \neq j \tag{4.15}$$

Oltre alle proprietà di completezza e disgiunzione espresse nella Equazione 4.15, ciascuna sotto-regione R_i deve costituire un dominio **omogeneo** secondo un predicato di similarità definito su proprietà locali — intensità, colore o texture — e, simultaneamente, essere **distinta** dalle regioni adiacenti.

Le tecniche di segmentazione possono essere organizzate in diverse famiglie. In questo capitolo verranno enfatizzati gli approcci riassunti nella Tabella 4.5, fondati principalmente su criteri di intensità, connettività e prossimità spaziale.

Tabella 4.5: Tassonomia semplificata dei principali approcci di segmentazione e raffinamento studiati in questo capitolo.

Approccio	Criterio di Segmentazione	Operatori di Riferimento
Sogliatura	Partizionamento dello spazio delle intensità	Criterio di Otsu, sogliatura globale e locale
Morfologia Matematica	Relazioni spaziali definite da elementi/funzioni strutturanti	Erosione, dilatazione, apertura, chiusura e ricostruzione
Basata su Regioni	Omogeneità locale e connettività spaziale	Etichettatura delle componenti connesse, Trasformata della Distanza e *Watershed*

Fino a questo punto, lo sviluppo pratico si è concentrato sulla **sogliatura**, mediante la combinazione tra equalizzazione adattiva CLAHE e il metodo globale di Otsu. Questa fase è stata integrata da operatori di **ricostruzione morfologica** basati su dilatazioni geodetiche, implementati dalle funzioni `mm.infrec`, `mm.clohole` e `mm.edgeoff`, producendo una maschera binaria pulita e adeguata all'analisi.

Tuttavia, in scenari in cui oggetti distinti appaiono connessi nella maschera binaria — sia per contatto fisico, sovrapposizione parziale o per stretti ponti di pixel prodotti dalla segmentazione —, la sogliatura non è più sufficiente per individualizzare ciascun oggetto. In tali casi, molteplici oggetti finiscono per comporre un'unica componente connessa, rendendo difficili le fasi successive di misurazione e interpretazione.

Per superare questa limitazione, le prossime sezioni introducono tre strumenti complementari: l'**Etichettatura delle Componenti Connesse**, la **Trasformata della Distanza** e l'algoritmo di segmentazione ***Watershed*** basato su marcatori. Nell'insieme, queste tecniche consentono di separare oggetti adiacenti, identificare le regioni individualmente ed estrarre descrittori geometrici coerenti per l'analisi quantitativa.

4.4.1 Etichettatura

L'**etichettatura delle componenti connesse** (*connected component labeling*) è l'operatore che assegna un identificatore intero unico a ciascun insieme di pixel appartenenti alla stessa componente connessa in un'immagine binaria.

> **Definizione formale**
>
> Data un'immagine binaria f e una relazione di connettività definita da un elemento strutturante B (tipicamente connettività-4 o connettività-8), l'algoritmo di etichettatura della Figura 4.15 produce un'immagine g nella quale tutti i pixel appartenenti alla stessa componente connessa ricevono la stessa etichetta intera positiva, mentre pixel appartenenti a componenti distinte ricevono etichette diverse.

La connettività definisce quali pixel sono considerati vicini diretti di un pixel (x, y). Le definizioni più utilizzate sono:

- **Connettività-4:** considera solo i quattro vicini ortogonali (nord, sud, est e ovest).
- **Connettività-8:** considera i quattro vicini ortogonali e i quattro diagonali, per un totale di otto vicini.

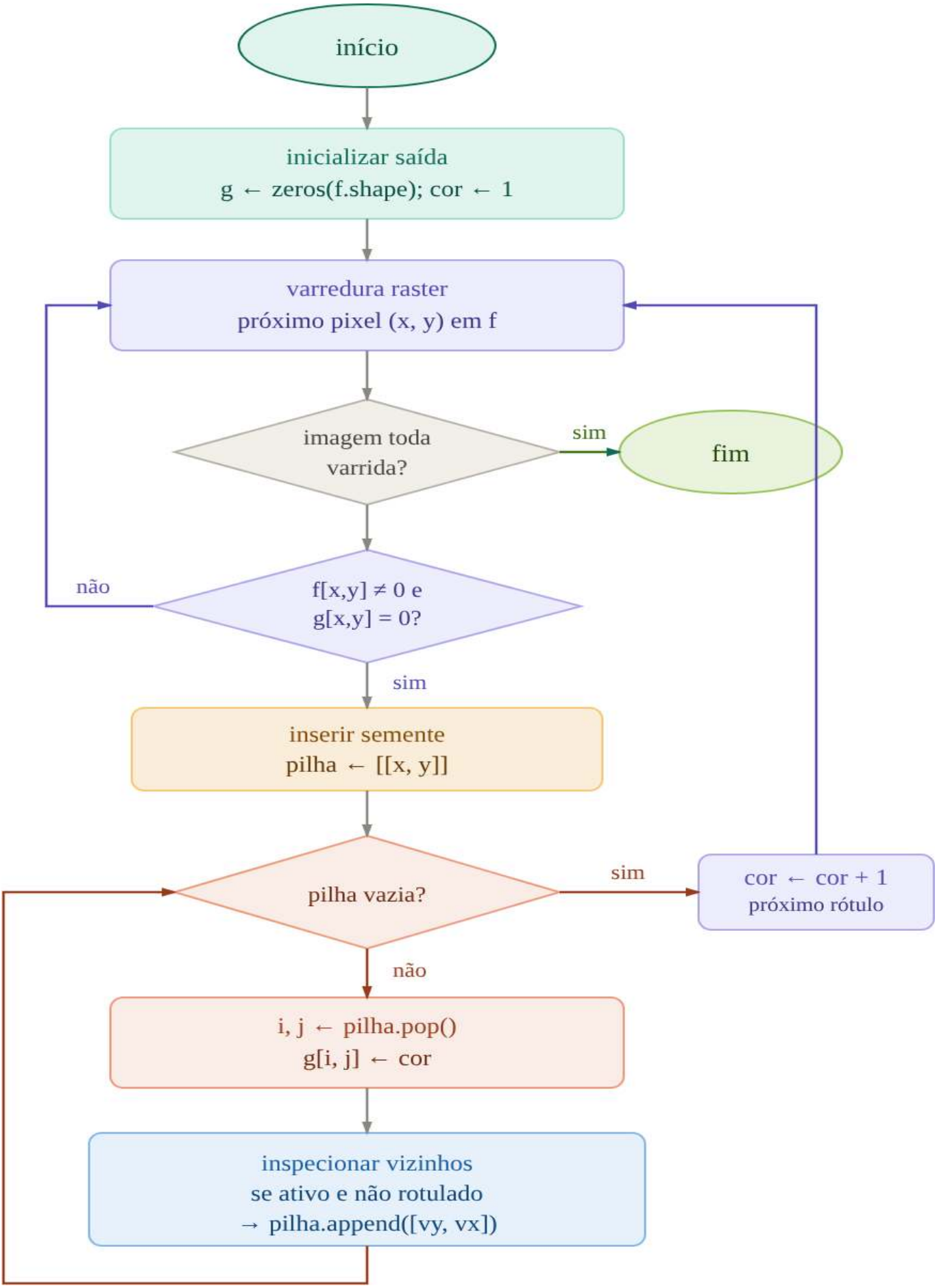
La scelta della connettività influenza direttamente la formazione delle componenti connesse e, di conseguenza, il risultato dell'etichettatura, come illustrato nella Figura 4.17. Un esempio aggiuntivo può essere esplorato interattivamente nel simulatore presentato nella Figura 4.16.

L'implementazione in `morph.py` fornisce due versioni di questo operatore. La funzione `mm.label0` riproduce esplicitamente l'algoritmo di *flood-fill* utilizzando una pila e consente di controllare la connettività tramite l'elemento strutturante adottato. Invece, `mm.label` delega l'operazione all'implementazione ottimizzata di OpenCV (`mm.label0`). In entrambi i casi, il risultato è un'immagine etichettata nella quale ciascuna componente connessa riceve un identificatore intero distinto.

L'helper `_viz` itera sulla finestra strutturante `b` centrata in (i, j), generando solo i vicini validi entro i limiti dell'immagine — la connettività desiderata è interamente determinata dalla forma di `b` passata all'algoritmo.

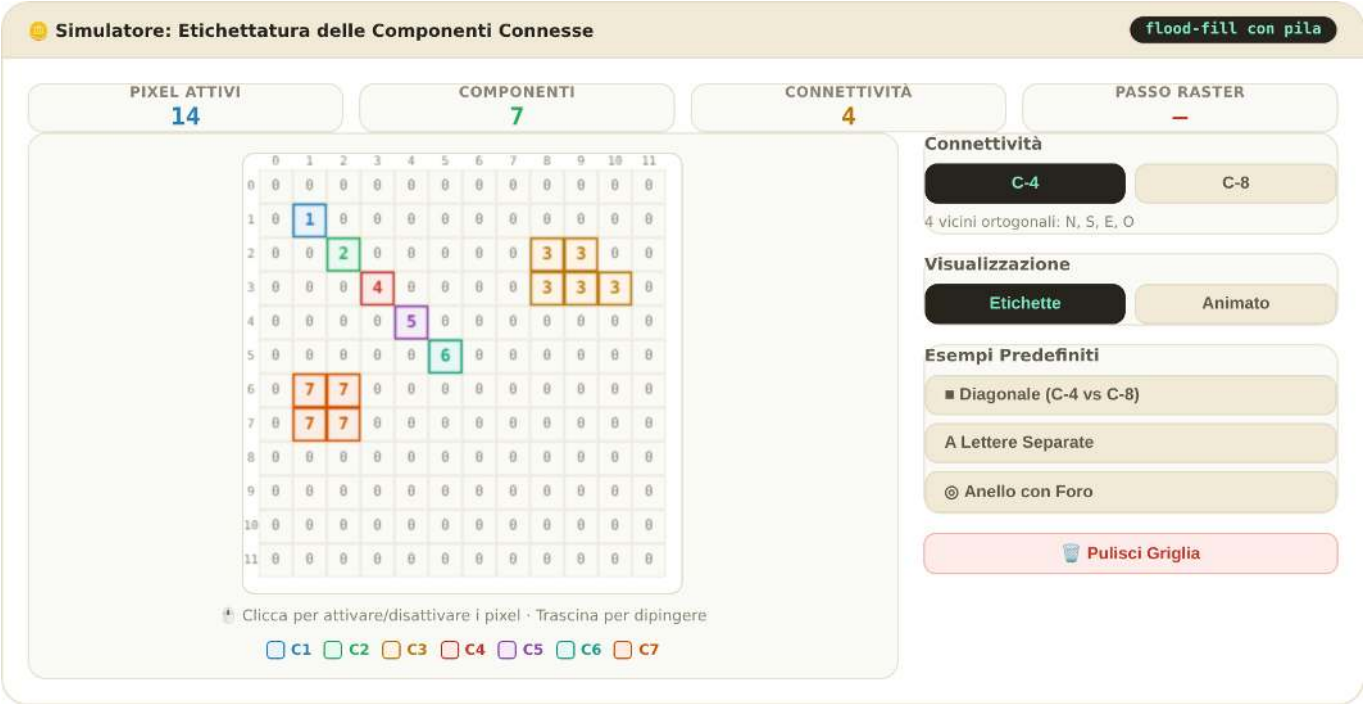
Esempio didattico — effetto della connettività:

```
# Immagine binaria 10×10 con componenti diagonalmente adiacenti
f = np.array([
    [0,0,0,0,0,0,0,0,0,0],
    [0,1,0,0,0,0,0,0,0,0],
    [0,0,1,0,0,0,0,0,0,0],  # diagonale con riga precedente
    [0,0,0,1,0,0,1,1,0,0],
    [0,0,0,0,0,0,1,1,0,0],
    [0,0,0,0,0,0,0,0,0,0],
    [0,1,1,1,0,0,0,0,0,0],
    [0,1,0,1,0,0,0,1,0,0],
    [0,1,1,1,0,0,0,0,1,0],  # diagonale con riga precedente
    [0,0,0,0,0,0,0,0,0,0]], dtype=np.uint8) * 255

# Elemento strutturante a croce (connettività-4) e quadrato (connettività-8)
B4 = mm.secross()   # connettività-4
B8 = mm.sebox()     # connettività-8

# Etichettatura con label0 (didattico) e label (cv2)
lbl4_didatico = mm.label0(f, B4)
lbl8_didatico = mm.label0(f, B8)

_, lbl4_cv2 = cv2.connectedComponents(f, connectivity=4)
_, lbl8_cv2 = cv2.connectedComponents(f, connectivity=8)

# Validazione incrociata
print(f"  label0 (C4) == cv2 (C4): {np.array_equal(lbl4_didatico, lbl4_cv2)}")
print(f"  label0 (C8) == cv2 (C8): {np.array_equal(lbl8_didatico, lbl8_cv2)}")
print(f"   Componenti C4: {lbl4_cv2.max()}  |  Componenti C8: {lbl8_cv2.max()}")

# Normalizzazione per visualizzazione
def norm_label(lbl):
    out = np.zeros_like(lbl, dtype=np.uint8)
    for i, v in enumerate(np.unique(lbl)[1:], 1):
        out[lbl == v] = int(i * 255 / lbl.max())
    return out

mm.show(
    [f, norm_label(lbl4_cv2), norm_label(lbl8_cv2)],
    titles=[
        "f originale",
        f"Etichettatura C4\n({lbl4_cv2.max()} componenti)",
        f"Etichettatura C8\n({lbl8_cv2.max()} componenti)"
    ],
    cols=3, figsize=(12, 4), axis=True
)
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-alg-rotulagem2.html

Figura 4.15: Algoritmo di etichettatura per *flood-fill* con pila.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-04-rotulacao.html

Figura 4.16: Simulatore interattivo di etichettatura delle componenti connesse (*connected component labeling*): visualizzazione dell'espansione flood-fill, connettività-4 e connettività-8.

```
label0 (C4) == cv2 (C4): True
label0 (C8) == cv2 (C8): True
 Componenti C4: 7  |  Componenti C8: 4
```

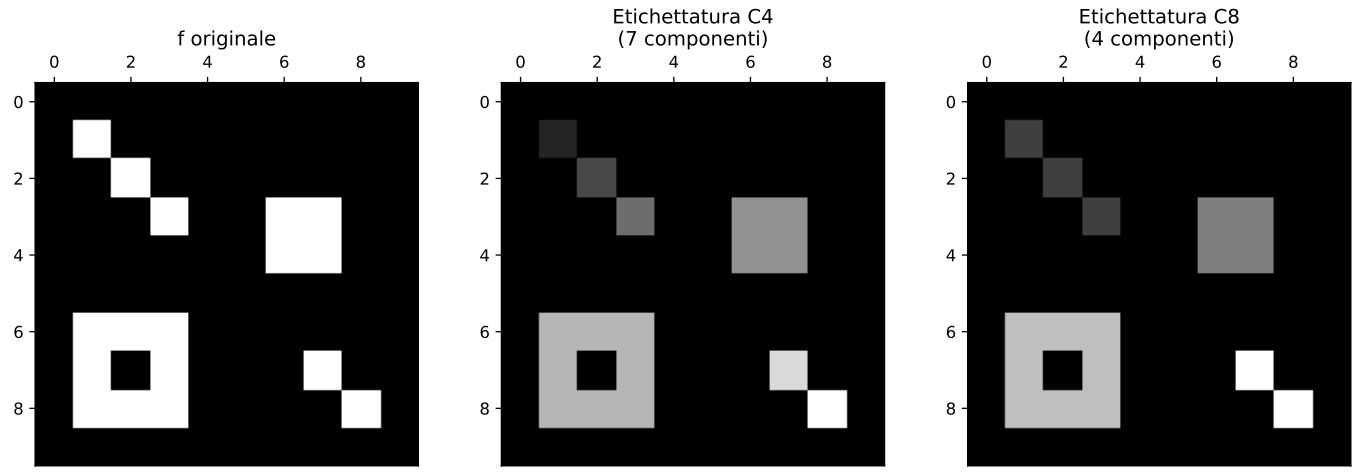


Figura 4.17: Effetto della connettività sull'etichettatura: l'immagine binaria 10×10 contiene pixel diagonalmente adiacenti. Con connettività-4, questi pixel formano componenti distinte; con connettività-8, si fondono in un'unica componente.

4.4.2 Trasformata della Distanza

La **Trasformata della Distanza** (TD) è un operatore che, applicato a un'immagine binaria f, produce un'immagine in livelli di grigio D nella quale a ciascun pixel appartenente all'oggetto ($f(x, y) \neq 0$) viene assegnato come valore la distanza geometrica fino al pixel di sfondo ($f(x', y') = 0$) più vicino:

$$D(x,y) = \min_{(x',y')\,:\,f(x',y')=0} d\big((x,y),\,(x',y')\big)$$

dove $d(\cdot,\cdot)$ è una metrica di distanza — tipicamente la distanza Euclidea (L_2). Il risultato è una rappresentazione topografica degli oggetti: i pixel situati all'interno assumono valori elevati, mentre i pixel vicini ai bordi presentano bassi valori di distanza. I **massimi locali** di D corrispondono ai punti più lontani dal bordo dell'oggetto, spesso vicini ai suoi centri geometrici o ai centri di massima inscrizione — proprietà particolarmente utile per la generazione automatica di marcatori nell'algoritmo *watershed*.

Definizione formale tramite erosioni

La TD ammette inoltre un'interpretazione morfologica iterativa, come nell'algoritmo della Figura 4.18. Si consideri una funzione strutturante b il cui valore centrale è nullo e i cui vicini possiedono costi negativi associati allo spostamento. Applicando erosioni successive con questa particolare funzione strutturante, i valori dei pixel degli oggetti (che devono assumere la distanza massima possibile dall'immagine) vengono progressivamente ridotti secondo i costi definiti da b. Il valore accumulato di questa propagazione rappresenta quindi la distanza dallo sfondo secondo la metrica indotta dalla funzione strutturante.

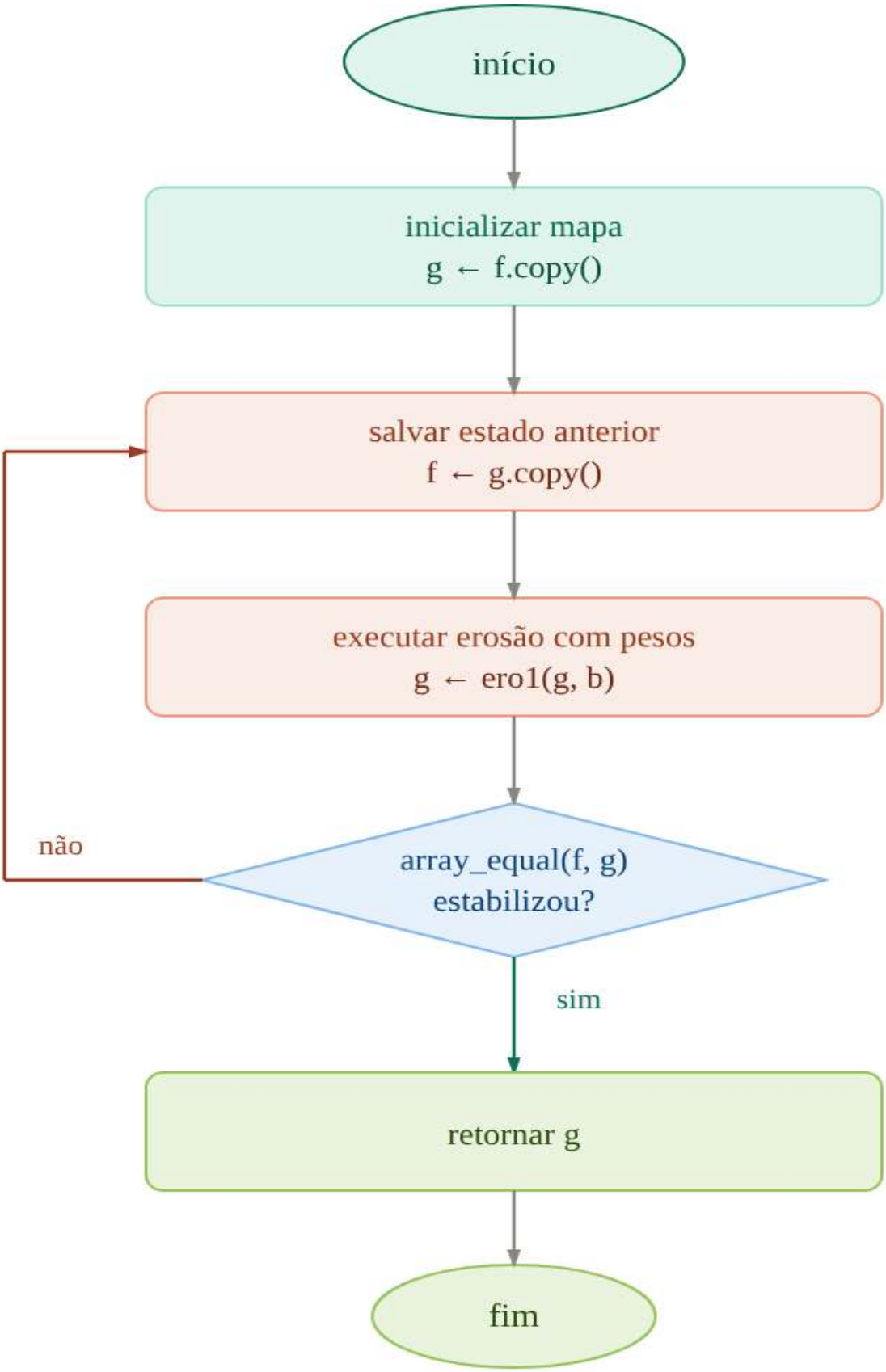
Questa interpretazione è implementata in `mm.dist1()`, che accumula erosioni successive utilizzando l'operazione `mm.ero1()`. Invece `mm.dist()` delega il calcolo della distanza Euclidea all'operatore ottimizzato di OpenCV `mm.dist(f)`, dove `f` è l'immagine binaria di ingresso, la distanza L2 specifica la metrica Euclidea (L_2) e `5` indica l'uso di una maschera 5×5 per approssimare la distanza con elevata precisione.

La funzione `dist1` produce una trasformata della distanza discreta la cui metrica è determinata dalla geometria e dai pesi della funzione strutturante utilizzata. Ad esempio, utilizzando una funzione strutturante a croce con costo unitario per i quattro vicini ortogonali, si ottiene la distanza di ***Manhattan*** (L_1). Altre scelte di vicinanza e pesi inducono metriche differenti. Invece `mm.dist()` calcola un'approssimazione efficiente della distanza Euclidea (L_2).

Poiché richiede successive erosioni su tutta l'immagine, l'approccio `dist1` possiede un costo computazionale significativamente maggiore rispetto a `mm.dist()`, essendo impiegato in questo libro principalmente a fini didattici e per evidenziare la relazione tra morfologia matematica e trasformate della distanza.

A Figura 4.19 presenta un simulatore interattivo della TD: è possibile posizionare il cursore su diversi pixel dell'oggetto e osservare, in tempo reale, il valore della distanza associato a quella posizione, cioè la distanza fino al pixel di sfondo più vicino. La Figura 4.20 presenta un esempio pratico di questa esecuzione in ambiente Python.

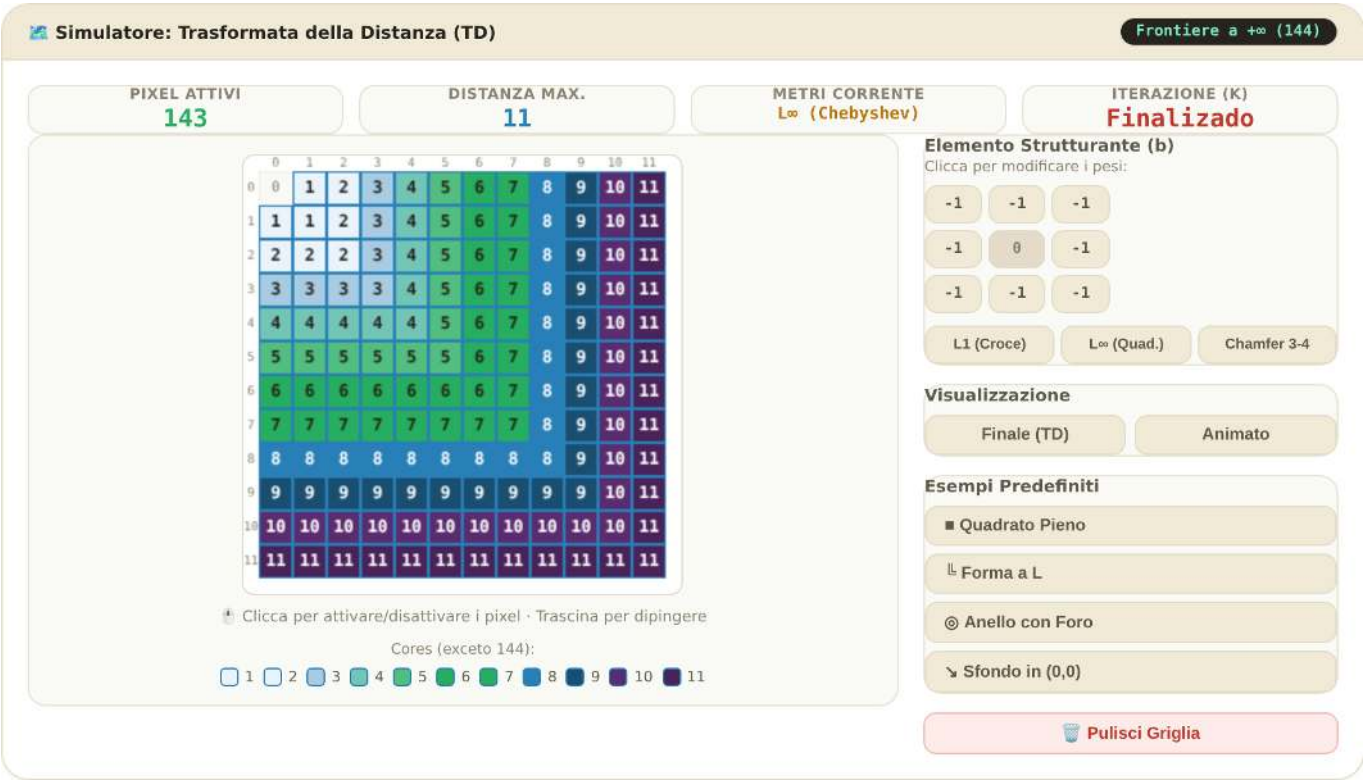
```
f = np.ones((10, 10), dtype=np.uint8) * 255
f[0, 0] = 0

B_cruz = np.array([
    [-np.inf, -1, -np.inf],
    [-1,       0, -1],
    [-np.inf, -1, -np.inf]
], dtype=float)

d_iter = mm.dist1(f, B_cruz)
d_l2   = mm.dist(f)

print(f"Max dist1 (erosioni) : {int(d_iter.max())} px")
print(f"Max dist (L2) : {int(d_l2.max())} px")

mm.show(
    [f, d_iter, d_l2],
    titles=["f originale", "mm.dist1 (erosioni con croce)", "mm.dist (L2)"],
    cols=3, figsize=(12, 4), axis=True
```



Ponto Fixo Numérico: A distância emerge da propagação matemática do valor zero.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-alg-distancia2.html

Figura 4.18: Algoritmo della Trasformata della Distanza.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-04-distancia.html

Figura 4.19: Simulatore interattivo della Trasformata della Distanza (TD) iterativa tramite erosione in toni di grigio. I pixel fuori dall'immagine assumono il valore massimo (144), propagando i costi a partire dallo sfondo interno.

```
)
```

```
Max dist1 (erosioni) : 18 px
Max dist (L2) : 12 px
```

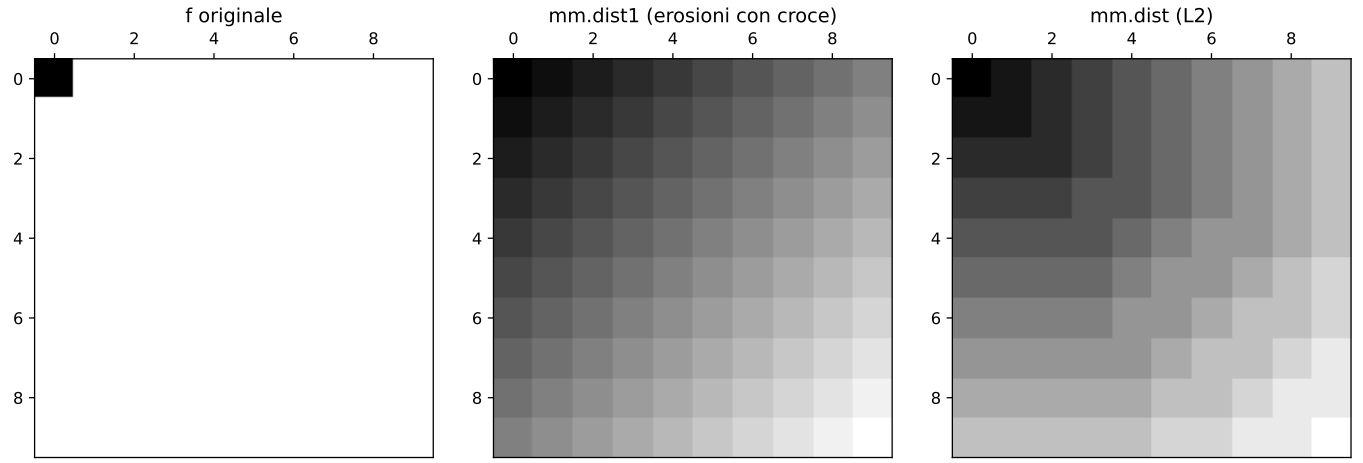


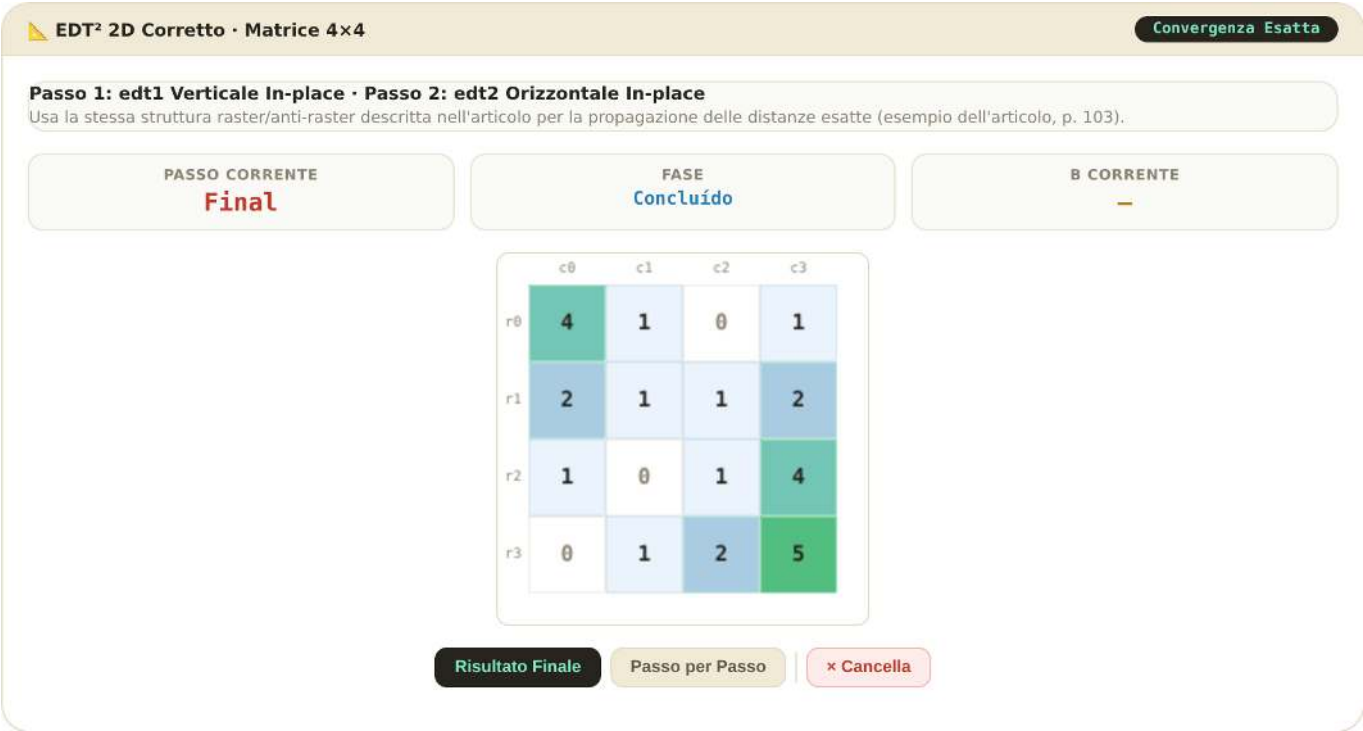
Figura 4.20: Trasformata della Distanza in immagine binaria 10×10. A sinistra: originale (*foreground* = 255). Al centro: mm.dist1 iterativa (erosioni con croce). A destra: mm.dist (L2).

L'annotazione dei valori numerici direttamente sui pixel consente di verificare come `dist1` propaghi le distanze secondo la metrica indotta dalla funzione strutturante utilizzata. Nel caso dell'elemento a croce con costo unitario, i valori ottenuti corrispondono alla distanza di *Manhattan* (L_1). Sebbene `dist1` e `mm.dist` producano valori numerici distinti poiché adottano metriche differenti, entrambe le trasformate preservano

la struttura topografica degli oggetti, facendo sì che i loro massimi si verifichino in regioni centrali simili. Questa proprietà giustifica l'uso di `mm.dist` nelle applicazioni pratiche, grazie alla sua elevata efficienza computazionale.

4.4.3 Trasformata della Distanza Euclidea in quattro passaggi

Il simulatore della Figura 4.21 implementa l'algoritmo della TDE di Lotufo (2001) in due fasi. Nella prima fase, la funzione `edt1` esegue una trasformazione unidimensionale verticale in modo sequenziale (*in-place*), percorrendo ciascuna colonna in *raster* (↓) e *anti-raster* (↑) per calcolare le distanze nella direzione verticale (due passaggi: Sud e Nord). Nella seconda fase, la funzione `edt2` utilizza questo risultato come input ed esegue una propagazione orizzontale tramite code: per ogni riga della matrice, due code di priorità, `Eq` e `Wq`, vengono inizializzate scorrendo gli indici di colonna in direzioni opposte (`Eq` da `W-1` a `1`, `Wq` da `2` a `W`), così che ogni pixel aggiornato accodi immediatamente i suoi vicini per un riprocessamento all'interno dello stesso ciclo (altri due passaggi: Est e Ovest). Questo meccanismo a coda consente di applicare erosioni successive con pesi dispari crescenti (`b = 1, 3, 5, ...`, incrementati a ogni iterazione del ciclo esterno) senza che la propagazione rimanga bloccata su valori obsoleti, poiché ogni ciclo risolve completamente la catena di dipendenze orizzontali prima del successivo incremento di `b`. La convergenza di questa propagazione produce la Trasformata della Distanza Euclidea sull'intera matrice, combinando l'informazione verticale ottenuta in `edt1` con la propagazione orizzontale a coda eseguita in `edt2`.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-04-tde.html

Figura 4.21: Simulatore interattivo 2D (4x4) con sincronizzazione rigorosa delle code di propagazione orizzontale (b) per ottenere la convergenza esatta descritta nell'articolo.

4.4.3.1 Trasformata di Distanza Geodetica

La trasformata di distanza geodetica associa a ogni pixel la distanza minima fino a un marcatore, sotto il vincolo imposto da una maschera. In questo modo, la propagazione avviene esclusivamente attraverso i pixel consentiti, preservando la connettività del dominio.

La Figura 4.22 illustra questo processo in un labirinto: (a) la maschera `g`; (b) la distanza geodetica `D1` calcolata dall'ingresso; (c) la distanza `D2` calcolata dall'uscita; e (d) il percorso minimo ottenuto da queste due trasformate.

Il percorso ottimale è determinato dalla somma delle distanze (`D1 + D2`). I pixel appartenenti alla traiettoria

minima sono quelli per cui tale somma assume il suo valore più basso, definendo una connessione tra ingresso e uscita con lunghezza geodetica minima.

Questo principio consente di risolvere labirinti senza la necessità di esplorare esplicitamente tutte le possibilità di percorso. La soluzione emerge direttamente dalla propagazione delle distanze in un dominio ristretto. Questo approccio è particolarmente rilevante in labirinti di elevata complessità, come quelli costruiti a partire da strutture quasi-cristalline e cicli hamiltoniani descritti da Singh (2024). In Zampirolli (2025), questo stesso formalismo viene impiegato per risolvere un labirinto complesso; successivamente, il metodo è illustrato in una versione semplificata del problema.

```
import numpy as np

# 1 = corridoio, 0 = parete
g = np.array([
 [0,1,0,0,0,0,0,0,0,0],
 [0,1,1,1,1,1,0,1,1,1],
 [0,0,0,0,0,1,0,1,0,1],
 [0,1,1,1,0,1,1,1,0,1],
 [0,1,0,1,0,0,0,0,0,1],
 [0,1,0,1,1,1,1,1,1,1],
 [0,1,0,0,0,0,0,0,1,0],
 [0,1,1,1,1,1,1,0,1,0],
 [0,0,0,0,0,0,1,1,1,0],
 [0,0,0,0,0,0,0,0,1,0]
], dtype=np.uint8)

# marcatore dell'ingresso
entrada = np.zeros_like(g, dtype=np.uint8)
entrada[0,1] = 1

# marcatore dell'uscita
saida = np.zeros_like(g, dtype=np.uint8)
saida[9,8] = 1

# distanze geodetiche
D1 = mm.gdist(g, entrada)
D2 = mm.gdist(g, saida)

# somma delle distanze
S = D1 + D2

# valore minimo valido della somma
dmin = np.min(S[S > 0])

# pixel appartenenti a un percorso ottimale
caminho = (S == dmin)

print("Distanza geodetica minima:", dmin)

mm.show(
    [g, D1, D2, caminho],
    titles=[
        "Labirinto",
        "Distanza dell'Ingresso",
        "Distanza dell'Uscita",
        f"Percorso Minimo\n(d={dmin})"
    ],
    cols=4,
    figsize=(14,4),
    axis=True
```

```
)
```

```
Distanza geodetica minima: 15
```

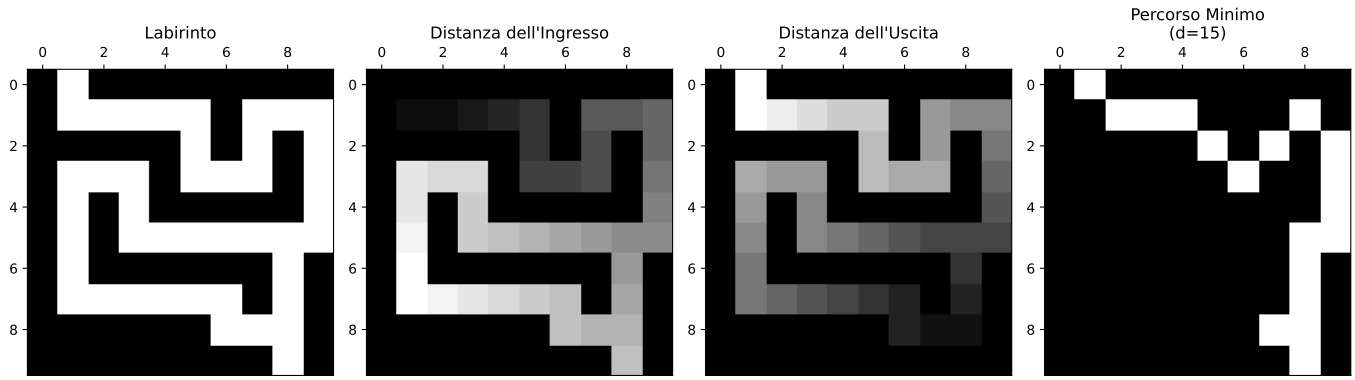


Figura 4.22: Percorso geodetico minimo in un labirinto. Le distanze geodetiche sono calcolate dall'ingresso e dall'uscita usando mm.gdist. I pixel la cui somma delle due distanze è uguale alla distanza minima tra i marcatori appartengono a un percorso ottimale.

4.4.4 Segmentazione per *Watershed*

L'algoritmo ***Watershed*** interpreta un'immagine in livelli di grigio come una superficie topografica, in cui valori elevati corrispondono a montagne e valori bassi corrispondono a valli o bacini di drenaggio (*catchment basins*). Nel contesto della segmentazione basata su marcatori, i massimi della Trasformata della Distanza sono spesso utilizzati per identificare regioni interne agli oggetti, fornendo semi affidabili per il processo di inondazione.

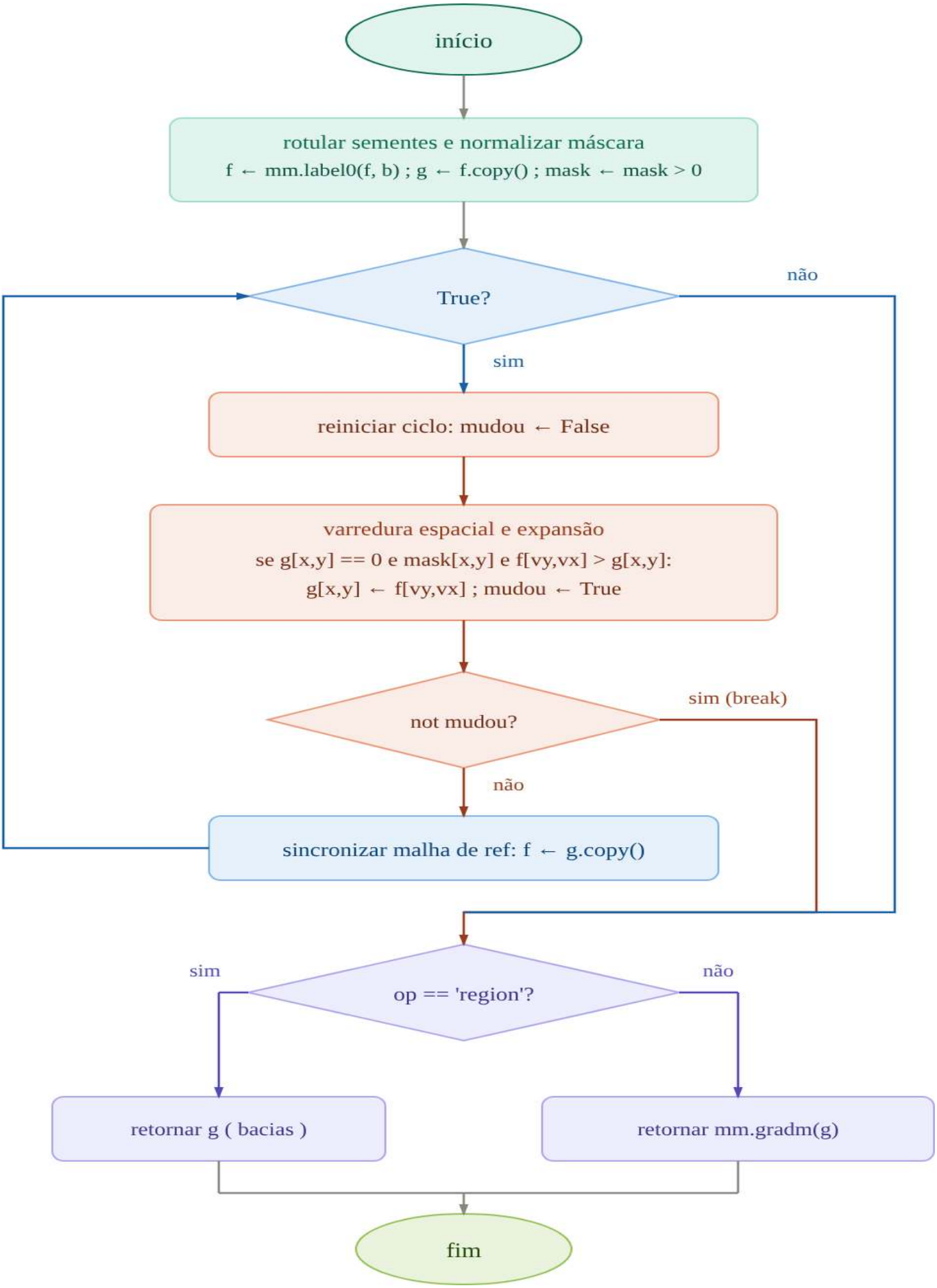
La segmentazione viene quindi realizzata tramite una simulazione concettuale di inondazione progressiva a partire da questi marcatori. Man mano che i bacini associati a semi diversi si espandono, le regioni vicine entrano eventualmente in contatto. In quel momento vengono costruite barriere virtuali, denominate *watershed lines*, che delimitano gli oggetti della scena. Questo meccanismo consente di separare oggetti adiacenti o parzialmente sovrapposti, anche quando essi formano una singola componente connessa dopo la sogliatura.

L'implementazione didattica presentata in questo capitolo esplora inizialmente il concetto di crescita delle regioni (*region growing*) confinato da una maschera binaria, come dettagliato nell'algoritmo interattivo della Figura 4.23.

Versione didattica versus implementazione classica

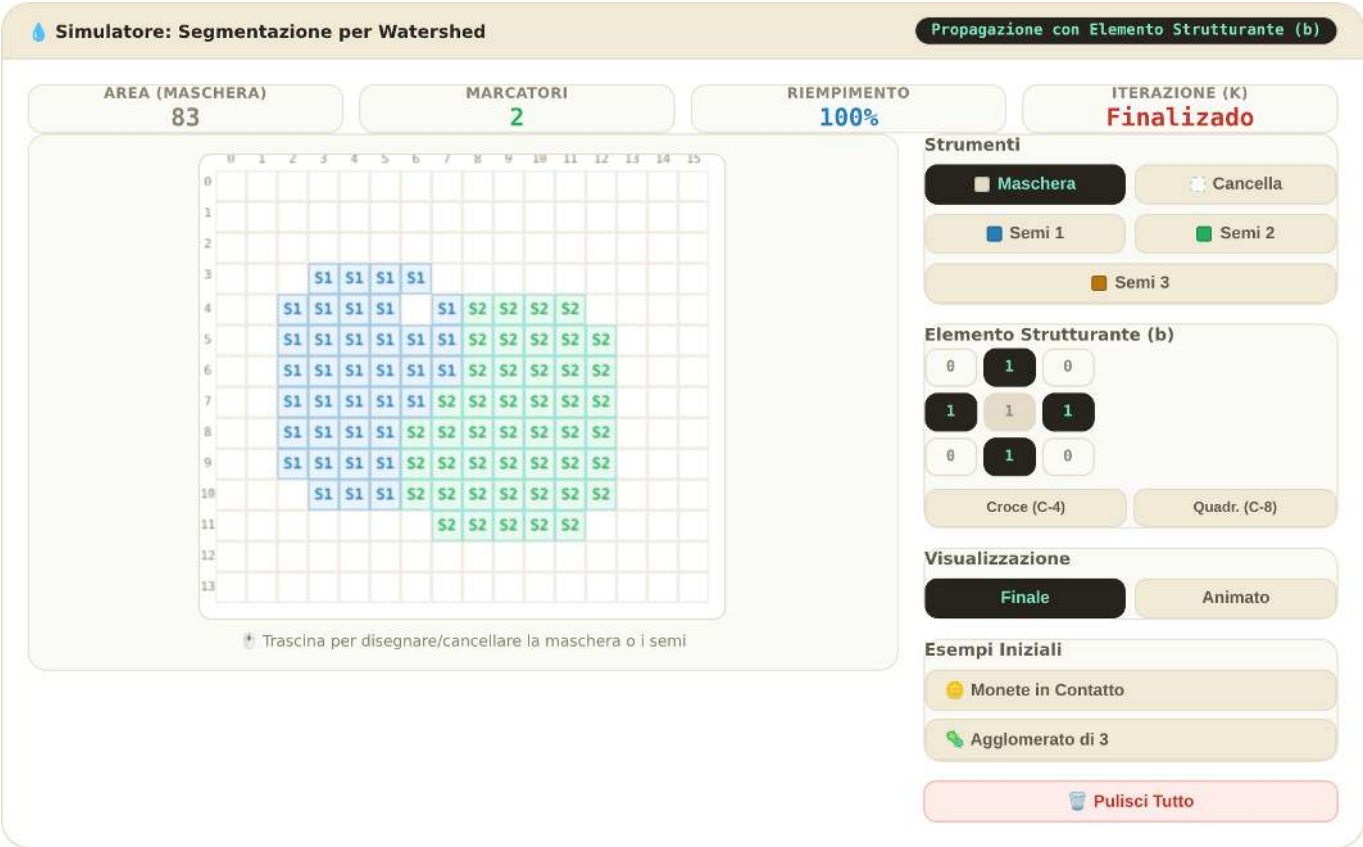
La funzione `mm.watershed0` non implementa l'algoritmo *watershed* classico. Il suo scopo è illustrare, in modo semplificato, la propagazione dei marcatori tramite crescita delle regioni (*region growing*), consentendo di visualizzare come diversi semi competono per l'occupazione dello spazio disponibile. La crescita è delimitata da una maschera binaria di supporto e monitorata da un controllo di stagnazione, generando un risultato simile a una partizione di Voronoi limitata alla geometria degli oggetti di input. La funzione `mm.watershed`, invece, utilizza l'implementazione ottimizzata di OpenCV (`mm.watershed`), che esegue l'inondazione su una superficie topografica definita dall'immagine di input. In questo caso, la propagazione dei marcatori è influenzata dai valori dei pixel, facendo sì che le linee di separazione si formino naturalmente sulle creste del rilievo.

Figura 4.24 presenta un simulatore iterativo che illustra la propagazione dei marcatori nella regione di interesse. Ogni marcatore agisce come una sorgente di allagamento che espande la propria area di influenza fino a incontrare regioni provenienti da altri semi. Nell'algoritmo di OpenCV, i pixel appartenenti alle linee di divisione sono identificati dal valore `-1`, che rappresenta i confini tra bacini adiacenti.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-alg-watershed2.html

Figura 4.23: Algoritmo Didático di *Watershed* per Crescita di Regioni Limitato da Maschera.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-04-watershed.html

Figura 4.24: Simulatore interattivo dell'Algoritmo *Watershed* per propagazione morfologica. Disegna la maschera, posiziona i marcatori e regola l'Elemento Strutturante per osservare l'inondazione. Quando i bacini si incontrano simultaneamente, il pareggio viene risolto assumendo una delle regioni in modo casuale.

Pipeline Morfologico del *Watershed*

Il *watershed* basato su marcatori integra normalmente un flusso più ampio di segmentazione. Nelle immagini reali, fasi di pre-elaborazione sono spesso necessarie per migliorare il contrasto, ridurre il rumore e generare marcatori affidabili. Questo flusso completo è riassunto nella Tabella 4.6.

Tabella 4.6: *Pipeline* completo del *watershed* basato su marcatori per immagini reali.

Fase	Operazione	Finalità
1	CLAHE + Smussamento	Miglioramento del contrasto e riduzione del rumore
2	Sogliatura	Separazione iniziale tra oggetto e sfondo
3	Apertura/Chiusura	Rimozione di rumori e piccole imperfezioni
4	Dilatazione della Maschera	Identificazione dello Sfondo Certo (*Sure Background*)
5	Trasformata della Distanza + Soglia	Identificazione dell'Oggetto Certo (*Sure Foreground*)
6	Regione Incerta	Differenza tra Sfondo Certo e Oggetto Certo
7	`mm.watershed`	Propagazione dei marcatori attraverso la regione incerta

Per enfatizzare esclusivamente i concetti di Trasformata della Distanza, marcatori e inondazione topografica, l'esempio della Figura 4.25 utilizza un'immagine binaria sintetica e adotta un flusso semplificato, riassunto nella Tabella 4.7.

Tabella 4.7: *Pipeline* semplificato utilizzato nell'esempio didattico della Figura 4.25.

Fase	Operazione	Finalità
1	Trasformata della Distanza	Costruzione della superficie topografica
2	Soglia della TD	Estrazione dei marcatori (*Sure Foreground*)
3	Dilatazione	Determinazione dello Sfondo Certo (*Sure Background*)
4	Regione Incerta	Differenza tra sfondo e marcatori
5	`mm.watershed`	Propagazione dei marcatori e generazione delle frontiere

```
# 1. Immagine sintetica e Trasformata della Distanza
f_sint = np.zeros((20, 20), dtype=np.uint8)
f_sint = mm.circle(f_sint, (6,  10), 5, 255, -1)
f_sint = mm.circle(f_sint, (14, 10), 5, 255, -1)
dist = mm.dist(f_sint)

# 2. Marcatori (picchi della distanza)
m = (dist > 0.8 * dist.max()).astype(np.uint8) * 255

# 3. Esecuzione del Watershed condizionale (m=marcatori prima, mask=f_sint)
w_reg  = mm.watershed0(m, mask=f_sint, op='region')
w_line = mm.watershed0(m, mask=f_sint, op='line')
```

```
#w_reg  = mm.watershedB(m, mask=f_sint, op='region')
#w_line = mm.watershedB(m, mask=f_sint, op='line')

w_reg  = mm.watershed(m, mask=f_sint, op='region')
w_line = mm.watershed(m, mask=f_sint, op='line')

# 4. Visualizzazione dei risultati
mm.show([f_sint, dist, m, w_reg, w_line], cols=5, figsize=(16, 4),
        titles=["Originale", "Distanza", "Marcatori", "Regioni", "Linee"])
```

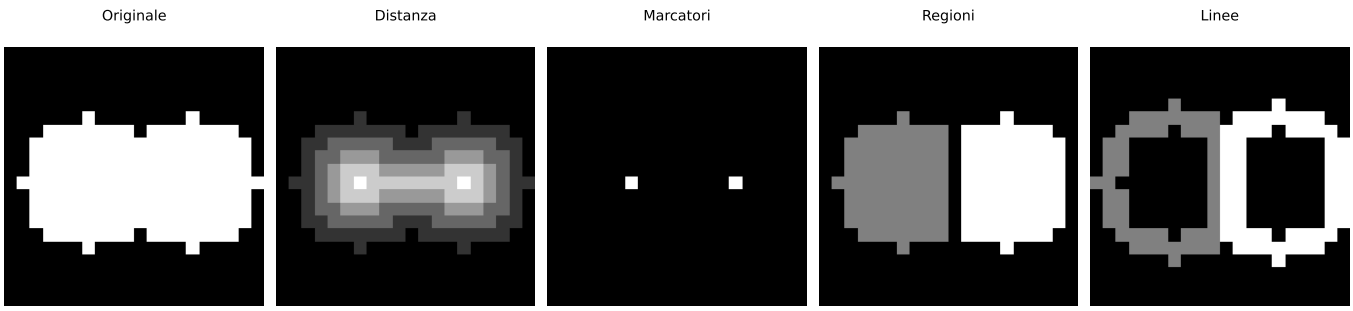


Figura 4.25: *Pipeline watershed* delimitato da maschera su immagine binaria 20×20.

Applicazione su monete sovrapposte:

```
img_base = img_coins_gray

# 1. Simulare monete sovrapposte/collegate (usando la maschera binaria base)
img_sobrepostas = mm.dil(img_final, mm.sebox(40))

# 2. Apertura morfologica per pulire i rumori
opening = mm.open(img_sobrepostas, mm.sebox(2))

# 3. Trasformata della distanza
dist = mm.dist(opening)
dist_vis=(255*(dist/dist.max())).astype(np.uint8) if dist.max() > 0 else dist.astype(np.uint8)

# 4. Picchi sicuri (Marcatori delle monete)
picos = (dist > 0.5 * dist.max()).astype(np.uint8) * 255

# 5. Esecuzione del Watershed (Adattato per usare la nuova firma)
# Passiamo 'opening' direttamente in mask, poiché delimita l'ambito di espansione delle monete
ws_region = mm.watershed(picos, mask=opening, op='region')
ws_line   = mm.watershed(picos, mask=opening, op='line')

# 6. Conteggio degli oggetti (Ignora lo sfondo 0)
labels = np.unique(ws_region)
labels = labels[labels > 0]
print(f"Oggetti rilevati: {len(labels)}")

# 7. Annotazione Finale
img_annotated = cv2.cvtColor(img_base, cv2.COLOR_GRAY2BGR)

for idx, label_id in enumerate(labels):
    mask_reg = (ws_region == label_id).astype(np.uint8)
    if mask_reg.sum() < 500: continue

    cy, cx = np.mean(np.where(mask_reg), axis=1).astype(int)
    cv2.putText(img_annotated, str(idx + 1), (cx - 25, cy + 20),
                cv2.FONT_HERSHEY_SIMPLEX, 3.2, (0, 255, 0), 8, cv2.LINE_AA)
```

```
# Disegna le linee di separazione in rosso
mask_ann = mm.dil(ws_line, np.ones((11, 11), np.uint8))
img_annotated[mask_ann > 0 ] = [255, 0, 0]

# Visualizzazione
mm.show(
    [img_base, img_sobrepostas, dist_vis, picos, ws_region, img_annotated],
    titles=["Originale", "Sovrapposte", "Distanza", "Marcatori", "Watershed", "Annotato"],
    cols=3, rows=2, figsize=(12, 8)
)
```

```
Oggetti rilevati: 12
```

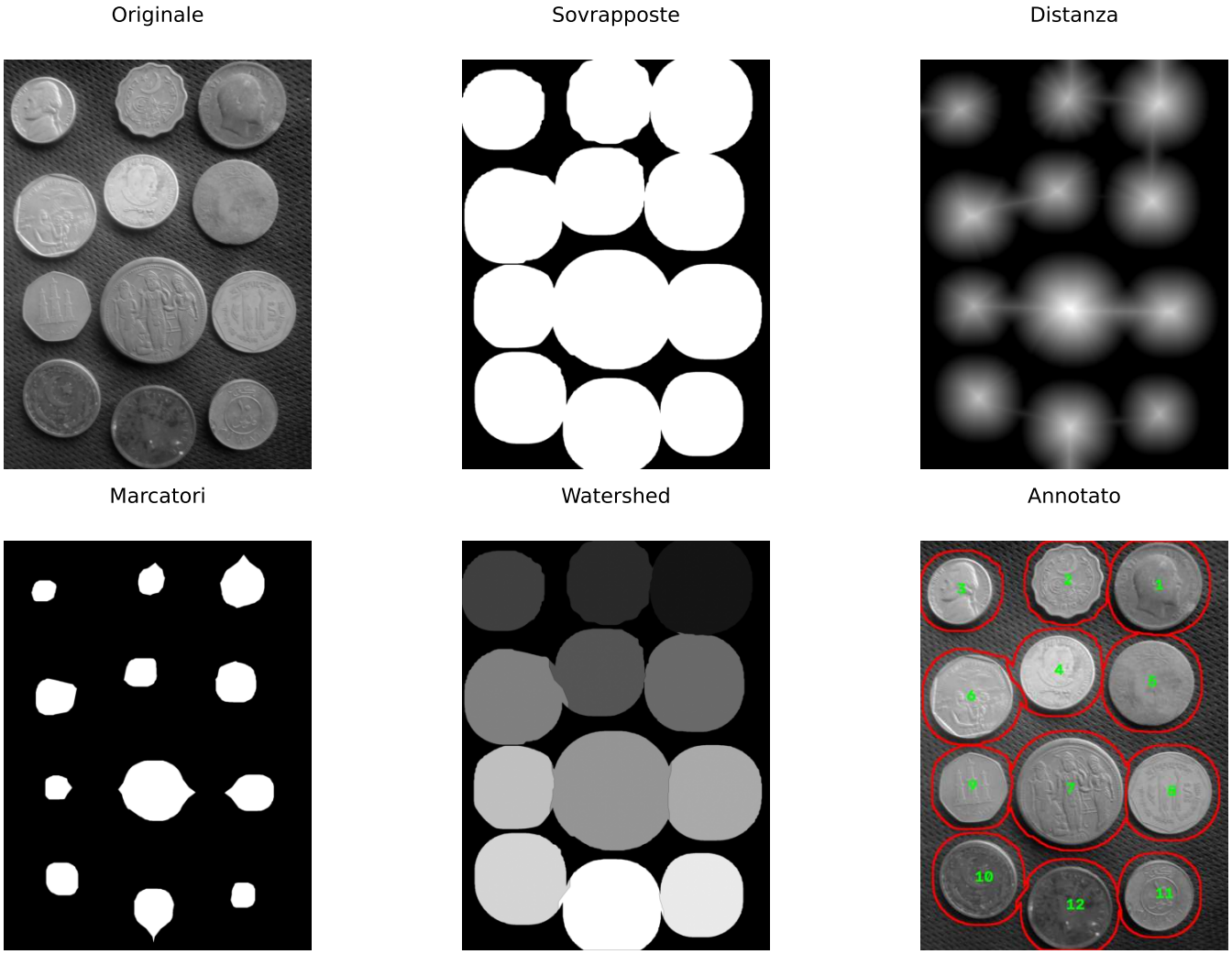


Figura 4.26: *Pipeline Watershed* per separazione di monete sovrapposte: dalla maschera binaria dilatata fino ai contorni finali annotati sull'immagine originale.

4.5 Estrazione di Componenti e Descrittori di Forma

Dopo la segmentazione e il raffinamento morfologico, il passo successivo consiste nell'identificare individualmente ciascun oggetto presente nell'immagine ed estrarne le proprietà geometriche. Questa fase è fondamentale per compiti di misurazione, classificazione e riconoscimento di pattern.

Un **componente connesso** è un insieme massimale di pixel appartenenti all'oggetto che rimangono

mutuamente connessi secondo una relazione di connettività precedentemente definita (connettività a 4 o a 8). Dopo l'etichettatura, ogni componente riceve un identificatore univoco, consentendo che le sue caratteristiche vengano analizzate singolarmente.

OpenCV offre due approcci complementari per questa analisi, riassunti nella Tabella 4.8.

Tabella 4.8: Confronto tra gli approcci basati su componenti connessi e su contorni.

	`connectedComponentsWithStats`	`findContours`
Restituisce	etichetta per pixel e statistiche per componente	sequenza di punti che descrive il contorno
Descrittori diretti	area, *bounding box* e centroide	perimetro, forma e gerarchia
Oggetti a contatto	tende a fondere regioni connesse	tende a produrre un unico contorno esterno
Uso tipico	conteggio, filtraggio ed etichettatura	analisi geometrica e descrittori di forma

4.5.1 Etichettatura e Statistiche dei Componenti

La funzione `mm.label0` esegue simultaneamente l'etichettatura dei componenti connessi e l'estrazione dei descrittori di base di ciascuna regione. L'operatore restituisce:

- un'immagine delle etichette (`labels`);
- statistiche geometriche (`stats`);
- coordinate dei centroidi (`centroids`).

Le statistiche includono area, larghezza, altezza e posizione del *bounding box* minimo allineato agli assi dell'immagine.

La Figura 4.27 illustra l'applicazione di questo operatore dopo il *pipeline* di segmentazione delle monete. Ogni componente connesso riceve un colore distinto e la sua area viene annotata direttamente sull'immagine.

```
# 1. Etichettatura e statistiche
n, labels, stats, centroids = cv2.connectedComponentsWithStats(
    img_final, connectivity=8
)

# 2. Colorazione dei componenti - paletta FISSA (generata una volta con
# np.random.seed(4)) affinché il percorso C++ riproduca colore per colore senza
# dipendere dal generatore di numpy. Se n supera len(PALETA), cicla.
PALETA = np.array([
    [0, 0, 0], [224, 82, 193], [233, 155, 73], [190, 247, 244],
    [103, 94, 179], [51, 154, 59], [137, 96, 232], [250, 243, 205],
    [100, 141, 208], [228, 187, 163], [202, 108, 214], [131, 105, 104],
    [86, 153, 81]], dtype=np.uint8)
colors    = PALETA[np.arange(n) % len(PALETA)].copy()
colors[0] = [0, 0, 0]                          # sfondo nero
img_colored   = colors[labels]
img_annotated = img_colored.copy()

# 3. Tabella dei descrittori
print(f"Componenti rilevati (escluso lo sfondo): {n - 1}")
print(f"\n{'ID':>4} {'Área':>8} {'cx':>6} {'cy':>6} {'w':>6} {'h':>6}")
print("-" * 42)
for i in range(1, n):
    area = stats[i, cv2.CC_STAT_AREA]
    cx, cy = int(centroids[i, 0]), int(centroids[i, 1])
    w, h   = stats[i, cv2.CC_STAT_WIDTH], stats[i, cv2.CC_STAT_HEIGHT]
    print(f"{i:>4} {area:>8} {cx:>6} {cy:>6} {w:>6} {h:>6}")
```

```
    for cor, esp in [((0,0,0), 10), ((255,0,0), 5)]:
        cv2.putText(img_annotated, f"{i}: {area}",
                    (cx - 150, cy + 18),
                    cv2.FONT_HERSHEY_SIMPLEX, 2.0, cor, esp, cv2.LINE_AA)

mm.show(
    [img_coins_gray, img_final, img_colored, img_annotated],
    titles=["Originale", "Segmentazione Finale", "Componenti Connessi", "Aree Annotate"],
    cols=4, figsize=(18, 6)
)
```

```
Componenti rilevati (escluso lo sfondo): 12

  ID     Área     cx     cy      w      h
-----------------------------------------
   1   231969   1484    280    553    542
   2   158967    917    250    453    468
   3   139229    250    312    433    419
   4   175550    857    821    474    465
   5   222882   1442    889    539    531
   6   210043    316    977    527    516
   7   343376    934   1559    667    665
   8   213641   1555   1574    530    515
   9   147880    328   1543    432    438
  10   187280    363   2111    487    492
  11   150110   1487   2215    433    444
  12   215387    932   2292    528    522
```

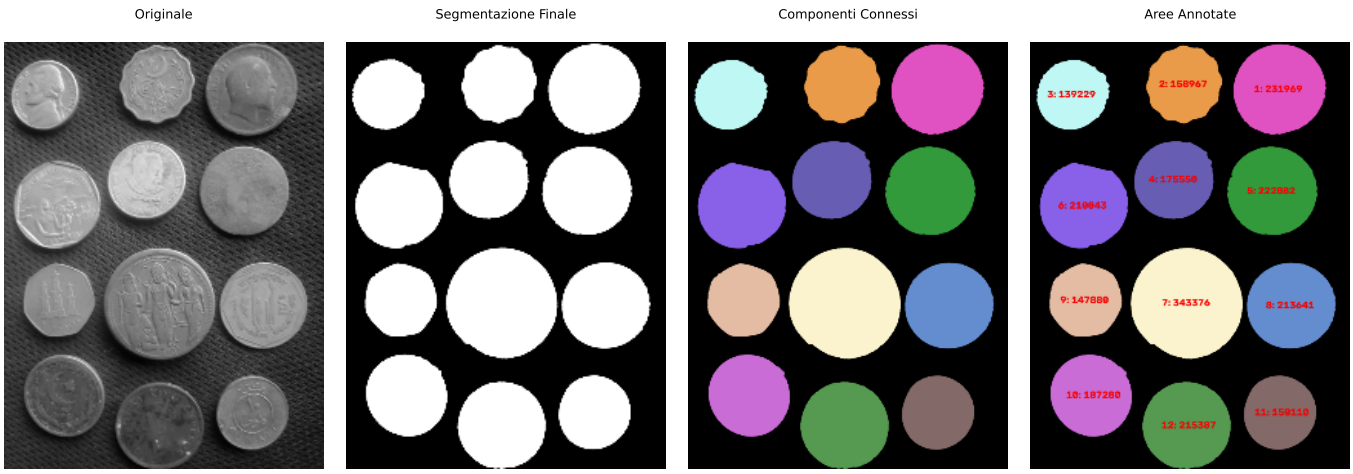


Figura 4.27: Componenti connessi estratti dopo il *pipeline* CLAHE → Otsu → pulizia morfologica. Ogni oggetto è colorato con un colore distinto e annotato con la propria area in pixel.

4.5.2 Descrittori di Forma

Mentre `connectedComponentsWithStats` opera sulle regioni, l'estrazione dei contorni agisce direttamente sui loro confini. L'operatore restituisce, per ciascun oggetto, una sequenza ordinata di punti che ne descrive il contorno.

A partire da questa rappresentazione è possibile calcolare descrittori geometrici che non sono forniti direttamente dall'etichettatura delle componenti:

- **Area** (area del contorno)
- **Perimetro** (perimetro);
- **Circolarità** ($C = \frac{4\pi A}{P^2}$, dove A è l'area e P il perimetro);

- **Approssimazione poligonale** (approssimazione poligonale);
- **Inviluppo convesso** (*convex hull*);
- **Gerarchia dei contorni**, che consente di rappresentare relazioni padre-figlio tra regioni e i loro fori interni.

La circolarità assume un valore massimo pari a 1 per un cerchio perfetto e diminuisce man mano che l'oggetto diventa più allungato o presenta irregolarità sui suoi bordi.

La Figura 4.28 presenta i contorni estratti dalle monete segmentate, insieme ai valori di circolarità calcolati per ciascun oggetto.

```
contornos, hierarquia = cv2.findContours(
    img_final, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
)

img_contornos  = cv2.cvtColor(img_coins_gray, cv2.COLOR_GRAY2BGR)
img_circulares = img_contornos.copy()

print(f"Contorni rilevati: {len(contornos)}")
print(f"\n{'ID':>4} {'Área':>8} {'Perímetro':>10} {'Circularidade':>14}")
print("-" * 42)

for i, cnt in enumerate(contornos, start=1):
    area  = cv2.contourArea(cnt)
    perim = cv2.arcLength(cnt, closed=True)
    circ  = (4 * np.pi * area / perim**2) if perim > 0 else 0
    M     = cv2.moments(cnt)
    cx    = int(M["m10"] / M["m00"]) if M["m00"] > 0 else 0
    cy    = int(M["m01"] / M["m00"]) if M["m00"] > 0 else 0

    print(f"{i:>4} {area:>8.0f} {perim:>10.1f} {circ:>14.3f}")

    cv2.drawContours(img_contornos,  [cnt], -1, (0, 255, 0), 3)
    cv2.drawContours(img_circulares, [cnt], -1, (0, 255, 0), 3)
    for cor, esp in [((0,0,0), 8), ((255,0,0), 3)]:
        cv2.putText(img_circulares, f"{circ:.2f}",
                    (cx - 80, cy + 15),
                    cv2.FONT_HERSHEY_SIMPLEX, 3.6, cor, esp, cv2.LINE_AA)

mm.show(
    [img_final, img_contornos, img_circulares],
    titles=["Segmentazione Finale", "Contorni", "Circolarità"],
    cols=3, figsize=(18, 6)
)
```

```
Contorni rilevati: 12

  ID     Área  Perímetro  Circularidade
------------------------------------------
   1   214626     1769.6          0.861
   2   149480     1465.1          0.875
   3   186570     1654.9          0.856
   4   147252     1458.6          0.870
   5   212886     1756.3          0.867
   6   342424     2232.5          0.863
   7   209282     1767.7          0.842
   8   222108     1809.7          0.852
   9   174854     1616.4          0.841
  10   138602     1471.5          0.804
```

```
 11    158306      1538.7          0.840
 12    231181      1847.4          0.851
```

Figura 4.28: Contorni estratti con *findContours*. Ogni moneta è annotata con la sua circolarità — valori vicini a 1 confermano la forma circolare.

4.5.3 Collegamento con la Rilevazione Moderna degli Oggetti

I descrittori estratti nelle sezioni precedenti — in particolare *bounding boxes*, centroidi, aree e misure di forma — stabiliscono un ponte naturale tra la segmentazione morfologica classica e i sistemi moderni di rilevazione degli oggetti. Sebbene le tecniche studiate in questo capitolo utilizzino operazioni su pixel e regioni segmentate, molte delle rappresentazioni prodotte sono direttamente compatibili con i formati impiegati nei modelli contemporanei di visione artificiale.

I rilevatori basati su apprendimento profondo, come la famiglia YOLO (*You Only Look Once*) (REDMON, 2016), operano direttamente su immagini a colori e producono, per ciascun oggetto rilevato, una *bounding box* descritta dal centro (cx, cy) e dalle dimensioni (w, h), oltre a una classe e a un punteggio di confidenza. Questa rappresentazione condivide la stessa struttura geometrica di base ottenuta tramite `connectedComponentsWithStats`, sebbene sia prodotta da un modello appreso e non da una segmentazione esplicita.

La Figura 4.29 illustra come le *bounding boxes* ottenute per morfologia possano essere esportate nel formato YOLO per comporre insiemi di dati utilizzati nell'addestramento o nella valutazione dei rilevatori.

```python
# Ricalcola etichette/statistiche dalla segmentazione finale
n, labels, stats, centroids = cv2.connectedComponentsWithStats(
    img_final, connectivity=8
)

H_img, W_img = img_coins_gray.shape
img_bbox = cv2.cvtColor(img_coins_gray, cv2.COLOR_GRAY2BGR)

CLASSE = 0   # 0 = moneta (unica categoria in questo esempio)

print(f"{'cls':>4} {'cx_n':>8} {'cy_n':>8} {'w_n':>8} {'h_n':>8}  ← formato YOLO")
print("-" * 54)

yolo_linhas = []
for i in range(1, n):
    x0 = stats[i, cv2.CC_STAT_LEFT]
    y0 = stats[i, cv2.CC_STAT_TOP]
```

```python
    w  = stats[i, cv2.CC_STAT_WIDTH]
    h  = stats[i, cv2.CC_STAT_HEIGHT]

    cx_n = (x0 + w / 2) / W_img
    cy_n = (y0 + h / 2) / H_img
    w_n  = w / W_img
    h_n  = h / H_img

    yolo_linhas.append(f"{CLASSE} {cx_n:.4f} {cy_n:.4f} {w_n:.4f} {h_n:.4f}")
    print(f"{CLASSE:>4} {cx_n:>8.4f} {cy_n:>8.4f} {w_n:>8.4f} {h_n:>8.4f}")

    cv2.rectangle(img_bbox, (x0, y0), (x0 + w, y0 + h), (0, 255, 0), 4)
    cv2.putText(img_bbox, f"moeda", (x0 + 8, y0 + 60),
                cv2.FONT_HERSHEY_SIMPLEX, 3.8, (255, 0, 0), 5, cv2.LINE_AA)

# Esporta file di annotazione nel formato YOLO
with open("moedas.txt", "w") as f:
    f.write("\n".join(yolo_linhas))
print("\nAnnotazione salvata in monete.txt")

mm.show(
    [img_coins_gray, img_final, img_bbox],
    titles=["Originale", "Segmentazione Finale", "Bounding Boxes (formato YOLO)"],
    cols=3, figsize=(18, 6)
)
```

```
 cls     cx_n     cy_n      w_n      h_n  ← formato YOLO
-----------------------------------------------------
   0   0.7753   0.1102   0.2880   0.2117
   0   0.4784   0.0984   0.2359   0.1828
   0   0.1331   0.1225   0.2255   0.1637
   0   0.4464   0.3217   0.2469   0.1816
   0   0.7523   0.3471   0.2807   0.2074
   0   0.1664   0.3812   0.2745   0.2016
   0   0.4862   0.6104   0.3474   0.2598
   0   0.8115   0.6154   0.2760   0.2012
   0   0.1724   0.6023   0.2250   0.1711
   0   0.1893   0.8258   0.2536   0.1922
   0   0.7763   0.8652   0.2255   0.1734
   0   0.4854   0.8953   0.2750   0.2039

Annotazione salvata in monete.txt
```

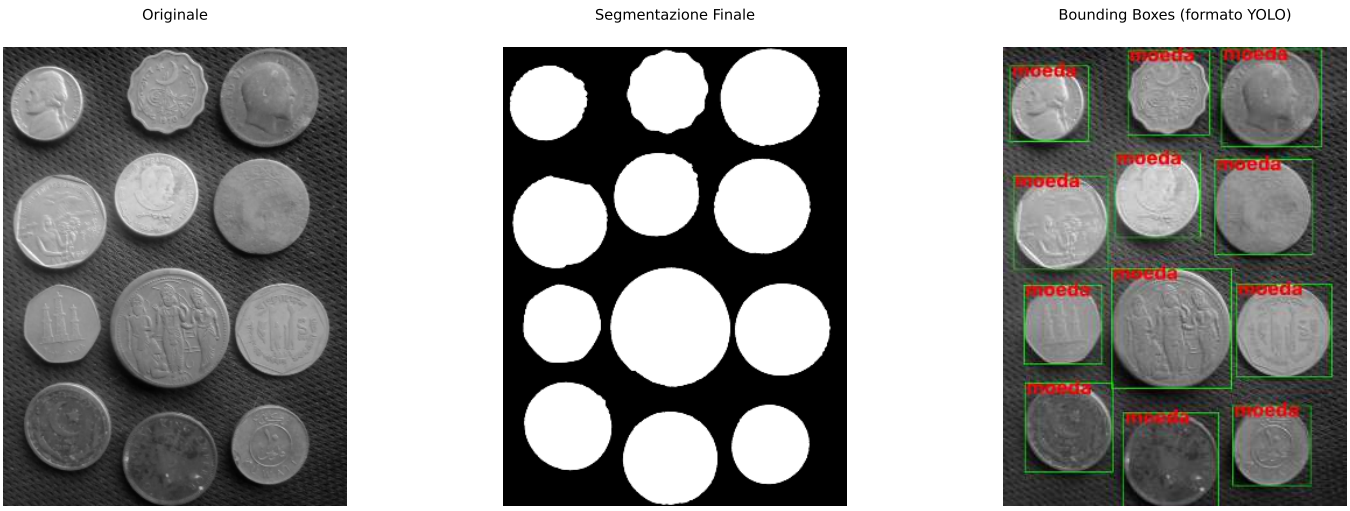


Figura 4.29: *Bounding boxes* derivate dai componenti connessi sovrapposti all'immagine originale. Le annotazioni vengono esportate nel formato YOLO (*classe cx cy w h*), con coordinate normalizzate nell'intervallo [0,1].

Il formato YOLO memorizza ogni oggetto in una riga contenente cinque campi:

```
classe cx cy w h
```

dove (cx, cy) rappresenta il centro della *bounding box* e (w, h) le sue dimensioni. Tutti i valori geometrici sono normalizzati nell'intervallo $[0, 1]$ rispetto alla larghezza e all'altezza dell'immagine. La **classe** è un identificatore intero associato a una categoria definita dal dataset (ad esempio, `0 → moneta`). Quando vi sono più categorie — come moneta d'oro (`0`), moneta d'argento (`1`) e disco di plastica (`2`) — è sufficiente assegnare l'identificatore corrispondente a ciascun oggetto prima dell'esportazione, mantenendo esattamente lo stesso formato di annotazione. Nell'esempio precedente, queste informazioni sono state memorizzate nel file `moedas.txt`.

Il flusso presentato in questo capitolo — segmentazione → etichettatura → estrazione delle *bounding box* — corrisponde concettualmente alla fase di **annotazione** (*labeling*) impiegata nella costruzione dei set di addestramento per i moderni detector. Strumenti specializzati, come Label Studio e Roboflow, automatizzano questo processo in immagini complesse, ma la logica fondamentale rimane la stessa: associare a ciascun oggetto una regione di interesse e una classe. In scenari controllati, con sfondo uniforme e oggetti ben separati, le tecniche morfologiche possono persino generare annotazioni automaticamente o fungere da punto di partenza per l'etichettatura manuale, riducendo significativamente lo sforzo di costruzione del dataset. Nelle applicazioni reali più complesse, tuttavia, la validazione umana rimane necessaria per garantire la qualità delle annotazioni.

ℹ Valutazione: IoU (*Intersection over Union*)

Un modo semplice per valutare la qualità di una segmentazione consiste nel confrontarla con una maschera di riferimento (*ground truth*). La metrica più utilizzata a questo scopo è la **IoU** (*Intersection over Union*):

$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \tag{4.16}$$

dove A rappresenta la segmentazione prodotta dall'algoritmo e B la segmentazione di riferimento.
Il valore della IoU varia tra 0 e 1. Quanto più alto è il valore, tanto maggiore è la sovrapposizione tra le maschere. Una IoU pari a 1 indica una corrispondenza perfetta tra la segmentazione ottenuta e il riferimento.

La stessa metrica è ampiamente utilizzata anche nel rilevamento degli oggetti, applicandola alle *bounding boxes* previste e annotate. In questo ambito, valori di IoU superiori a 0,5 sono spesso adottati come criterio minimo per considerare corretta una rilevazione.

Oltre la Morfologia

Le tecniche studiate in questo capitolo segmentano gli oggetti sfruttando la connettività spaziale, le operazioni morfologiche e il rilievo topografico. Esistono, tuttavia, approcci alternativi basati sul raggruppamento di caratteristiche, come l'algoritmo *k-means*, i modelli di miscela gaussiana (GMM) e metodi più recenti basati sull'apprendimento profondo. Queste tecniche verranno riprese nella Parte II del libro, dedicata alla Visione Artificiale.

4.6 Riassunto

In questo capitolo sono state presentate le principali tecniche di segmentazione e morfologia matematica, concludendo lo studio dell'elaborazione delle immagini nel dominio spaziale.

- **Pre-elaborazione e sogliatura:** La combinazione tra equalizzazione adattiva CLAHE e il metodo di Otsu si è dimostrata efficace per indurre una separazione bimodale nell'istogramma e semplificare la binarizzazione di immagini con illuminazione non uniforme.
- **Erosione e dilatazione:** Operatori morfologici fondamentali basati sulla ricerca di minimi e massimi locali in un intorno definito dall'elemento strutturante B. Sono operatori duali rispetto al complemento e sono stati implementati tramite le funzioni `mm.ero` e `mm.dil`.
- **Apertura e chiusura:** Composizioni di erosione e dilatazione che consentono di rimuovere rumori, smussare i contorni e colmare piccole lacune, preservando la struttura globale degli oggetti.
- **Ricostruzione morfologica:** Processo geodetico iterativo che propaga un marcatore all'interno dei limiti imposti da una maschera, costituendo la base di operatori come `mm.clohole` e `mm.edgeoff`.
- **Pipeline di pulizia binaria:** Flusso consolidato composto da CLAHE → Otsu → apertura → `mm.clohole` → apertura ristretta → `mm.edgeoff`, che produce maschere adatte all'analisi quantitativa.
- **Morfologia in scala di grigi:** Estensione algebrica basata su minimi e massimi ponderati, che consente operatori come il gradiente morfologico e i filtri *top-hat* per l'enfatizzazione di strutture locali.
- **Trasformata della Distanza e Watershed:** La Trasformata della Distanza ha permesso di generare marcatori automatici per l'algoritmo *watershed*, consentendo la separazione di oggetti adiacenti o parzialmente sovrapposti.
- **Componenti connesse e descrittori:** L'etichettatura delle regioni (`mm.label0`) e l'estrazione dei contorni hanno permesso di calcolare descrittori geometrici come area, centroide, perimetro, circolarità e *bounding boxes*.
- **Collegamento con la Visione Artificiale Moderna:** Le *bounding boxes* estratte tramite morfologia sono state esportate nel formato YOLO, evidenziando il legame tra tecniche classiche di segmentazione e sistemi moderni di rilevamento degli oggetti.

Il Capitolo 5 introdurrà tecniche di elaborazione nel **dominio della frequenza**, affrontando la **Trasformata di Fourier**, il filtraggio spettrale e i fondamenti della **compressione delle immagini**, inclusi DCT, JPEG e *wavelets*.

4.7 🤖 Uso del Gemini Notebook come Tutor Complementare

In questa edizione, l'uso del **Gemini Notebook** è incentivato come strumento complementare di apprendimento. Basato sull'intelligenza artificiale, il sistema utilizza esclusivamente i documenti forniti dall'autore come fonte di conoscenza, producendo risposte allineate al contenuto e all'approccio adottato nel corso di questo capitolo.

🎓 Studia con il Tutor Intelligente

🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 04

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato costruito esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, o seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ **Avviso sui Contenuti Generati dall'IA**

Sebbene sia uno strumento prezioso di supporto allo studio, il Gemini Notebook può occasionalmente produrre risposte incomplete, imprecise o errate. Si raccomanda di validare le informazioni consultando il materiale del capitolo, libri, articoli scientifici e altre fonti accademiche affidabili. Quando possibile, esegui e sperimenta gli esempi pratici presentati nel corso del testo per consolidare la comprensione dei concetti.

4.8 Elenco di Esercizi

1. **(10%)** Implementare manualmente il criterio di Otsu senza utilizzare `mm.threshold`. Calcolare la varianza tra le classi $\sigma_B^2(T)$ per tutte le soglie $T \in [0, 255]$ usando `mm.hist`, identificare la soglia ottimale T^* e confrontare il risultato con il valore ottenuto da OpenCV. Tracciare σ_B^2 in funzione di T ed evidenziare il punto di massimo.

2. **(15%)** Applicare la sogliatura adattativa con blocchi di dimensione 11, 31 e 51 a un'immagine contenente illuminazione non uniforme. Confrontare i risultati con la sogliatura globale di Otsu e discutere i vantaggi e i limiti di ciascun approccio.

3. **(15%)** Eseguire il *pipeline* watershed sull'immagine delle monete variando la soglia applicata alla Trasformata della Distanza (0.3, 0.5 e 0.7 volte il valore massimo). Spiegare come questo parametro influenzi la generazione dei marcatori, la separazione di oggetti adiacenti e l'occorrenza di sovra-segmentazione.

4. **(15%)** Utilizzando `mm.drawImg`, costruire una dimostrazione visiva passo passo dell'erosione di un'immagine binaria 7×7 con elemento strutturante quadrato 3×3. Per ogni posizione analizzata, indicare se l'elemento strutturante è completamente contenuto nell'oggetto e giustificare il valore attribuito al pixel di uscita.

5. **(15%)** Dimostrare sperimentalmente la dualità tra erosione e dilatazione verificando l'identità $(A \ominus B)^c = A^c \oplus \hat{B}$ utilizzando `mm.ero`, `mm.dil` e `mm.bnot`. Calcolare la differenza pixel per pixel tra i due membri dell'equazione e presentare il risultato utilizzando `mm.histImg` o una visualizzazione equivalente.

6. **(15%)** Implementare manualmente il gradiente morfologico utilizzando solo `mm.ero` e `mm.dil`, confrontando il risultato con `mm.gradm(img, B)`. Valutare l'effetto di diversi elementi strutturanti (quadrato 3×3, disco 5×5 e linea 1×9) sulla rilevazione dei bordi.

7. **(15%)** Costruire un *pipeline* completo per il conteggio e la classificazione di monete per dimensione (piccola, media e grande) utilizzando area e circolarità come descrittori. Generare manualmente una maschera di riferimento (*ground truth*) e calcolare la metrica IoU (*Intersection over Union*) per valutare la qualità della segmentazione. Presentare i risultati in tabella e tramite visualizzazioni prodotte con `mm.show`.

Riferimenti del Capitolo

La base teorica di questo capitolo si fonda sulle seguenti opere:

- Gonzalez (2018) per i concetti di segmentazione, sogliatura di Otsu, Trasformata della Distanza, *watershed*, morfologia matematica e descrittori di forma.
- Matheron (1975) e Serra (1982) per la fondazione teorica originale, la formulazione algebrica e lo sviluppo della Morfologia Matematica.
- Szeliski (2022) per la segmentazione basata su regioni, l'etichettatura delle componenti connesse, il *watershed* basato su marcatori e la valutazione della segmentazione tramite la metrica IoU.
- Bradski (2008) per l'utilizzo pratico della libreria OpenCV, incluse funzioni come `mm.dist`, `mm.watershed`, `mm.label0` e l'estrazione dei contorni.
- Redmon (2016) per l'introduzione ai moderni rilevatori della famiglia YOLO e la loro relazione con descrittori geometrici come le *bounding boxes* estratte mediante segmentazione.
- Singh (2024) per la costruzione di labirinti complessi basati su cicli hamiltoniani su mosaici quasicristallini, utilizzati come esempio applicativo della Trasformata della Distanza Geodetica e degli algoritmi di ricerca di percorsi.
- Zampirolli (2025) per l'implementazione degli operatori morfologici, delle trasformate geodetiche e della risoluzione di labirinti tramite propagazione delle distanze in domini ristretti.

Executar Colab | Abrir GitHub

4.9 💻 Parte Pratica con Esercizi di Programmazione

Questa lista trasforma i concetti del Capitolo 4 in un percorso pratico di segmentazione e morfologia matematica. Gli EP iniziano con la sogliatura e avanzano fino all'etichettatura e ai descrittori delle componenti, utilizzando sempre matrici di piccole dimensioni affinché ogni pixel possa essere verificato a mano.

! Regola comune degli EP morfologici

Nelle operazioni con vicinato, **non eseguire il padding**. Per ogni pixel, valutare solo le posizioni dell'elemento strutturante che ricadono all'interno del dominio dell'immagine. Questa è la stessa idea delle implementazioni didattiche in `morph.py`, come `mm.dil0`, `mm.ero0`, `mm.dil1` e `mm.label0`: il vicinato viene ritagliato dal dominio valido dell'immagine.

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EP)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
```

```
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, eseguire `TestSuite("EP04_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare direttamente codice Python, senza salvare un file, utilizzare `run_code(codigo)` passando il codice come *stringa* in una variabile `codigo`:

```
codigo = """
from morph import mm
# ... tuo codice qui ...
"""
TestSuite("EP04_01").run_code(codigo)
```

4.9.1 EP04_01 Soglia Globale con Soglia Fissa

Negli **scanner di documenti** e nei **sistemi di lettura di codici a barre**, la prima fase dell'elaborazione consiste sempre nel separare ciò che è "oggetto" (inchiostro, testo, barre) da ciò che è "sfondo" (carta, imballaggio). La **soglia globale** fa esattamente questo: confronta ogni pixel con un'unica soglia T e decide, in tempo reale, se esso appartiene alla classe chiara o alla classe scura. È l'operatore di segmentazione più semplice — eppure è alla base di gran parte dei *pipeline* industriali di ispezione visiva. Vedi in Figura 4.30 una simulazione di questo EP.

4.9.1.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Soglia:** Leggere l'intero T (soglia di decisione).
3. **Dati:** Leggere i valori interi della matrice originale riga per riga.
4. **Mappatura:** Per ogni pixel p, calcolare il nuovo valore tramite l'equazione:

$$p' = \begin{cases} 255, & \text{se } p > T \\ 0, & \text{se } p \leq T \end{cases}$$

5. **Output:** Visualizzare la matrice binarizzata con dimensioni $L \times C$.

4.9.1.2 📌 Vincoli Computazionali

- **Binarizzazione:** L'output contiene **solo** i valori 0 o 255.
- **Confronto rigoroso:** Il criterio usa $> T$ (i pixel uguali a T diventano sfondo).
- **Tipo:** Il risultato finale deve essere intero.
- **Osservazione:** Questo EP segue la convenzione di OpenCV (`cv2.THRESH_BINARY`): solo i pixel con valore **maggiore di** T diventano bianchi (`255`); i pixel con valore **uguale a** T restano neri (`0`).

4.9.1.3 🧠 Fondamenti Teorici

Parametro	Tipo	Impatto Visivo
T **piccolo**	Intero	La maggior parte dei pixel diventa bianca
T **grande**	Intero	La maggior parte dei pixel diventa nera
T **ben scelto**	Intero	Separa nettamente oggetto e sfondo

4.9.1.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero T.
- Righe successive: Elementi interi della matrice originale.

Output:

- Matrice binarizzata in L righe e C colonne, valori 0 o 255 separati da spazi.

4.9.1.5 📌 Esempi

Input	Output	Osservazione
2 4 100 0 99 100 180 255 30 120 80	0 0 0 255 255 0 255 0	$T = 100$: solo i pixel con valore maggiore di 100 diventano bianchi; pertanto, 99 e 100 diventano neri.
1 3 0 0 50 255	0 255 255	$T = 0$: solo i pixel con valore strettamente maggiore di 0 diventano bianchi.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0401-limiar.html

Figura 4.30: Simulatore EP04_01: Soglia Globale con Soglia Fissa (p' = (p > T) ? 255 : 0)

```
%%writefile EP04_01.py
# Codice Python
```

```
Overwriting EP04_01.py
```

```
TestSuite("EP04_01.py").run()
```

```
<IPython.core.display.HTML object>
```

4.9.2 EP04_02 📊 Soglia Automatica di Otsu

Scegliere manualmente la soglia T funziona quando l'illuminazione è stabile, ma in **microscopia digitale** e nell'**ispezione di strisci di sangue**, ogni campione ha un contrasto diverso — una soglia fissa fallirebbe da un'immagine all'altra. Il **metodo di Otsu** risolve questo problema trovando, da solo, la soglia che **massimizza la separazione statistica** tra le due classi di pixel, rendendo la segmentazione automatica e adattiva. Vedi in Figura 4.31 una simulazione di questo EP.

4.9.2.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Dati:** Leggere i valori interi della matrice originale riga per riga.
3. **Istogramma:** Costruire l'istogramma $h[i]$, $i = 0, \dots, 255$, contando quanti pixel hanno valore i.
4. **Ricerca della soglia:** Per ogni candidato T da 1 a 255, calcolare la **varianza tra le classi**:

$$\sigma_B^2(T) = \frac{n_0 \cdot n_1}{N^2} (m_0 - m_1)^2$$

   dove n_0, n_1 sono le quantità di pixel con valore $< T$ e $\geq T$, m_0, m_1 sono le loro medie, e $N = L \times C$.
5. **Scelta:** La soglia ottimale T^* è quella che massimizza $\sigma_B^2(T)$ (in caso di pareggio, mantenere la **prima** trovata).
6. Applicazione: Binarizzare l'immagine usando T*, applicando:

$$p' = \begin{cases} 255, & \text{se } p > T^* \\ 0, & \text{se } p \leq T^* \end{cases}$$

4.9.2.2 📌 Vincoli Computazionali

- **Candidati validi:** Ignorare T che lasci $n_0 = 0$ o $n_1 = 0$ (classe vuota).
- **Pareggio:** Mantenere sempre la **prima** T che ha raggiunto il valore massimo di σ_B^2.
- **Tipo:** T^* e la matrice di uscita devono essere interi.
- **Convenzione OpenCV:** La binarizzazione segue `cv2.THRESL_BINARY`; i pixel con valore esattamente uguale a T^* diventano neri.

4.9.2.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto
$\sigma_B^2(T)$ **alta**	Classi ben separate in T	T è un buon candidato come soglia
Istogramma bimodale	Due "colline" distinte	Otsu trova la valle tra di esse
Istogramma unimodale	Una sola "collina"	Otsu sceglie comunque *qualche* T, ma la segmentazione è poco affidabile

4.9.2.4 📦 Specifica di Input e Output (VPL)

Input:

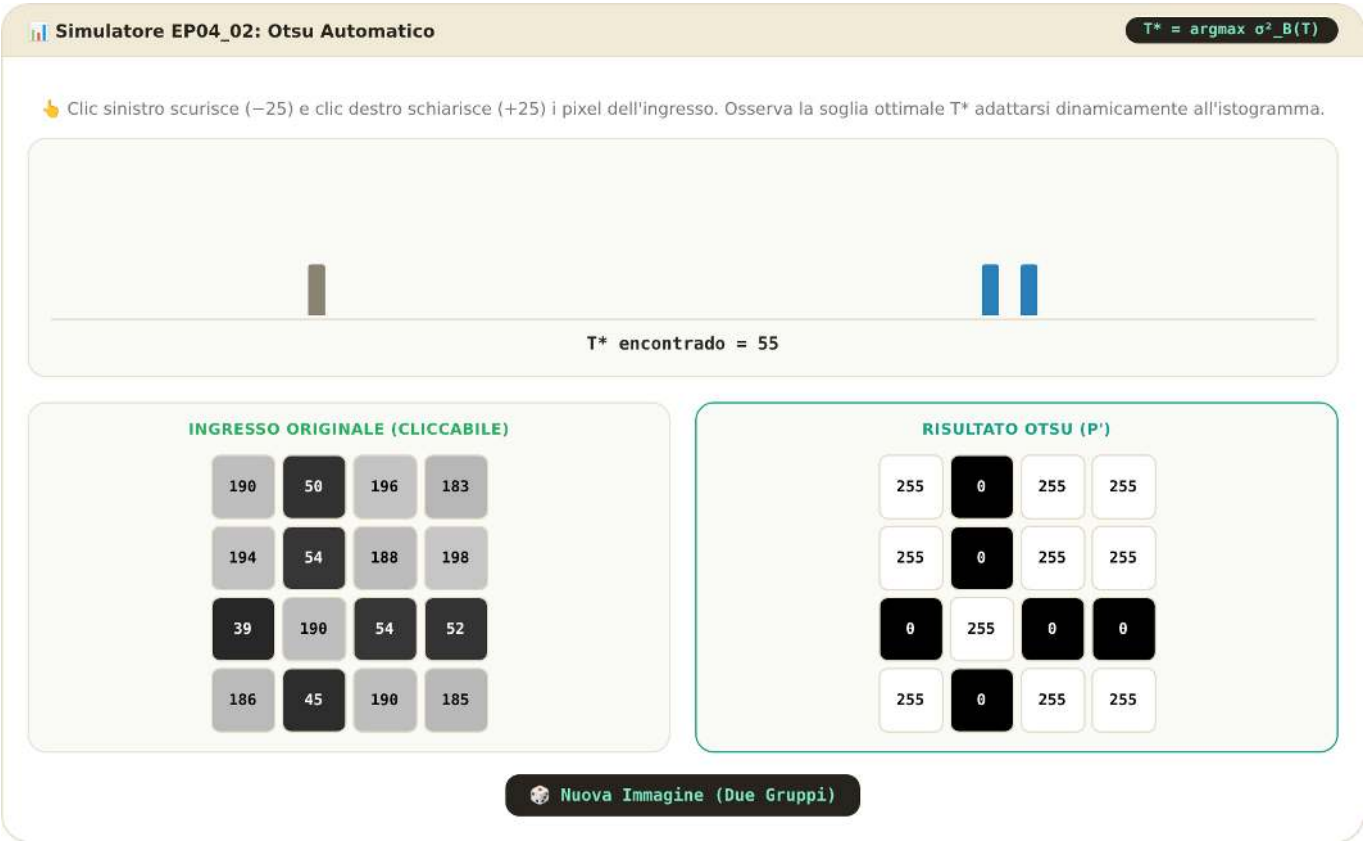
- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi interi della matrice originale.

Output:

- Matrice binarizzata in L righe e C colonne, valori 0 o 255.

4.9.2.5 📌 Esempi

Input	Output	Osservazione
4 4 12 12 12 200 12 12 200 200 12 200 200 200 200 200 200 200	0 0 0 255 0 0 255 255 0 255 255 255 255 255 255 255	Istogramma bimodale chiaro: 12 e 200
1 2 10 250	0 250	Solo due valori: T^* si colloca sul maggiore



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0402-otsu.html

Figura 4.31: Simulatore EP04_02: Sogliatura Automatica di Otsu (T* = argmax ²_B(T))

```
%%writefile EP04_02.py
# Codice Python
```

```
Overwriting EP04_02.py
```

```
TestSuite("EP04_02.py").run()
```

```
<IPython.core.display.HTML object>
```

4.9.3 EP04_03 🌱 Dilatazione Binaria Piana (mm.dil0)

Nella **microscopia di particelle** e nell'**OCR di targhe automobilistiche usurate**, tratti sottili o discontinui devono essere "ingrossati" affinché il riconoscimento funzioni. La **dilatazione morfologica** fa esattamente questo: espande le regioni chiare utilizzando un elemento strutturante B — la stessa operazione implementata in `morph.py` come `mm.dil0(f, B)`, utilizzata quando B è **piano** (senza pesi, solo 0/1). Vedi in Figura 4.32 una simulazione di questo EP.

4.9.3.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) di f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice f (l'immagine originale), riga per riga.
5. **Riflessione:** Costruire B_{ref}, la versione di B riflessa di 180° (righe e colonne invertite) — esattamente come fa `mm.dil0` internamente.
6. **Vicinato senza padding:** Per ogni pixel (y, x), percorrere le posizioni (by, bx) di B_{ref} centrate su (y, x), usando lo spostamento

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\tfrac{L_B}{2} + 0{,}5, \quad o_x = -\tfrac{C_B}{2} + 0{,}5$$

   Scartare ogni (v_y, v_x) al di fuori di $[0, L) \times [0, C)$ — **non riempire con zeri**.
7. **Mappatura:** Calcolare ogni pixel di uscita come il **massimo** tra $f(y, x)$ e tutti gli $f(v_y, v_x)$ validi la cui posizione corrispondente in B_{ref} vale 1:

$$g(y, x) = \max\Big(f(y, x), \max_{\substack{(v_y, v_x) \text{ valido} \\ B_{ref}(by, bx)=1}} f(v_y, v_x)\Big)$$

8. **Uscita:** Visualizzare la matrice g con dimensioni $L \times C$.

4.9.3.2 📌 Vincoli Computazionali

- **Senza padding:** Non inventare mai vicini al di fuori dell'immagine; utilizzare solo quelli che esistono realmente.
- **Riflessione obbligatoria:** B deve essere riflesso prima dell'applicazione (è ciò che distingue `mm.dil0` da una semplice ricerca del massimo).
- **Robustezza ai bordi:** Se nessuna posizione valida di $B_{ref} = 1$ ricade all'interno del dominio per un dato pixel, questo **mantiene il suo valore originale**.

4.9.3.3 🧠 Fondamento Teorico

Concetto	Significato	Impatto Visivo
Dilatazione	$g \geq f$ sempre (estensiva)	Le regioni chiare crescono, i buchi scuri si restringono
B più grande	Vicinato più ampio	Crescita più aggressiva
Riflessione di B	$B_{ref}(y, x) = B(-y, -x)$	Garantisce la definizione formale di Minkowski della dilatazione

4.9.3.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Successive L_B righe: elementi interi (0 o 1) della matrice B.
- Successive L righe: elementi interi della matrice f.

Output:

- Matrice g in L righe e C colonne, valori interi separati da spazio.

4.9.3.5 📌 Esempi

Input	Output	Osservazione
3 3 3 3 0 1 0 1 1 1 0 1 0 0 0 0 0 9 0 0 0 0	0 9 0 9 9 9 0 9 0	B a croce simmetrico: punto isolato si espande a croce
1 4 1 3 1 1 1 10 200 5 80	200 200 200 80	B orizzontale: ogni pixel "attira" il massimo dei vicini della riga

```
%%writefile EP04_03.py
# Codice Python
```

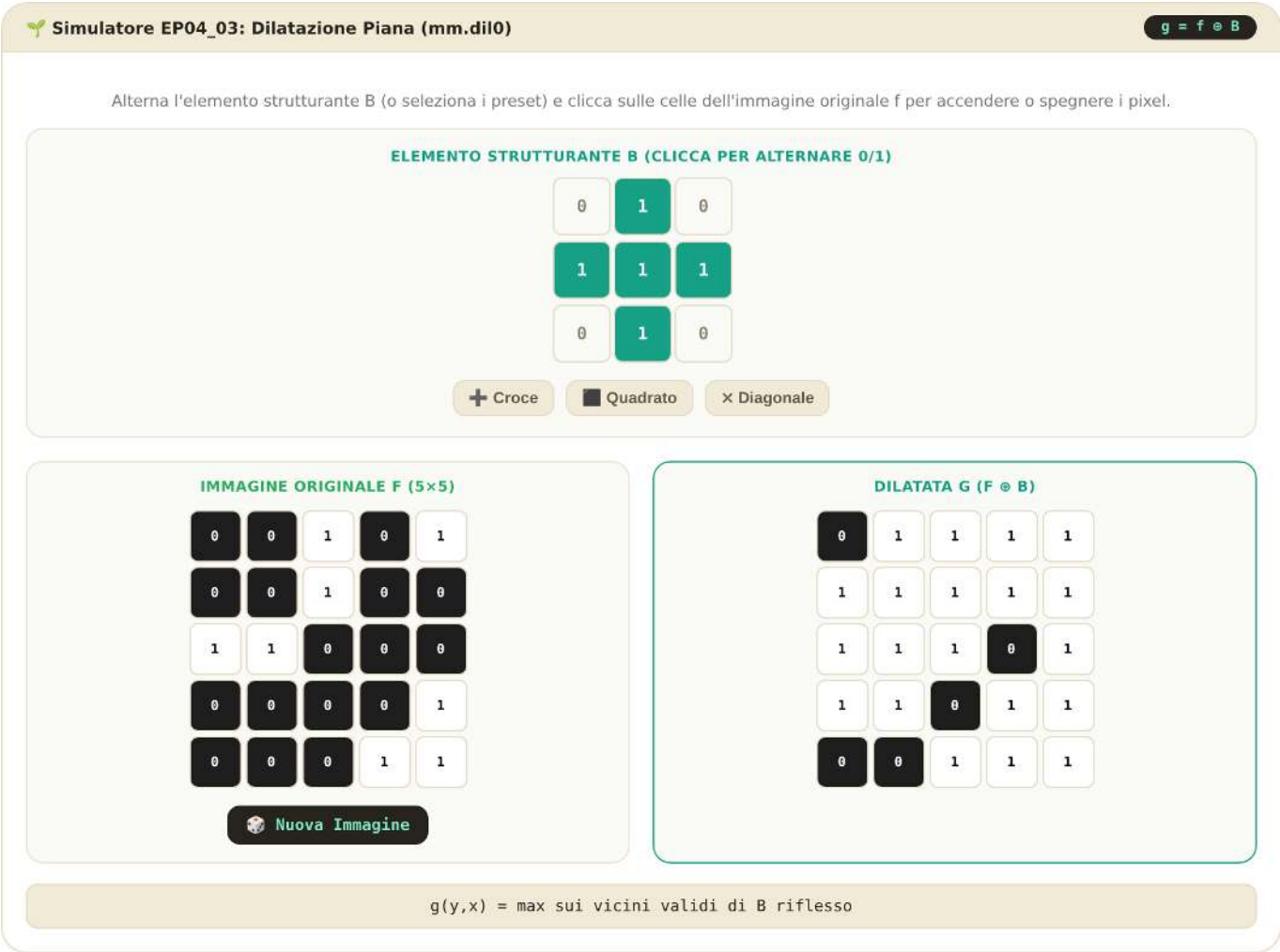
```
Overwriting EP04_03.py
```

```
TestSuite("EP04_03.py").run()
```

```
<IPython.core.display.HTML object>
```

4.9.4 EP04_04 🪨 Erosione Binaria Piatta (mm.ero0)

Se la dilatazione ingrossa, l'**erosione** assottiglia. Nei **sistemi di conteggio cellulare**, viene utilizzata per **separare cellule che si toccano**: "consumando" i bordi di ogni regione, le connessioni sottili tra gli oggetti scompaiono prima ancora che venga effettuato qualsiasi conteggio. In `morph.py`, questa è l'operazione `mm.ero0(f, B)` — il **duale** esatto della dilatazione, e l'unica delle due che **non** riflette l'elemento strutturante. Vedi nella Figura 4.33 una simulazione di questo EP.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0403-dilatacao.html

Figura 4.32: Simulatore EP04_03: Dilatazione Binaria Piana (g = f B)

4.9.4.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) da f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice f (l'immagine originale), riga per riga.
5. **Vicinato senza padding (senza riflessione!):** Per ogni pixel (y, x), percorrere le posizioni (by, bx) di B **nell'ordine originale** (senza riflettere), usando lo stesso spostamento dell'EP04_03:

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\frac{L_B}{2} + 0{,}5, \quad o_x = -\frac{C_B}{2} + 0{,}5$$

Scartare ogni (v_y, v_x) al di fuori di $[0, L) \times [0, C)$. 6. **Mappatura:** Calcolare ogni pixel di uscita come il **minimo** tra $f(y, x)$ e tutti gli $f(v_y, v_x)$ validi la cui posizione corrispondente in B vale 1:

$$g(y, x) = \min\Big(f(y, x), \min_{\substack{(v_y, v_x) \text{ valido} \\ B(by, bx)=1}} f(v_y, v_x)\Big)$$

7. **Uscita:** Visualizzare la matrice g con dimensioni $L \times C$.

4.9.4.2 📌 Vincoli Computazionali

- **Senza riflessione:** Diversamente dalla dilatazione, B viene utilizzato **esattamente come letto** — riflettere qui sarebbe un errore concettuale grave.
- **Senza padding:** I vicini al di fuori dell'immagine vengono semplicemente ignorati, mai trattati come 0.
- **Robustezza ai bordi:** Se nessuna posizione valida di $B = 1$ ricade all'interno del dominio, il pixel mantiene il suo valore originale.

4.9.4.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto Visivo
Erosione	$g \leq f$ sempre (anti-estensiva)	Le regioni chiare si restringono, il rumore puntuale scompare
Dualità	$\text{ero}(f, B) = -\text{dil}(-f, B_{ref})$	Erosione e dilatazione sono "specchi" matematici
B **più grande**	Erosione più aggressiva	Gli oggetti sottili scompaiono completamente

4.9.4.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

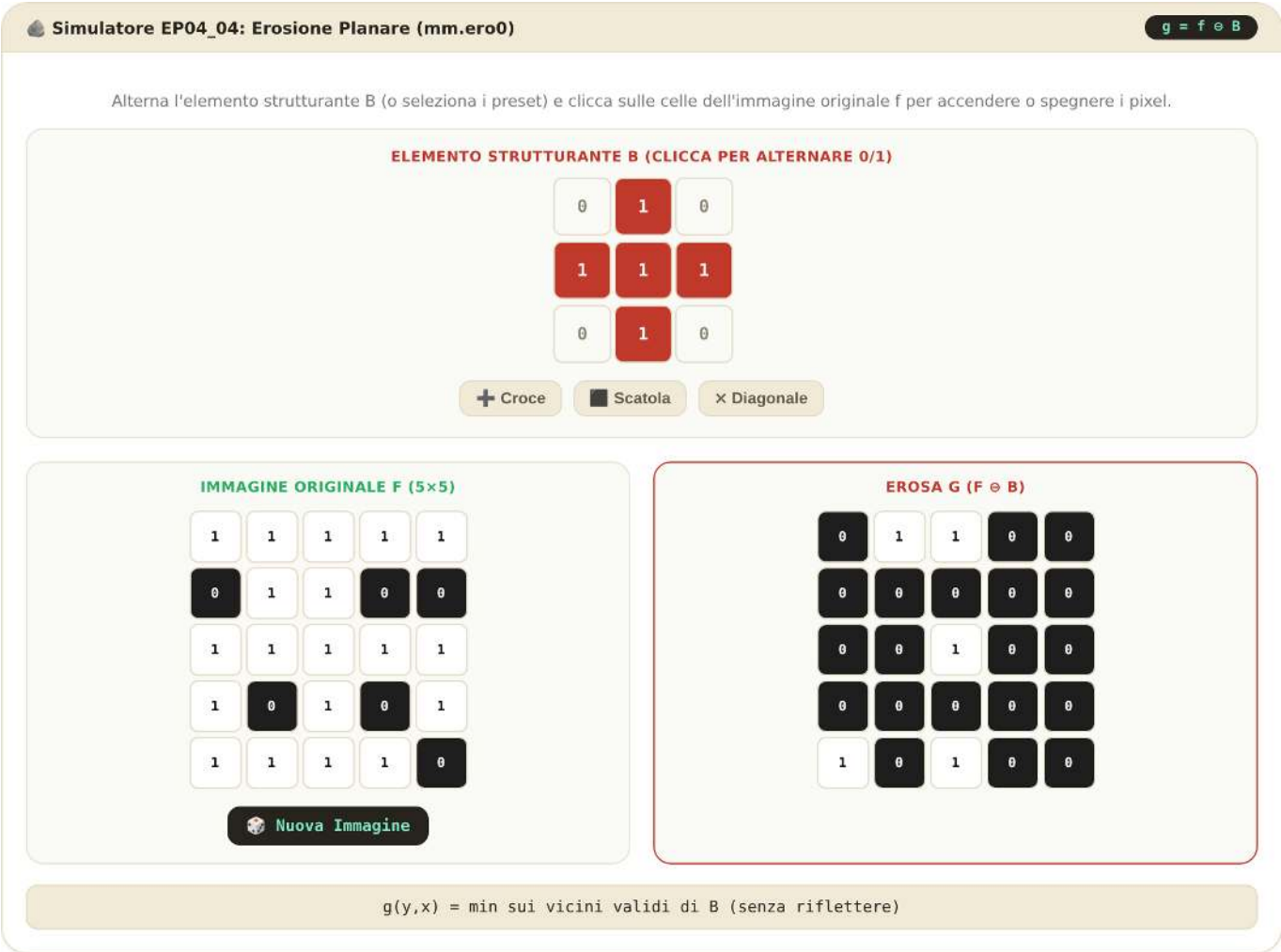
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (0 o 1) della matrice B.
- Prossime L righe: elementi interi della matrice f.

Uscita:

- Matrice g in L righe e C colonne, valori interi separati da spazio.

4.9.4.5 📌 Esempi

Ingresso	Uscita	Osservazione
3 3 3 3 0 1 0 1 1 1 0 1 0 9 9 9 9 0 9 9 9 9	9 0 9 0 0 0 9 0 9	Il “buco” centrale (0) si propaga a croce
1 4 1 3 1 1 1 10 200 5 80	10 5 5 80	*B* orizzontale: ogni pixel “attira” il minimo dei vicini della riga



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0404-erosao.html

Figura 4.33: Simulatore EP04_04: Erosione Binaria Piana (g = f B)

```
%%writefile EP04_04.py
# Codice Python
```

```
Overwriting EP04_04.py
```

```
TestSuite("EP04_04.py").run()
```

```
<IPython.core.display.HTML object>
```

4.9.5 EP04_05 🧹 Apertura Morfologica (Rimozione del Rumore)

Le immagini acquisite da **sensori a basso costo**, come quelli dei droni agricoli, sono spesso disseminate di piccoli punti di rumore — pixel isolati che non rappresentano nulla di reale. Applicare l'erosione seguita dalla dilatazione con lo **stesso** elemento strutturante produce l'**apertura**: essa "pulisce" punti e sottili protuberanze, ma restituisce all'oggetto principale praticamente la sua dimensione originale. È la combinazione classica utilizzata nella **pre-elaborazione di immagini satellitari** prima di qualsiasi conteggio dell'area coltivata. Vedi in Figura 4.34 una simulazione di questo EP.

4.9.5.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) da f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice binaria f (valori 0 o 1), riga per riga.
5. **Erosione:** Calcolare $e = f \ominus B$, usando esattamente l'algoritmo di EP04_04 (senza riflettere B, senza padding).
6. **Dilatazione:** Calcolare $g = e \oplus B$, usando esattamente l'algoritmo di EP04_03 (riflettendo B, senza padding) — ma ora applicato su e, non su f.
7. **Output:** Visualizzare la matrice risultante g (l'**apertura** di f tramite B) con dimensioni $L \times C$.

4.9.5.2 📌 Vincoli Computazionali

- **Ordine fisso:** È **sempre** prima erosione, poi dilatazione — l'ordine inverso definisce un altro operatore (chiusura, nel prossimo EP).
- **Stesso B:** L'elemento strutturante utilizzato nell'erosione e nella dilatazione deve essere identico.
- **Nessun padding in nessuna delle due fasi.**

4.9.5.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto Visivo
Anti-estensività	$g \subseteq f$ sempre	L'apertura non crea mai nuovi pixel, li rimuove soltanto
Idempotenza	apertura(apertura(f)) = apertura(f)	Applicarla di nuovo non cambia più nulla
Punti isolati	Più piccoli di B	Vengono completamente eliminati
Nucleo dell'oggetto	Più grande di B	Viene recuperato quasi intatto dalla dilatazione finale

4.9.5.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.

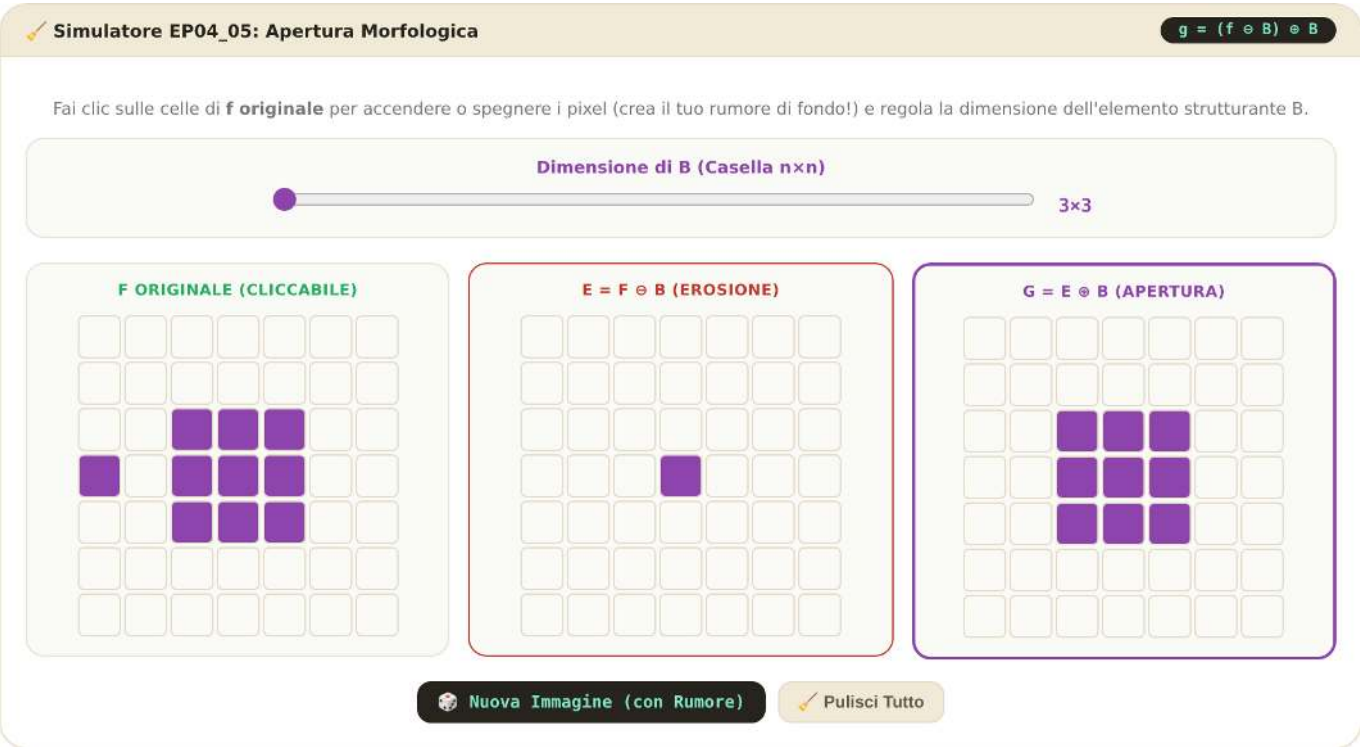
- Prossime L_B righe: elementi interi (0 o 1) della matrice B.
- Prossime L righe: elementi interi (0 o 1) della matrice f.

Output:

- Matrice risultante in L righe e C colonne, valori 0 o 1.

4.9.5.5 📌 Esempi

Input	Output	Osservazione
7 7 3 3 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Punti isolati e la sottile protuberanza scompaiono; il quadrato centrale sopravvive



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0405-abertura.html

Figura 4.34: Simulatore EP04_05: Apertura Morfologica (g = (f B) B)

```
%%writefile EP04_05.py
# Codice Python
```

```
Overwriting EP04_05.py
```

```
TestSuite("EP04_05.py").run()
```

```
<IPython.core.display.HTML object>
```

4.9.6 EP04_06 🧩 Chiusura Morfologica (Riempimento delle Lacune)

Nella **digitalizzazione delle impronte digitali**, i solchi della pelle talvolta vengono interrotti da sporco o secchezza, creando piccole lacune nella curva continua che dovrebbe esistere. La **chiusura** — dilatazione seguita da erosione con lo stesso elemento strutturante — è l'operatore duale dell'apertura: **riempie piccoli buchi e rientranze strette**, senza alterare significativamente il contorno esterno dell'oggetto. È la fase standard prima di estrarre lo scheletro di un'impronta digitale. Vedi una simulazione di questo EP in Figura 4.35.

4.9.6.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) da f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice binaria f (valori 0 o 1), riga per riga.
5. **Dilatazione:** Calcolare $d = f \oplus B$, usando esattamente l'algoritmo del EP04_03 (riflettendo B, senza padding).
6. **Erosione:** Calcolare $g = d \ominus B$, usando esattamente l'algoritmo del EP04_04 (senza riflettere B, senza padding) — ora applicato su d, non su f.
7. **Uscita:** Visualizzare la matrice risultante g (la **chiusura** di f per B) con dimensioni $L \times C$.

4.9.6.2 📌 Vincoli Computazionali

- **Ordine fisso:** È **sempre** prima la dilatazione, poi l'erosione — l'ordine inverso è l'apertura del EP04_05.
- **Stesso B:** L'elemento strutturante usato nella dilatazione e nell'erosione deve essere identico.
- **Nessun padding in nessuna delle due fasi.**

4.9.6.3 🧠 Fondamento Teorico

Concetto	Significato	Impatto Visivo
Estensività	$g \supseteq f$ sempre	La chiusura non rimuove mai pixel, solo aggiunge
Idempotenza	$\text{chiudi}(\text{chiudi}(f)) = \text{chiudi}(f)$	Applicare di nuovo non cambia più nulla
Piccoli buchi	Minori di B	Vengono completamente riempiti
Dualità	$\text{chiudi}(f) = \overline{\text{apri}(\bar{f})}$	È l'apertura applicata al "negativo" dell'immagine

4.9.6.4 📦 Specifica di Input e Output (VPL)

Input:

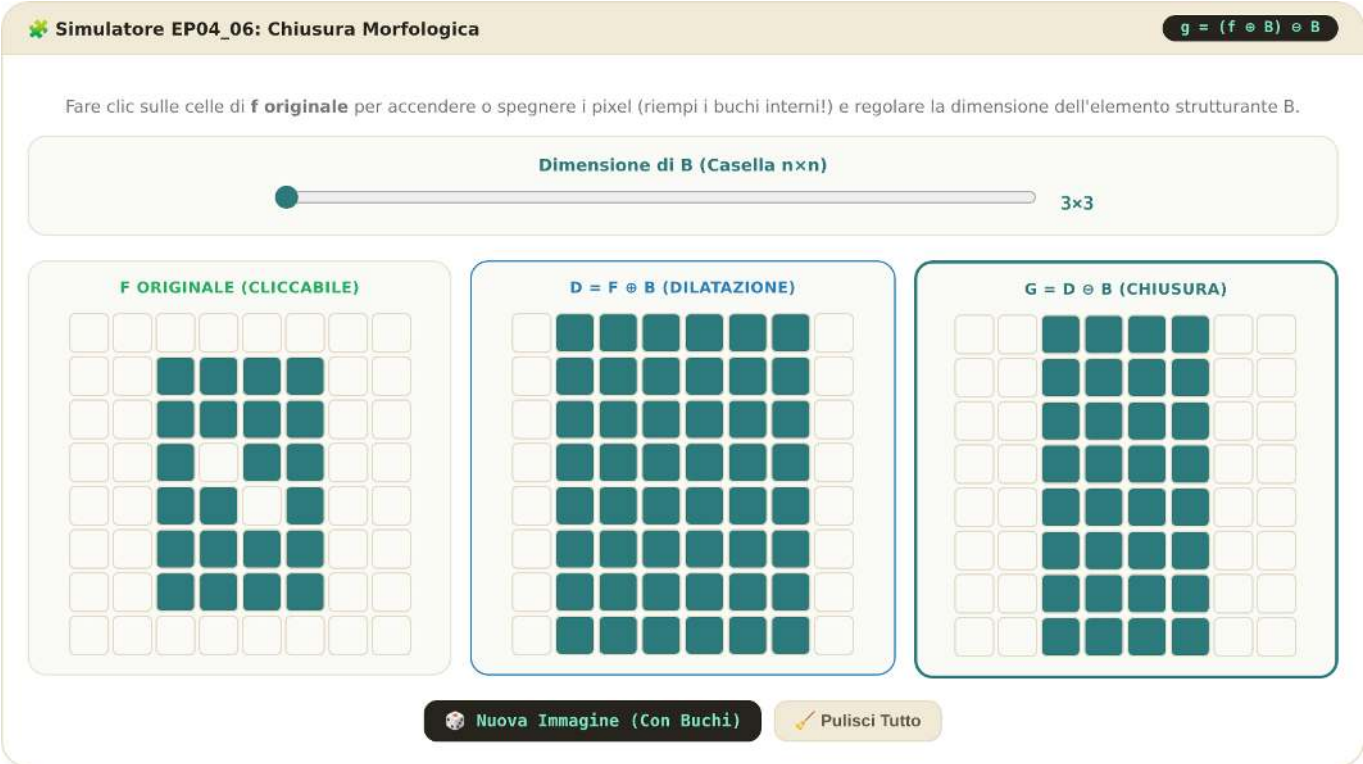
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (0 o 1) della matrice B.
- Prossime L righe: elementi interi (0 o 1) della matrice f.

Output:

- Matrice risultante in L righe e C colonne, valori 0 o 1.

4.9.6.5 📌 Esempi

Input	Output	Osservazione
8 8 3 3 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 1 1 0 0 0 0 1 1 0 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0	I due buchi interni non adiacenti vengono completamente riempiti



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0406-fechamento.html

Figura 4.35: Simulatore EP04_06: Chiusura Morfologica (g = (f B) B)

```
%%writefile EP04_06.py
# Codice Python
```

```
Overwriting EP04_06.py
```

```
TestSuite("EP04_06.py").run()
```

```
<IPython.core.display.HTML object>
```

4.9.7 EP04_07 Dilatazione ed Erosione con Pesi (mm.dil1 / mm.ero1)

Finora, l'elemento strutturante diceva solo "questo vicino conta" o "non conta" — ma nei **modelli digitali di elevazione** (usati nei GIS e nella pianificazione del drenaggio urbano), ogni vicino dovrebbe avere un **peso diverso** a seconda della distanza o della direzione del rilievo. Le versioni **ponderate** della dilatazione e dell'erosione, implementate in `morph.py` come `mm.dil1(f, b)` e `mm.ero1(f, b)`, sommano (o sottraggono) il peso di ciascun vicino prima di prendere il massimo (o il minimo) — generalizzando tutto ciò che è stato fatto negli EP precedenti. Vedere in Figura 4.36 una simulazione di questo EP.

4.9.7.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) di f.
2. **Dimensioni di b:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante ponderato.
3. **Pesi:** Leggere la matrice b di pesi **interi** (possono essere negativi, zero o positivi), riga per riga.
4. **Dati:** Leggere la matrice f (l'immagine originale), riga per riga.
5. **Vicinato senza padding:** Per ogni pixel (y, x), percorrere **tutte** le posizioni (by, bx) di b (non solo dove varrebbe 1 — qui **tutto** il peso partecipa), usando lo stesso spostamento degli EP precedenti:

$$v_y = y + by + o_y, \quad v_x = x + bx + o_x, \quad o_y = -\tfrac{L_B}{2} + 0{,}5, \quad o_x = -\tfrac{C_B}{2} + 0{,}5$$

   Scartare ogni (v_y, v_x) fuori da $[0, L) \times [0, C)$.
6. **Dilatazione ponderata:** Calcolare

$$g_{dil}(y, x) = \max\Big(f(y, x), \max_{(v_y, v_x)\ \text{valido}} \big(f(v_y, v_x) + b(by, bx)\big)\Big)$$

7. **Erosione ponderata:** Calcolare, **usando lo stesso b e senza riflessione**:

$$g_{ero}(y, x) = \min\Big(f(y, x), \min_{(v_y, v_x)\ \text{valido}} \big(f(v_y, v_x) - b(by, bx)\big)\Big)$$

8. **Uscita:** Mostrare **prima** la matrice completa g_{dil}, e **poi** la matrice completa g_{ero}.

4.9.7.2 📌 Vincoli Computazionali

- **Nessuna delle due riflette b** — la versione ponderata non usa riflessione, nemmeno nella dilatazione (diversamente da `mm.dil0`).
- **Tutti i pesi partecipano:** Non esiste qui il filtro "$B = 1$"; anche il peso 0 entra nel calcolo.
- **Senza padding:** i vicini fuori dall'immagine sono ignorati, mai virtualmente riempiti.
- **Tipo:** L'uscita può contenere valori negativi o maggiori di 255 — **non** c'è *clipping* in questo EP.
- **Suggerimento:** Per rimuovere i messaggi di overflow quando si superano i limiti del tipo uint8, includere all'inizio del codice:

```
import warnings
warnings.filterwarnings("ignore")
```

4.9.7.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto Visivo
Peso positivo	"Spinge" il valore del vicino verso l'alto nella dilatazione	Simula un rilievo che sale in quella direzione
Peso negativo	Riduce il contributo del vicino	Simula distanza o attenuazione direzionale
Dualità ponderata	$\text{ero1}(f, b) = -\text{dil1}(-f, b)$	La simmetria tra le due operazioni si mantiene anche con i pesi

4.9.7.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (possono essere negativi) della matrice b.
- Prossime L righe: elementi interi della matrice f.

Output:

- Prima la matrice g_{dil} in L righe e C colonne.
- Successivamente la matrice g_{ero} in L righe e C colonne.

4.9.7.5 📌 Esempi

Input	Output	Osservazione
3 3 3 3 0 1 0 1 2 1 0 1 0 10 20 30 40 50 60 70 80 90	50 60 61 80 90 91 81 91 92 8 9 19 9 10 20 39 40 50	Il peso centrale 2 accelera la crescita nella dilatazione e il restringimento nell'erosione

```
%%writefile EP04_07.py
# Codice Python
```

```
Overwriting EP04_07.py
```

```
TestSuite("EP04_07.py").run()
```

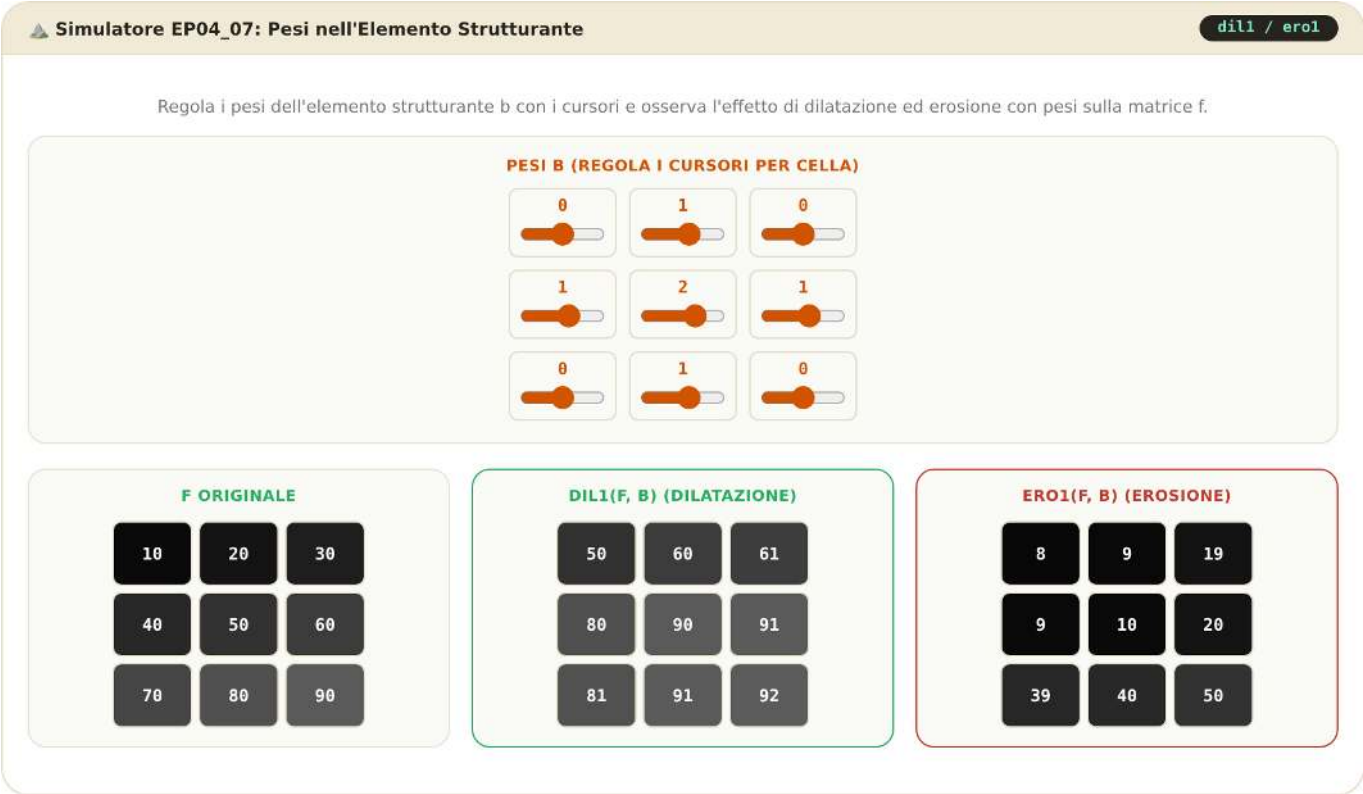
```
<IPython.core.display.HTML object>
```

4.9.8 EP04_08 🌋 Gradiente Morfologico, Top-hat e Black-hat

Nell'**ispezione automatica dei circuiti stampati**, tre domande ricorrono continuamente: dove sono i **bordi** dei componenti? Quali **dettagli chiari e piccoli** (come i punti di saldatura) si distinguono dallo sfondo? Quali **incavi scuri** (come le cricche) lo sfondo nasconde? Una singola coppia erosione/dilatazione risponde a tutte e tre: il **gradiente morfologico** evidenzia i contorni, il **top-hat** rivela i picchi stretti, e il **black-hat** rivela le valli strette — tre strumenti, un solo intorno. Vedi in Figura 4.37 una simulazione di questo EP.

4.9.8.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** Leggere gli interi L (righe) e C (colonne) di f.
2. **Dimensioni di B:** Leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** Leggere la matrice B con valori 0 o 1, riga per riga.
4. **Dati:** Leggere la matrice f (l'immagine originale, in scala di grigi), riga per riga.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0407-pesos.html

Figura 4.36: Simulatore EP04_07: Dilatazione ed Erosione con Pesi (mm.dil1 / mm.ero1)

5. **Operatori di base:** Calcolare, esattamente come negli EP 04_03 fino a 04_06:
   - $d = f \oplus B$ (dilatazione),
   - $e = f \ominus B$ (erosione),
   - apertura $= e \oplus B$,
   - chiusura $= d \ominus B$.
6. **Gradiente morfologico:** $\text{grad}(y, x) = d(y, x) - e(y, x)$.
7. **Top-hat:** $\text{tophat}(y, x) = f(y, x) - \text{apertura}(y, x)$.
8. **Black-hat:** $\text{blackhat}(y, x) = \text{chiusura}(y, x) - f(y, x)$.
9. **Output:** Mostrare, **in questo ordine**, le tre matrici complete: gradiente, top-hat, black-hat.

4.9.8.2 📌 Vincoli Computazionali

- **Nessun padding in nessuna fase intermedia** — dilatazione, erosione, apertura e chiusura seguono le stesse regole di vicinato degli EP precedenti.
- **Nessun *clipping*:** le tre uscite possono contenere qualsiasi valore intero (il gradiente è sempre ≥ 0, ma anche top-hat e black-hat lo sono).
- **Riutilizzo:** d e e devono essere calcolati **una sola volta** e riutilizzati per costruire apertura, chiusura e gradiente.

4.9.8.3 🧠 Fondamenti Teorici

Operatore	Formula	Cosa rivela
Gradiente	$d - e$	Bordi: zero in regioni piatte, alto nelle transizioni
Top-hat	$f - \text{apertura}(f)$	Elementi **chiari e sottili**, più piccoli di B

Operatore	Formula	Cosa rivela
Black-hat	chiusura$(f) - f$	Elementi **scuri e sottili**, più piccoli di B

4.9.8.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero L_B.
- Riga 4: Intero C_B.
- Prossime L_B righe: elementi interi (0 o 1) della matrice B.
- Prossime L righe: elementi interi della matrice f.

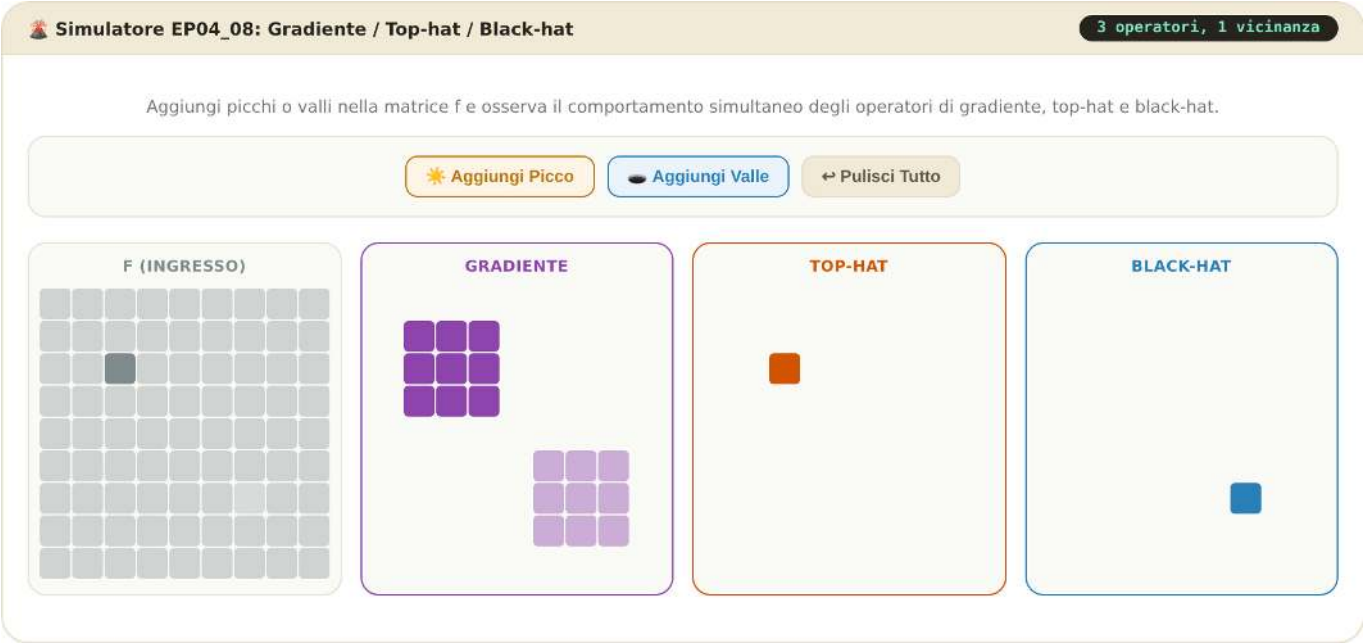
Output:

- Matrice gradiente in L righe e C colonne.
- Matrice top-hat in L righe e C colonne.
- Matrice black-hat in L righe e C colonne.

4.9.8.5 📌 Esempi

Input	Output	Osservazione
9 9 3 3 1 1 1 1 1 1 1 1 1 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 80 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 2 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10	(gradiente: alone $3 \times 3 = 70$ attorno a $(2, 2)$ e alone $3 \times 3 = 8$ attorno a $(6, 6)$, resto 0) (top-hat: unico 70 in $(2, 2)$, resto 0) (black-hat: unico 8 in $(6, 6)$, resto 0)	Picco isolato diventa top-hat; valle isolata diventa black-hat; entrambi appaiono nel gradiente

```
%%writefile EP04_08.py
# Codice Python
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0408-gradiente.html

Figura 4.37: Simulatore EP04_08: Gradiente morfologico, Top-hat e Black-hat

```
Overwriting EP04_08.py
```

```
TestSuite("EP04_08.py").run()
```

```
<IPython.core.display.HTML object>
```

4.9.9 EP04_09 Trasformata della Distanza e il "Nucleo" dell'Oggetto

Nella **robotica mobile**, quando si pianifica un percorso all'interno di un corridoio, il robot vuole sapere non solo *dove* c'è spazio libero, ma anche **quanto distante** ogni punto libero sia dalla parete più vicina. I percorsi più sicuri tendono a passare attraverso il "nucleo" del corridoio, lontano dagli ostacoli.

La **trasformata della distanza morfologica** assegna a ogni pixel un valore che rappresenta la sua distanza dal bordo più vicino, secondo la metrica definita dall'elemento strutturante. I pixel vicini al bordo ricevono valori bassi, mentre i pixel più interni ricevono valori più alti. Il pixel con il valore massimo corrisponde alla regione più protetta dell'oggetto, spesso associata al suo centro morfologico.

Vedere in Figura 4.38 una simulazione di questo EP.

4.9.9.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** leggere gli interi L (righe) e C (colonne) dell'immagine f.
2. **Dimensioni di B:** leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** leggere la matrice b, contenente valore 0 al centro e valori negativi nelle altre posizioni.
4. **Immagine:** leggere la matrice binaria f (valori 0 o 1), riga per riga.
5. **Preparazione:** moltiplicare l'immagine per $L \times C$, garantendo che i pixel interni abbiano un valore iniziale sufficientemente alto per la propagazione delle distanze.
6. **Trasformata della distanza:** calcolare la matrice delle distanze utilizzando il metodo `mm.dist1(f,b)`.
7. **Output:** visualizzare la matrice risultante dalla trasformata della distanza.

4.9.9.2 📌 Vincoli Computazionali

- Utilizzare l'implementazione dell'erosione ponderata fornita dalla libreria.

- L'elemento strutturante può contenere valori negativi arbitrari.
- La trasformata deve essere ottenuta applicando iterativamente erosioni ponderate fino a raggiungere un punto fisso.

⚠️ **Nota Cruciale sulla Lettura delle Matrici:** Poiché l'elemento strutturante può contenere valori interi negativi (ad esempio, `-1` e `-99`), **non utilizzare la funzione `mm.readImg` per leggere la matrice** b. Questa funzione converte i dati nel tipo `uint8`, causando *underflow* e corrompendo i valori negativi. Leggere le L_B righe di b manualmente utilizzando il tipo predefinito `int`. L'immagine f può continuare a essere letta normalmente con `mm.readImg`.

4.9.9.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto Visivo
dist(y, x)	Distanza morfologica fino al bordo più vicino secondo la metrica definita da b	I pixel più interni ricevono valori più alti
Valore massimo	Pixel più distante dal bordo	Approssima il centro morfologico dell'oggetto
Elemento strutturante ponderato	Definisce i costi di spostamento tra pixel vicini	Determina la metrica della distanza utilizzata
Oggetti sottili	Regioni strette dell'oggetto	Producono valori bassi di distanza

4.9.9.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: intero L.
- Riga 2: intero C.
- Riga 3: intero L_B.
- Riga 4: intero C_B.
- Prossime L_B righe: elementi interi della matrice b.
- Prossime L righe: elementi binari (0 o 1) della matrice f.

⚠️ **Nota di implementazione:** Gli elementi della matrice f (0 o 1) devono essere moltiplicati per **255** per generare un'immagine binaria adeguata (0 e 255) prima di applicare la Trasformata della Distanza (TD).

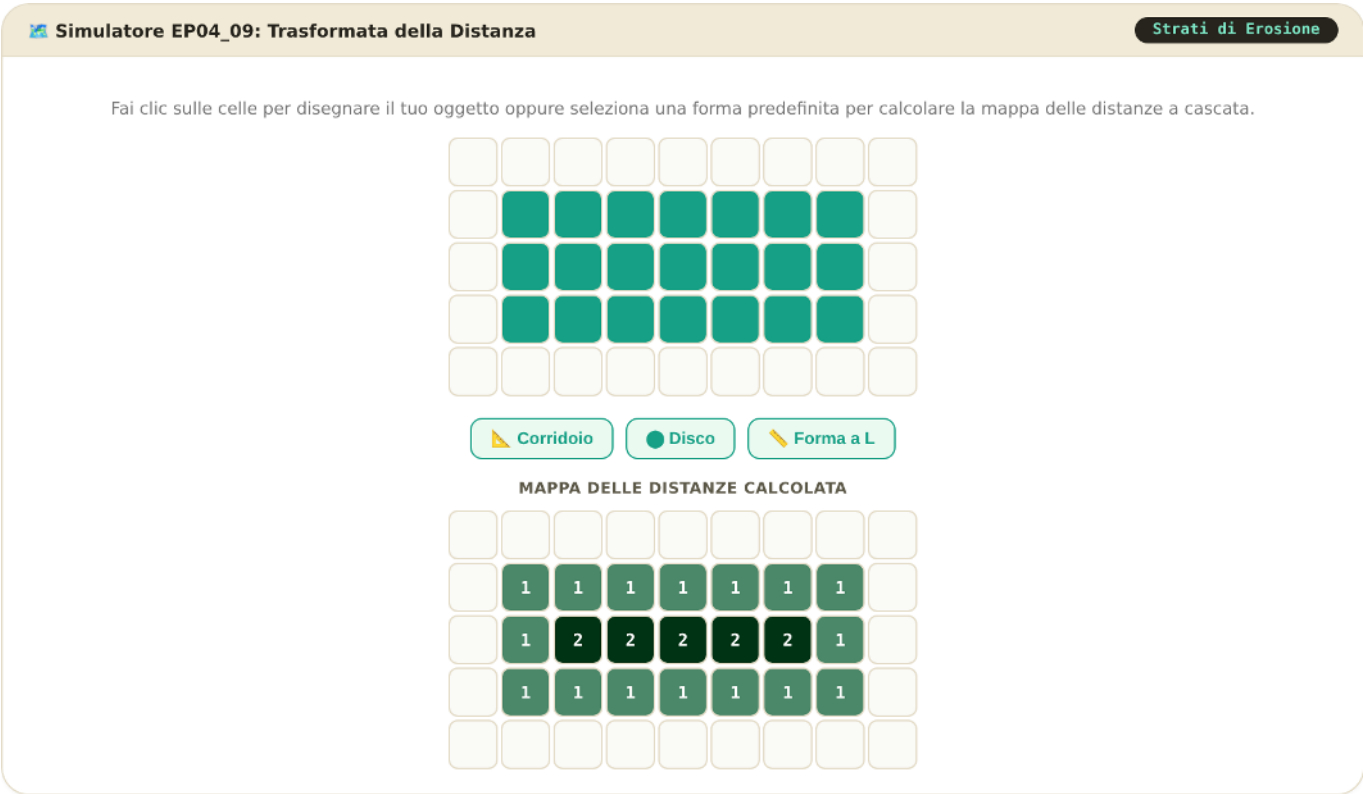
Output:

- Matrice della trasformata della distanza in L righe e C colonne.

4.9.9.5 📌 Esempio

Input	Output	Osservazione
5 9 3 3 -99 -1 -99 -1 0 -1 -99 -1 -99 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 1 2 2 2 2 2 1 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0	Risultato della trasformata della distanza.

Nota: il valore `-99` agisce come un'approssimazione pratica di $-\infty$, impedendo la propagazione attraverso le diagonali. In questo modo, solo i vicini orizzontali e verticali contribuiscono alla distanza, producendo la distanza di Manhattan.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0409-distancia.html

Figura 4.38: Simulatore EP04_09: Trasformata della Distanza (Livelli di Erosione)

```
%%writefile EP04_09.py
# Codice Python
```

```
Overwriting EP04_09.py
```

```
TestSuite("EP04_09.py").run()
```

```
<IPython.core.display.HTML object>
```

4.9.10 EP04_10 🪙 Separazione dei *Blob*, Etichettatura e Descrittori

In una **linea di produzione di monete**, è comune che i pezzi si tocchino l'un l'altro sul nastro trasportatore, formando un'unica macchia connessa nell'immagine — un conteggio ingenuo sbaglierebbe il totale. La soluzione classica combina operazioni morfologiche e analisi di connettività: prima un'**erosione** riduce o spezza le connessioni fragili tra gli oggetti, e poi l'**etichettatura delle componenti connesse** separa ciascun oggetto in una regione distinta. Infine, **descrittori geometrici** (area e bounding box) riassumono ciascuna componente trovata.

Vedi in Figura 4.39 una simulazione di questo EP.

4.9.10.1 📋 Linee Guida di Implementazione

1. **Dimensioni dell'immagine:** leggere gli interi L (righe) e C (colonne) da f.
2. **Dimensioni di B:** leggere gli interi L_B (righe) e C_B (colonne) dell'elemento strutturante.
3. **Elemento strutturante:** leggere la matrice B, contenente valori 0 o 1, riga per riga.
4. **Dati:** leggere la matrice binaria f (valori 0 o 1), riga per riga.
5. **Separazione:** calcolare
$$f_{ero} = f \ominus B$$
usando erosione binaria piatta (come nell'EP04_04), eliminando connessioni fragili tra gli oggetti.
6. **Etichettatura:** su f_{ero}, identificare le componenti connesse usando la connettività definita dall'intorno B. L'etichettatura deve seguire una scansione *raster*: quando si trova un pixel 1 non ancora etichettato, assegnare una nuova etichetta intera crescente a partire da 1 e propagare tale etichetta a tutta la regione connessa.
7. **Descrittori:** per ogni etichetta k, calcolare:
   - **Area:** numero di pixel appartenenti all'etichetta;
   - **Bounding box:**
$$(y_{min}, x_{min}, y_{max}, x_{max})$$
8. **Output:** mostrare il numero totale di etichette e, successivamente, una riga per etichetta nel formato:
$$k,\ \text{area},\ y_{min},\ x_{min},\ y_{max},\ x_{max}$$

4.9.10.2 📌 Vincoli Computazionali

- L'erosione deve essere applicata prima dell'etichettatura.
- La connettività è fissa e definita dall'intorno sopra descritto.
- L'elemento strutturante B non interferisce con la connettività dell'etichettatura.
- Nessun padding in alcuna fase.
- L'ordine delle etichette segue la prima scoperta durante la scansione *raster*.

4.9.10.3 🧠 Fondamenti Teorici

Concetto	Significato	Impatto
Ponte sottile	Connessione stretta tra oggetti	Può essere rimosso dall'erosione morfologica
Connettività	Definita dall'insieme $\mathcal{N}(y, x)$	Determina quali pixel appartengono alla stessa componente
Area	Numero di pixel per componente	Stima diretta della dimensione dell'oggetto
Bounding box	Estensione spaziale dell'etichetta	Riassunto geometrico della componente

4.9.10.4 📦 Specifica di Input e Output (VPL)

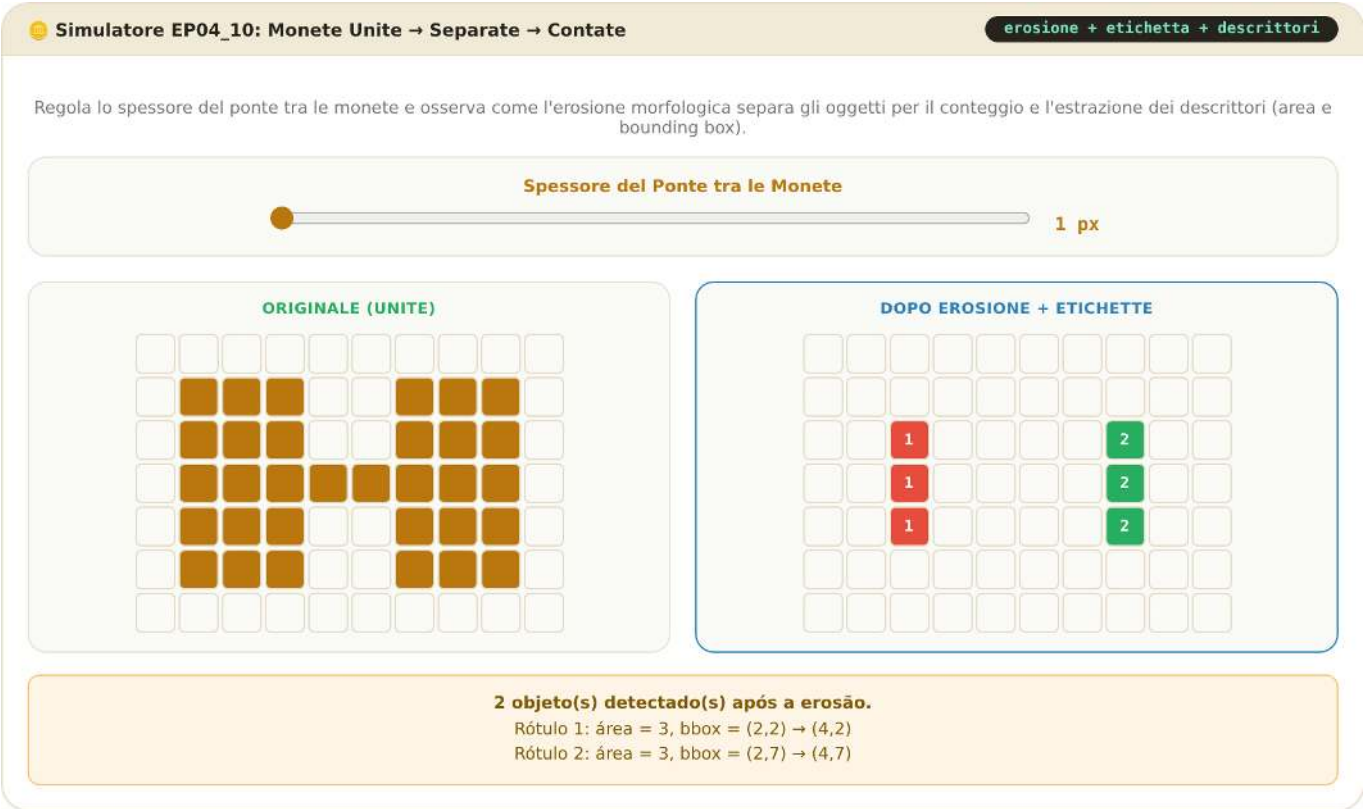
Input:

- Riga 1: intero L
- Riga 2: intero C
- Riga 3: intero L_B
- Riga 4: intero C_B
- Prossime L_B righe: matrice B
- Prossime L righe: matrice f

Output:

- Riga 1: numero totale di etichette trovate
- Righe successive:

$$k,\ \text{area},\ y_{min},\ x_{min},\ y_{max},\ x_{max}$$



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap04/sim-ep0410-rotulacao.html

Figura 4.39: Simulatore EP04_10: Separazione di Blob, Etichettatura e Descrittori

```
%%writefile EP04_10.py
# Codice Python
```

```
Overwriting EP04_10.py
```

```
TestSuite("EP04_10.py").run()
```

```
<IPython.core.display.HTML object>
```

Capitolo 5

Trasformate e Compressione

Executar Colab Abrir GitHub

Nei capitoli precedenti, tutte le operazioni sono state eseguite **nel dominio spaziale**, in cui gli algoritmi agiscono direttamente sui valori di intensità dei pixel.

In questo capitolo verrà presentato un approccio complementare: il **dominio della frequenza**, nel quale l'immagine è rappresentata dalle variazioni spaziali di intensità, e non solo dai valori individuali dei pixel.

Il concetto di **frequenza spaziale** descrive la rapidità con cui l'intensità varia lungo l'immagine. Le variazioni lente corrispondono a **basse frequenze**, mentre bordi, dettagli fini e rumori corrispondono ad **alte frequenze**.

Questa rappresentazione si basa sul fatto che qualsiasi immagine digitale discreta può essere decomposta in una combinazione di **funzioni ortogonali**. La **Trasformata di Fourier** utilizza una **base di esponenziali complesse bidimensionali** (equivalenti a sinusoidi con orientamento e frequenza specifici). Altre trasformate, come la **Trasformata del Coseno (DCT)** e la **Trasformata *Wavelet* (DWT)**, utilizzano diverse famiglie di funzioni di base — coseni bidimensionali nel caso della DCT, e funzioni con supporto compatto nel caso delle *wavelet*.

Tra le principali applicazioni di questa rappresentazione si evidenziano:

1. **Filtraggio nel dominio della frequenza**, per attenuare o enfatizzare determinate bande di frequenza;
2. **Analisi multirisoluzione tramite trasformate *wavelet***, che rappresenta strutture a diverse scale;
3. **Compressione delle immagini**, mediante la riduzione del numero di coefficienti necessari per rappresentare l'immagine.

5.1 Obiettivi

Al termine di questo capitolo, sarai in grado di:

- **Interpretare lo spettro di Fourier** di un'immagine, distinguendo magnitudine, fase e componenti di frequenza;
- **Applicare il Teorema della Convoluzione** per effettuare filtraggi nel dominio della frequenza utilizzando la Trasformata Veloce di Fourier (FFT);
- **Progettare e analizzare filtri nel dominio della frequenza**, comprendendo il funzionamento di filtri passa-basso, passa-alto e *notch*;
- **Comprendere l'analisi multirisoluzione tramite trasformate *wavelet*** e la loro applicazione nella rappresentazione gerarchica delle immagini;
- **Descrivere il processo di compressione delle immagini**, inclusa la Trasformata Discreta del Coseno (DCT) e la quantizzazione dei coefficienti;
- **Selezionare formati di archiviazione delle immagini**, come JPEG, PNG e WebP, in base ai requisiti dell'applicazione.

5.2 Configurazione dell'Ambiente

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup()
from morph import mm
import numpy as np
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0
```

5.3 Trasformata di Fourier Discreta 2D

L'analisi di Fourier si basa sul principio secondo cui qualsiasi segnale periodico può essere rappresentato come una somma di funzioni sinusoidali con differenti frequenze, ampiezze e fasi. Questo concetto si applica anche alle immagini digitali, consentendo di rappresentarle nel **dominio della frequenza** invece che nel dominio spaziale.

La Figura 5.1 illustra questa decomposizione per un segnale unidimensionale. Nel caso di un'immagine, la Trasformata Discreta di Fourier (DFT) converte la matrice delle intensità $f(x, y)$ in un insieme di coefficienti che descrive il contributo delle diverse frequenze spaziali presenti nell'immagine.

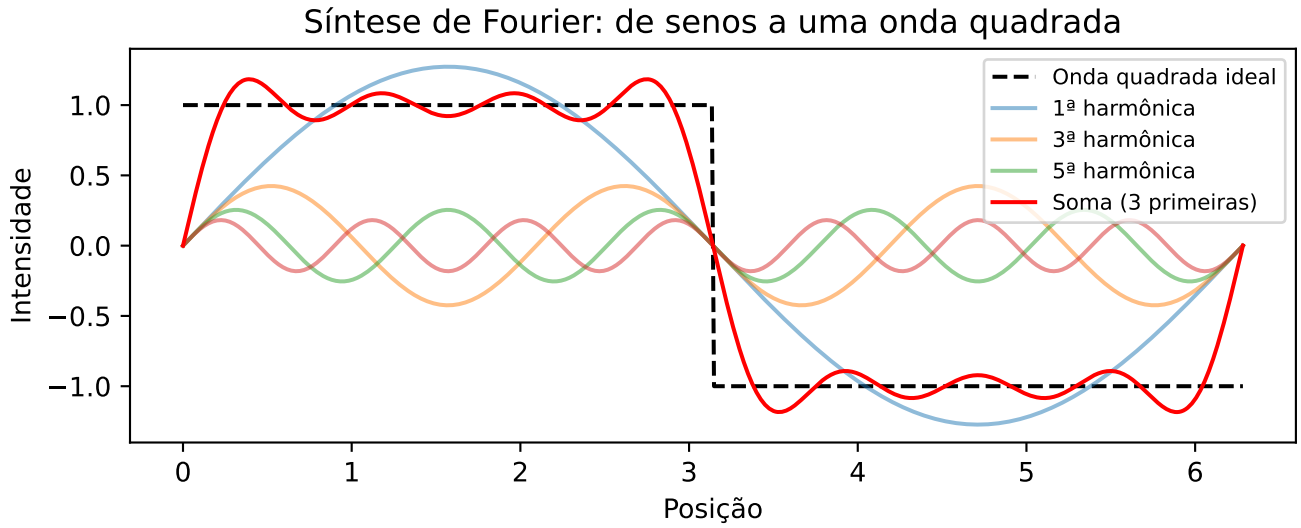


Figura 5.1: Decomposizione di Fourier 1D: un'onda quadra (linea tratteggiata) è approssimata dalla somma delle prime sinusoidi (linee colorate). Più termini ci sono, migliore è l'approssimazione.

5.3.1 Simulatore: Ricostruire Segnali con Sinusoidi

Prima di studiare le immagini bidimensionali, il simulatore della Figura 5.2 illustra il principio dell'analisi di Fourier per segnali unidimensionali: **una forma d'onda può essere approssimata dalla somma di sinusoidi con diverse frequenze e ampiezze**.

Man mano che si aggiungono nuovi termini, la somma delle sinusoidi (curva nera) si avvicina alla forma d'onda di riferimento (tratteggiata). Il grafico inferiore mostra lo spettro delle ampiezze, indicando il contributo di ciascuna frequenza alla ricostruzione del segnale.

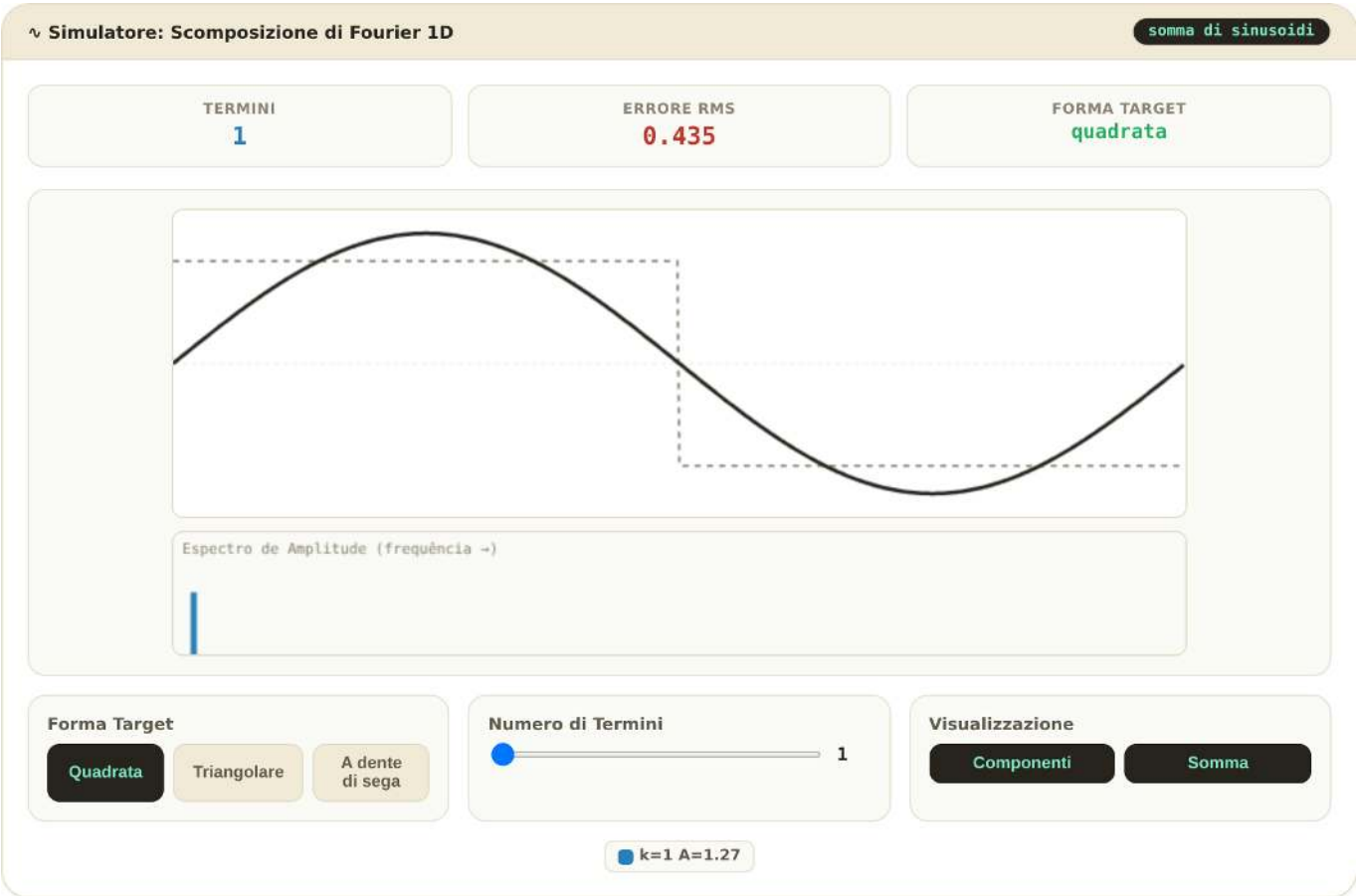
Attività

Esplora il simulatore e rispondi:

1. Quanti termini sono necessari per ottenere una buona approssimazione dell'onda quadra?
2. Quale delle tre forme d'onda converge più rapidamente? Giustifica la tua risposta.
3. Come cambia lo spettro delle ampiezze passando dall'onda quadra a quella triangolare?

Risposte

1. Quanti termini sono necessari per una buona approssimazione dell'onda quadra?
Con circa 15-20 termini, la forma dell'onda si avvicina già bene al riferimento. Tuttavia, in prossimità delle discontinuità permane una piccola oscillazione, nota come **fenomeno di Gibbs**, che non scompare nemmeno aggiungendo più termini.
2. Quale forma converge più rapidamente? Perché?
L'**onda triangolare** converge più rapidamente, poiché le ampiezze delle sue armoniche decadono più velocemente di quelle dell'onda quadra e dell'onda a dente di sega. Di conseguenza, pochi termini già producono una buona approssimazione.
3. Come cambia lo spettro tra l'onda quadra e quella triangolare?
Entrambe presentano solo **armoniche dispari**, ma nell'onda triangolare le ampiezze diminuiscono molto più rapidamente. Pertanto, poche armoniche sono sufficienti per ricostruire il segnale con buona precisione.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-05-freq.html

Figura 5.2: Simulatore interattivo della decomposizione di Fourier 1D: visualizzazione della somma di sinusoidi con diverse frequenze, ampiezze e fasi. Aggiungi termini e osserva la convergenza verso forme d'onda arbitrarie.

5.3.2 Interpretazione dello spettro di frequenza

Applicando la Trasformata Discreta di Fourier (DFT) a un'immagine e visualizzando il modulo dei suoi coefficienti (vedi Figura 5.5), si ottiene lo **spettro di ampiezza**, che mostra la distribuzione delle frequenze spaziali presenti nell'immagine.

Il coefficiente situato all'origine della DFT, denominato **componente DC** (*Direct Current*), corrisponde alla frequenza nulla e rappresenta l'intensità media dell'immagine. Per convenzione, tale coefficiente è memorizzato nell'angolo superiore sinistro dello spettro. Per facilitarne l'interpretazione, si applica l'operazione **FFT Shift**, che sposta la componente DC al centro dell'immagine. Dopo questo spostamento, le basse frequenze si concentrano nella regione centrale, mentre le alte frequenze si trovano in prossimità dei bordi, come riassunto nella Tabella 5.1.

Tabella 5.1: Corrispondenza tra le regioni dello spettro di ampiezza dopo l'applicazione dell'FFT Shift.

Regione dello spettro	Componenti predominanti	Esempi nell'immagine
Centro (basse frequenze)	Variazioni spaziali lente	Illuminazione, regioni omogenee e forme globali
Regione intermedia (medie frequenze)	Variazioni di scala intermedia	Trame e pattern ripetitivi
Bordi (alte frequenze)	Variazioni spaziali rapide	Contorni, dettagli fini e rumore

Questa organizzazione facilita l'interpretazione dello spettro e la progettazione di filtri. L'attenuazione delle basse frequenze riduce le variazioni globali di intensità, mentre l'attenuazione delle alte frequenze smussa l'immagine riducendo i dettagli fini e parte del rumore.

5.3.3 L'Esperimento della Griglia: Costruire un'Immagine da un Singolo Coefficiente

Prima di presentare la formulazione matematica della Trasformata Discreta di Fourier (DFT), è utile analizzare la sua inversa, denominata Trasformata Discreta Inversa di Fourier (IDFT). Si consideri uno spettro in cui tutti i coefficienti siano nulli, eccetto uno. Un esempio di questa costruzione è presentato nel codice della Figura 5.3 e può essere esplorato interattivamente nel simulatore della Figura 5.4..

L'immagine ricostruita è una sinusoide bidimensionale. La posizione del coefficiente nello spettro determina la sua **orientazione** e la sua **frequenza spaziale**, mentre la sua magnitudine e la sua fase definiscono, rispettivamente, la sua ampiezza e il suo spostamento spaziale. Pertanto, ciascun coefficiente della DFT rappresenta una componente sinusoidale, e l'immagine originale può essere ricostruita mediante la somma di tutte queste componenti.

```
import cv2

N_grid = 100
espectro_vazio = np.zeros((N_grid, N_grid), dtype=complex)

# Accendendo un singolo punto (frequenza) fuori dal centro
u0, v0 = 10, 5
espectro_vazio[N_grid//2 - v0, N_grid//2 - u0] = 1000

# Tornando al dominio spaziale (IDFT)
onda_2d = np.real(np.fft.ifft2(np.fft.ifftshift(espectro_vazio)))

onda_vis = cv2.normalize(onda_2d, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
espectro_vis = cv2.normalize(
    np.abs(espectro_vazio), None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

# Evidenziazione visiva del punto
espectro_color = cv2.cvtColor(espectro_vis, cv2.COLOR_GRAY2BGR)
```

```
cv2.circle(espectro_color, (N_grid//2 - u0, N_grid//2 - v0), 2, (0, 0, 255), -1)

mm.show([espectro_color, onda_vis],
        titles=["Spettro (1 punto attivo)", "Onda 2D risultante (IDFT)"],
        cols=2, figsize=(10, 4))
```

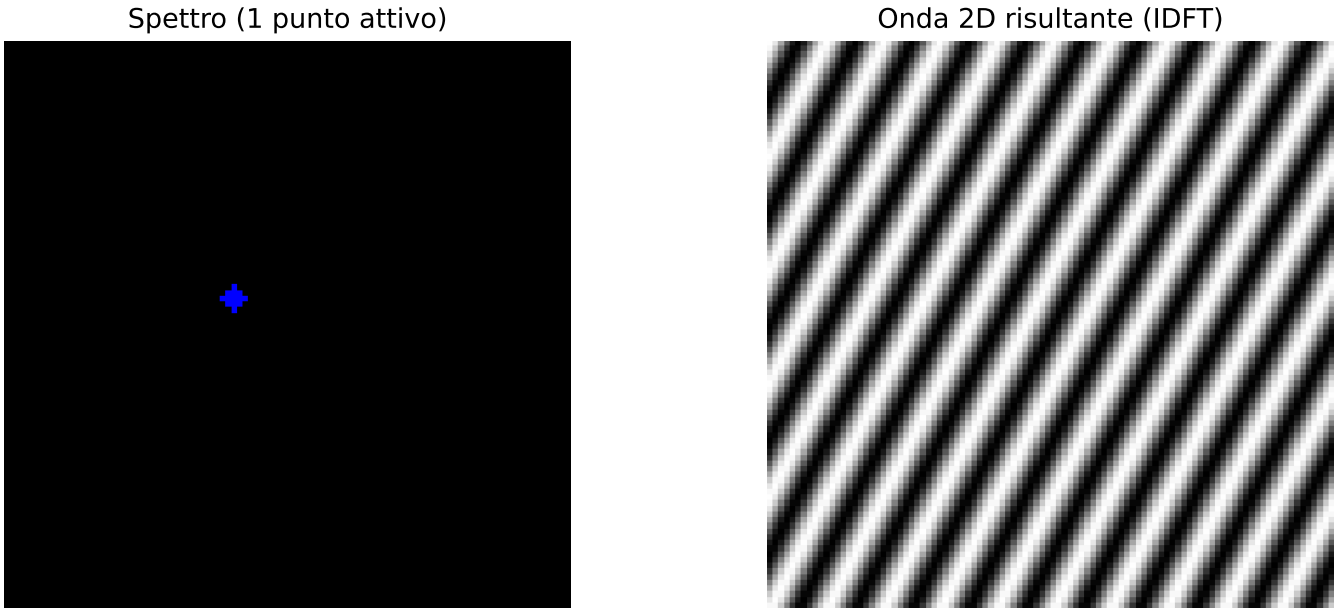
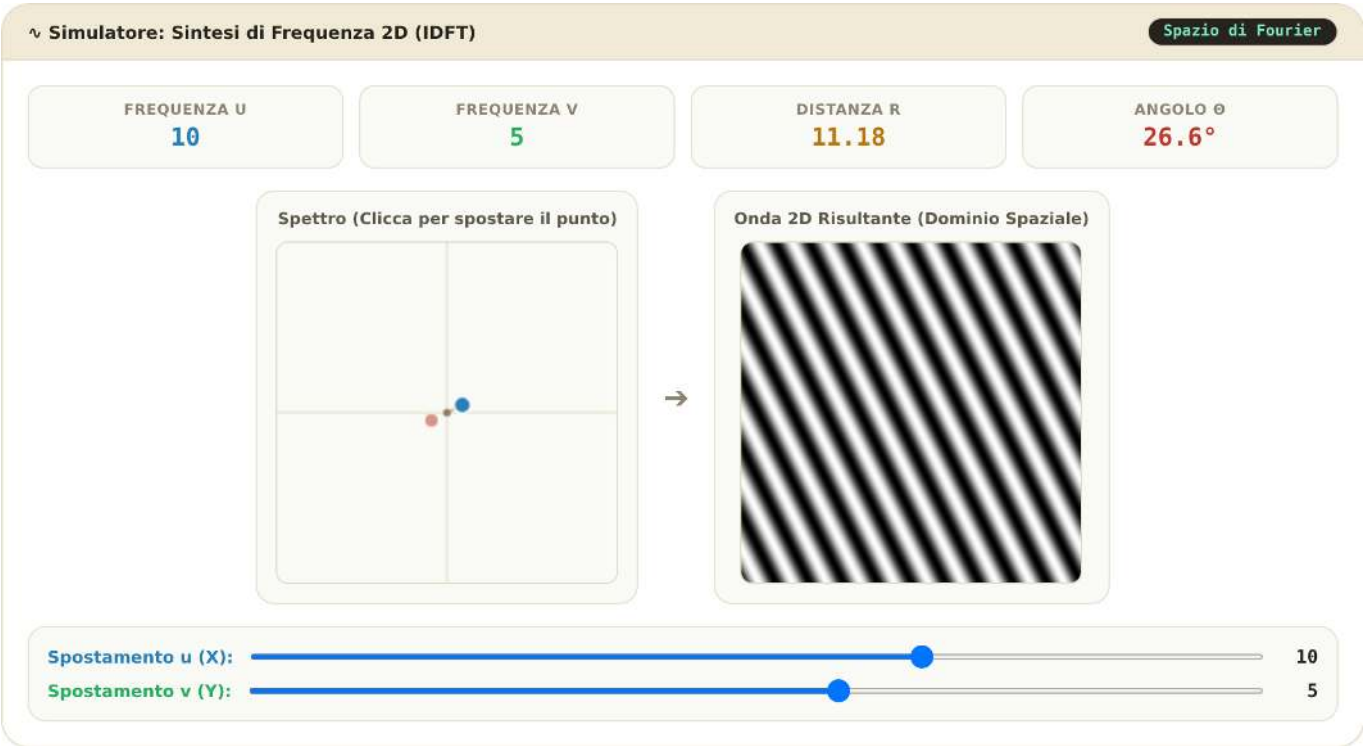


Figura 5.3: Ogni frequenza nello spettro (punto isolato) corrisponde a un'onda sinusoidale 2D ruotata nel dominio spaziale.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-05-grade-2d.html

Figura 5.4: Simulatore interattivo della sintesi di Fourier 2D. Modifica la posizione orizzontale (u) e verticale (v) del coefficiente nello spettro di frequenze centrato e osserva come la distanza dal centro determina la frequenza spaziale (spessore) e l'angolo determina l'orientamento dell'onda sinusoidale generata.

5.3.4 Definizione Matematica

Si consideri un'immagine $f(x, y)$ con dimensioni $M \times N$. La sua **Trasformata Discreta di Fourier 2D** (DFT) è definita da:

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{5.1}$$

dove $u = 0, 1, \dots, M-1$ e $v = 0, 1, \dots, N-1$ rappresentano le frequenze discrete nelle direzioni orizzontale e verticale, rispettivamente. Il termine esponenziale corrisponde a una sinusoide bidimensionale, la cui frequenza e orientazione sono determinate dagli indici (u, v).

La **Trasformata Discreta Inversa di Fourier 2D** (IDFT) ricostruisce l'immagine originale a partire dai suoi coefficienti:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v)\, e^{j2\pi(\frac{ux}{M}+\frac{vy}{N})} \tag{5.2}$$

Le Equazioni Equazione 5.1 e Equazione 5.2 mostrano che la DFT e la IDFT formano una coppia di trasformazioni: la prima converte l'immagine nel dominio della frequenza, mentre la seconda ricostruisce esattamente l'immagine originale a partire dai suoi coefficienti.

Sul simbolo j

Il termine j denota l'**unità immaginaria**, definita da $j^2 = -1$. In ingegneria e nell'elaborazione dei segnali, si adotta j invece di i per evitare conflitti con la notazione della corrente elettrica. Il suo utilizzo nell'esponenziale complessa, governato dalla formula di Eulero ($e^{j\theta} = \cos\theta + j\sin\theta$), consente di rappresentare in modo compatto l'ampiezza e la fase di ciascuna frequenza spaziale presente nell'immagine.

Che cos'è il componente DC?

Il coefficiente $F(0, 0)$, denominato **componente DC** (*Direct Current*), è uguale alla somma delle intensità di tutti i pixel dell'immagine (vedi Figura 5.5):

$$F(0, 0) = MN\,\bar{f},$$

dove $\bar{f}$ è l'intensità media dell'immagine. Per questo motivo, il componente DC rappresenta il livello medio di intensità e, nella maggior parte delle immagini naturali, possiede la maggiore ampiezza dello spettro.
Gli altri coefficienti rappresentano variazioni attorno a tale media. Dopo l'applicazione del **FFT Shift**, il componente DC viene spostato al centro dello spettro, concentrando le basse frequenze nella regione centrale e le alte frequenze ai bordi.

5.3.5 Magnitudine e Fase

Ogni coefficiente della Trasformata Discreta di Fourier (DFT) è un numero complesso e può essere scritto come

$$F(u, v) = R(u, v) + j\,I(u, v),$$

dove $R(u, v)$ e $I(u, v)$ corrispondono, rispettivamente, alle parti **reale** e **immaginaria** del coefficiente. Dall'Equazione Equazione 5.1 si ottiene

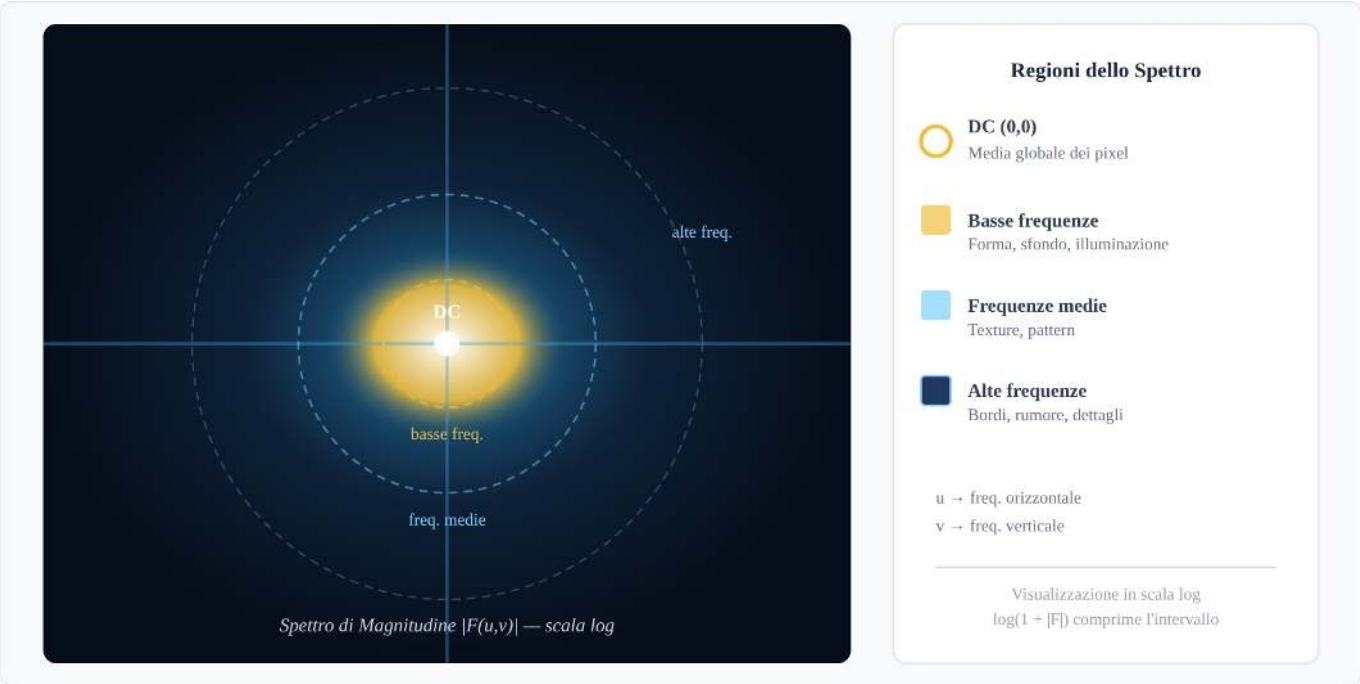


Figura 5.5: Diagramma concettuale dello spettro di Fourier 2D centrato.

$$R(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cos\Big(2\pi \left(\frac{ux}{M} + \frac{vy}{N}\right)\Big),$$

e

$$I(u,v) = -\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \sin\Big(2\pi \left(\frac{ux}{M} + \frac{vy}{N}\right)\Big).$$

Da questa rappresentazione si definiscono due grandezze fondamentali:

- **Magnitudine**, che indica l'intensità della componente di frequenza,

$$|F(u,v)| = \sqrt{R(u,v)^2 + I(u,v)^2};$$

- **Fase**, che determina l'allineamento (o lo spostamento) spaziale della componente,

$$\phi(u,v) = \operatorname{atan2}(I(u,v),\, R(u,v)).$$

Pertanto, ogni coefficiente può anche essere scritto nella sua forma polare,

$$F(u,v) = |F(u,v)|\, e^{j\phi(u,v)}.$$

Lo spettro di Fourier può quindi essere visualizzato mediante due immagini distinte: lo **spettro di magnitudine**, generalmente utilizzato per analizzare la distribuzione delle frequenze, e lo **spettro di fase**, che descrive l'organizzazione spaziale delle componenti sinusoidali.

Sebbene lo spettro di magnitudine sia il più utilizzato per l'ispezione visiva, la fase contiene gran parte delle informazioni strutturali dell'immagine. La combinazione di magnitudine e fase consente di ricostruire esattamente l'immagine originale tramite la IDFT.

5.3.6 Cosa trasportano l'ampiezza e la fase?

Una dimostrazione classica consiste nel combinare l'ampiezza di un'immagine con la fase di un'altra e ricostruire il risultato. Questo esperimento evidenzia che:

- **La fase** preserva la struttura spaziale dell'immagine, inclusa la posizione degli oggetti, i loro contorni e la loro geometria. Piccole alterazioni nella fase possono provocare grandi cambiamenti visivi.
- **L'ampiezza** controlla come l'energia è distribuita tra le frequenze spaziali, influenzando principalmente il contrasto e la tessitura.

Quando un'immagine viene ricostruita con l'ampiezza di A e la fase di B, il risultato tende a **somigliare più a B che ad A**, evidenziando che la fase è il componente principale responsabile dell'organizzazione spaziale della scena. Tuttavia, l'ampiezza rimane importante, poiché modula il contrasto delle strutture ricostruite. Pertanto, una ricostruzione fedele dipende dalla combinazione coerente tra ampiezza e fase.

Un esempio di questo comportamento è presentato in Figura 5.6.

Analogia con l'Audio: Limitazioni e Precauzioni

La fase di un segnale svolge ruoli distinti nell'audio e nelle immagini:

- **Audio stereo o multicanale:** la fase relativa tra i canali è fondamentale per la percezione della posizione delle sorgenti sonore, tramite le differenze interaurali di tempo (ITD, *Interaural Time Differences*).
- **Audio monaurale:** la fase assoluta esercita una scarsa influenza percettiva diretta.
- **Immagini (DFT):** la fase è il fattore principale responsabile dell'organizzazione spaziale della scena, mentre l'ampiezza modula il contrasto e la distribuzione dell'energia tra le frequenze.

In entrambi i domini, l'**ampiezza** è legata all'intensità delle componenti di frequenza: nell'audio, influenza il timbro e l'intensità percepita; nelle immagini, influenza il contrasto e la tessitura.

```
#   Esperimento: L'Importanza della Fase
#   Caricamento dell'immagine
url     = "https://upload.wikimedia.org/wikipedia/commons/2/25/GAZI.MD.AHAD_11.jpg"
caminho = "imagens/coins.jpg"

if not os.path.exists(caminho):
    os.makedirs("imagens", exist_ok=True)
    img_obj = mm.read(url, pil=True)
    mm.write(img_obj, caminho)
else:
    img_obj = mm.read(caminho, pil=True)

img_color = np.array(img_obj)
img_gray  = mm.gray(img_color)

img_a = cv2.resize(img_gray, (400, 400))

# Creare un'immagine B sintetica (motivo geometrico)
img_b = np.zeros((400, 400), dtype=np.uint8)
cv2.rectangle(img_b, (100, 100), (300, 300), 255, -1)
cv2.circle(img_b, (200, 200), 150, 128, 10)

FA = np.fft.fft2(img_a)
FB = np.fft.fft2(img_b)

# Scambio di Fase
rec_A_mag_B_fase = np.real(np.fft.ifft2(np.abs(FA) * np.exp(1j * np.angle(FB))))
rec_B_mag_A_fase = np.real(np.fft.ifft2(np.abs(FB) * np.exp(1j * np.angle(FA))))

mm.show(
```

```
    [img_a, img_b, rec_A_mag_B_fase, rec_B_mag_A_fase],
    titles=["Immagine A", "Immagine B", "Mag(A) + Fase(B)", "Mag(B) + Fase(A)"],
    cols=4, figsize=(16, 4)
)

print("  La fase preserva bordi e contorni; la magnitudine controlla contrasto e")
print("texture. Nell'audio stereo, la fase influisce sulla localizzazione spaziale; nelle")
print("immagini, determina l'organizzazione della scena.")
```

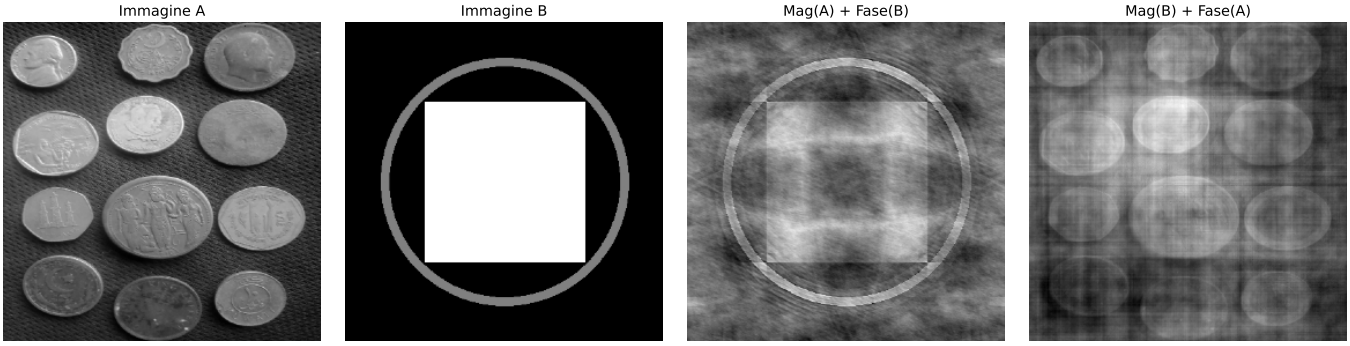


Figura 5.6: Esperimento di cambio di fase: Immagine A (monete) e Immagine B (motivo geometrico) ricostruite con magnitudini e fasi scambiate. Il risultato mostra che la struttura visiva è **molto più sensibile alla fase** che alla magnitudine: quando la fase di B viene mantenuta, l'immagine risultante preserva l'organizzazione spaziale di B, anche con la magnitudine di A. La magnitudine, a sua volta, influenza principalmente il contrasto e la texture. Si noti che la qualità della ricostruzione non è perfetta — ci sono artefatti visibili —, evidenziando l'interdipendenza tra fase e magnitudine per una rappresentazione fedele dell'immagine.

```
  La fase preserva bordi e contorni; la magnitudine controlla contrasto e
texture. Nell'audio stereo, la fase influisce sulla localizzazione spaziale; nelle
immagini, determina l'organizzazione della scena.
```

5.4 Teorema della Convoluzione e Strategie di Filtraggio

Il **Teorema della Convoluzione** stabilisce una relazione fondamentale tra il dominio spaziale e quello delle frequenze:

$$f(x,y) \circledast h(x,y) \overset{\mathcal{F}}{\longleftrightarrow} F(u,v)\,H(u,v) \tag{5.3}$$

dove $\circledast$ rappresenta la **convoluzione circolare discreta**. Pertanto, la convoluzione tra un'immagine $f(x,y)$ e un filtro $h(x,y)$ può essere sostituita dalla moltiplicazione dei loro spettri.

In pratica, per ottenere lo stesso risultato della convoluzione lineare eseguita nel dominio spaziale, si applica il **zero-padding** prima della Trasformata Rapida di Fourier (FFT), evitando artefatti ai bordi dell'immagine.

Tuttavia, la filtrazione nel dominio delle frequenze non è sempre l'alternativa più efficiente. Per filtri come quello gaussiano e il filtro della media (*Box Filter*), la proprietà di **separabilità** consente di ridurre significativamente il costo computazionale della convoluzione nel dominio spaziale.

5.4.1 *Kernel* Separabile vs. Non Separabile

Un ***kernel* separabile** può essere scritto come il prodotto esterno di due vettori unidimensionali,

$$H = v\,h^T,$$

consentendo di sostituire la convoluzione bidimensionale con due convoluzioni unidimensionali consecutive: una nella direzione orizzontale e una in quella verticale.

Un ***kernel* non separabile**, invece, non ammette tale decomposizione e, pertanto, la sua convoluzione deve essere eseguita direttamente sull'intorno bidimensionale.

In pratica, per un *kernel* di dimensione $K \times K$, la convoluzione diretta richiede K^2 moltiplicazioni per pixel, mentre un *kernel* separabile ne richiede solo $2K$, riducendo significativamente il costo computazionale.

5.4.2 Analisi dell'Efficienza Computazionale

Si consideri un'immagine di dimensioni $M \times N$ e un filtro quadrato di dimensione $K \times K$. La Tabella 5.2 confronta la complessità delle principali strategie di filtraggio.

Tabella 5.2: Confronto della complessità della convoluzione diretta, separabile e tramite Trasformata Rapida di Fourier (FFT).

Metodo di filtraggio	Complessità asintotica	Dipendenza da K	Applicazione tipica
Spatiale non separabile	$\mathcal{O}(MNK^2)$	Quadratica	*Kernel* piccoli e non separabili
Spatiale separabile	$\mathcal{O}(MNK)$	Lineare	Filtri Gaussiano e della media
Tramite FFT	$\mathcal{O}(MN\log(MN))$	Indipendente da K	*Kernel* grandi

Per *kernel* piccoli, la convoluzione spaziale, specialmente quando il filtro è separabile, risulta solitamente più efficiente grazie al basso costo delle operazioni. All'aumentare della dimensione del *kernel*, il filtraggio tramite FFT diventa più vantaggioso, poiché il suo costo è praticamente indipendente dalla dimensione del filtro.

5.4.3 Discussione dei risultati sperimentali

Il grafico ottenuto nella prova con l'immagine delle monete (2560×1920), presentato nella Figura 5.7, conferma il comportamento previsto dall'analisi della complessità computazionale.

1. **Convoluzione non separabile ($\mathcal{O}(MNK^2)$)**
   La convoluzione diretta presenta una crescita quadratica con la dimensione del *kernel*. Per valori piccoli di K, il costo è basso, ma aumenta rapidamente man mano che il *kernel* cresce, diventando impraticabile per applicazioni in tempo reale.

2. **Filtraggio tramite FFT ($\mathcal{O}(MN\log(MN))$)**
   Il costo della FFT dipende solo dalla dimensione dell'immagine, essendo indipendente da K. Pertanto, le sue prestazioni rimangono approssimativamente costanti al variare del *kernel*, rendendola vantaggiosa per filtri grandi o non separabili.

3. **Convoluzione separabile ($\mathcal{O}(MNK)$)**
   La scomposizione del *kernel* in due filtri monodimensionali riduce significativamente il costo computazionale. In pratica, questo approccio tende ad essere il più efficiente per filtri separabili, specialmente in implementazioni ottimizzate.

In generale, la scelta del metodo dipende dalla dimensione e dalla struttura del *kernel*. I filtri separabili sono più efficienti nel dominio spaziale, mentre la FFT diventa più vantaggiosa per *kernel* grandi o per convoluzioni multiple nel dominio della frequenza.

$$g = \mathcal{F}^{-1}[\mathcal{F}(f) \cdot \mathcal{F}(h)] \quad \text{(FFT)} \qquad g = f \circledast h \quad \text{(convoluzione diretta)} \qquad g = (f \circledast v) \circledast h^T \quad \text{(separabile)} \tag{5.4}$$

dove:

- $f(x, y)$ rappresenta l'immagine di ingresso;
- $h(x, y)$ è il *kernel* bidimensionale del filtro;
- v e h^T sono, rispettivamente, i vettori verticale e orizzontale che compongono il *kernel* separabile.

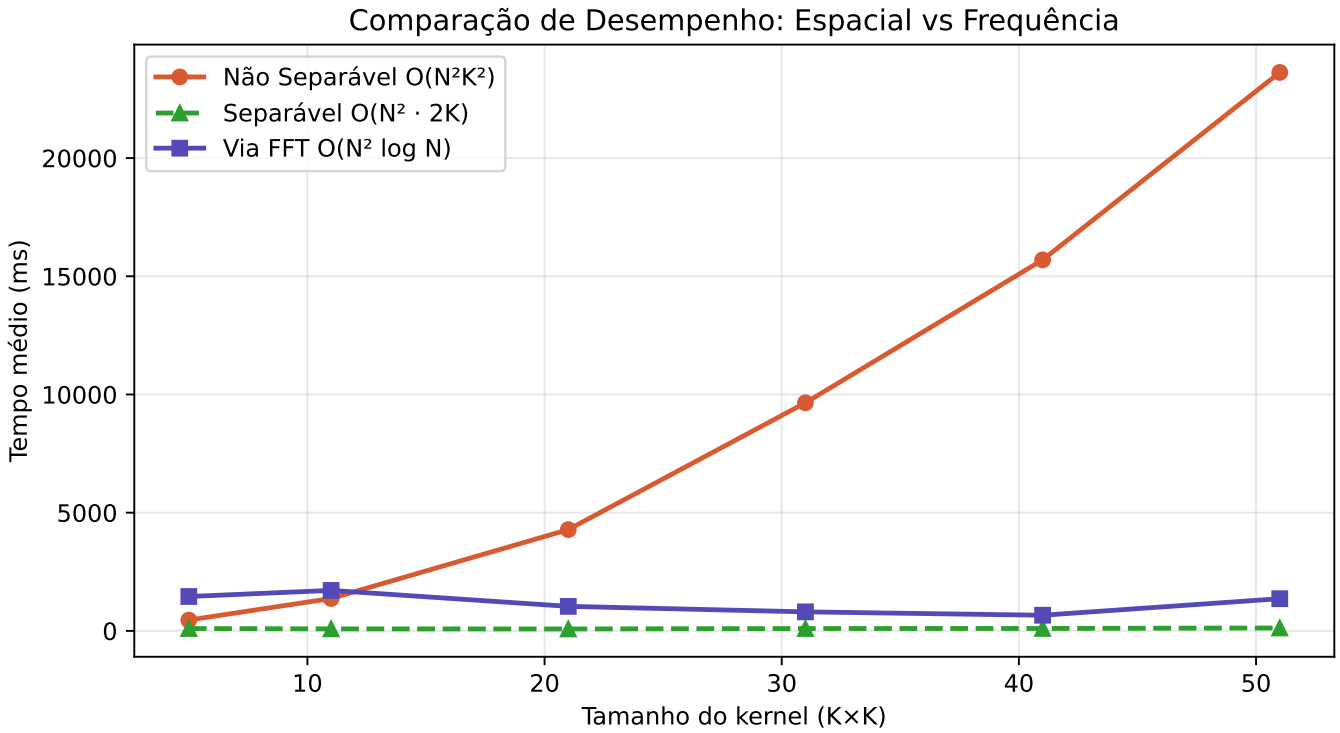


Figura 5.7: Confronto di efficienza: Convoluzione Non Separabile (Spaziale 2D), Separabile (Spaziale 1D) e tramite FFT.

! Il problema della convoluzione circolare (*wrap-around*)

La Trasformata Discreta di Fourier (DFT) assume che l'immagine sia **estesa periodicamente nello spazio**, cioè che i suoi bordi si ripetano indefinitamente.
In questa condizione, la moltiplicazione nel dominio della frequenza corrisponde a una **convoluzione circolare** nel dominio spaziale. Di conseguenza, regioni opposte dell'immagine (alto e basso, sinistra e destra) iniziano a interagire artificialmente, come illustrato nella Figura 5.8..
L'applicazione del *zero-padding* prima della FFT riduce questo effetto estendendo l'immagine con valori nulli ai bordi, avvicinando il risultato alla convoluzione lineare. Questo comportamento può essere interpretato alla luce del Teorema della Convoluzione, presentato nella Figura 5.9..

```
#   Definire M e N
M, N = img_gray.shape  # riga aggiunta

# Simulazione di un filtro di dislocamento brutale
H_shift = np.zeros_like(img_gray, dtype=complex)
for u in range(M):
    for v in range(N):
        H_shift[u, v] = np.exp(-1j * 2 * np.pi * (u*120/M + v*120/N))

# Filtraggio SENZA padding (causa il wrap-around)
F_img = np.fft.fft2(img_gray)
img_vazada = np.real(np.fft.ifft2(F_img * H_shift))

img_vazada_vis = cv2.normalize(img_vazada, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
mm.show([img_gray, img_vazada_vis],
```

```
        titles=["Originale", "Filtraggio s/ Padding (Perdita)"], cols=2, figsize=(10, 4))
```

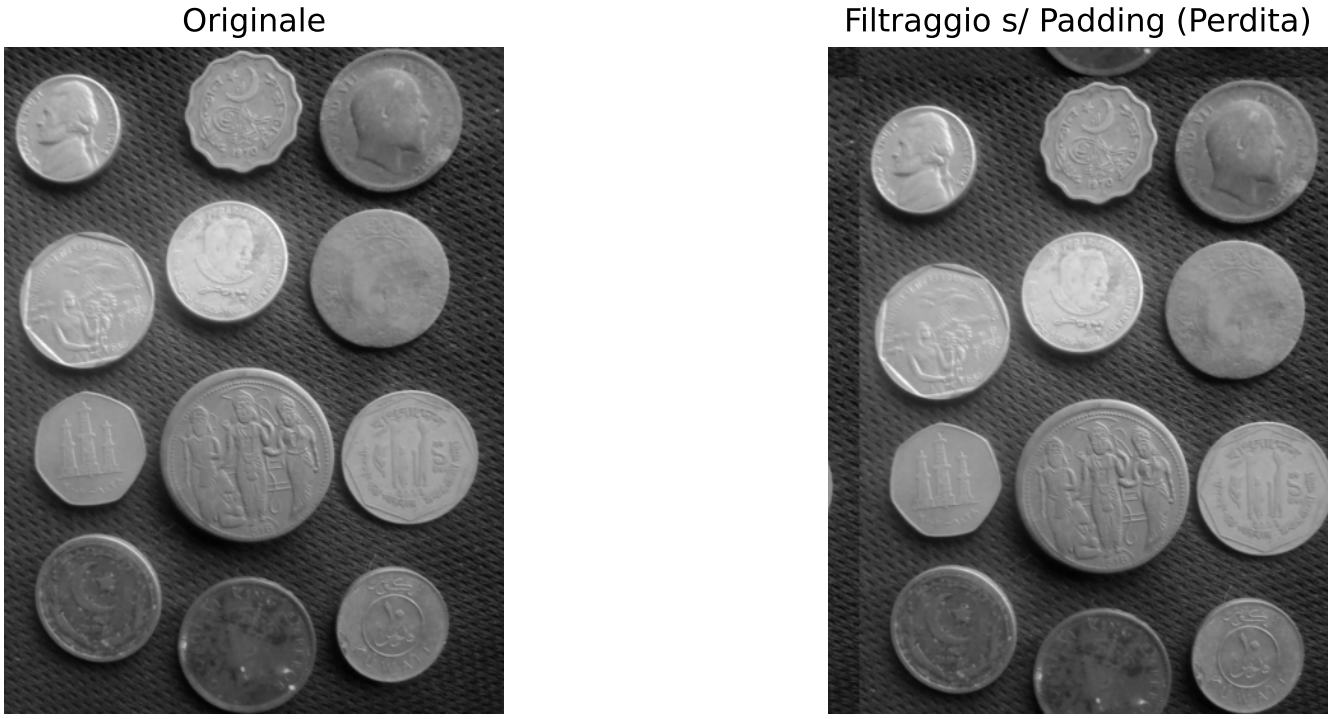


Figura 5.8: Senza *padding*, un dislocamento severo fa sì che l'immagine fuoriesca sul lato opposto (convoluzione circolare).

```
#   Kernel Gaussiano 11×11
sigma  = 3.0
K      = 11
ks     = np.arange(K) - K // 2
gauss1d = np.exp(-ks**2 / (2 * sigma**2))
gauss1d /= gauss1d.sum()
kernel  = np.outer(gauss1d, gauss1d)    # kernel 2D separabile

#   Metodo 1: Convoluzione spaziale diretta
f_float  = img_gray.astype(np.float64)
conv_esp = cv2.filter2D(f_float, -1, kernel, borderType=cv2.BORDER_CONSTANT)

#   Metodo 2: Moltiplicazione in frequenza (via FFT)
M, N     = f_float.shape
# Padding per convoluzione lineare (evita aliasing circolare)
Mpad     = 2 ** int(np.ceil(np.log2(M + K - 1)))
Npad     = 2 ** int(np.ceil(np.log2(N + K - 1)))

# Posiziona il kernel con l'origine in (0,0) e padding con zeri
kernel_pad         = np.zeros((Mpad, Npad))
kh, kw             = kernel.shape
kernel_pad[:kh, :kw] = kernel

F_img   = np.fft.fft2(f_float,  (Mpad, Npad))
F_kern  = np.fft.fft2(kernel_pad)
conv_freq = np.real(np.fft.ifft2(F_img * F_kern))

# Ritaglio per compensare lo spostamento introdotto dal posizionamento del kernel
offset   = K // 2
conv_freq_crop = conv_freq[offset:offset+M, offset:offset+N]
```

```
#   Verifica numerica
diff = np.abs(conv_esp - conv_freq_crop)
print(f"Differenza massima  (|conv_spaz - conv_freq|): {diff.max():.2e}")
print(f"Differenza media   (|conv_spaz - conv_freq|): {diff.mean():.2e}")
print(f"→ Teorema della Convoluzione verificato numericamente.")

conv_esp_vis  = cv2.normalize(conv_esp,       None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
conv_freq_vis = cv2.normalize(conv_freq_crop, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
diff_vis      = cv2.normalize(diff,           None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

mm.show(
    [img_gray, conv_esp_vis, conv_freq_vis, diff_vis],
    titles=[
        "Originale",
        "Convoluzione spaziale",
        "Moltiplicazione in frequenza",
        f"Differenza (máx={diff.max():.1e})"
    ],
    cols=4, figsize=(16, 5)
)
```

```
Differenza massima  (|conv_spaz - conv_freq|): 2.56e-13
Differenza media   (|conv_spaz - conv_freq|): 2.76e-14
→ Teorema della Convoluzione verificato numericamente.
```

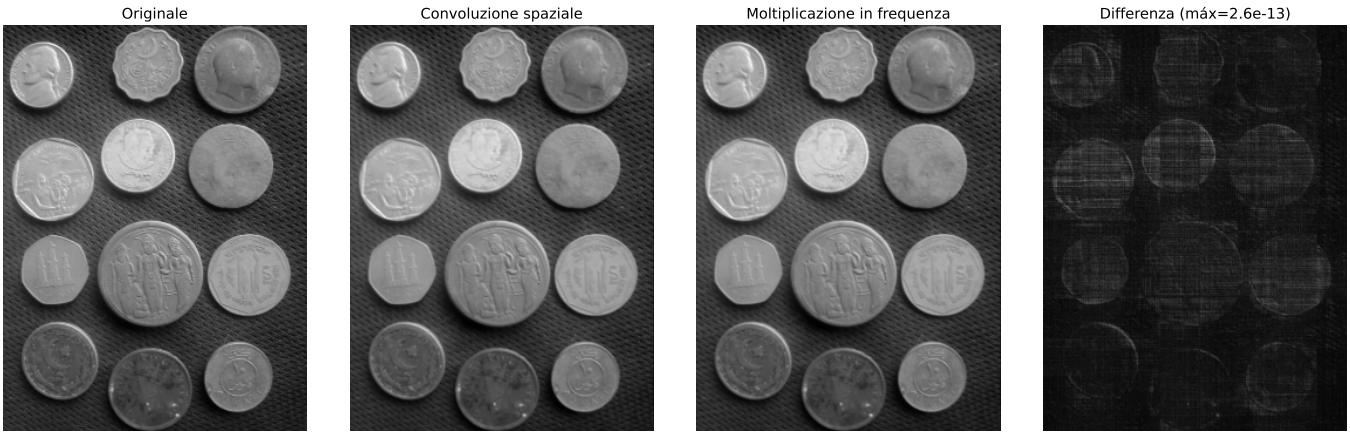


Figura 5.9: Verifica del Teorema della Convoluzione: la differenza pixel per pixel tra la convoluzione spaziale (cv2.filter2D) e la moltiplicazione in frequenza (FFT) è numericamente nulla — confermando l'equivalenza teorica.

Sulla differenza numerica

La differenza residua dell'ordine di 10^{-13} non viola il Teorema della Convoluzione, ma riflette **limitazioni computazionali** inerenti all'aritmetica a virgola mobile (doppia precisione, $\sim 10^{-16}$) e all'**ordine delle operazioni** tra i due metodi:

- **Convoluzione spaziale:** somma ponderata di vicini con arrotondamenti successivi.
- **Convoluzione in frequenza:** coinvolge tre trasformate FFT e una moltiplicazione complessa, soggetta a errori di troncamento e quantizzazione.

Pertanto, l'uguaglianza teorica è esatta, ma l'implementazione numerica produce una differenza praticamente nulla (errore relativo $< 10^{-12}$), confermando il teorema entro la precisione della macchina.

5.5 Filtri nel Dominio delle Frequenze

Un filtro nel dominio delle frequenze può essere interpretato come una **funzione di trasferimento applicata allo spettro dell'immagine**. In questa rappresentazione, ogni coefficiente di frequenza viene moltiplicato per un valore compreso tra 0 e 1, che determina la sua attenuazione o preservazione. La forma di questa funzione definisce l'effetto visivo del filtro.

Taglio netto e *ringing*. I filtri ideali con transizione istantanea a una frequenza di taglio D_0 producono discontinuità nel dominio delle frequenze. Questa discontinuità si riflette nel dominio spaziale come oscillazioni in prossimità dei bordi, note come *ringing*. Questo effetto è associato alla convoluzione con funzioni di supporto infinito nello spazio, come la funzione *sinc*, come illustrato nella Figura 5.10.

Filtri con transizione graduale. Alternative come i filtri Gaussiano e Butterworth attenuano la transizione tra regioni di passaggio e di reiezione, riducendo il *ringing*. Di contro, tale attenuazione implica un confine di separazione meno definito tra frequenze preservate e attenuate.

```
# Simulando il Filtro Ideale in Frequenza (Cilindro) e la sua rappresentazione Spaziale (Sinc)
N_grid = 2**7
u = np.arange(-N_grid//2, N_grid//2)
U, V = np.meshgrid(u, u)
D = np.sqrt(U**2 + V**2)

# Frequenza: Cilindro Ideale (1 al centro, 0 fuori dal raggio 20)
H_freq = np.zeros((N_grid, N_grid))
H_freq[D <= 20] = 1

# Spazio: L'inversa risulta nella famigerata Sinc 2D
h_space = np.fft.fftshift(np.real(np.fft.ifft2(np.fft.ifftshift(H_freq))))

fig, ax = plt.subplots(1, 2, subplot_kw={'projection': '3d'}, figsize=(12, 4))
ax[0].plot_surface(U, V, H_freq, cmap='viridis', edgecolor='none')
ax[0].set_title("Frequência: Filtro Ideal (Cilindro)")
ax[0].set_zlim(0, 1.2)

ax[1].plot_surface(U, V, h_space, cmap='plasma', edgecolor='none')
ax[1].set_title("Espaço Real: Ondulações da Sinc (Causa do Ringing)")
plt.tight_layout(); plt.show()
```

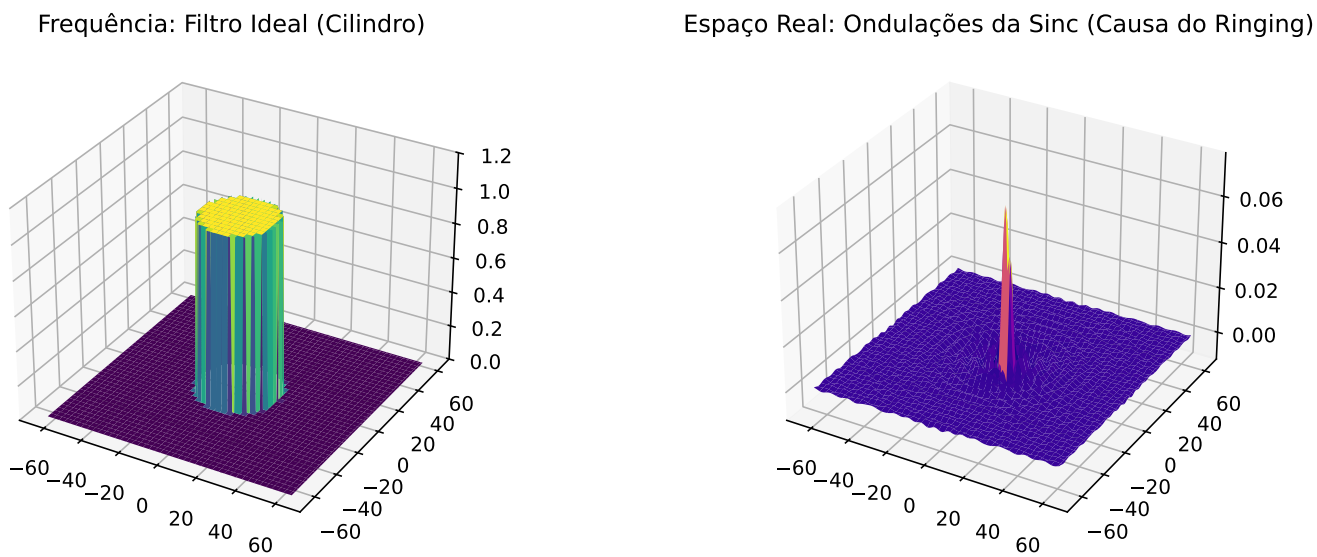


Figura 5.10: La Dualità Pericolosa: Il taglio netto in Frequenza (Cilindro) si trasforma obbligatoriamente in una Sinc spaziale. Le sue ondulazioni causano il *ringing* fantasma sui bordi dell'immagine.

5.5.1 Filtri Passa-Basso

I filtri passa-basso attenuano le componenti ad alta frequenza, producendo un appiattimento dell'immagine e una riduzione del rumore. Dopo la centralizzazione dello spettro (FFT Shift), la distanza di ciascun punto dal centro è data da:

$$D(u,v) = \sqrt{\left(u - \frac{M}{2}\right)^2 + \left(v - \frac{N}{2}\right)^2} \tag{5.5}$$

Filtro Ideale (LPFI):

$$H_{\text{ideal}}(u,v) = \begin{cases} 1, & D(u,v) \leq D_0 \\ 0, & D(u,v) > D_0 \end{cases} \tag{5.6}$$

Il taglio netto a D_0 introduce discontinuità nel dominio delle frequenze, causando oscillazioni nel dominio spaziale note come *ringing*. Questo effetto è associato alla convoluzione con funzioni a supporto infinito.

Filtro Gaussiano (LPFG):

$$H_{\text{gauss}}(u,v) = e^{-D^2(u,v)/(2\sigma^2)} \tag{5.7}$$

La regolarità della funzione gaussiana nel dominio delle frequenze evita discontinuità, eliminando il *ringing* e producendo una transizione graduale tra frequenze preservate e attenuate.

Filtro di Butterworth (LPFB) di ordine n:

$$H_{\text{BW}}(u,v) = \frac{1}{1 + [D(u,v)/D_0]^{2n}} \tag{5.8}$$

Il parametro n controlla la gradualità della transizione tra l'attenuazione e il passaggio delle frequenze. Valori piccoli producono transizioni morbide, mentre valori grandi avvicinano il comportamento al filtro ideale, con un maggiore rischio di *ringing*. Un esempio comparativo è mostrato in Figura 5.11.

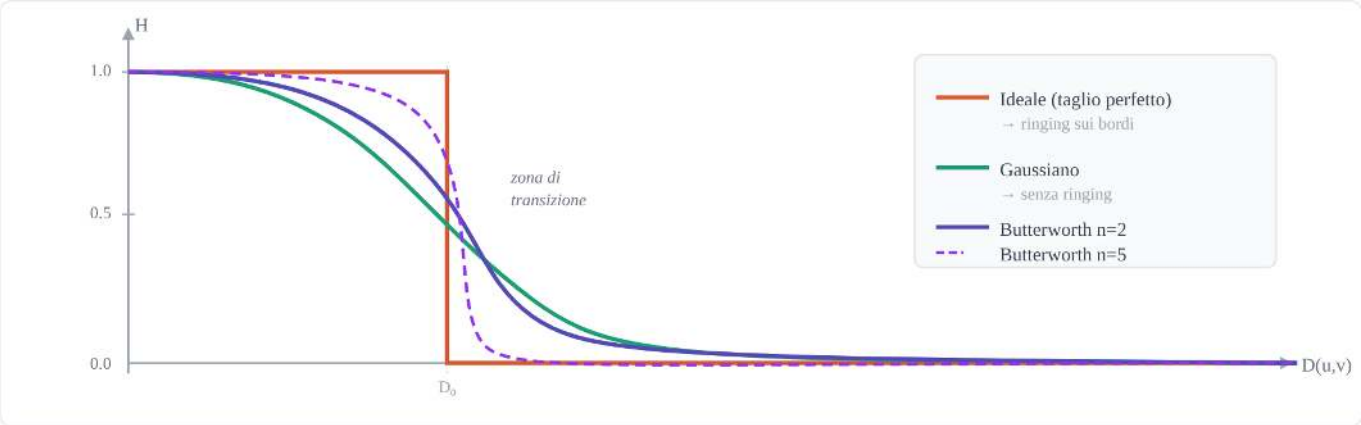


Figura 5.11: Filtri passa-basso.

5.5.2 Filtri Passa-Alto e Passa-Banda

I filtri passa-alto possono essere ottenuti da un filtro passa-basso complementare, definito come:

$$H_{\text{HP}}(u,v) = 1 - H_{\text{LP}}(u,v)$$

Questo tipo di filtro preserva le componenti ad alta frequenza, evidenziando bordi e dettagli, mentre attenua le regioni a variazione graduale.

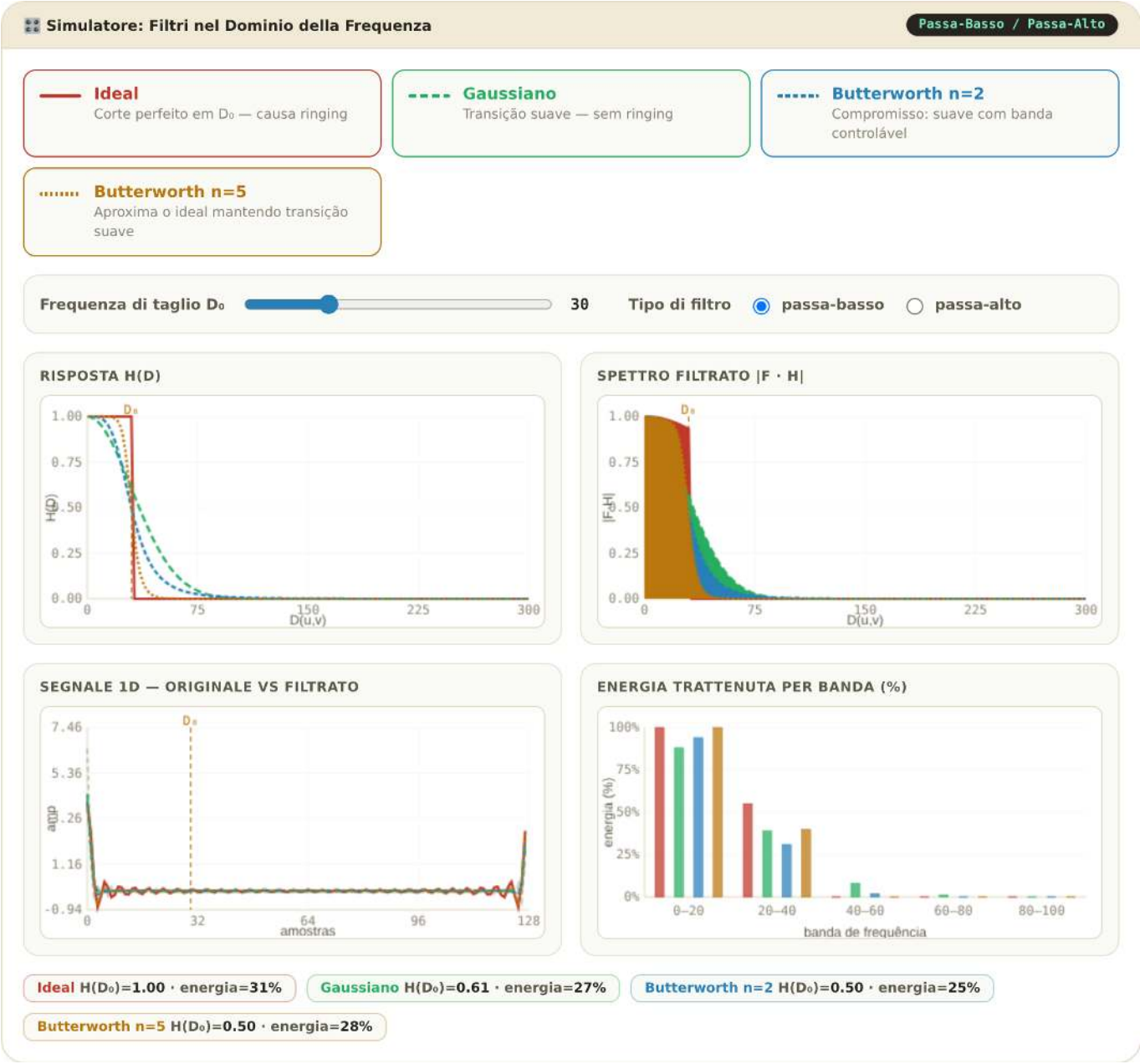
I filtri passa-banda preservano solo una banda intermedia di frequenze, limitata da due raggi D_L e D_H:

$$H_{\mathrm{BP}}(u,v) = H_{\mathrm{LP}}^{(D_H)}(u,v) \cdot \left[1 - H_{\mathrm{LP}}^{(D_L)}(u,v)\right]$$

Questo tipo di filtraggio è utile quando si desidera rimuovere simultaneamente le componenti a bassa e ad alta frequenza, preservando solo le strutture di scala intermedia.

Un'applicazione importante è la rimozione del **rumore periodico**, in cui i pattern regolari appaiono come picchi localizzati nello spettro di magnitudine. Questi picchi possono essere attenuati mediante filtri *notch* (reietta-banda), posizionati specificamente sulle frequenze indesiderate.

Esempi di filtri nel dominio della frequenza sono presentati nel simulatore della Figura 5.12, Figura 5.13 e Figura 5.14..



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-05-filtros.html

Figura 5.12: Simulatore interattivo di filtri nel dominio della frequenza.

```
import io

def distancia_centro(M, N):
```

```
    """Matrice delle distanze dal centro dello spettro."""
    u = np.arange(M) - M // 2
    v = np.arange(N) - N // 2
    V, U = np.meshgrid(v, u)
    return np.sqrt(U**2 + V**2)

def aplicar_filtro_freq(img, H):
    """Applica il filtro H (centrato) all'immagine tramite FFT."""
    F  = np.fft.fftshift(np.fft.fft2(img.astype(np.float64)))
    Fg = F * H
    g  = np.real(np.fft.ifft2(np.fft.ifftshift(Fg)))
    return cv2.normalize(g, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)


M, N = img_gray.shape
D    = distancia_centro(M, N)
D0   = 30      # frequenza di taglio
n_bw = 2       # ordine del Butterworth

#   Funzioni di trasferimento
H_ideal = (D <= D0).astype(np.float64)
H_gauss = np.exp(-D**2 / (2 * D0**2))
H_bw    = 1.0 / (1.0 + (D / D0)**(2 * n_bw))

#   Immagini filtrate
img_ideal = aplicar_filtro_freq(img_gray, H_ideal)
img_gauss = aplicar_filtro_freq(img_gray, H_gauss)
img_bw    = aplicar_filtro_freq(img_gray, H_bw)

#   Profili di H(u,v)
def fig2img(fig):
    b = io.BytesIO(); fig.savefig(b, format='png', dpi=100); plt.close(fig); b.seek(0)
    return (plt.imread(b)[:,:,:3]*255).astype(np.uint8)

fig, ax = plt.subplots(figsize=(6, 3))
linha   = M // 2
ax.plot(H_ideal[linha, :], label="Ideal",       color="#D85A30", lw=1.5, ls="--")
ax.plot(H_gauss[linha, :], label="Gaussiano",   color="#1D9E75", lw=1.5)
ax.plot(H_bw[linha, :],    label="Butterworth", color="#534AB7", lw=1.5)
ax.axvline(N//2-D0, color="#aaa", lw=0.8, ls=":")
ax.axvline(N//2+D0, color="#aaa", lw=0.8, ls=":")
ax.set(title="Perfis H(u,v) - linha central", xlabel="v", ylabel="H(u,v)")
ax.legend(fontsize=8); plt.tight_layout()
perfil_img = fig2img(fig)

#   Filtri H visualizzati
def H_vis(H):
    return cv2.normalize((H*255).astype(np.uint8), None, 0, 255, cv2.NORM_MINMAX)

mm.show(
    [img_gray, img_ideal, img_gauss, img_bw,
     H_vis(H_ideal), H_vis(H_gauss), H_vis(H_bw), perfil_img],
    titles=[
        "Originale", "LPF Ideale", "LPF Gaussiano", "LPF Butterworth (n=2)",
        "H Ideale",  "H Gaussiano","H Butterworth", "Profili H(u,v)"
    ],
    cols=4, figsize=(16, 9)
)
```

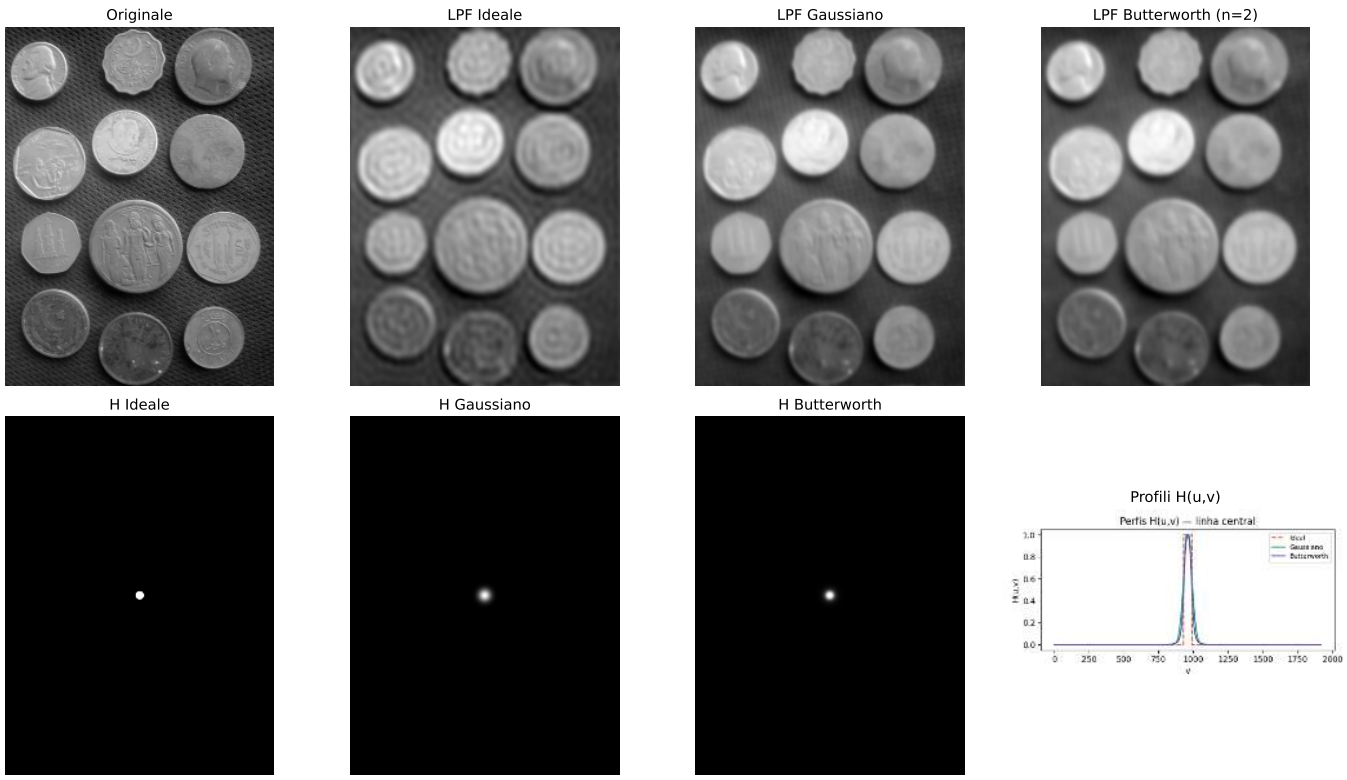


Figura 5.13: Confronto tra filtri passa-basso: Ideale (D =30), Gaussiano (D =30) e Butterworth (D =30, n=2). Profili de H(u,v) lungo una linea centrale e immagini filtrate corrispondenti.

```
# Filtro passa-alta: complemento do passa-baixa Gaussiano
# Reutiliza aplicar_filtro_freq() definida na célula anterior
H_alta   = 1 - H_gauss
img_alta = aplicar_filtro_freq(img_gray, H_alta)

mm.show(
    [img_gray, img_alta],
    titles=["Originale", "Passa-alto Gaussiano ($D_0=30$)"],
    cols=2
)
```

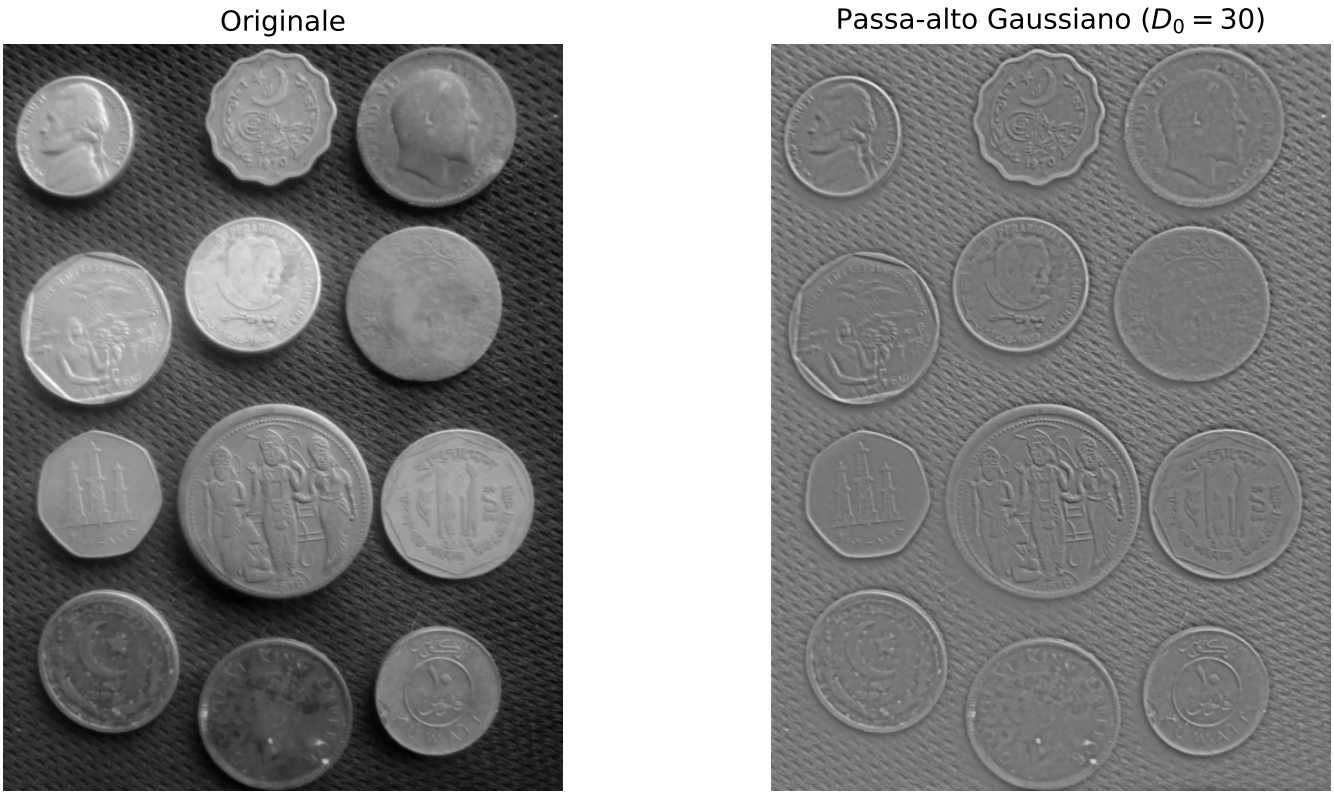


Figura 5.14: Filtro passa-alta Gaussiano. (a) Original; (b) Filtro passa-alta (D =30) - as bordas das moedas e fundo texturizado são realçados.

5.5.3 Rimozione del Rumore Periodico

Il rumore periodico — associato a interferenze elettriche, pattern regolari dei sensori o artefatti di scansione — appare nello spettro di Fourier come **picchi puntuali simmetrici rispetto al centro**.

Il filtro **reietta-banda (*notch*)** attenua selettivamente queste frequenze, preservando le altre componenti dell'immagine. Un esempio di applicazione è presentato nella Figura 5.15..

```
#   Immagine con rumore periodico sintetico
h_img, w_img = img_gray.shape
x = np.arange(w_img)
y = np.arange(h_img)
X, Y = np.meshgrid(x, y)

# Usare frequenze intere e consistenti (importante per una restaurazione perfetta)
u0, v0 = 20, 20  # frequenze esatte del rumore

ruido = 40 * np.sin(2 * np.pi * (u0 * X / w_img + v0 * Y / h_img))
img_ruidosa = np.clip(img_gray.astype(np.float64) + ruido, 0, 255).astype(np.uint8)

#   Spettro dell'immagine rumorosa
F_r = np.fft.fftshift(np.fft.fft2(img_ruidosa.astype(np.float64)))
mag_r = np.log1p(np.abs(F_r))
mag_vis = cv2.normalize(mag_r, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

#   Maschera notch
mascara = np.ones((h_img, w_img), dtype=np.float64)
r_notch = 8  # raggio del notch (regolazione fine se necessario)

def suprimir_pico(mask, cy, cx, r):
```

```
    """Azera un disco di raggio r centrato in (cy, cx)"""
    yy, xx = np.ogrid[:mask.shape[0], :mask.shape[1]]
    dist = np.sqrt((yy - cy)**2 + (xx - cx)**2)
    mask[dist <= r] = 0
    return mask

# Coordinate centrali
cy, cx = h_img // 2, w_img // 2

# Sopprimere i 4 picchi simmetrici (importante!)
for dy, dx in [(v0, u0), (-v0, -u0), (v0, -u0), (-v0, u0)]:
    mascara = suprimir_pico(mascara, cy + dy, cx + dx, r_notch)

mascara_vis = (mascara * 255).astype(np.uint8)

#   Filtraggio e ricostruzione
F_filtrada = F_r * mascara
img_rest = np.real(np.fft.ifft2(np.fft.ifftshift(F_filtrada)))
img_rest_vis = cv2.normalize(img_rest, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

# Valutazione
psnr = cv2.PSNR(img_gray, img_rest_vis)
ssim = cv2.SSIM(img_gray, img_rest_vis) if hasattr(cv2, 'SSIM') else "N/A"
#ssim = ssim_sk(img_gray, img_rest_vis, data_range=255)

print(f"PSNR (originale vs restaurata): {psnr:.2f} dB")

#   Visualizzazione
mm.show(
    [img_ruidosa, mag_vis, mascara_vis, img_rest_vis],
    titles=[
        "Con rumore periodico",
        "Spettro (log)",
        "Maschera notch",
        f"Restaurata (PSNR={psnr:.1f} dB)"
    ],
    cols=4,
    figsize=(16, 4)
)
```

```
PSNR (originale vs restaurata): 32.93 dB
```

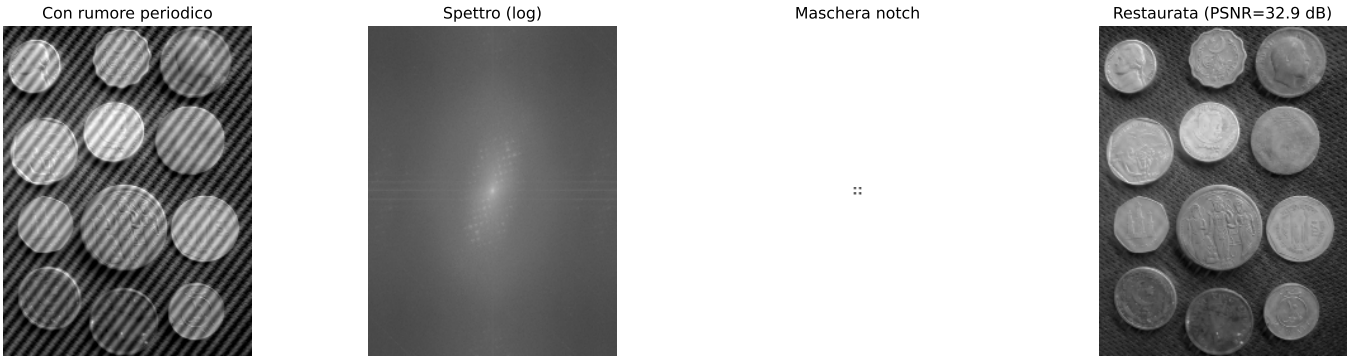


Figura 5.15: Rimozione del rumore periodico tramite filtro *notch* nel dominio della frequenza: (a) immagine con rumore sinusoidale, (b) spettro che mostra i picchi del rumore, (c) maschera *notch* centrata sui picchi, (d) immagine restaurata.

📌 Sintesi — Filtri Spettrali

Filtro	Effetto visivo	Artefatto	Uso
Passa-basso ideale	Attenuazione intensa	*Ringing*	Illustrativo
Passa-basso Gaussiano	Attenuazione graduale	Non presenta *ringing*	Attenuazione generale
Passa-basso Butterworth	Attenuazione controllata	*Ringing* (ordini elevati)	Compromesso tra attenuazione e selettività
Passa-alto	Enfatizzazione dei bordi	Amplificazione del rumore	Rilevamento dei contorni
Notch	Rimozione selettiva delle frequenze	Possibili distorsioni locali	Rimozione del rumore periodico

La progettazione dei filtri nel dominio della frequenza consiste nella definizione di maschere spettrali. Tuttavia, effetti nel dominio spaziale, come *ringing* e sfocatura, emergono direttamente da queste scelte nello spettro.

5.6 *Wavelet* e Multirisoluzione

La Trasformata di Fourier decompone il segnale in frequenze **globali**: ogni coefficiente $F(u, v)$ riceve contributi dall'intera immagine, senza informazioni esplicite sulla localizzazione spaziale di tali frequenze. Pertanto, strutture localizzate, come i bordi, sono rappresentate in modo distribuito nello spettro.

Le ***wavelet* (ondine)** superano questa limitazione utilizzando funzioni base **localizzate nello spazio**, che possono essere traslate e scalate. Queste funzioni possiedono **supporto compatto**, ossia sono diverse da zero solo in una regione finita del dominio, consentendo una rappresentazione simultanea in termini di **frequenza e localizzazione spaziale**.

5.6.1 Il Limite della Trasformata di Fourier: localizzazione spaziale

La Trasformata di Fourier descrive con precisione **quali frequenze sono presenti** in un segnale, ma non rappresenta esplicitamente **dove queste frequenze si verificano nello spazio**.

Nell'esperimento presentato nella Figura 5.16, due immagini con strutture localizzate in posizioni diverse producono spettri di magnitudine praticamente identici. Ciò accade perché la rappresentazione di Fourier è globale: ogni coefficiente riceve un contributo dall'intera immagine.

Di conseguenza, lo spettro di magnitudine non rappresenta esplicitamente la localizzazione dei bordi o di altre strutture, ma solo la distribuzione delle frequenze presenti. Questa limitazione ha motivato lo sviluppo di rappresentazioni multirisoluzione, come la Trasformata *Wavelet* Discreta (DWT), in grado di descrivere simultaneamente la frequenza e la localizzazione spaziale delle strutture dell'immagine.

```
img_sinal1 = np.zeros((128, 128)); img_sinal1[:, 20:25] = 1; img_sinal1[100:105, :] = 1
img_sinal2 = np.zeros((128, 128)); img_sinal2[:, 90:95] = 1; img_sinal2[30:35, :] = 1

mag1 = np.log1p(np.abs(np.fft.fftshift(np.fft.fft2(img_sinal1))))
mag2 = np.log1p(np.abs(np.fft.fftshift(np.fft.fft2(img_sinal2))))

mm.show([img_sinal1, mag1, img_sinal2, mag2],
        titles=["Señale A", "Spettro A", "Señale B (Spostato)", "Spettro B"],
        cols=4, figsize=(14, 4)
        )
```

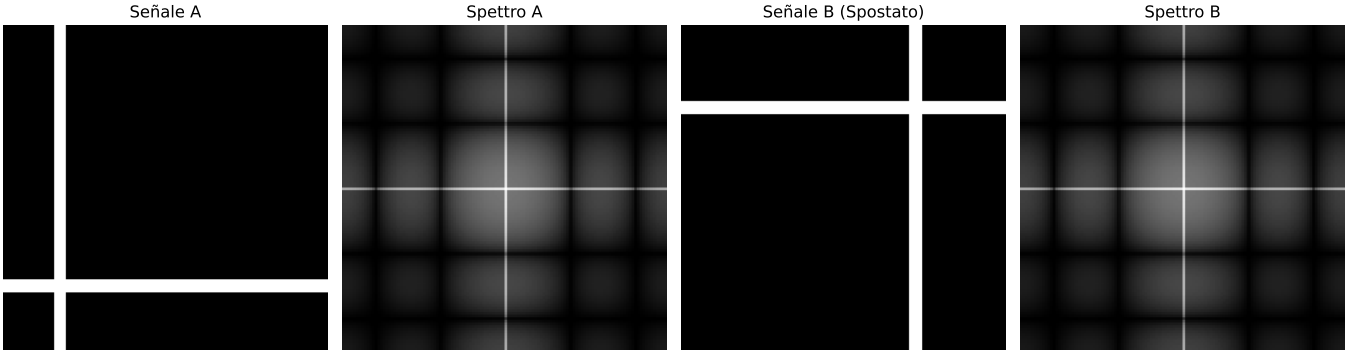


Figura 5.16: Fourier globale è cieco per la posizione. Gli spettri non dicono dove sono i bordi.

5.6.2 Trasformata *Wavelet* Discreta 2D

La Trasformata *Wavelet* Discreta (DWT) applica, separatamente nelle direzioni orizzontale e verticale, due filtri complementari: un **passa-basso** h (approssimazione) e un **passa-alto** g (dettagli), seguiti da sottocampionamento per un fattore di 2 in ciascuna dimensione. Questo processo produce quattro sottobande, i cui nomi indicano la combinazione dei filtri applicati in ciascuna direzione (L = *Low-pass*, passa-basso; H = *High-pass*, passa-alto). Le caratteristiche di ciascuna sottobanda sono riassunte nella Tabella 5.3.

$$\mathrm{DWT}(f) = \{ \underbrace{\mathrm{LL}}_{\text{approssimazione}}, \underbrace{\mathrm{LH}}_{\text{dettagli orizzontali}}, \underbrace{\mathrm{HL}}_{\text{dettagli verticali}}, \underbrace{\mathrm{HH}}_{\text{dettagli diagonali}} \}.$$

Tabella 5.3: Sottobande prodotte dalla Trasformata *Wavelet* Discreta 2D (DWT), indicando i filtri applicati in ciascuna direzione e il contenuto predominante di ciascuna componente.

Sottobanda	Filtri applicati	Contenuto visivo
LL	basso × basso	Approssimazione dell'immagine (versione smussata e ridotta)
LH	basso × alto	Bordi orizzontali e variazioni verticali
HL	alto × basso	Bordi verticali e variazioni orizzontali
HH	alto × alto	Dettagli diagonali e trame

La decomposizione può essere applicata ricorsivamente sulla sottobanda LL, generando una rappresentazione multirisoluzione. Dopo J livelli, si ottiene una struttura con $3J + 1$ sottobande, in cui ogni nuovo livello riduce la risoluzione della componente di approssimazione.

> **Collegamento con le CNN**
>
> La decomposizione multirisoluzione delle *wavelet* possiede una relazione concettuale con le rappresentazioni gerarchiche utilizzate nelle reti neurali convoluzionali (CNN). In entrambi i casi, successive fasi di filtraggio e riduzione della risoluzione producono descrizioni sempre più astratte dell'immagine. Tuttavia, le ***wavelet* utilizzano filtri matematicamente definiti e ricostruibili**, mentre le **CNN apprendono i propri filtri durante l'addestramento**.

5.6.3 Famiglie di *Wavelet*

Diverse famiglie di *wavelet* presentano compromessi distinti tra **supporto spaziale**, regolarità e capacità di compressione. Il **supporto** corrisponde all'estensione della funzione *wavelet* nel dominio spaziale: quanto più

piccolo è il supporto, tanto più localizzata è la funzione; quanto più grande, tanto più liscia tende a essere la sua rappresentazione, sebbene con un costo computazionale maggiore. La Tabella 5.4 confronta alcune delle famiglie più utilizzate.

Tabella 5.4: Confronto tra famiglie di *wavelet*, evidenziando la lunghezza del supporto, il numero di momenti nulli, la simmetria e le applicazioni tipiche.

Wavelet	Lunghezza del supporto	Momenti nulli	Simmetria	Uso tipico
Haar	2	1	Asimmetrica	Introduzione e analisi di base
Daubechies db4	8	4	Asimmetrica	Compressione e analisi generale
Symlet sym4	8	4	Quasi simmetrica	Ricostruzione di segnali
Biortogonale 5/3	5/3	2/2	Simmetrica	JPEG 2000 senza perdita
Biortogonale 9/7	9/7	4/4	Simmetrica	JPEG 2000 con perdita

I **momenti nulli** misurano la capacità della *wavelet* di rappresentare regioni uniformi dell'immagine con pochi coefficienti diversi da zero. Una *wavelet* con p momenti nulli annulla esattamente i polinomi di grado fino a $p-1$. Di conseguenza, quanto maggiore è il numero di momenti nulli, tanto maggiore tende a essere l'efficienza di compressione nelle regioni omogenee, sebbene ciò implichi generalmente funzioni con supporto più esteso.

La Figura 5.17 presenta le funzioni di base (*wavelet*) $\psi(t)$ nel dominio spaziale. Queste funzioni possiedono **supporto compatto**, cioè sono diverse da zero solo in una regione finita del dominio, contrariamente alle sinusoidi della Trasformata di Fourier, che si estendono su tutto il dominio.

```
#   Import pywt
try:
    import pywt
except ImportError:
    import subprocess
    subprocess.run(["pip", "install", "PyWavelets", "-q"])
    import pywt

wavelet_haar = pywt.Wavelet('haar')
wavelet_db4  = pywt.Wavelet('db4')

phi_h, psi_h, x_h = wavelet_haar.wavefun(level=4)
phi_d, psi_d, x_d = wavelet_db4.wavefun(level=4)

fig, ax = plt.subplots(1, 2, figsize=(10, 3))
ax[0].plot(x_h, psi_h, 'b', lw=2); ax[0].set_title("Ondaleta Haar ( )")
ax[1].plot(x_d, psi_d, 'g', lw=2); ax[1].set_title("Ondaleta Daubechies 4 ( )")
plt.tight_layout(); plt.show()
```

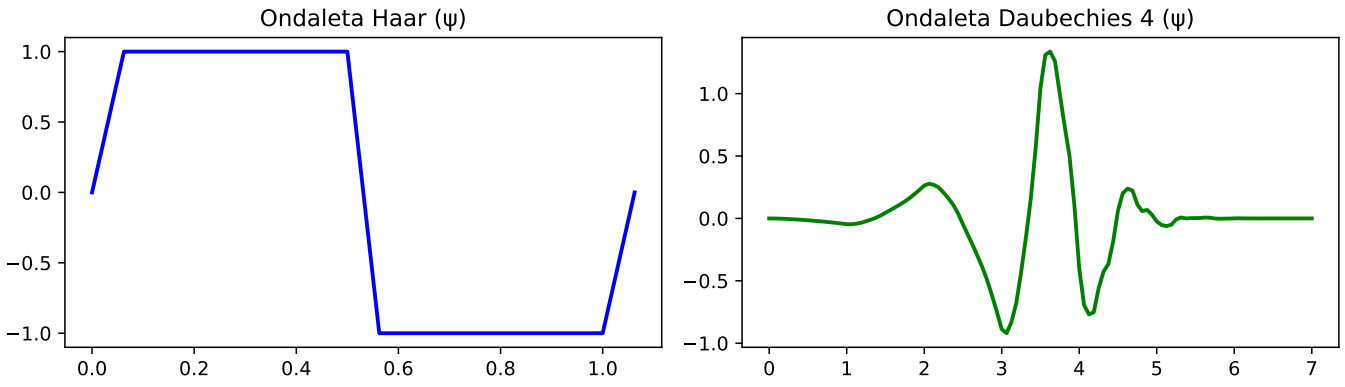


Figura 5.17: Funzioni della *Wavelet* ( ). Nota come rapidamente decadano a zero (supporto compatto), al contrario delle sinusoidi infinite di Fourier.

Il diagrama della Figura 5.18 illustra l'analisi multirisoluzione eseguita dalla DWT, in cui la sottobanda di approssimazione (LL) viene successivamente decomposta, formando una rappresentazione gerarchica su due livelli.

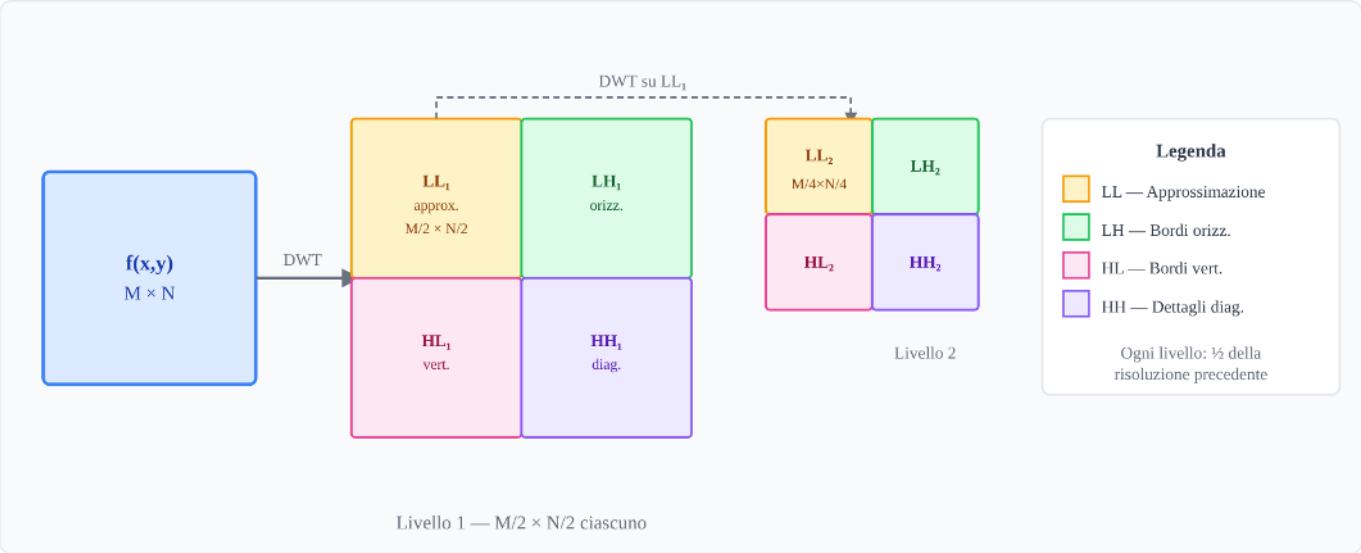


Figura 5.18: Diagramma della decomposizione *wavelet* 2D su due livelli.

Il simulatore Figura 5.19 consente di esplorare in modo interattivo la Trasformata *Wavelet* Discreta 2D (DWT) utilizzando la *wavelet* di Haar. La decomposizione in sottobande evidenzia la separazione tra la componente di approssimazione e le componenti di dettaglio dell'immagine.

I diversi modelli di ingresso permettono di osservare il comportamento direzionale dei filtri. In immagini con bordi orizzontali e verticali, le sottobande LH e HL evidenziano, rispettivamente, le variazioni verticali e orizzontali dell'intensità. Nelle regioni a variazione graduale, la maggior parte dell'energia si concentra nella sottobanda di approssimazione LL, mentre le sottobande di dettaglio presentano coefficienti prossimi allo zero.

L'analisi multirisoluzione può essere osservata anche aumentando il numero di livelli di decomposizione. In tal caso, solo la sottobanda LL_1 viene nuovamente decomposta, generando le sottobande LL_2, LH_2, HL_2 e HH_2, che formano il secondo livello della rappresentazione gerarchica.

In pattern costituiti da regioni omogenee di grande estensione, come una sfumatura graduale o una scacchiera composta da blocchi di grandi dimensioni, l'energia rimane prevalentemente concentrata nella sottobanda LL. Nella sfumatura, ciò avviene perché le differenze tra pixel adiacenti sono piccole. Nella scacchiera, invece, i pixel possiedono praticamente la stessa intensità all'interno di ciascun blocco, cosicché solo i confini tra

blocchi producono coefficienti non nulli nelle sottobande di dettaglio. Poiché tali confini occupano solo una piccola frazione dell'immagine, il loro contributo all'energia totale rimane ridotto.

Per rendere possibile l'analisi visiva di queste variazioni sottili, il simulatore incorpora un controllo del guadagno di contrasto dei dettagli (variabile da 1 a 8). Questo parametro funziona come un fattore di amplificazione lineare applicato esclusivamente ai coefficienti delle sottobande di dettaglio (LH, HL e HH) prima della loro visualizzazione a schermo. Negli scenari di transizione graduale (come il gradiente) o di uniformità locale (come l'interno dei blocchi della scacchiera), le differenze numeriche calcolate dal filtro passa-alto di Haar producono coefficienti molto prossimi allo zero, il che renderebbe i quadranti corrispondenti scuri e impercettibili a occhio nudo. Moltiplicando questi valori per il guadagno, il simulatore recupera visivamente le strutture ad alta frequenza nascoste ed enfatizza l'orientamento dei bordi residui.

Il grafico dell'energia per sottobanda quantifica tale distribuzione tra la componente di approssimazione e le componenti di dettaglio, dimostrando che il guadagno visivo non altera la metrica originale dell'energia. Nelle immagini naturali, la maggior parte dell'energia si concentra nella sottobanda LL, mentre le sottobande LH, HL e HH rappresentano principalmente bordi, texture e altre variazioni locali dell'intensità.

5.6.4 Analisi Multirisoluzione con la DWT 2D

La Trasformata *Wavelet* Discreta 2D (DWT) decompone un'immagine in componenti di approssimazione e dettaglio, organizzate in modo gerarchico su diverse scale e orientazioni. Poiché le sottobande di dettaglio in immagini naturali spesso presentano coefficienti a basso contrasto, gli esempi pratici che seguono utilizzano un pattern geometrico sintetico generato in Python. Questo approccio replica il comportamento del simulatore della Figura 5.19, rendendo visivamente espliciti gli effetti del filtraggio spaziale e della decomposizione multirisoluzione.

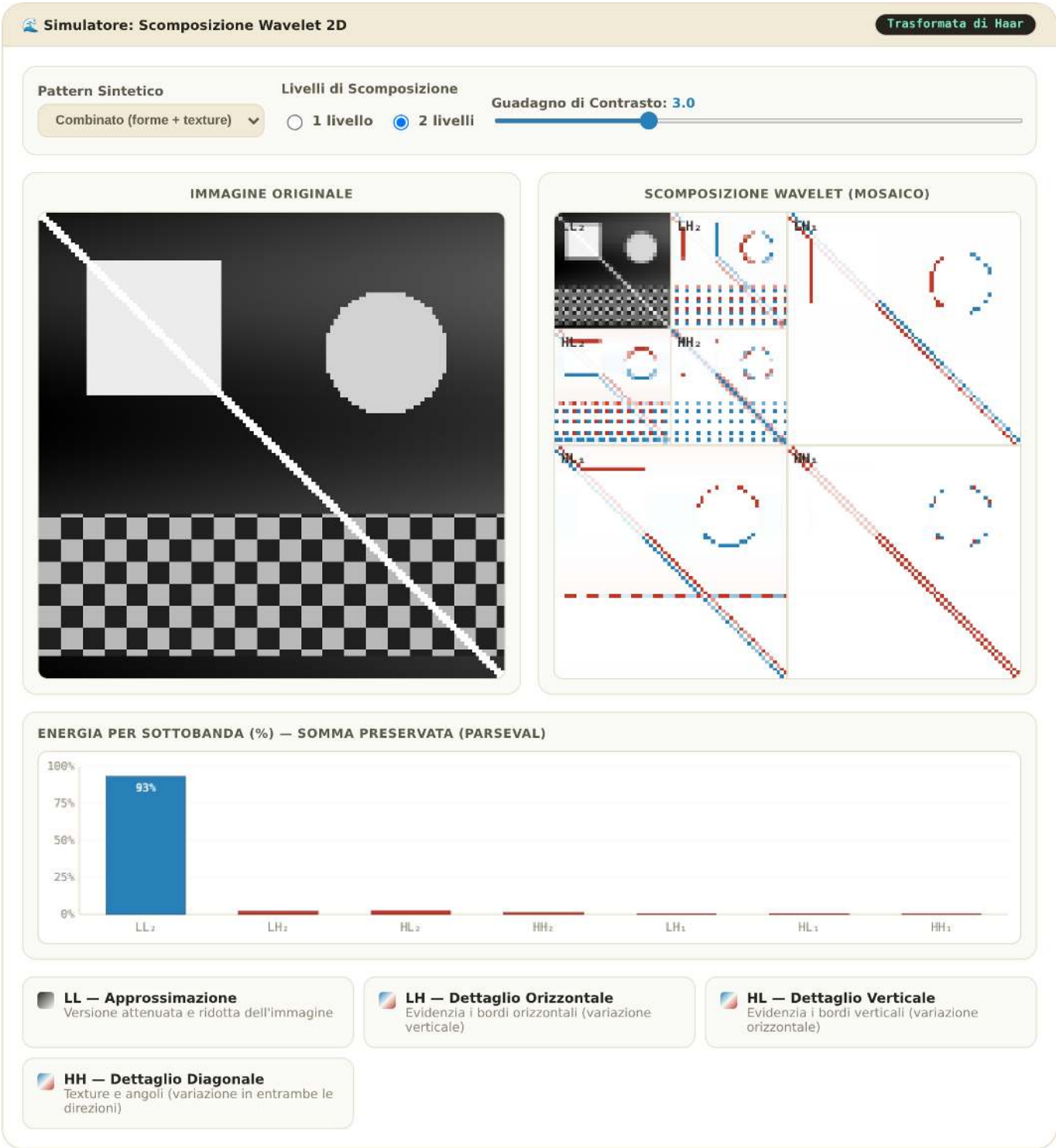
5.6.4.1 Scomposizione a Mosaico su Più Livelli

La Figura 5.20 illustra la struttura gerarchica della DWT su due livelli utilizzando la *wavelet* di Haar. Il processo si basa sull'applicazione combinata di filtri passa-basso e passa-alto nelle direzioni orizzontale e verticale, seguiti da un sottocampionamento con fattore 2.

Nel primo livello, l'immagine originale genera la sottobanda di approssimazione (LL_1) e le componenti di dettaglio orizzontale (LH_1), verticale (HL_1) e diagonale (HH_1). Nell'analisi multirisoluzione, la sottobanda LL_1 viene nuovamente filtrata e sottocampionata, producendo il secondo livello di scomposizione (LL_2, LH_2, HL_2 e HH_2).

Per consentire l'interpretazione visiva delle componenti di dettaglio, il codice estrae il valore assoluto dei loro coefficienti e applica una normalizzazione lineare (*min-max*) per occupare l'intera gamma dinamica dei toni di grigio [0, 255]. Questa operazione trasforma le regioni omogenee (coefficienti nulli) in nero ed evidenzia in bianco i bordi e le texture estratte a ciascuna scala e orientazione.

```
try:
    import pywt
    HAS_PYWT = True
except ImportError:
    import subprocess
    subprocess.run(["pip", "install", "PyWavelets", "-q"])
    import pywt
    HAS_PYWT = True

import numpy as np
import cv2

#   Geração da Imagem Sintética (Mesmo padrão 'combined' do simulador)
def gerar_imagem_sintetica(N=256):
    img = np.zeros((N, N), dtype=np.float64)
    for y in range(N):
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-05-wavelet.html

Figura 5.19: Simulazione della decomposizione *wavelet* 2D.

```
        for x in range(N):
            v = 55 + 35 * (x / N) + 15 * np.sin(y / 24)
            # Quadrado
            if 24 < x < 100 and 24 < y < 100:
                v = 225
            # Círculo
            cx, cy, r = 190, 76, 34
            if (x - cx)**2 + (y - cy)**2 < r**2:
                v = 205
            # Textura periódica (inferior)
            if y > 164 and y < 244:
                p = 12
                v = 185 if ((x // p + y // p) % 2 == 0) else 65
            # Linha diagonal
            if abs(x - y) < 4:
                v = 240
            img[y, x] = np.clip(v, 0, 255)
    return img.astype(np.uint8)

# Substitui a imagem escura de moedas pelo padrão sintético claro
img_gray = gerar_imagem_sintetica(256)

#   Decomposição wavelet 2 níveis
wavelet   = "haar"
img_float = img_gray.astype(np.float64)

# Nível 1
coefs1 = pywt.dwt2(img_float, wavelet)
LL1, (LH1, HL1, HH1) = coefs1

# Nível 2 (aplicado sobre LL1)
coefs2 = pywt.dwt2(LL1, wavelet)
LL2, (LH2, HL2, HH2) = coefs2

def sb_vis(sb):
    """Normalizza la sottobanda per la visualizzazione [0,255]."""
    return cv2.normalize(np.abs(sb), None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

print(f"Forma originale     : {img_gray.shape}")
print(f"LL1 (livello 1)     : {LL1.shape}  |  LH1/HL1/HH1: {LH1.shape}")
print(f"LL2 (livello 2)     : {LL2.shape}    |  LH2/HL2/HH2: {LH2.shape}")

imgs_dwt   = [img_gray, sb_vis(LL1), sb_vis(LH1), sb_vis(HL1), sb_vis(HH1),
              sb_vis(LL2), sb_vis(LH2), sb_vis(HL2), sb_vis(HH2)]
titles_dwt = ["Original",
              "LL  (aprox.)", "LH  (horiz.)", "HL  (vert.)", "HH  (diag.)",
              "LL  (aprox.)", "LH  (horiz.)", "HL  (vert.)", "HH  (diag.)"]

mm.show(imgs_dwt, titles=titles_dwt, cols=5, figsize=(16, 7))
```

```
Forma originale     : (256, 256)
LL1 (livello 1)     : (128, 128)  |  LH1/HL1/HH1: (128, 128)
LL2 (livello 2)     : (64, 64)    |  LH2/HL2/HH2: (64, 64)
```

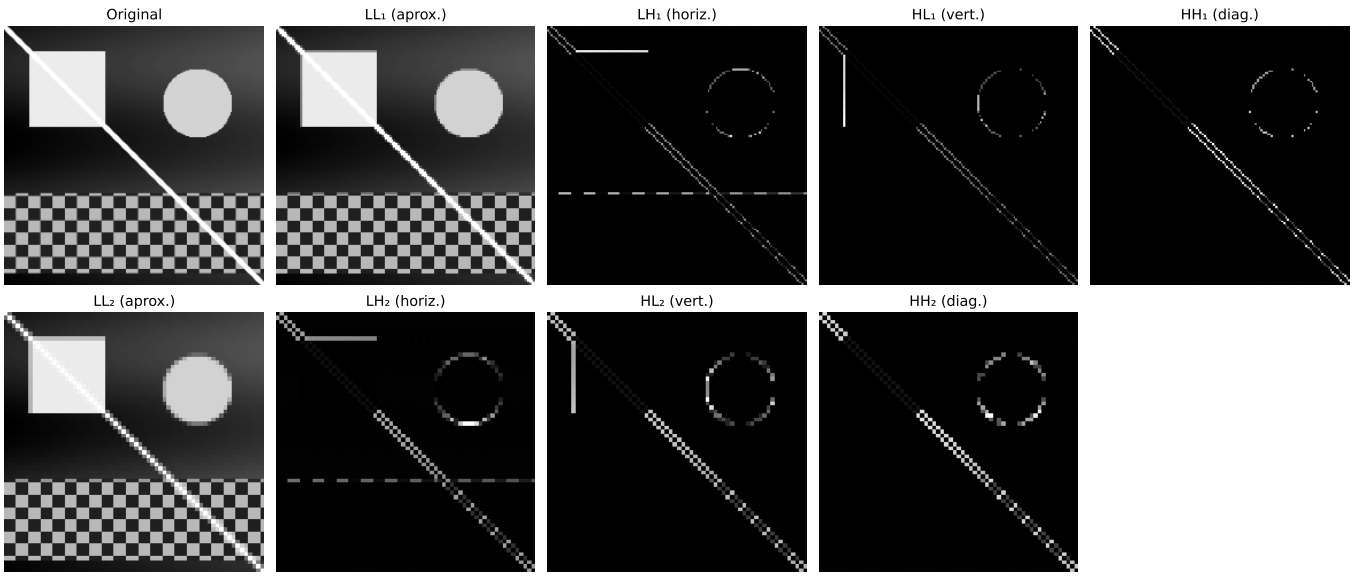


Figura 5.20: Decomposição *wavelet* 2D de 2 níveis com *wavelet* Haar: subbandas LL, LH, HL, HH em cada nível. As subbandas de detalhe revelam estruturas orientadas em diferentes escalas utilizando um padrão sintético.

5.6.4.2 Il Compromesso tra Localizzazione e Morbidezza

La scelta della funzione di base (*wavelet*) influenza direttamente il modo in cui le caratteristiche dell'immagine vengono distribuite e codificate dai coefficienti della DWT. La Figura 5.21 confronta i risultati pratici ottenuti applicando quattro famiglie distinte sul pattern geometrico sintetico: `haar`, `db4`, `sym4` e `bior2.2`.

Poiché possiede un supporto corto e una forma a funzione a gradino, la *wavelet* di Haar produce coefficienti altamente localizzati in corrispondenza delle discontinuità spaziali, generando bordi sottili e nitidi nelle sottobande di dettaglio. Al contrario, famiglie come Daubechies (`db4`) e Symlets (`sym4`), che presentano un supporto maggiore (filtri più lunghi) e un numero più elevato di momenti nulli, generano risposte più morbide e distribuite attorno alle transizioni, il che può introdurre lievi oscillazioni o effetti di sfocatura sui confini netti.

Questo comportamento evidenzia il classico compromesso (*trade-off*) dell'analisi multirisoluzione: supporti più piccoli favoriscono la localizzazione spaziale esatta dei bordi, mentre supporti più ampi e un maggior numero di momenti nulli tendono a produrre rappresentazioni più sparse e morbide. Questa morbidezza e la capacità di attenuazione delle alte frequenze garantiscono una maggiore efficienza nella compattazione dell'energia, caratteristiche fondamentali per applicazioni di compressione dei dati e rimozione del rumore (*denoising*).

```
import numpy as np
import cv2
import pywt

# Garantisce che img_gray e img_float utilizzino lo stesso motivo sintetico chiaro
if 'gerar_imagem_sintetica' in globals():
    img_gray = gerar_imagem_sintetica(256)
else:
    # Fallback qualora il blocco precedente non sia stato eseguito nella stessa sessione
    def gerar_imagem_sintetica(N=256):
        img = np.zeros((N, N), dtype=np.float64)
        for y in range(N):
            for x in range(N):
                v = 55 + 35 * (x / N) + 15 * np.sin(y / 24)
                if 24 < x < 100 and 24 < y < 100: v = 225
```

```
                cx, cy, r = 190, 76, 34
                if (x - cx)**2 + (y - cy)**2 < r**2: v = 205
                if y > 164 and y < 244:
                    p = 12
                    v = 185 if ((x // p + y // p) % 2 == 0) else 65
                if abs(x - y) < 4: v = 240
                img[y, x] = np.clip(v, 0, 255)
        return img.astype(np.uint8)
    img_gray = gerar_imagem_sintetica(256)

img_float = img_gray.astype(np.float64)

wavelets_comp = ["haar", "db4", "sym4", "bior2.2"]
imgs_comp, titles_comp = [], []

for wname in wavelets_comp:
    LL, (LH, HL, HH) = pywt.dwt2(img_float, wname)
    imgs_comp  += [sb_vis(LL), sb_vis(HH)]
    titles_comp += [f"{wname} - LL ", f"{wname} - HH "]

mm.show(imgs_comp, titles=titles_comp, cols=4, figsize=(14, 8))
```

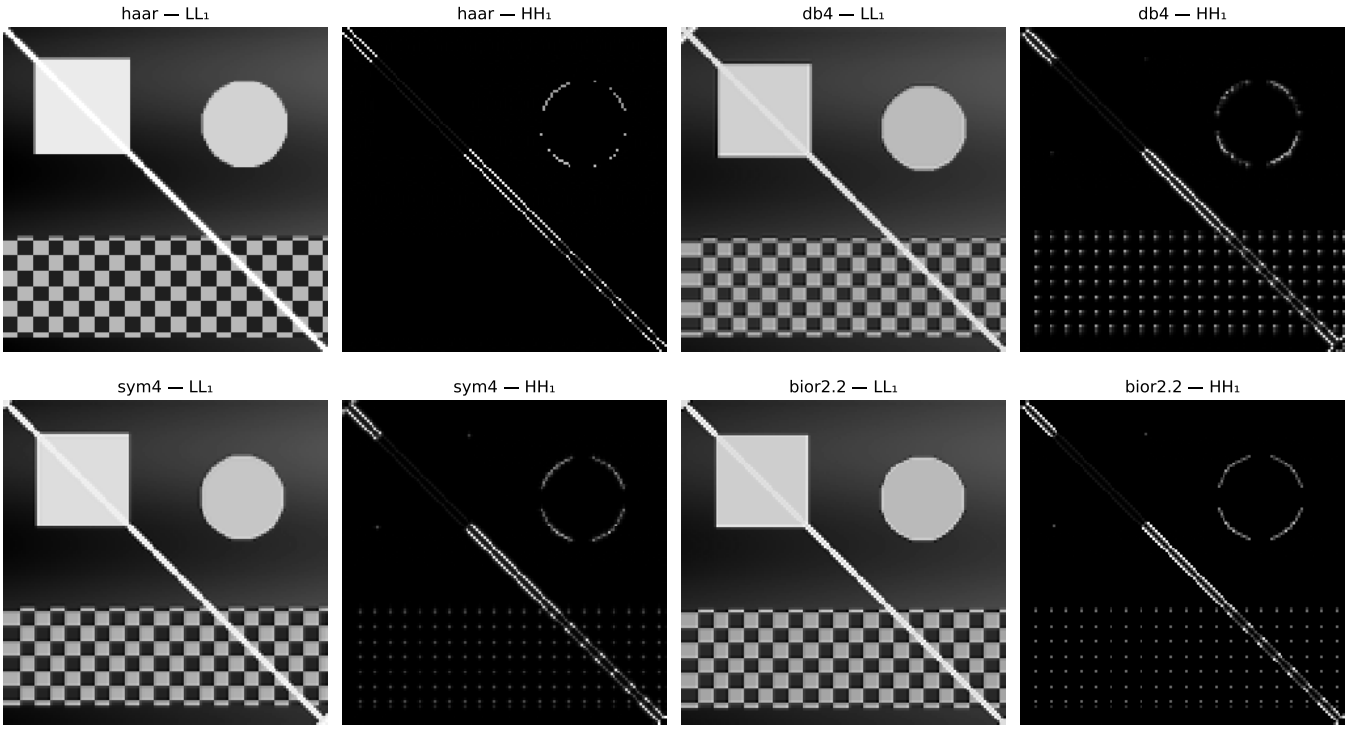


Figura 5.21: Confronto tra famiglie di *wavelet*: Haar, db4, sym4 e bior2.2. Sottobanda LL (approssimazione) e HH (diagonale) per ciascuna scelta, illustrando il compromesso tra compattezza e levigatezza basato sul motivo sintetico.

5.6.4.3 Limiarizzazione dei Coefficienti e Compressione

Una delle principali applicazioni della Trasformata *Wavelet* Discreta (DWT) è la compressione dei dati, favorita dalla capacità di rappresentazione **sparsa** dei coefficienti. La Figura 5.22 illustra l'effetto della limiarizzazione netta (*hard thresholding*), tecnica in cui i coefficienti di dettaglio con magnitudine inferiore a una soglia T vengono integralmente azzerati prima del processo di sintesi eseguito dalla Trasformata *Wavelet* Discreta Inversa (IDWT).

Man mano che la soglia T viene aumentata, un volume crescente di coefficienti ad alta frequenza viene

azzerato. Poiché concentrano meno energia, la rimozione di queste componenti riduce considerevolmente la quantità di informazione necessaria per rappresentare l'immagine, mantenendo intatta la componente di approssimazione globale (la sottobanda LL più profonda) per preservare la struttura macro. Visivamente, questo scarto di coefficienti si manifesta attraverso la scomparsa progressiva delle trame fini e la levigatura delle transizioni brusche di intensità.

La fedeltà dell'immagine ricostruita rispetto all'originale è quantificata dalla metrica del **Picco del Rapporto Segnale-Rumore (PSNR, *Peak Signal-to-Noise Ratio*)**, espressa in decibel (dB). Valori più elevati di PSNR indicano una distorsione minore e una maggiore prossimità matematica al segnale originale. L'esperimento pratico evidenzia il decadimento graduale del PSNR all'aumentare dell'aggressività della limiarizzazione, consentendo di valutare numericamente la soglia ottimale per il bilanciamento tra compressione e degrado visivo.

```
import numpy as np
import cv2
import pywt

# Garantisce che img_gray utilizzi lo stesso pattern sintetico chiaro
if 'gerar_imagem_sintetica' in globals():
    img_gray = gerar_imagem_sintetica(256)
else:
    # Fallback nel caso il blocco precedente non sia stato eseguito nella stessa sessione
    def gerar_imagem_sintetica(N=256):
        img = np.zeros((N, N), dtype=np.float64)
        for y in range(N):
            for x in range(N):
                v = 55 + 35 * (x / N) + 15 * np.sin(y / 24)
                if 24 < x < 100 and 24 < y < 100: v = 225
                cx, cy, r = 190, 76, 34
                if (x - cx)**2 + (y - cy)**2 < r**2: v = 205
                if y > 164 and y < 244:
                    p = 12
                    v = 185 if ((x // p + y // p) % 2 == 0) else 65
                if abs(x - y) < 4: v = 240
                img[y, x] = np.clip(v, 0, 255)
        return img.astype(np.uint8)
    img_gray = gerar_imagem_sintetica(256)

def dwt_threshold_reconstruct(img, wavelet='db4', nivel=2, threshold=0.0):
    """Decompone, applica la soglia e ricostruisce tramite IDWT."""
    coefs = pywt.wavedec2(img.astype(np.float64), wavelet, level=nivel)
    # Copia e applica hard thresholding su tutti i dettagli
    coefs_t = [coefs[0]]   # LL finale non viene sogliato
    for detalhe in coefs[1:]:
        coefs_t.append(tuple(pywt.threshold(sb, threshold, mode='hard') for sb in detalhe))
    rec = pywt.waverec2(coefs_t, wavelet)
    # Ritaglio alla dimensione originale
    rec = rec[:img.shape[0], :img.shape[1]]
    return np.clip(rec, 0, 255).astype(np.uint8)

thresholds = [0, 10, 30, 60, 100]
imgs_thr   = [img_gray]
titles_thr = ["Original"]

for t in thresholds:
    rec  = dwt_threshold_reconstruct(img_gray, threshold=t)
    psnr = cv2.PSNR(img_gray, rec)
    imgs_thr.append(rec)
    titles_thr.append(f"T={t}  PSNR={psnr:.1f} dB")
```

```
mm.show(imgs_thr, titles=titles_thr, cols=3, figsize=(14, 10))
```

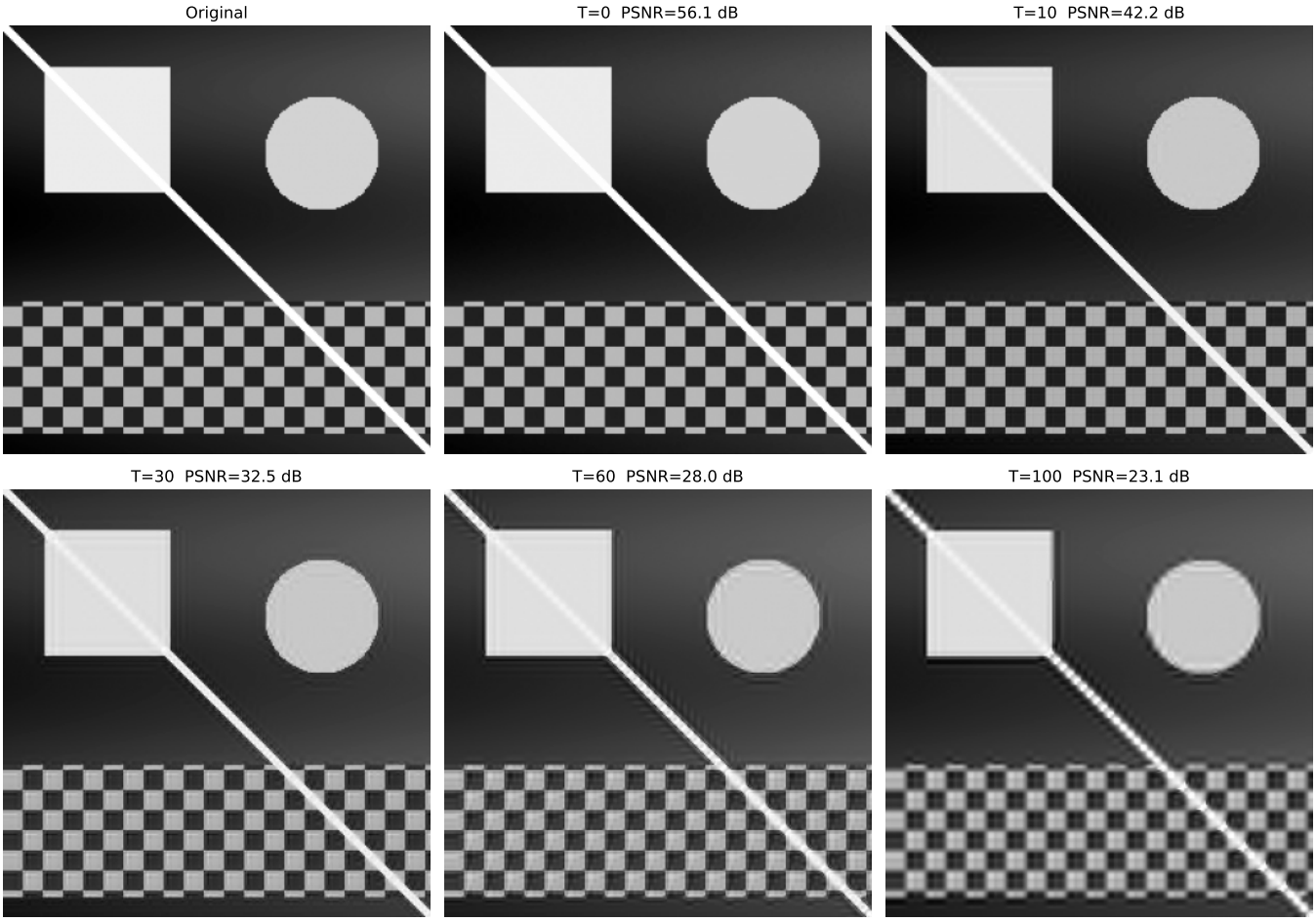


Figura 5.22: Ricostruzione *wavelet* con sogliatura dei coefficienti (*hard thresholding*): all'aumentare della soglia, più dettagli vengono azzerati, producendo immagini progressivamente più morbide. La metrica PSNR quantifica la perdita di qualità sul pattern sintetico.

Sintesi — Fourier vs. *Wavelet*: quando utilizzare ciascun approccio?

La Tabella 5.5 sintetizza le principali differenze strutturali e operative tra la Trasformata Discreta di Fourier (DFT) e la Trasformata *Wavelet* Discreta (DWT).

Tabella 5.5: Confronto tra la Trasformata Discreta di Fourier (DFT) e la Trasformata *Wavelet* Discreta (DWT), evidenziandone le principali caratteristiche e applicazioni.

Criterio	Fourier (DFT)	*Wavelet* (DWT)
Funzioni di base	Sinusoidi di supporto infinito	Funzioni di supporto compatto
Localizzazione spaziale	Non esplicita (globale)	Esplicita (locale)
Filtraggio spettrale	Eccellente per il controllo fine delle frequenze	Basata su sottobande (scale)
Compressione delle immagini	Base della DCT (JPEG tradizionale)	Base della DWT (JPEG 2000)
Analisi multiscala	No	Sì
Rimozione del rumore periodico	Altamente efficiente	Poco indicata
Segnali non stazionari	Limitata	Altamente efficiente

In termini pratici, la DFT si consolida come lo strumento ideale per l'analisi spettrale pura, la progettazione di filtri selettivi nel dominio della frequenza e l'attenuazione di rumori periodici e armonici. D'altro canto, la DWT eccelle in scenari che richiedono la rigorosa preservazione della localizzazione spaziale delle caratteristiche associata al loro contenuto frequenziale, distinguendosi nella compressione dei dati, nell'analisi multirisoluzione e nell'elaborazione di transizioni brusche. Pertanto, entrambe le trasformate devono essere comprese come tecniche perfettamente complementari, che tracciano percorsi distinti e specifici per la risoluzione di problemi nell'ambito dell'elaborazione digitale di immagini e visione artificiale (PDI-VC).

Analogie con l'Audio: Limitazioni e Precauzioni

Nel tracciare analogie tra l'elaborazione delle immagini e l'audio, è importante considerare le differenze fondamentali:

- Nei sistemi audio stereo/multicanale, la fase tra i canali è cruciale per la percezione della localizzazione spaziale (differenze interaurali di fase e di tempo).
- Nei sistemi monoaurali, la fase ha un'influenza percettiva limitata — l'orecchio umano è relativamente insensibile alla fase assoluta di componenti sinusoidali isolate.
- Nelle immagini, la fase della DFT è sempre fondamentale per la localizzazione spaziale delle strutture, indipendentemente dal fatto che l'immagine sia monocromatica o a colori.

L'analogia tra la fase nell'audio e la fase nelle immagini deve essere utilizzata con cautela, evidenziando che, sebbene entrambe trasportino informazioni sull'organizzazione spaziale/temporale del segnale, i meccanismi percettivi sono fondamentalmente differenti.

5.7 Compressione delle Immagini

Mentre le *wavelet* stabiliscono il fondamento teorico dello standard JPEG 2000, lo standard JPEG tradizionale si basa sulla **Trasformata Discreta dei Coseni (DCT, *Discrete Cosine Transform*)**. Nonostante le differenze strutturali, entrambi gli approcci condividono lo stesso principio fondamentale: compattare l'energia dell'immagine in un numero ridotto di coefficienti e scartare le componenti di minore rilevanza con impatto visivo minimo.

L'obiettivo centrale della compressione è ridurre il volume di dati necessario per l'archiviazione o la trasmissione di un'immagine. Tale processo è reso possibile dall'identificazione e dall'eliminazione di **ridondanze** strutturali e percettive.

5.7.1 Tassonomia delle Ridondanze

Lo sviluppo di algoritmi di compressione si fonda sull'identificazione e sull'eliminazione di tre categorie principali di ridondanza, sintetizzate nella Tabella 5.6.

Tabella 5.6: Categorie di ridondanza nelle immagini digitali e i rispettivi meccanismi di sfruttamento.

Tipo	Definizione	Approccio di Sfruttamento
Spaziale (interpixel)	Elevata correlazione e dipendenza statistica tra pixel adiacenti.	DCT, DWT e codifica predittiva.
Spettrale (intercanale)	Correlazione statistica tra i canali di colore di una stessa immagine.	Trasformazioni dello spazio colore (es: RGB in YC_bC_r).
Psicovisuale	Insensibilità del sistema visivo umano (SVH) alle variazioni ad alta frequenza e a basso contrasto.	Processi di quantizzazione selettiva dei coefficienti.

A seconda della preservazione dell'informazione originale dopo il processo di decodifica, i metodi di compressione si dividono in due classi fondamentali:

- **Senza perdita (*lossless*):** Garantisce una ricostruzione bit per bit identica all'immagine originale. Viene impiegata in scenari in cui l'integrità dei dati è strettamente critica, come nelle immagini mediche, nella diagnostica per immagini e nell'archiviazione di documenti testuali.
- **Con perdita (*lossy*):** Ammette l'introduzione di una distorsione controllata nel segnale in cambio di tassi di compressione sostanzialmente più elevati. È l'approccio standard per le fotografie di consumo e lo *streaming* video, ecosistemi nei quali il SVH tollera piccole attenuazioni dell'alta frequenza senza percezione di degrado della qualità visiva.

5.7.2 Trasformata del Coseno Discreta (DCT-II 2D)

La **Trasformata del Coseno Discreta** (DCT) costituisce l'operazione centrale dello standard JPEG. A differenza della DFT, che utilizza una base complessa, la DCT si basa su funzioni trigonometriche puramente reali. Per un blocco immagine $f(x, y)$ di dimensioni $N \times N$, la **DCT-II 2D** mappa il segnale spaziale nel dominio delle frequenze spaziali, generando la matrice dei coefficienti $C(u, v)$ tramite:

$$C(u, v) = \alpha(u)\,\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y)\, \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right] \tag{5.9}$$

dove i fattori di normalizzazione ortogonale sono dati da $\alpha(0) = \sqrt{1/N}$ e $\alpha(k) = \sqrt{2/N}$ per $k > 0$.

Ogni coefficiente $C(u, v)$ quantifica il contributo — o "peso" — di una specifica frequenza spaziale all'interno di quel blocco. Il termine $C(0, 0)$ è denominato **componente DC** e rappresenta l'intensità media del blocco (frequenza nulla). I restanti coefficienti, chiamati **componenti AC** (*Alternating Current*), corrispondono a frequenze spaziali progressivamente più elevate.

5.7.3 Le Funzioni di Base della DCT

Da una prospettiva geometrica, la Equazione 5.9 realizza la proiezione del blocco di pixel su un insieme di funzioni ortogonali. Per il caso standard del JPEG ($N = 8$), il blocco spaziale viene decomposto in una combinazione lineare di **64 funzioni di base** bidimensionali, denotate da $B_{u,v}(x, y)$ e generate dal prodotto di funzioni cosinusoidali:

$$B_{u,v}(x, y) = \cos\left[\frac{\pi(2x+1)u}{16}\right] \cos\left[\frac{\pi(2y+1)v}{16}\right]$$

In questo modo, l'operazione inversa può essere interpretata come la ricostruzione esatta del blocco originale tramite la somma ponderata di queste 64 matrici di base, dove ogni coefficiente $C(u, v)$ funge da peso analitico della rispettiva componente armonica.

La **frequenza spaziale** indicata dagli indici (u, v) determina il numero di cicli di oscillazione lungo le dimensioni orizzontali e verticali del blocco. Come illustrato nella Figura 5.23 — il cui codice isola ciascuna base applicando la trasformazione inversa su impulsi unitari —, queste 64 funzioni sono organizzate in una matrice 8×8. L'angolo superiore sinistro ($u = 0, v = 0$) mostra il pattern uniforme a frequenza nulla (DC), mentre il progredire verso destra (asse u) o verso il basso (asse v) mappa variazioni armoniche progressivamente maggiori, rappresentando transizioni rapide, bordi e trame nelle orientazioni orizzontali, verticali e diagonali.

> **DCT vs DFT: Vantaggio della Compattazione dell'Energia**
>
> Sia la DCT che la DFT mappano un blocco spaziale $N \times N$ in una matrice di coefficienti della stessa dimensione. Tuttavia, per immagini naturali, la DCT presenta una maggiore efficienza nella **compattazione dell'energia** alle basse frequenze. Ciò avviene perché la DCT assume implicitamente una simmetria pari del segnale ai bordi del blocco, il che equivale a un'estensione periodica continua, minimizzando l'effetto di dispersione spettrale (*ringing*). Di conseguenza, la maggior parte dei coefficienti

AC decade rapidamente verso valori prossimi allo zero, ottimizzando il *pipeline* di compressione senza introdurre una degradazione visiva percepibile.

```
from scipy.fft import dct, idct  # riga aggiunta

fig, axes = plt.subplots(8, 8, figsize=(6, 6))
fig.subplots_adjust(hspace=0.05, wspace=0.05)
for i in range(8):
    for j in range(8):
        coef = np.zeros((8, 8)); coef[i, j] = 1
        b = idct(idct(coef.T, norm='ortho').T, norm='ortho')
        axes[i, j].imshow(b, cmap='gray')
        axes[i, j].axis('off')
plt.suptitle("As 64 Bases da DCT 8x8", y=0.92, fontsize=12, fontweight='bold')
plt.show()
```

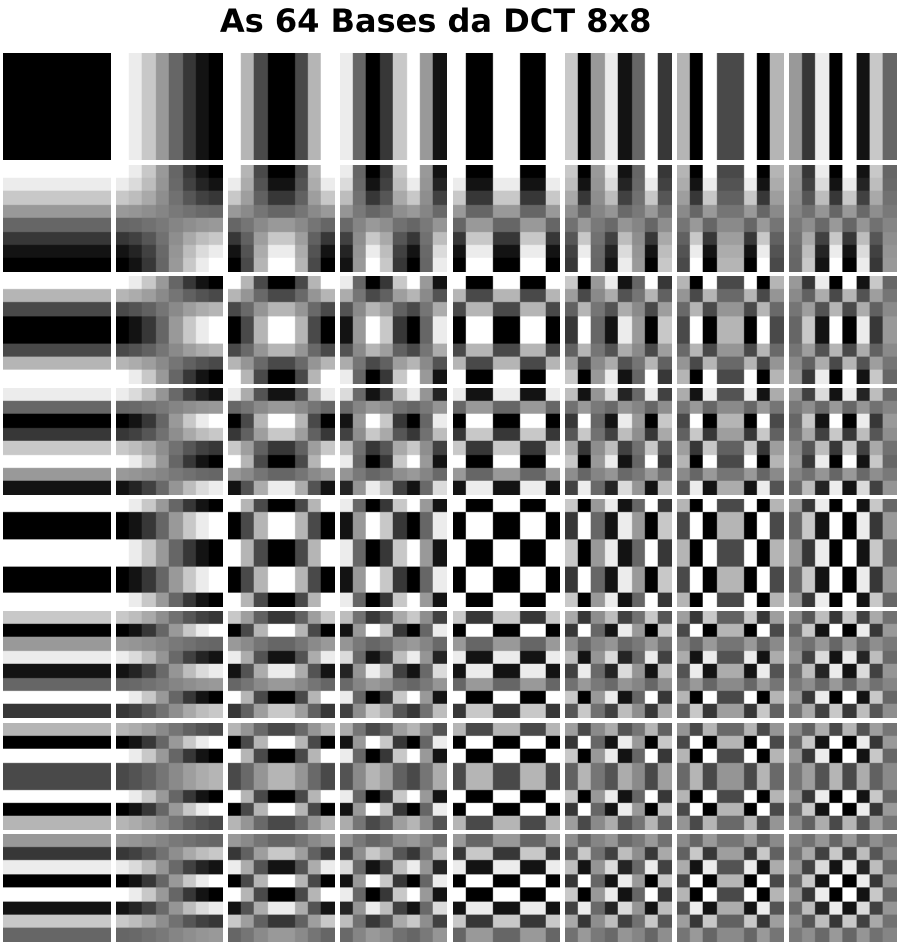


Figura 5.23: L'alfabeto visivo del JPEG: Le 64 funzioni di base della DCT-II. Il coefficiente DC si trova in alto a sinistra (liscio). Scendendo e avanzando verso destra, l'oscillazione spaziale aumenta drasticamente.

5.7.4 Concentrazione di Energia e Ricostruzione Progressiva

Prima dell'applicazione della DCT, i pixel del blocco di intensità vengono di routine traslati (sottraendo 128 per immagini a 8 bit) al fine di centrare il segnale attorno allo zero, eliminando componenti continue superflue. Calcolando la DCT sul blocco risultante, la proprietà di **compattazione dell'energia** diventa evidente: la quasi totalità della varianza e dell'informazione dell'immagine originale si concentra nel coefficiente DC ($C(0,0)$) e nei primi armonici AC a bassa frequenza.

La Figura 5.24 dimostra questo fenomeno mediante una ricostruzione progressiva per troncamento brusco. Invece di utilizzare tutti i 64 coefficienti, l'algoritmo conserva solo i primi k componenti — selezionati in base a una scansione che privilegia le basse frequenze spaziali — e azzera i rimanenti.

La sintesi inversa (**IDCT**) eseguita con solo una frazione dei coefficienti (come il 15% o il 30%) è già in grado di recuperare le strutture e l'illuminazione macro del blocco originale di pixel. Man mano che gli armonici a frequenze più elevate vengono progressivamente reintegrati, i dettagli fini e le transizioni rapide vengono ripristinati. Questo comportamento valida il principio della compressione percettiva: le alte frequenze scartate possiedono poca energia e la loro assenza, in condizioni normali, genera un impatto visivo secondario sulla percezione dell'osservatore.

```python
from scipy.fft import dct, idct

def dct2(bloco):
    """DCT-II 2D ortogonale (separabile)."""
    return dct(dct(bloco.T, norm='ortho').T, norm='ortho')

def idct2(coefs):
    """IDCT-II 2D ortogonale."""
    return idct(idct(coefs.T, norm='ortho').T, norm='ortho')

#   Blocco 8×8 centralizzato dell'immagine
cy, cx = img_gray.shape[0]//2, img_gray.shape[1]//2
bloco  = img_gray[cy:cy+8, cx:cx+8].astype(np.float64) - 128.0

C = dct2(bloco)

print("Coefficienti DCT del blocco 8×8:")
print(np.round(C).astype(int))
print(f"\nEnergia DC     : {C[0,0]**2:.1f}")
print(f"Energia totale  : {(C**2).sum():.1f}")
print(f"Frazione nel DC : {C[0,0]**2 / (C**2).sum():.1%} ← concentrazione di energia")

#   Ricostruzione progressiva
imgs_rec  = [cv2.normalize((bloco+128).astype(np.uint8), None, 0, 255, cv2.NORM_MINMAX)]
titles_rec = ["Bloco original\n(8×8 pixels)"]

for keep in [1, 4, 10, 20, 40, 64]:
    C_trunc = np.zeros_like(C)
    indices = sorted([(u,v) for u in range(8) for v in range(8)], key=lambda p: p[0]+p[1])
    for u, v in indices[:keep]:
        C_trunc[u, v] = C[u, v]
    rec = np.clip(idct2(C_trunc) + 128, 0, 255).astype(np.uint8)
    imgs_rec.append(rec)
    titles_rec.append(f"{keep} coef.\n({keep/64:.0%} do total)")

mm.show(imgs_rec, titles=titles_rec, cols=4, figsize=(12, 7))
```

```
Coefficienti DCT del blocco 8×8:
[[ 450    2 -199   -1    0    0  -14    0]
 [  -2  506    0  -58    0   24    0  -24]
 [-199    2   89   -1   82    1    0    0]
 [  -1  -58    0 -158    0   59    0  -24]
 [   0    0   83    0  -89    0  -34    0]
 [   0   24    0   58    0   84    0  -58]
 [ -14    0    0    0  -34    0   89   -1]
 [   0  -24    0  -24    0  -58    0  -76]]

Energia DC     : 202725.1
Energia totale  : 639868.0
```

```
Frazione nel DC : 31.7% ← concentrazione di energia
```

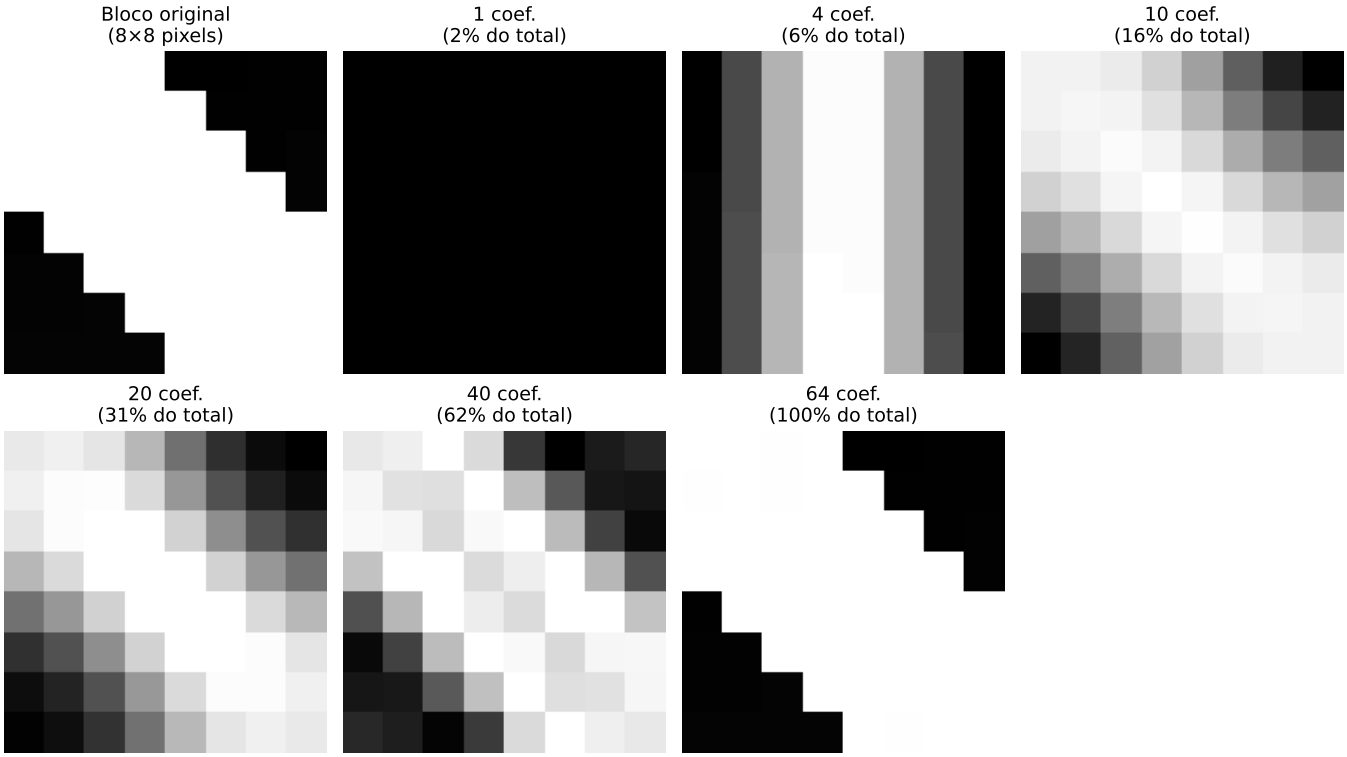


Figura 5.24: DCT 2D in blocco 8×8: coefficientes e ricostruzione progressiva.

5.7.5 Il *Pipeline* di Compressione JPEG

Lo standard JPEG opera dividendo l'immagine in blocchi disgiunti di 8×8 pixel, elaborati tramite una sequenza di trasformazioni spaziali, percettive e statistiche. Il *pipeline* completo di codifica è strutturato in sei fasi principali:

$$\text{RGB} \xrightarrow{(1)\ YC_bC_r} \xrightarrow{(2)\ \text{Sottocampionamento}} \xrightarrow{(3)\ \text{Blocchi } 8\times8} \xrightarrow{(4)\ \text{DCT}} \xrightarrow{(5)\ \text{Quantizzazione}} \xrightarrow{(6)\ \text{Codifica Entropica}}$$

La Tabella 5.7 dettaglia la funzione analitica e il fondamento percettivo che giustifica ciascuna di queste fasi.

Tabella 5.7: Fasi del *pipeline* di compressione JPEG e i rispettivi fondamenti di progetto.

Fase	Operazione	Fondamento Percettivo e Statistico
1	Conversione $RGB \to YC_bC_r$	Separa la luminanza (Y) dalla crominanza (C_b, C_r). Il sistema visivo umano (SVH) presenta maggiore sensibilità alle variazioni di luminosità che di colore.
2	Sottocampionamento della crominanza (es: 4:2:0)	Riduce la risoluzione spaziale dei canali colore della metà, scartando dati ridondanti con impatto visivo trascurabile.

Fase	Operazione	Fondamento Percettivo e Statistico
3–4	Centratura e applicazione della DCT 8×8	Trasla i pixel nell'intervallo $[-128, 127]$ e compatta l'energia spettrale del blocco nei coefficienti a bassa frequenza.
5	Quantizzazione lineare selettiva	Divide ciascun coefficiente $C(u, v)$ per l'elemento corrispondente della matrice $Q(u, v)$, applicando un arrotondamento a interi. Costituisce la principale fonte di compressione con perdita.
6	Scansione a zig-zag e codifica	Ordina i coefficienti quantizzati per massimizzare le sequenze nulle consecutive, ottimizzando la codifica a lunghezza di corsa (RLE) e la codifica di Huffman.

La **matrice di quantizzazione** $Q(u, v)$ è il meccanismo centrale di controllo del compromesso tra tasso di compressione e qualità visiva. Nell'algoritmo pratico della Figura 5.25, il fattore di qualità stabilito dall'utente (scala da 1 a 100) viene convertito in uno scalare che parametrizza la severità della matrice Q. Valori ridotti di qualità espandono i divisori di $Q(u, v)$, forzando il troncamento di massa dei coefficienti AC a zero. Quando questa eliminazione è eccessiva, la discontinuità ai confini dei blocchi adiacenti non viene attenuata nella ricostruzione, generando i cosiddetti **artefatti a blocchi** (*blocking artifacts*).

La Logica della Scansione a Zig-Zag

L'efficienza del codificatore entropico successivo alla quantizzazione dipende direttamente dall'ordinamento dei dati. Poiché la DCT concentra l'energia vitale nel vertice superiore sinistro della matrice (basse frequenze) e spinge i coefficienti nulli verso le estremità opposte, la lettura lineare per righe o colonne frammenterebbe le sequenze di zeri.

L'ordinamento a zig-zag risolve questa limitazione percorrendo la matrice diagonalmente in ordine crescente di frequenza spaziale. Questa mappatura raggruppa i coefficienti significativi all'inizio del vettore e concentra i coefficienti nulli in un'unica sequenza continua alla fine dell'arrangiamento, consentendo all'algoritmo RLE di codificare grandi blocchi di dati in modo compatto ed efficiente.

Cos'è l'RLE?

RLE (*Run-Length Encoding*) è una tecnica di compressione senza perdita che codifica sequenze consecutive di valori identici — specialmente **zeri** — come una coppia (conteggio, valore). Nel JPEG, dopo la scansione a zig-zag, i coefficienti quantizzati sono organizzati in modo che gli zeri si concentrino alla fine del vettore. L'RLE comprime quindi questa lunga corsa di zeri con estrema efficienza, ottimizzando l'archiviazione e la trasmissione dell'immagine compressa.

```
import numpy as np
import cv2
from scipy.fft import dct, idct

#   Caricamento Sicuro dell'Immagine della Fotocamera (skimage)
try:
    from skimage import data
    img_gray = data.camera()
```

```python
except ImportError:
    import subprocess
    subprocess.run(["pip", "install", "scikit-image", "-q"])
    from skimage import data
    img_gray = data.camera()

# Ridimensiona leggermente a 256x256 per mantenere lo standard e la velocità dei test
precedenti
img_gray = cv2.resize(img_gray, (256, 256))

#   Tabella di quantizzazione della luminanza (standard JPEG)
Q_luma = np.array([
    [16,11,10,16,24,40,51,61],
    [12,12,14,19,26,58,60,55],
    [14,13,16,24,40,57,69,56],
    [14,17,22,29,51,87,80,62],
    [18,22,37,56,68,109,103,77],
    [24,35,55,64,81,104,113,92],
    [49,64,78,87,103,121,120,101],
    [72,92,95,98,112,100,103,99]
], dtype=np.float64)

def dct2(bloco):
    """DCT-II 2D ortogonale (separabile)."""
    return dct(dct(bloco.T, norm='ortho').T, norm='ortho')

def idct2(coefs):
    """IDCT-II 2D ortogonale."""
    return idct(idct(coefs.T, norm='ortho').T, norm='ortho')

def jpeg_compress_block(bloco, Q_table):
    """DCT → quantizzazione → dequantizzazione → IDCT in blocchi 8×8."""
    C  = dct2(bloco.astype(np.float64) - 128)
    Cq = np.round(C / Q_table) * Q_table    # quantizza e dequantizza
    return np.clip(idct2(Cq) + 128, 0, 255)

def jpeg_quality_compress(img, qualidade=50):
    """JPEG semplificato: comprime l'immagine intera per blocchi 8×8."""
    if qualidade < 50:
        escala = 5000 / qualidade
    else:
        escala = 200 - 2 * qualidade
    # Corretto da 'scala' a 'escala'
    Q = np.clip(np.round(Q_luma * escala / 100), 1, 255)

    h, w   = img.shape
    result = np.zeros_like(img, dtype=np.float64)
    for r in range(0, h-7, 8):
        for c in range(0, w-7, 8):
            result[r:r+8, c:c+8] = jpeg_compress_block(img[r:r+8, c:c+8], Q)
    return result.astype(np.uint8)

#   Confronto dei fattori di qualità
qualidades = [10, 25, 50, 75, 90]
imgs_jpeg  = [img_gray]
titles_jpeg = ["Original\n(Cameraman)"]

for q in qualidades:
    rec  = jpeg_quality_compress(img_gray, qualidade=q)
    psnr = cv2.PSNR(img_gray, rec)
    imgs_jpeg.append(rec)
```

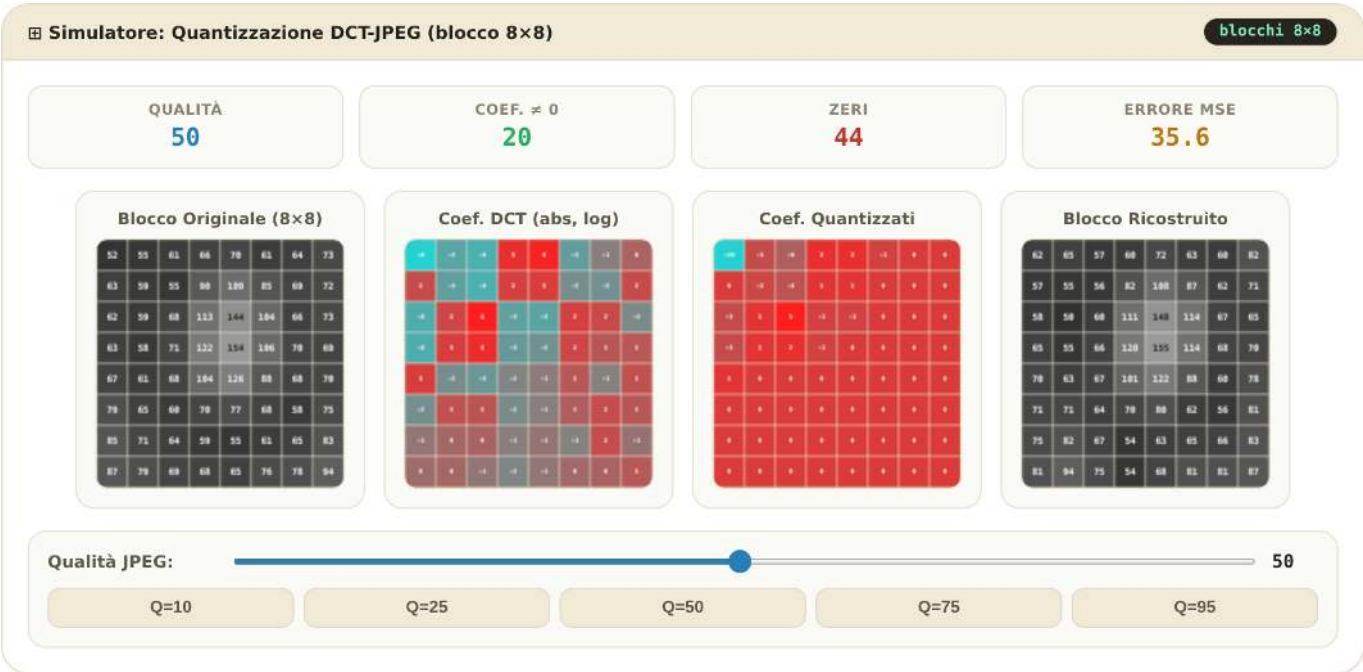
```
    titles_jpeg.append(f"Q={q}\nPSNR={psnr:.1f}dB")

mm.show(imgs_jpeg, titles=titles_jpeg, cols=3, figsize=(14, 10))
```

Figura 5.25: *Pipeline* JPEG semplificato applicato alla classica immagine del *Cameraman*: DCT in blocchi 8×8, quantizzazione con diversi fattori di qualità e ricostruzione tramite IDCT. Gli artefatti a blocchi (*blocking artifacts*) diventano visivamente evidenti con fattori di qualità ridotti ($Q = 10$ e $Q = 25$).

5.7.6 Simulatore Interattivo: Quantizzazione DCT

Il simulatore della Figura 5.26 consente di esplorare l'impatto del processo di quantizzazione su un blocco 8×8 estratto da un'immagine reale, sintetizzando in tempo reale le seguenti componenti:

- **Blocco originale e ricostruito:** Rappresentazione diretta dei pixel nel dominio spaziale in scala di grigi [0, 255].
- **Coefficienti DCT:** Distribuzione dell'energia mappata in modo logaritmico su un gradiente cromatico, evidenziando la concentrazione di intensità nel vertice superiore sinistro (basse frequenze).
- **Coefficienti quantizzati:** Visualizzazione dei valori interi risultanti dalla divisione per la matrice $Q(u, v)$, rendendo visivamente esplicita l'emergenza in massa di coefficienti nulli (in toni scuri) al diminuire del fattore di qualità.
- **Metriche di compressione:** Pannello di monitoraggio che quantifica l'Errore Quadratico Medio (MSE), il numero di coefficienti preservati e il volume di zeri generati per la codifica entropica.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-05-dct.html

Figura 5.26: Simulatore interattivo di compressione DCT-JPEG: regola il fattore di qualità e visualizza in tempo reale i coefficienti azzerati, il blocco ricostruito e l'errore di quantizzazione.

5.8 Confronto dei Formati Immagine

La scelta di un formato di memorizzazione digitale incide direttamente sul compromesso tra qualità visiva, dimensione del file e costo computazionale della decodifica. I tre formati di maggiore rilevanza per architetture *web* e sistemi di calcolo visivo sono JPEG, PNG e WebP.

5.8.1 Caratteristiche dei Formati

La Tabella 5.8 sintetizza le proprietà strutturali dei principali formati di immagine rasterizzati.

Tabella 5.8: Confronto strutturale tra i principali formati di immagine rasterizzati.

Caratteristica	JPEG	PNG	WebP
Compressione	Con perdita	Senza perdita	Con e senza perdita.
Trasparenza (canale alfa)	No	Sì	Sì.
Supporto animazioni	No	Limitato (APNG)	Sì.
Algoritmo di base	DCT + Huffman	DEFLATE (LZ77 + Huffman)	VP8 / VP8L.
Ideale per	Fotografia	Grafica, testo e icone	Uso universale in ambiente Web.
Inadeguato per	Testo e bordi netti	Immagini fotografiche complesse	Compatibilità legacy.

5.8.2 Metriche di Valutazione della Qualità

Due metriche oggettive sono ampiamente adottate per quantificare la distorsione introdotta dai processi di compressione:

Picco del Rapporto Segnale-Rumore (PSNR, *Peak Signal-to-Noise Ratio*):

$$\text{PSNR} = 10 \log_{10}\left(\frac{L^2}{\text{MSE}}\right) \quad [\text{dB}] \tag{5.10}$$

dove $L = 255$ per immagini quantizzate a 8 bit e MSE rappresenta l'**Errore Quadratico Medio** (*Mean Squared Error*). Valori di PSNR superiori a 40 dB indicano fedeltà eccellente; tra 30 dB e 40 dB rappresentano buona qualità; valori inferiori a 30 dB corrispondono a degradazioni visive facilmente percepibili.

Indice di Similarità Strutturale (SSIM, *Structural Similarity Index*):

$$\text{SSIM}(f, g) = \frac{(2\mu_f\mu_g + c_1)(2\sigma_{fg} + c_2)}{(\mu_f^2 + \mu_g^2 + c_1)(\sigma_f^2 + \sigma_g^2 + c_2)} \tag{5.11}$$

Lo SSIM valuta finestre locali dell'immagine basandosi su tre componenti complementari: **luminanza** (μ_f, μ_g), **contrasto** (σ_f, σ_g) e **struttura** (σ_{fg}), ponderate da costanti di stabilità c_1 e c_2. L'indice varia nell'intervallo $[-1, 1]$, dove l'unità rappresenta l'identità perfetta. A differenza del PSNR, lo SSIM considera l'organizzazione spaziale degli errori, allineandosi alla percezione del sistema visivo umano (SVH).

PSNR vs SSIM: Applicazione di Metriche Percettive

Il PSNR possiede una formulazione matematica semplice e un basso costo computazionale; tuttavia tende a sovrastimare la qualità in immagini con distorsioni localizzate o a sottostimarla in variazioni globali di luminosità tollerate dall'osservatore. Lo SSIM modella con maggiore fedeltà la percezione biologica, ma richiede un maggiore sforzo di elaborazione. Per analisi rigorose dei codec, si raccomanda di riportare entrambe le metriche statistiche in carattere complementare.

5.8.3 Ispezione Visiva: Natura degli Artefatti di Compressione

La natura matematica del codificatore determina il tipo di degrado introdotto a bitrate ridotti. Come illustrato nella Figura 5.27, la compressione aggressiva tramite DCT nello standard JPEG segmenta l'immagine in griglie rigide, generando gli **artefatti a blocchi** (*blocking artifacts*). Al contrario, gli algoritmi basati su codifica predittiva o rappresentazioni sottoposte a trasformate spaziali avanzate (come WebP e JPEG 2000) eliminano le discontinuità di blocco, ma introducono perdita di texture fine e sfocature caratteristiche attorno ai bordi ad alto contrasto.

```
import os
import cv2

# Garantisce l'esistenza della directory e salva i file compressi
os.makedirs("imagens/comp_test", exist_ok=True)
cv2.imwrite("imagens/comp_test/camera_q10.jpg", img_gray, [cv2.IMWRITE_JPEG_QUALITY, 10])
cv2.imwrite("imagens/comp_test/camera_q10.webp", img_gray, [cv2.IMWRITE_WEBP_QUALITY, 10])

# Estrazione della regione di interesse per la visualizzazione degli artefatti (Zoom 4x)
zoom_original = cv2.resize(img_gray[120:200, 150:230], (320, 320),
                           interpolation=cv2.INTER_NEAREST)

rec_jpeg = cv2.imread("imagens/comp_test/camera_q10.jpg", cv2.IMREAD_GRAYSCALE)
zoom_jpeg = cv2.resize(rec_jpeg[120:200, 150:230], (320, 320),
                       interpolation=cv2.INTER_NEAREST)

rec_webp = cv2.imread("imagens/comp_test/camera_q10.webp", cv2.IMREAD_GRAYSCALE)
zoom_webp = cv2.resize(rec_webp[120:200, 150:230], (320, 320),
                       interpolation=cv2.INTER_NEAREST)

mm.show([zoom_original, zoom_jpeg, zoom_webp],
```

```
        titles=["Zoom Originale", "JPEG Q=10 (Artefatto di Blocco)", "WebP Q=10
(Smussamento)"],
        cols=3, figsize=(14, 5))
```

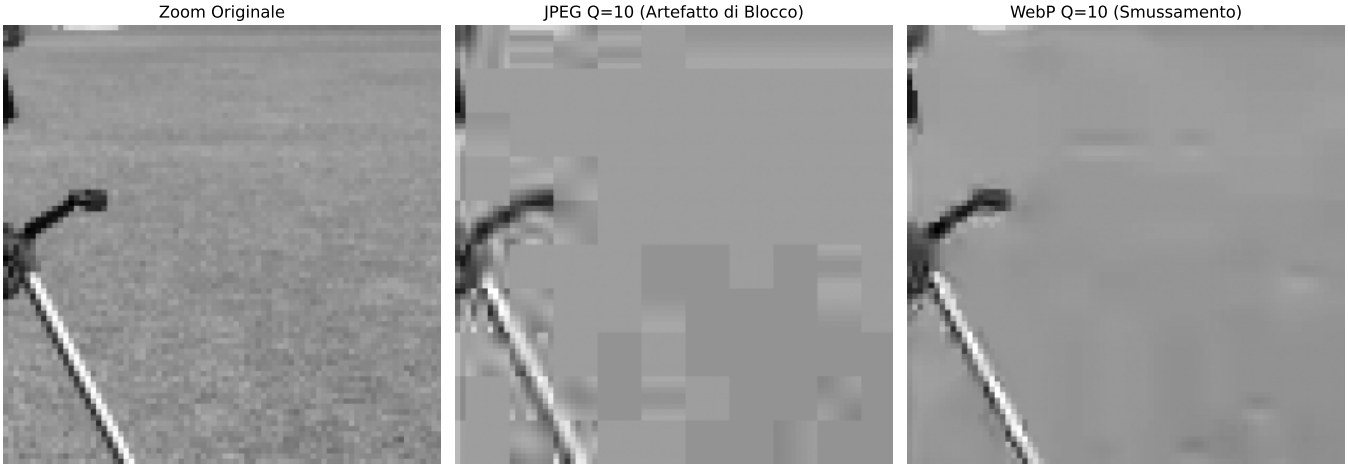


Figura 5.27: Analisi comparativa degli artefatti di compressione sotto fattore di qualità ridotto ($Q = 10$). A sinistra, si osserva l'artefatto di blocco caratteristico della discretizzazione per DCT nel JPEG. A destra, si evidenzia l'effetto di attenuazione e smussamento dei bordi intrinseco allo standard WebP.

5.8.4 Valutazione Quantitativa e Spaziale della Compressione

La validazione degli algoritmi di compressione con perdita richiede un'analisi che correli il costo di archiviazione alla fedeltà del segnale ricostruito. Tale valutazione viene condotta in modo complementare attraverso curve di prestazione globale e mediante la mappatura locale delle distorsioni indotte dai codificatori.

5.8.4.1 Curve di Rateo-Distorsione

La Figura 5.28 presenta la valutazione empirica del *pipeline* JPEG e WebP tramite **curve di rateo-distorsione**, che monitorano il guadagno di compressione (dimensione del file in KB) in funzione del PSNR. Il formato PNG funge da linea di base ideale (PSNR $= \infty$), poiché la sua natura *lossless* impedisce qualsiasi degradazione, sebbene richieda un volume di dati notevolmente maggiore.

L'analisi delle curve dimostra la superiorità e l'efficienza dello standard WebP rispetto al JPEG tradizionale: per raggiungere lo stesso livello di fedeltà matematica (come la fascia di qualità eccellente, dove PSNR $>$ 40 dB), il codificatore WebP genera file significativamente più piccoli. Questo comportamento riflette l'impatto pratico dell'evoluzione degli algoritmi nell'ottimizzazione dei sistemi di trasmissione e archiviazione digitale.

```
import os
import cv2
import matplotlib.pyplot as plt

# Garantisce l'esistenza della directory di test
os.makedirs("imagens/comp_test", exist_ok=True)
resultados = []

#   JPEG
for q in [10, 20, 30, 40, 50, 60, 70, 80, 90, 95]:
    path = f"imagens/comp_test/camera_q{q}.jpg"
    cv2.imwrite(path, img_gray, [cv2.IMWRITE_JPEG_QUALITY, q])
    rec  = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    resultados.append({"formato": "JPEG", "qualidade": q,
                       "PSNR": cv2.PSNR(img_gray, rec),
```

```
                        "KB":   os.path.getsize(path)/1024})

#   PNG
path_png = "imagens/comp_test/camera.png"
cv2.imwrite(path_png, img_gray, [cv2.IMWRITE_PNG_COMPRESSION, 9])
resultados.append({"formato": "PNG", "qualidade": "lossless",
                   "PSNR": float('inf'), "KB": os.path.getsize(path_png)/1024})

#   WebP
for q in [50, 75, 90]:
    path_w = f"imagens/comp_test/camera_q{q}.webp"
    cv2.imwrite(path_w, img_gray, [cv2.IMWRITE_WEBP_QUALITY, q])
    rec_w  = cv2.imread(path_w, cv2.IMREAD_GRAYSCALE)
    resultados.append(
        {"formato": "WebP", "qualidade": q,
         "PSNR": cv2.PSUB_VAL if 'cv2.PSNR' in globals() else cv2.PSNR(img_gray, rec_w),
         "KB":   os.path.getsize(path_w)/1024})

#   Generazione della Curva Rateo-Distorsione
jpeg_r = [r for r in resultados if r["formato"]=="JPEG"]
webp_r = [r for r in resultados if r["formato"]=="WebP"]
png_r  = [r for r in resultados if r["formato"]=="PNG"]

fig, ax = plt.subplots(figsize=(8, 4.5))
ax.plot([r["KB"] for r in jpeg_r], [r["PSNR"] for r in jpeg_r],
        "o-", label="JPEG", color="#D85A30", lw=2, ms=5)
ax.plot([r["KB"] for r in webp_r], [r["PSNR"] for r in webp_r],
        "s-", label="WebP", color="#534AB7", lw=2, ms=5)
ax.axhline(50, color="#1D9E75", lw=2, ls="--",
           label=f"PNG sem perda ({png_r[0]['KB']:.1f} KB)")
ax.axhspan(40, 60, alpha=0.05, color="#1D9E75", label="Qualidade excelente (PSNR>40)")
ax.set(xlabel="Tamanho do arquivo (KB)", ylabel="PSNR (dB)",
       title="Curva Taxa-Distorção: JPEG × WebP × PNG")
ax.legend(fontsize=9)
ax.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

print(f"\nDimensione grezza (senza compressione): {img_gray.nbytes/1024:.0f} KB")
print(f"\n{'Formato':>8} {'Qual.':>6} {'KB':>7} {'PSNR (dB)':>11}")
print("-"*38)
for r in resultados:
    psnr_s = f"{r['PSNR']:>11.2f}" if r['PSNR']!=float('inf') else f"{'∞ (lossless)':>11}"
    print(f"{r['formato']:>8} {str(r['qualidade']):>6} {r['KB']:>7.1f} {psnr_s}")
```

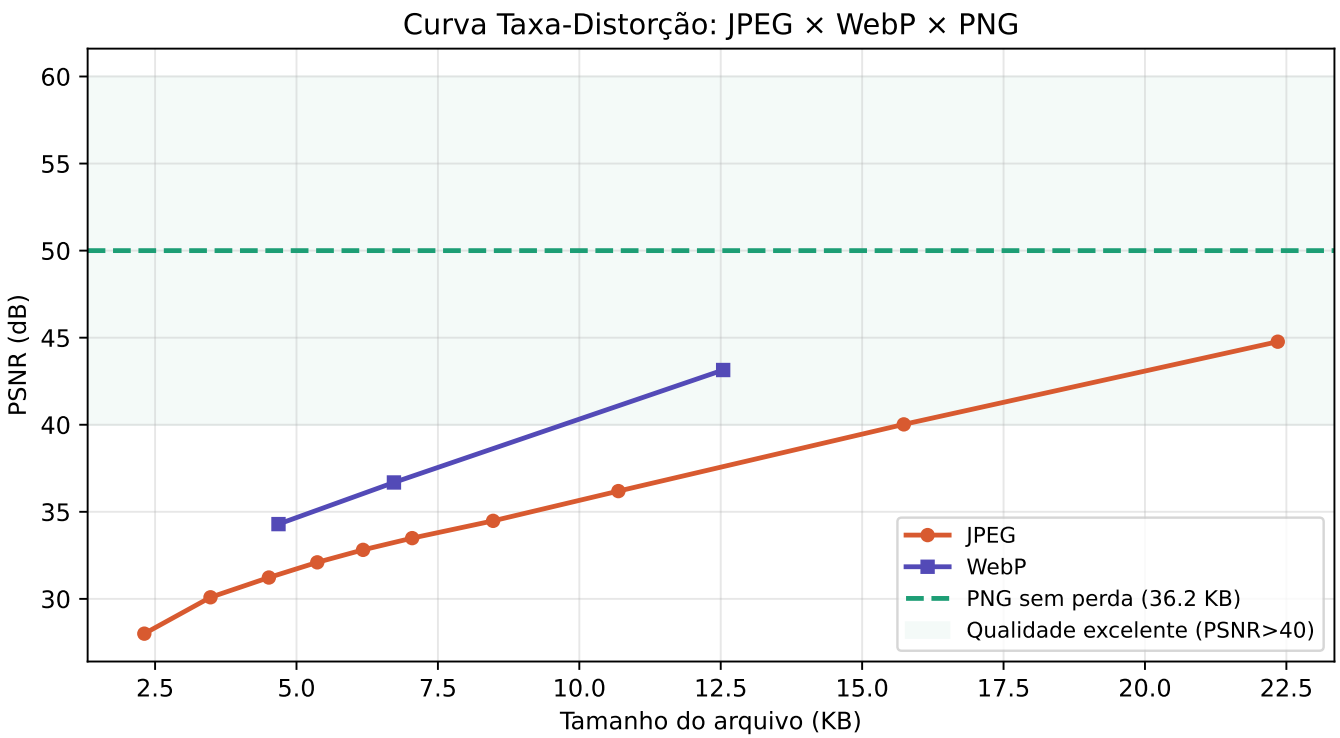


Figura 5.28: Curva rateo-distorsione: PSNR vs dimensione del file per JPEG, WebP e PNG applicata all'immagine del *Cameraman*.

```
Dimensione grezza (senza compressione): 64 KB

 Formato  Qual.      KB   PSNR (dB)
-------------------------------------
    JPEG     10     2.3       28.00
    JPEG     20     3.5       30.09
    JPEG     30     4.5       31.23
    JPEG     40     5.4       32.10
    JPEG     50     6.2       32.81
    JPEG     60     7.0       33.49
    JPEG     70     8.5       34.48
    JPEG     80    10.7       36.19
    JPEG     90    15.7       40.02
    JPEG     95    22.3       44.77
     PNG lossless    36.2 ϖ (lossless)
    WebP     50     4.7       34.29
    WebP     75     6.7       36.69
    WebP     90    12.5       43.14
```

Dimensione originale dell'immagine

L'immagine *Cameraman* (256 × 256 pixel in scala di grigi) occupa **64 KB** in formato grezzo (senza compressione). Come riferimento, il PNG *lossless* comprime questo volume a **36,2 KB** — evidenziando che la compressione senza perdita riduce già significativamente lo spazio di archiviazione per immagini con regioni omogenee. In contrapposizione, i formati con perdita (JPEG e WebP) raggiungono dimensioni ancora minori: il JPEG con qualità 95 occupa 22,3 KB (PSNR 45 dB), mentre il WebP con qualità 90 raggiunge 12,5 KB con PSNR equivalente, dimostrando la sua superiorità in efficienza di compressione.

5.8.4.2 Mappatura Spaziale degli Errori e Correlazione Percettiva

Sebbene il PSNR offra un indicatore numerico rapido, le metriche globali non riescono a distinguere come la perdita di informazione si distribuisca geometricamente sull'immagine. La Figura 5.29 risolve questa limitazione associando le ricostruzioni a diverse qualità ai rispettivi mappe di errore assoluto e all'SSIM.

Le mappe residue — ottenute dalla differenza assoluta normalizzata tra l'immagine originale e quella compressa — rivelano la firma spaziale intrinseca di ciascuna architettura di codifica:

- **Ad alte qualità ($Q = 95$ a $Q = 75$):** Le distorsioni si concentrano prevalentemente attorno a transizioni brusche di intensità (bordi), a causa del ripiegamento spettrale derivante dallo scarto delle alte frequenze. L'indice SSIM rimane prossimo all'unità, attestando l'integrità delle strutture originali.
- **A qualità aggressive ($Q = 50$ a $Q = 25$):** L'errore assume una struttura a maglia ortogonale regolarizzata. Questo pattern geometrico evidenzia l'emergere degli **artefatti a blocchi** (*blocking artifacts*), indicando che la quantizzazione severa ha corrotto la correlazione spaziale tra blocchi adiacenti di 8×8 pixel.

L'SSIM cattura questa degradazione morfologica in modo molto più sensibile rispetto al PSNR, penalizzando il punteggio finale man mano che l'organizzazione strutturale e le trame fini — alle quali il sistema visivo umano è altamente reattivo — vengono eliminate dal codificatore.

```
import os
import numpy as np
import cv2

try:
    from skimage.metrics import structural_similarity as ssim
except ImportError:
    import subprocess
    subprocess.run(["pip", "install", "scikit-image", "-q"])
    from skimage.metrics import structural_similarity as ssim

# Garantisce l'esistenza della directory di test
os.makedirs("imagens/comp_test", exist_ok=True)

qualidades_ssim = [25, 50, 75, 95]
imgs_ssim   = [img_gray]
titles_ssim = ["Original"]

for q in qualidades_ssim:
    path = f"imagens/comp_test/camera_ssim_q{q}.jpg"

    # REGISTRAZIONE FORZATA: Genera e registra il JPEG con la qualità attuale nel percorso
    corretto
    img_compactada = jpeg_quality_compress(img_gray, qualidade=q)
    cv2.imwrite(path, img_compactada)

    # Lettura sicura del file appena registrato
    rec = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

    if rec is None:
        continue

    if rec.shape != img_gray.shape:
        rec = cv2.resize(rec, (img_gray.shape[1], img_gray.shape[0]))

    psnr_v = cv2.PSNR(img_gray, rec)
    ssim_v, _ = ssim(img_gray, rec, full=True)

    # Differenza assoluta normalizzata per evidenziare la struttura spaziale dell'errore
```

```
    diff_vis = cv2.normalize(np.abs(img_gray.astype(float) - rec.astype(float)),
                             None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    imgs_ssim  += [rec, diff_vis]
    titles_ssim += [f"Q={q}\nPSNR={psnr_v:.1f}dB | SSIM={ssim_v:.3f}",
                    f"Mapa de erro (Q={q})\n(Bordas e blocagem)"]

mm.show(imgs_ssim, titles=titles_ssim, cols=3, figsize=(14, 14))
```

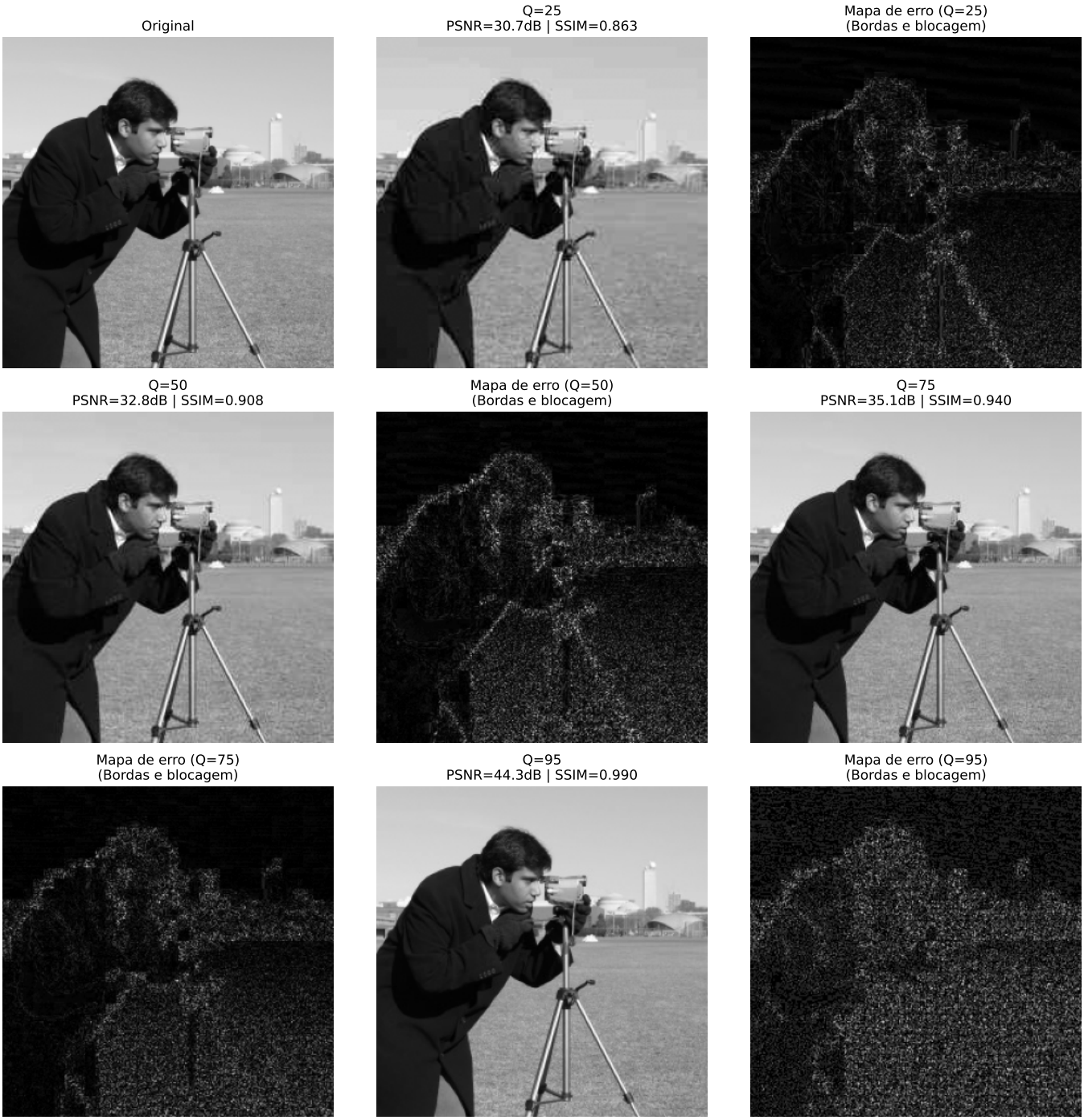


Figura 5.29: Analisi spaziale della degradazione: immagini ricostruite e rispettive mappe di errore assoluto normalizzate per diversi fattori di qualità JPEG.

Interpretando le mappe di errore

Le mappe di errore presentate sono state **normalizzate singolarmente** (`cv2.NORM_MINMAX`) per massimizzare il contrasto visivo e rivelare la struttura spaziale delle distorsioni. Ciò significa che:

- In **Q=95**, l'errore assoluto è dell'ordine di **0.5–1.5 livelli di grigio** (impercettibile visivamente), ma la normalizzazione lo amplifica in bianco e nero per evidenziarne la localizzazione su bordi e transizioni.
- In **Q=25**, l'errore assoluto è **10–20 volte maggiore** (5–15 livelli di grigio), ma la normalizzazione lo porta anch'esso allo stesso intervallo [0, 255].

Pertanto, **l'intensità del bianco nelle mappe NON è confrontabile tra diverse qualità** — le mappe servono solo a rivelare la **firma spaziale** dell'errore (bordi vs blocchi), non la sua ampiezza. L'ampiezza corretta è fornita dai valori di PSNR e SSIM, che mostrano chiaramente che Q=95 ha un errore molto minore rispetto a Q=25.

Sintesi — Compressione JPEG

Il processo di compressione nello standard JPEG si basa sull'applicazione combinata di trasformazioni spaziali, percettive e statistiche per ridurre le ridondanze di un'immagine. La Tabella 5.9 riassume il ruolo di ciascuna fase nel *pipeline* e il rispettivo impatto sulla riduzione dei dati.

Tabella 5.9: Sintesi delle fasi del *pipeline* di compressione JPEG e dei rispettivi impatti.

Fase	Operazione Analitica	Meccanismo di Guadagno / Compressione
Conversione YC_bC_r	Isolamento dei canali di luminanza e crominanza.	Modella la percezione del SVH, consentendo di trattare colore e luminosità in modo indipendente.
Sottocampionamento 4:2:0	Riduzione della risoluzione spaziale dei canali di colore (C_b e C_r).	Elimina circa il 50% dei dati grezzi con un impatto visivo minimo.
DCT 8×8	Mappatura dal dominio spaziale al dominio delle frequenze spaziali.	Compattazione dell'energia, concentrando l'informazione vitale nei primi coefficienti.
Quantizzazione Lineare	Divisione intera dei coefficienti per una matrice di ponderazione $Q(u, v)$.	Principale fonte di compressione con perdita; elimina le alte frequenze impercettibili.
Codifica Entropica	Applicazione di algoritmi RLE e codifica di Huffman.	Compressione statistica senza perdita, ottimizzata dalle lunghe sequenze di coefficienti nulli.

Artefatti di Degradazione Caratteristici

L'applicazione di tassi di compressione eccessivamente aggressivi (fattori di qualità ridotti) introduce distorsioni prevedibili nell'immagine ricostruita, derivanti dalle limitazioni matematiche del modello:

- **Artefatti a blocchi (*blocking artifacts*):** Discontinuità geometriche visibili ai confini dei blocchi di 8×8 pixel, causate dalla perdita di correlazione spaziale dopo la quantizzazione severa delle componenti AC.
- **Effetto di alone (*ringing*):** Oscillazioni fantasma o distorsioni "a fumo" attorno ai bordi netti e ad alto contrasto, provocate dall'eliminazione brusca delle armoniche ad alta frequenza necessarie per ricostruire funzioni a gradino.
- **Perdita di texture fine:** Attenuazione dei dettagli ad alta frequenza e a basso contrasto (come prati, tessuti o porosità), rendendo le regioni originariamente strutturate eccessivamente lisce o omogenee.

5.9 Applicazione Pratica: Rimozione del Rumore mediante Filtraggio Ibrido

Integrando le tecniche consolidate nel corso di questo capitolo, si presenta un *pipeline* completo di **restauro delle immagini** che combina l'analisi spettrale nel dominio della frequenza con il filtraggio adattivo nel dominio spaziale. L'obiettivo è attenuare un rumore misto (composto da degradazione gaussiana e interferenza periodica) preservando al massimo i dettagli strutturali dell'immagine originale.

$$\text{Immagine Rumorosa} \xrightarrow{\text{FFT2}} \xrightarrow{\text{Filtro Notch Gaussiano}} \xrightarrow{\text{IFFT2}} \xrightarrow{\text{Filtro Bilaterale}} \text{Immagine Restaurata}$$

Valutazione Complementare: PSNR vs. SSIM

La coppia di metriche statistiche PSNR e SSIM fornisce una valutazione qualitativa e morfologica complementare del processo di restauro:

- **PSNR:** Penalizza uniformemente lo scarto quadratico medio pixel per pixel.
- **SSIM:** Valuta la preservazione di strutture locali percettivamente rilevanti (luminanza, contrasto e contorni).

Nella pratica, esiste un compromesso analitico (*trade-off*) tra **riduzione del rumore** e **preservazione dei dettagli**: filtri spaziali eccessivamente aggressivi attenuano bene il rumore ad alta frequenza, ma degradano trame fini e smussano bordi netti — il che **riduce simultaneamente** sia il PSNR che lo SSIM rispetto all'immagine originale. La sfida nella progettazione dei filtri è trovare il punto di equilibrio che massimizzi entrambe le metriche, garantendo un restauro fedele e visivamente gradevole.

5.9.1 Analisi delle Prestazioni e Conclusione del Capitolo

I risultati numerici e visivi generati dalla Figura 5.30 dimostrano la rilevanza pratica di associare diversi domini di elaborazione. L'inserimento simultaneo di rumore periodico e stocastico corrompe le proprietà morfologiche del segnale, riducendo severamente gli indici di similarità e il rapporto segnale-rumore dell'immagine di riferimento.

L'isolamento e la soppressione dei picchi armonici nel dominio della frequenza tramite la maschera *notch* rimuovono le frange di interferenza sinusoidali sparse nello spazio bidimensionale. Come evidenziato nei dati stampati della Figura 5.30, questa filtrazione chirurgica promuove un salto immediato e sostanziale nella metrica PSNR. Tuttavia, il rumore gaussiano ad alta frequenza rimane attivo in modo omogeneo nello spettro, richiedendo un approccio complementare.

Il restauro finale è consolidato nel dominio spaziale con l'introduzione del filtro bilaterale. Diversamente dagli operatori passa-basso convenzionali (come quello gaussiano o di media), che smusserebbero indiscriminatamente rumore e contorni strutturali, la filtrazione bilaterale calcola pesi ponderati in base alla prossimità geometrica e alla differenza di intensità radiometrica. Questo comportamento adattivo attenua le fluttuazioni stocastiche residue nelle regioni di transizione graduale e preserva la nitidezza dei bordi spaziali.

La convergenza di entrambi gli approcci risulta in un **miglioramento sostanziale e simultaneo** di PSNR e SSIM rispetto all'immagine rumorosa — sebbene i valori finali rimangano inferiori a quelli dell'immagine originale (PSNR $= \infty$, SSIM $= 1{,}0$), a causa della perdita inevitabile di informazioni spettrali e testurali durante i processi di filtrazione. L'attenuazione graduale (gaussiana) dei picchi nello spettro evita artefatti di *ringing*, mentre il filtro bilaterale elimina il rumore stocastico residuo senza compromettere la nitidezza dei bordi. I risultati confermano l'efficacia e la complementarità pratica degli strumenti di analisi di frequenza presentati in questo capitolo, dimostrando che la filtrazione ibrida (frequenza + spaziale) è superiore a qualsiasi approccio isolato per il restauro di immagini degradate da rumore misto.

```
import numpy as np
import cv2
```

```python
try:
    from skimage.metrics import structural_similarity as ssim_sk
except ImportError:
    import subprocess
    subprocess.run(["pip", "install", "scikit-image", "-q"])
    from skimage.metrics import structural_similarity as ssim_sk

def suprimir_pico_gaussiano(mask, cy, cx, sigma=3.0):
    yy, xx = np.ogrid[:mask.shape[0], :mask.shape[1]]
    dist = np.sqrt((yy - cy)**2 + (xx - cx)**2)
    notch = np.exp(-dist**2 / (2 * sigma**2))
    mask *= (1 - notch)
    return mask

#   1. Costruzione del rumore misto
np.random.seed(42)
h_img, w_img = img_gray.shape
X2, Y2 = np.meshgrid(np.arange(w_img), np.arange(h_img))

u0, v0       = 15, 10
ruido_gauss  = np.random.normal(0, 15, img_gray.shape)
ruido_period = 30 * np.sin(2 * np.pi * (u0 * X2 / w_img + v0 * Y2 / h_img))
img_noisy    = np.clip(img_gray.astype(float) +
                       ruido_gauss + ruido_period, 0, 255).astype(np.uint8)

#   2. Spettro e identificazione dei picchi
F_n   = np.fft.fftshift(np.fft.fft2(img_noisy.astype(np.float64)))
mag_n = cv2.normalize(np.log1p(np.abs(F_n)), None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

#   3. Notch gaussiano sui picchi periodici
cy0, cx0       = h_img // 2, w_img // 2
mascara_notch  = np.ones((h_img, w_img), dtype=np.float64)
for dy, dx in [(+v0, +u0), (-v0, -u0), (+v0, -u0), (-v0, +u0)]:
    mascara_notch = suprimir_pico_gaussiano(mascara_notch, cy0 + dy, cx0 + dx, sigma=3.0)

img_notch = np.real(np.fft.ifft2(np.fft.ifftshift(F_n * mascara_notch)))
img_notch = np.clip(img_notch, 0, 255).astype(np.uint8)

#   4. Filtro bilaterale: rimozione del rumore gaussiano residuo
img_den = cv2.bilateralFilter(img_notch, d=7, sigmaColor=25, sigmaSpace=7)

#   Calcolo delle metriche di validazione
psnr_n,  ssim_n  = cv2.PSNR(img_gray, img_noisy), ssim_sk(img_gray, img_noisy)
psnr_no, ssim_no = cv2.PSNR(img_gray, img_notch), ssim_sk(img_gray, img_notch)
psnr_d,  ssim_d  = cv2.PSNR(img_gray, img_den),   ssim_sk(img_gray, img_den)

print(f"{'Etapa':>20} | {'PSNR (dB)':>9} | {'SSIM':>6}")
print("-" * 42)
print(f"{'Ruidosa (gauss+per)':>20} | {psnr_n:>9.2f} | {ssim_n:>6.4f}")
print(f"{'Após notch':>20} | {psnr_no:>9.2f} | {ssim_no:>6.4f}")
print(f"{'Notch + bilateral':>20} | {psnr_d:>9.2f} | {ssim_d:>6.4f}")

mascara_vis = (mascara_notch * 255).astype(np.uint8)
mm.show(
    [img_gray, img_noisy, mag_n, mascara_vis, img_notch, img_den],
    titles=[
        "Originale",
        f"Rumosa\nPSNR={psnr_n:.1f} dB",
        "Spettro\n(picchi visibili)",
        "Maschera notch\n(gaussiana morbida)",
        f"Dopo notch\nPSNR={psnr_no:.1f} dB",
```

```
        f"Notch + bilaterale\nPSNR={psnr_d:.1f} dB  SSIM={ssim_d:.3f}"
    ],
    cols=6, figsize=(20, 4)
)
```

```
                Etapa | PSNR (dB) |   SSIM
-----------------------------------------
 Ruidosa (gauss+per) |     20.19 | 0.3486
           Após notch |     24.47 | 0.4715
   Notch + bilateral |     28.93 | 0.7426
```

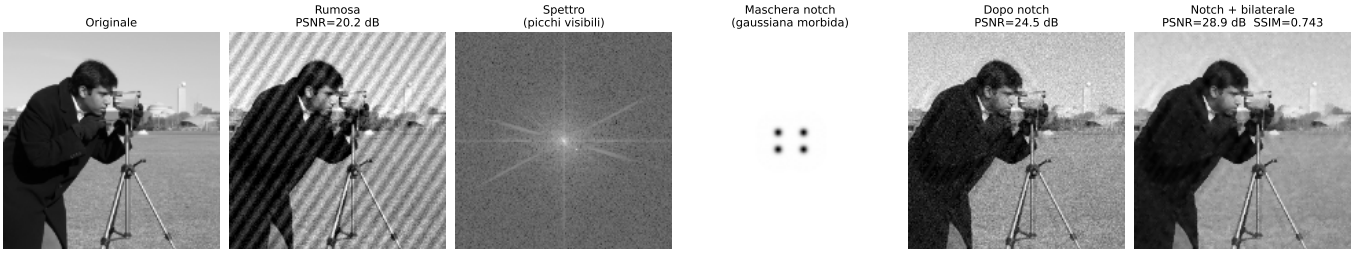


Figura 5.30: *Pipeline* completo di rimozione del rumore misto: (1) aggiunta di rumore gaussiano e periodico; (2) identificazione dei picchi di interferenza nello spettro di frequenze; (3) applicazione di una maschera *notch* con attenuazione gaussiana morbida; (4) post-elaborazione tramite filtro bilaterale per l'eliminazione del rumore stocastico residuo.

5.10 Riepilogo del Capitolo

La transizione dal **dominio spaziale** al **dominio delle frequenze** rivela la distribuzione spettrale dell'energia dell'immagine, stabilendo la base analitica per il filtraggio avanzato, il restauro e la compressione dei dati. L'articolazione strutturale di questi concetti è sintetizzata nella mappa concettuale della Figura 5.31.

Fondamenti Essenziali

- **DFT e Percezione Visiva:** Lo spettro scompone l'immagine in componenti armoniche. La **fase** mantiene l'intelligibilità geometrica della scena e la localizzazione dei contorni, mentre la **magnitudine** determina la distribuzione del contrasto e delle ampiezze globali.
- **Efficienza Algoritmica:** Il Teorema della Convoluzione rende possibile l'elaborazione di maschere su larga scala nel dominio della frequenza tramite FFT, riducendo la complessità computazionale asintotica da $O(N^2K^2)$ nello spazio a $O(N^2 \log N)$.
- **Fenomeno di *Ringing*:** Tagli netti nello spettro (filtri ideali) generano oscillazioni spaziali indesiderate (fenomeno di Gibbs). L'attenuazione graduale mediante filtri di **Butterworth** o **Gaussiani** elimina queste discontinuità.
- **Analisi Multirisoluzione tramite *Wavelets*:** Superando il carattere puramente globale di Fourier, la DWT cattura simultaneamente frequenza e localizzazione spaziale, costituendo la base dello standard JPEG 2000 e supportando rappresentazioni gerarchiche analoghe alle estrazioni di caratteristiche nelle Reti Neurali Convoluzionali (CNN).
- **Compressione Percettiva (DCT):** La pipeline JPEG sfrutta i limiti di contrasto del sistema visivo umano alle alte frequenze spaziali. La DCT isola l'energia di blocchi 8×8, consentendo alla quantizzazione di scartare i coefficienti AC di dettagli fini senza un danno percettivo severo.

Prossimi Passi: Il **Capitolo 6** inaugura la Parte II dell'opera, applicando gli strumenti di elaborazione delle immagini alla risoluzione di problemi reali di ispezione industriale. Verranno esplorate tecniche di **segmentazione e analisi delle forme** per il rilevamento automatico di difetti nelle linee di produzione — dall'identificazione di difetti superficiali nei pezzi alla lettura di *QRCode* nelle prove, consolidando il ponte tra la teoria presentata nella Parte I e le esigenze pratiche della visione computazionale.

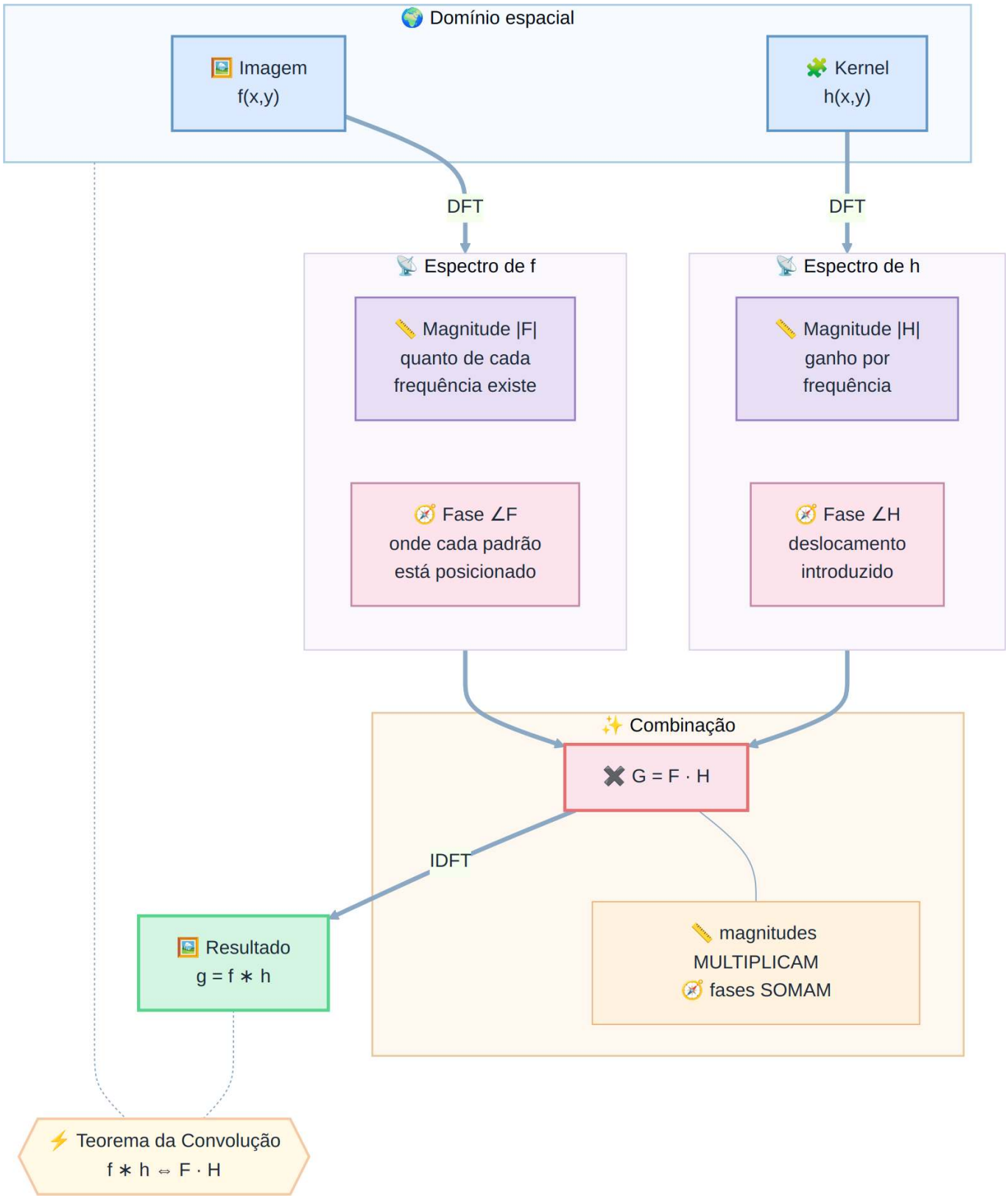


Figura 5.31: Mappa concettuale delle trasformazioni e delle proprietà nel dominio delle frequenze.

5.11 🤖 Uso del Gemini Notebook come Tutore Complementare

In questa edizione, si incoraggia l'uso della piattaforma **Gemini Notebook** come strumento complementare di apprendimento — **non come sostituto** della lettura attenta, della risoluzione degli esercizi o della sperimentazione pratica. Basato su architetture di intelligenza artificiale, il sistema utilizza esclusivamente il materiale didattico e i documenti forniti dall'autore come base di conoscenza, garantendo che le risposte generate siano concettualmente allineate al contenuto programmatico e all'approccio pedagogico adottato nel corso di quest'opera.

Accesso al Tutore Intelligente

🚀 ACCEDI A Gemini Notebook: CAPITOLO 05

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato realizzato esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, oppure seguendo il percorso in C++, le risposte del tutore potrebbero non corrispondere esattamente alla versione che stai leggendo.

Linee Guida sul Contenuto Generato dall'Intelligenza Artificiale

Sebbene gli strumenti di intelligenza artificiale costituiscano alleati efficienti nel processo di apprendimento e revisione, il contenuto generato è soggetto a incongruenze o imprecisioni tecniche. Pertanto, è indispensabile la consultazione sistematica di libri di testo, articoli scientifici e fonti accademiche indicizzate per una validazione rigorosa delle informazioni. Si raccomanda vivamente l'esecuzione e la modifica degli esempi pratici in Python forniti in questo capitolo come metodo primario di verifica sperimentale dei risultati.

5.12 Elenco di Esercizi

1. **(10%) Implementazione Diretta della DFT 2D:** Implementare analiticamente la Trasformata Discreta di Fourier 2D (DFT) senza l'ausilio di funzioni native di librerie (come `np.fft.fft2`), utilizzando strettamente la formulazione matematica definita in Equazione 5.1 per una matrice di dimensioni 16×16. Eseguire la validazione numerica confrontando i coefficienti generati con i risultati della funzione `np.fft.fft2`, assicurandosi che la deviazione assoluta massima sia inferiore a 10^{-8}. Misurare i tempi di esecuzione di entrambi i metodi e presentare una giustificazione teorica per la disparità osservata in termini di complessità asintotica.

2. **(15%) Soppressione del Rumore Periodico:** Aggiungere interferenze sinusoidali con frequenze spaziali $(u_0, v_0) \in \{(5, 10), (20, 5), (30, 30)\}$ all'immagine di test del *Cameraman*. Per ogni scenario di degradazione, progettare una maschera di filtraggio *notch* specifica nel dominio della frequenza per isolare e attenuare i picchi armonici indesiderati. Valutare quantitativamente l'efficacia del processo di restauro mediante il calcolo delle metriche PSNR e SSIM. Discutere analiticamente il compromesso (*trade-off*) tra l'attenuazione del rumore sinusoidale e l'indesiderata attenuazione delle caratteristiche strutturali legittime dell'immagine.

3. **(15%) Analisi Comparativa degli Operatori Passa-Basso:** Condurre uno studio comparativo tra i filtri passa-basso Ideale, Gaussiano e Butterworth (con ordini armonici $n = 1, 2, 4$), parametrizzati con frequenze di taglio $D_0 = 20, 40, 60$ pixel. Per ogni combinazione strutturale, calcolare gli indici PSNR e SSIM dell'immagine risultante rispetto al segnale originale di riferimento. Organizzare i dati quantitativi in una tabella strutturata e tracciare i grafici unidimensionali delle funzioni di trasferimento corrispondenti lungo il profilo orizzontale $H(u, 0)$.

4. **(15%) Banco di Filtri Multirisoluzione di Haar:** Sviluppare uno script per eseguire manualmente

la decomposizione *wavelet* discreta 2D di primo livello utilizzando la famiglia Haar. L'algoritmo deve calcolare i coefficienti dei filtri corrispondenti passa-basso (h) e passa-alto (g), applicandoli in modo separabile sulle righe e colonne della matrice, seguiti dall'operazione di decimazione (sottocampionamento spaziale per un fattore di 2). Validare numericamente l'accuratezza della propria implementazione confrontando le sottobande ottenute con l'output della funzione `pywt.dwt2(img, 'haar')`.

5. **(15%) Compressione Sparsa mediante Sogliatura Wavelet:** Applicare la tecnica di filtraggio per sogliatura netta (*hard thresholding*) sui coefficienti di dettaglio della decomposizione *wavelet*, adottando le soglie numeriche $T \in \{5, 10, 20, 40, 80\}$ per le famiglie Haar, Daubechies (`db4`) e Symlets (`sym4`). Dopo aver eseguito il processo di sintesi mediante la trasformata inversa (`pywt.waverec2`), calcolare i valori di PSNR e SSIM di ciascuna immagine ricostruita. Identificare e giustificare quale combinazione di famiglia *wavelet* e soglia T massimizza la similarità strutturale.

6. **(15%) Costruzione di un Codificatore JPEG Semplificato:** Implementare la pipeline completa di compressione dei dati simulando lo standard JPEG. Il flusso deve comprendere: conversione spaziale $RGB \to YC_bC_r$, sottocampionamento cromatico nella proporzione 4:2:0, segmentazione della luminanza in blocchi disgiunti di 8×8 pixel, applicazione della DCT-II 2D ortogonale e quantizzazione lineare basata sulla matrice normalizzata di luminanza scalata per i fattori di qualità desiderati. Eseguire la decodifica inversa e confrontare quantitativamente le ricostruzioni con i file generati dalla funzione `cv2.imencode` per i fattori di qualità di 20, 50 e 80.

7. **(15%) Analisi Percettiva su Contenuti Eterogenei:** Sviluppare un'immagine sintetica composta da tre regioni distinte e di caratteristiche spettrali contrastanti: una texture fotografica complessa (che rappresenta alte frequenze stocastiche), un'area di testo vettorizzato con bordi netti (che rappresenta transizioni a gradino pure) e un gradiente lineare continuo (che rappresenta basse frequenze omogenee). Sottoporre questa immagine mista ai processi di compressione nei formati JPEG, PNG e WebP. Valutare e interpretare i risultati correlando la dimensione finale del file su disco alle metriche PSNR e SSIM ottenute, giustificando quale formato mostra le prestazioni migliori per segnali di natura eterogenea e perché tale vantaggio si verifica in termini di compattazione dell'energia e preservazione percettiva.

Riferimenti del Capitolo

La base teorica e lo sviluppo analitico dei concetti trattati in questo capitolo si fondano sulle seguenti opere di riferimento:

- **Gonzalez (2018)** — Formulazioni classiche delle Trasformate Discrete di Fourier 2D (DFT), progettazione di filtri analitici nel dominio della frequenza, Trasformata Discreta del Coseno (DCT) e principi fondamentali dei sistemi di compressione delle immagini.
- **Oppenheim (2010)** — Teoria formale di segnali e sistemi applicati nel dominio discreto, che copre le proprietà matematiche della DFT e la modellazione analitica del Teorema della Convoluzione.
- **Mallat (1999)** — Fondamento matematico della teoria delle *wavelet*, formalizzazione dell'analisi multirisoluzione (MRA) e architettura dei banchi di filtri diadici.
- **Wallace (1991)** — Specifica originale e aspetti ingegneristici dello standard di compressione ISO/IEC JPEG, con particolare enfasi sui criteri psicovisuali per la progettazione delle matrici di quantizzazione DCT.
- **Szeliski (2022)** — Modellazione computazionale e caratterizzazione delle metriche moderne di fedeltà e qualità percettiva (PSNR e SSIM), nonché l'analisi comparativa dei formati di immagine rasterizzati ad alte prestazioni.

Executar Colab | Abrir GitHub

5.13 💻 Parte Pratica con Esercizi di Programmazione

La presente lista di esercizi di programmazione (EP) consolida le formulazioni teoriche presentate nel corso del Capitolo 5 — Trasformate e Compressione — attraverso un percorso pratico applicato. Gli esercizi sono strutturati a partire da matrici di dimensioni ridotte, consentendo la validazione analitica e l'ispezione manuale di ciascun coefficiente, mantenendo la coerenza metodologica adottata nei capitoli precedenti.

L'incatenamento degli esercizi riproduce rigorosamente il flusso concettuale del capitolo: si inizia con l'implementazione esplicita della Trasformata Discreta di Fourier (DFT) a partire dalla sua definizione matematica fondamentale; si prosegue con la progettazione di filtri passa-basso e maschere *notch* nel dominio della frequenza; si applica la quantizzazione dei coefficienti (nucleo della compressione con perdita); e si conclude con l'integrazione di queste fasi nella costruzione di un *pipeline* di compressione JPEG semplificato e nell'analisi percettiva dei formati immagine.

❗ Linee Guida per la Risoluzione degli Esercizi di Programmazione

In tutti gli esercizi di questo capitolo, le coordinate del **centro dello spettro** (origine delle frequenze spaziali dopo l'applicazione dello spostamento `fftshift`) devono essere determinate tramite divisione intera. Per una matrice con L righe e C colonne, la componente di frequenza nulla si trova nella posizione:

$$(c_y, c_x) = \left(\left\lfloor \frac{L}{2} \right\rfloor, \left\lfloor \frac{C}{2} \right\rfloor \right)$$

Questa convenzione è rigorosamente identica a quella adottata dalla funzione `np.fft.fftshift`. Inoltre, in tutte le fasi che richiedono discretizzazione o arrotondamento numerico (sia nella quantizzazione dei coefficienti AC sia nella ricostruzione finale dei pixel), si deve impiegare l'arrotondamento standard al numero intero più vicino (*round half away from zero*), mitigando ambiguità in valori con frazione esattamente pari a 0.5.

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EPs)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, esegui `TestSuite("EP05_01.estensione").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di

test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare il codice Python direttamente, senza salvare un file, usa `run_code(codice)` passando il codice come *stringa* in una variabile `codice`:

```
codice = """
from morph import mm
# ... il tuo codice qui ...
"""
TestSuite("EP05_01").run_code(codice)
```

5.13.1 EP05_01 🟢 Filtro Passa-Basso Ideale per Distanza nello Spettro

In uno ***scanner* di documenti antico**, il sensore cattura carta stropicciata e la trama delle fibre insieme al testo — rumore ad alta frequenza che "inquina" lo spettro ai bordi. Il tecnico della manutenzione non ha accesso all'immagine originale, ma solo allo **spettro di magnitudo già calcolato** dal software dello *scanner*. Il suo compito è semplice e chirurgico: mantenere solo il **cerchio centrale** delle basse frequenze (la struttura globale del documento) ed eliminare tutto ciò che si trova al di fuori del raggio D_0, rimuovendo la trama fine senza nemmeno dover toccare l'immagine spaziale.

Questo è il **Filtro Passa-Basso Ideale (LPFI)**: l'operazione spettrale più diretta del capitolo, ma anche quella che meglio rivela l'anatomia di uno spettro centrato.

5.13.1.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne) dello spettro di magnitudo — già fornito **centrato** (equivalente all'uscita di `np.fft.fftshift`).
2. **Frequenza di taglio:** Leggere l'intero D_0.
3. **Dati:** Leggere i valori interi della matrice di magnitudo, riga per riga.
4. **Centro dello spettro:** Calcolare $(c_y, c_x) = (L \,//\, 2,\ C \,//\, 2)$.
5. **Distanza:** Per ogni posizione (u, v), calcolare
$$D(u,v) = \sqrt{(u - c_y)^2 + (v - c_x)^2}$$
6. **Maschera ideale:** Applicare
$$H(u,v) = \begin{cases} 1, & D(u,v) \le D_0 \\ 0, & D(u,v) > D_0 \end{cases}$$
7. **Filtraggio:** Il valore di uscita è $\text{mag}'(u,v) = \text{mag}(u,v) \cdot H(u,v)$.
8. **Uscita:** Visualizzare la matrice filtrata con dimensioni $L \times C$.

5.13.1.2 📌 Vincoli Computazionali

- **Confronto non stretto:** il criterio usa $D(u,v) \le D_0$ (il confine appartiene al filtro, cioè viene mantenuto).
- **Tipo:** tutti i valori di ingresso e uscita sono interi; la distanza è calcolata in virgola mobile solo internamente.
- **Nessun arrotondamento della magnitudo:** poiché l'ingresso è già intero e la maschera è binaria (0 o 1), l'uscita non richiede mai arrotondamento.

5.13.1.3 🧠 Fondamenti Teorici

Regione	Distanza dal centro	Effetto del filtro
Centro ($D \le D_0$)	Basse frequenze	Preservate — struttura globale mantenuta

Regione	Distanza dal centro	Effetto del filtro
Bordi ($D > D_0$)	Alte frequenze	Azzerate — trama e rumore rimossi
D_0 **piccolo**	—	L'immagine ricostruita sarebbe molto sfocata
D_0 **grande**	—	Poca filtrazione; quasi tutta l'energia preservata

5.13.1.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero D_0.
- Righe successive: Elementi interi della matrice di magnitudo (centrata).

Uscita:

- Matrice filtrata con L righe e C colonne, separati da spazi.

5.13.1.5 📌 Esempi

Ingresso	Uscita	Osservazione
3 3 1 10 20 30 40 50 60 70 80 90	0 20 0 40 50 60 0 80 0	Centro $(1, 1)$. Gli angoli hanno $D = \sqrt{2} \approx 1.41 > 1$, quindi vengono azzerati; i vicini ortogonali hanno $D = 1 \leq 1$ e sono mantenuti.
1 3 0 5 9 7	0 9 0	$L = 1, C = 3$: centro in $(0, 1)$. Solo la posizione centrale stessa ($D = 0$) sopravvive a $D_0 = 0$.

```
%%writefile EP05_01.py
# Codice Python
```

```
Overwriting EP05_01.py
```

```
TestSuite("EP05_01.py").run()
```

```
<IPython.core.display.HTML object>
```

5.13.2 EP05_02 🟡 Filtro *Notch*: Rimozione dei Picchi Periodici

Una telecamera di **ispezione industriale** acquisisce immagini di circuiti stampati, ma l'alimentazione della linea di produzione introduce un'**interferenza elettrica periodica** — un pattern di strisce quasi impercettibile a occhio nudo, che però appare nello spettro di Fourier come **coppie di picchi luminosi** posizionati simmetricamente attorno al centro. Il team di visione artificiale non può rielaborare l'acquisizione: deve **localizzare e cancellare chirurgicamente** queste coppie di picchi nello spettro, preservando tutta l'altra informazione utile dell'immagine.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-ep0501.html

Figura 5.32: Simulatore EP05_01: Filtro Passa-Basso Ideale nello Spettro

Questo è il ruolo del **filtro rigetta-banda *notch***: a differenza del passa-basso (che interessa una regione continua), esso agisce su **punti specifici e sui loro simmetrici**, lasciando intatto il resto dello spettro.

5.13.2.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne) dello spettro di magnitudine centrato.
2. **Dati:** Leggere i valori interi della matrice di magnitudine, riga per riga.
3. **Picchi:** Leggere l'intero K (numero di coppie di picchi da rimuovere).
4. **Per ciascuno dei K picchi:** leggere tre interi Δv, Δu, r — spostamento verticale, spostamento orizzontale e raggio del *notch*.
5. **Centro dello spettro:** $(c_y, c_x) = (L\,//\,2,\ C\,//\,2)$.
6. **Soppressione simmetrica:** per ogni picco, azzerare **tutte** le posizioni (u, v) tali che la distanza dal punto $(c_y + \Delta v,\ c_x + \Delta u)$ sia $\leq r$, **e anche** tutte le posizioni con distanza $\leq r$ dal punto simmetrico $(c_y - \Delta v,\ c_x - \Delta u)$.
7. **Uscita:** Visualizzare la matrice risultante con dimensioni $L \times C$.

5.13.2.2 📌 Vincoli Computazionali

- **Simmetria obbligatoria:** ogni picco indicato genera **due** dischi azzerati (il punto e il suo simmetrico rispetto al centro) — dimenticare il simmetrico è l'errore più comune.
- **Sovrapposizione:** se due dischi si sovrappongono, la posizione rimane azzerata (non c'è "somma" o ripristino).
- **Confronto non stretto:** una posizione viene azzerata se distanza $\leq r$.
- **Ordine di lettura:** i K picchi devono essere elaborati nell'ordine in cui compaiono nell'input, ma il risultato finale non dipende dall'ordine (le operazioni di azzeramento sono commutative).

5.13.2.3 🧠 Fondamenti Teorici

Concetto	Ruolo nel filtro *notch*
Picco in $(\Delta v, \Delta u)$	Frequenza dell'interferenza periodica rilevata visivamente nello spettro

Concetto	Ruolo nel filtro *notch*
Punto simmetrico $(-\Delta v, -\Delta u)$	Ogni DFT di segnale reale è hermitiana: i picchi compaiono sempre in coppie simmetriche rispetto al centro
Raggio r	Controlla la "larghezza" della reiezione — un r grande rimuove più energia attorno al picco, ma anche informazione utile

5.13.2.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Righe successive: Elementi interi della matrice di magnitudine (centrata), L righe.
- Riga successiva: Intero K.
- K righe successive: tre interi Δv, Δu, r (separati da spazi).

Output:

- Matrice risultante in L righe e C colonne, separati da spazi.

5.13.2.5 📌 Esempi

Input	Output	Osservazione
5 5 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 1 1 1 0	1 2 3 4 5 6 0 8 9 10 11 12 13 14 15 16 17 18 0 20 21 22 23 24 25	Centro $(c_y, c_x) = (2, 2)$. Il picco indicato $(\Delta v, \Delta u) = (1, 1)$ genera il punto $(3, 3)$ (valore 19) e il suo simmetrico $(1, 1)$ (valore 7), entrambi azzerati con $r = 0$ (solo i punti esatti).

```
%%writefile EP05_02.py
# Codice Python
```

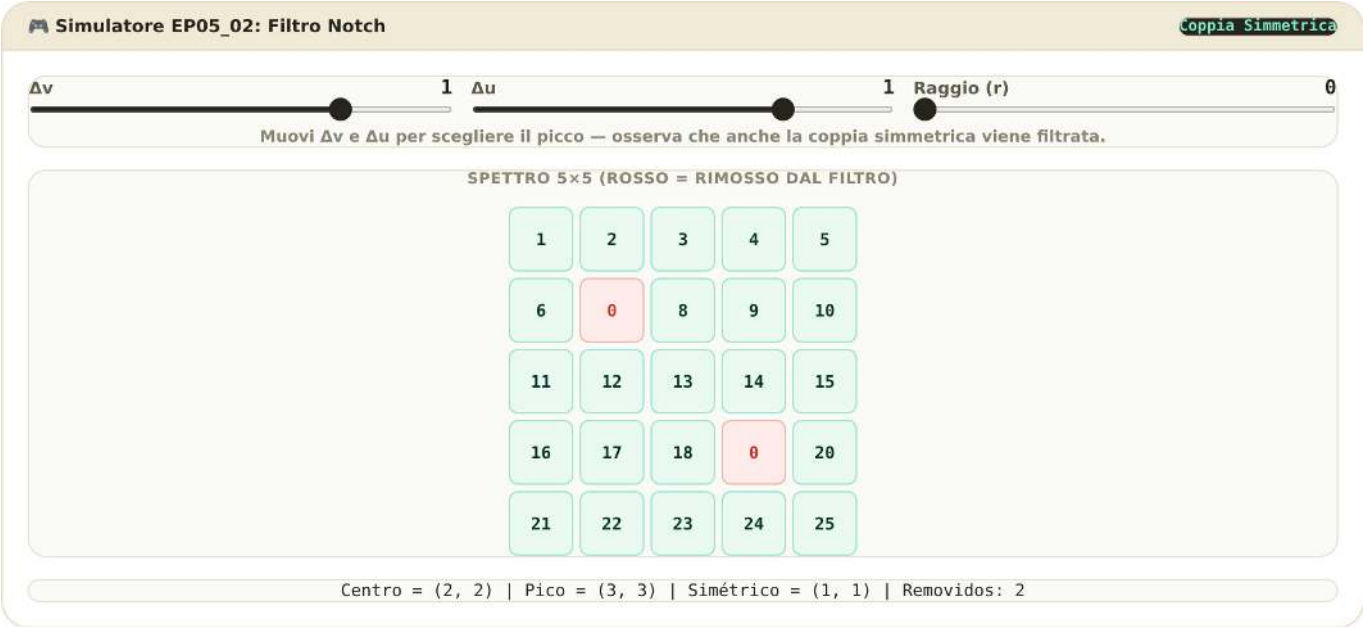
```
Overwriting EP05_02.py
```

```
TestSuite("EP05_02.py").run()
```

```
<IPython.core.display.HTML object>
```

5.13.3 EP05_03 🟠 Quantizzazione DCT: la Vera Fonte di Compressione

Un'applicazione di **galleria fotografica** deve ridurre le dimensioni di migliaia di immagini prima di caricarle sul cloud, senza ricodificare tutto da zero. L'ingegnere responsabile ha già i **coefficienti DCT** di ogni blocco 4×4 calcolati (la fase computazionalmente onerosa è già stata eseguita) — manca solo applicare la **tabella di quantizzazione**, la fase che scarta realmente informazioni e genera compressione. I coefficienti ad alta frequenza, meno percettibili all'occhio umano, ricevono divisori grandi e tendono a diventare **zero**; i coefficienti a bassa frequenza, più percettibili, ricevono divisori piccoli e sopravvivono quasi intatti.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-ep0502.html

Figura 5.33: Simulatore EP05_02: Filtro Notch

Dovrai implementare esattamente questa fase: **quantizzare e dequantizzare** (dividere, arrotondare, moltiplicare di nuovo) — il cuore della compressione *lossy* del JPEG.

5.13.3.1 📋 Linee Guida di Implementazione

1. **Dimensione del blocco:** Leggere l'intero N (blocco $N \times N$).
2. **Coefficienti:** Leggere la matrice C dei coefficienti DCT, N righe con N interi ciascuna (possono essere negativi).
3. **Tabella di quantizzazione:** Leggere la matrice Q, N righe con N interi positivi ciascuna.
4. **Quantizzazione:** Per ogni posizione (u, v), calcolare l'indice quantizzato

$$\tilde{C}(u, v) = \text{round}\left(\frac{C(u, v)}{Q(u, v)}\right)$$

   usando l'arrotondamento standard all'intero più vicino (i valori intermedi `.5` non si verificano mai nei casi di test).
5. **Dequantizzazione (ricostruzione):** Calcolare

$$C'(u, v) = \tilde{C}(u, v) \times Q(u, v)$$

6. **Output:** Visualizzare la matrice ricostruita C', $N \times N$, di interi.

5.13.3.2 📌 Vincoli Computazionali

- ***Round-trip* completo:** l'output è il coefficiente **ricostruito** ($\tilde{C} \times Q$), non l'indice quantizzato isolato.
- **Divisione in virgola mobile:** la divisione $C(u, v)/Q(u, v)$ deve essere eseguita in virgola mobile prima dell'arrotondamento — la divisione intera troncata produrrà un risultato errato.
- **Segno preservato:** i coefficienti negativi mantengono il segno dopo la quantizzazione e la ricostruzione.
- $Q(u, v) > 0$ **sempre:** non è necessario gestire la divisione per zero.

5.13.3.3 🧠 Fondamenti Teorici

Coefficiente	Frequenza	Valore tipico di Q	Effetto della quantizzazione
$C(0,0)$	DC (media del blocco)	Piccolo	Quasi sempre sopravvive — domina l'energia
$C(u,v)$ con $u+v$ basso	Bassa frequenza	Piccolo/medio	Parzialmente preservato
$C(u,v)$ con $u+v$ alto	Alta frequenza	Grande	Spesso diventa zero — fonte della compressione

5.13.3.4 📦 Specifica di Input e Output (VPL)

Input:

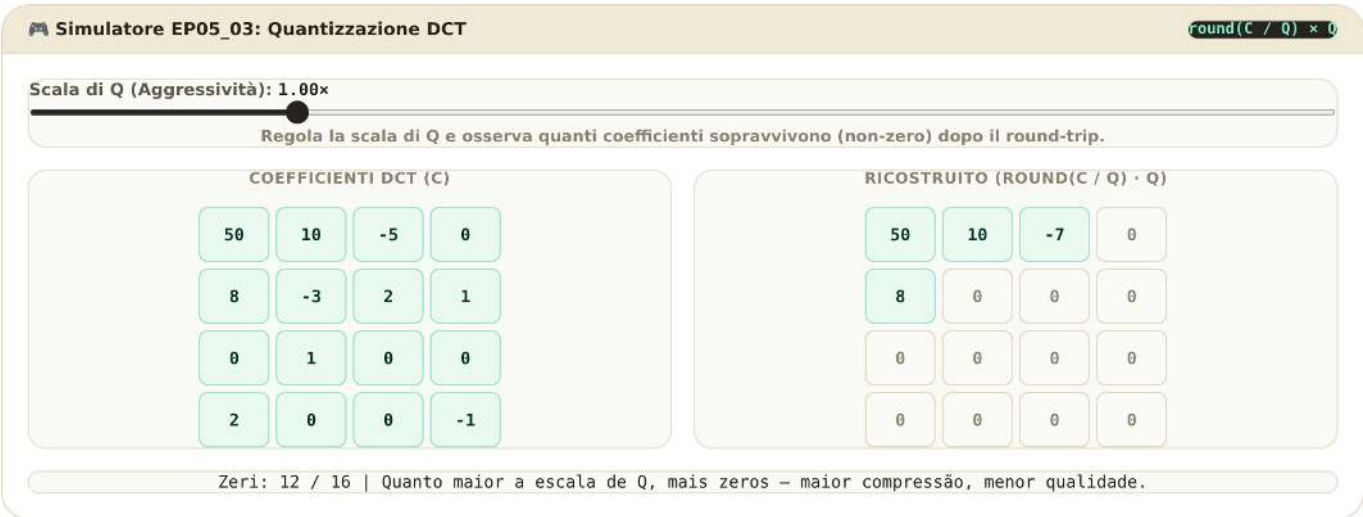
- Riga 1: Intero N.
- N righe successive: matrice C (coefficienti DCT, interi, possono essere negativi).
- N righe successive: matrice Q (tabella di quantizzazione, interi positivi).

Output:

- Matrice ricostruita C', $N \times N$, interi separati da spazio.

5.13.3.5 📌 Esempi

Input	Output	Osservazione
4 50 10 -5 0 8 -3 2 1 0 1 0 0 2 0 0 -1 2 5 7 8 4 7 8 11 6 8 11 12 9 11 12 14	50 10 -7 0 8 0 0 0 0 0 0 0 0 0 0 0	$C(0,0) = 50/2 = 25 \to 25{\times}2 = 50$ (preservato). $C(0,2) = -5/7 \approx -0.71 \to -1 \to -1 \times 7 = -7$. Invece $C(1,1) = -3/7 \approx -0.43 \to 0$: azzerato dalla quantizzazione — la maggior parte del blocco diventa zero, illustrando la compattazione dell'energia nell'angolo superiore sinistro.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-ep0503.html

Figura 5.34: Simulatore EP05_03: Quantizzazione DCT (*round-trip*)

```
%%writefile EP05_03.py
# Codice Python
```

```
Overwriting EP05_03.py
```

```
TestSuite("EP05_03.py").run()
```

```
<IPython.core.display.HTML object>
```

5.13.4 EP05_04 🔴 Implementazione della DFT 2D a partire dalla definizione

Un laboratorio di ricerca in **astronomia computazionale** ha ricevuto, da una missione remota, un piccolo sensore sperimentale i cui dati grezzi non possono essere elaborati tramite librerie moderne di FFT — l'ambiente di validazione è isolato e consente solo operazioni aritmetiche di base. Il team deve **reimplementare la Trasformata di Fourier Discreta 2D a partire dalla definizione matematica stessa**, cella per cella, per poi confrontare bit per bit con `np.fft.fft2` in un altro ambiente.

Questo è l'esercizio più concettuale della lista: non ci sono scorciatoie. Dovrai implementare direttamente la doppia sommatoria della Equazione 5.1, evidenziando *perché* la FFT esiste — e il costo computazionale che essa evita.

5.13.4.1 📋 Linee guida per l'implementazione

1. **Dimensioni:** Leggere i numeri interi M (righe) e N (colonne) dell'immagine $f(x, y)$.
2. **Dati:** Leggere i valori interi di $f(x, y)$, riga per riga.
3. **DFT 2D:** Per ogni coppia di frequenze (u, v) con $u = 0, \dots, M-1$ e $v = 0, \dots, N-1$, calcolare

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y)\, e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

   usando l'identità di Eulero $e^{-j\theta} = \cos(\theta) - j\sin(\theta)$ per separare la parte reale e quella immaginaria — **non utilizzare alcuna funzione FFT predefinita**.
4. **Magnitudine:** Calcolare $|F(u, v)| = \sqrt{\mathrm{Re}(F)^2 + \mathrm{Im}(F)^2}$ e arrotondare all'intero più vicino.
5. **Output:** Visualizzare la matrice delle magnitudini arrotondate, $M \times N$, nello stesso ordine (senza `fftshift` — il componente DC rimane in $(0, 0)$).

5.13.4.2 📌 Vincoli computazionali

- **Vietato l'uso di librerie FFT:** l'implementazione deve calcolare esplicitamente le doppie sommatorie (cicli annidati), anche se più lenta.
- **Senza `fftshift`:** l'output mantiene la convenzione grezza della DFT, con il componente DC in $F(0, 0)$ (angolo superiore sinistro).
- **Arrotondamento:** la magnitudine finale deve essere arrotondata all'intero più vicino; nei casi di test non vi è ambiguità `.5`.
- **Precisione:** piccoli errori di virgola mobile (ordine di 10^{-6}) prima dell'arrotondamento sono previsti e non influenzano il risultato intero finale.

5.13.4.3 🧠 Fondamenti teorici

Elemento	Significato
$F(0, 0)$	Componente DC — somma di tutti i pixel, $F(0, 0) = \sum f(x, y)$
Parte reale $\mathrm{Re}(F)$	Proiezione del segnale sui coseni
Parte immaginaria $\mathrm{Im}(F)$	Proiezione del segnale sui seni

Elemento	Significato
Complessità di questa implementazione	$\mathcal{O}((MN)^2)$ — ecco perché la FFT, con $\mathcal{O}(MN\log(MN))$, è indispensabile nelle immagini reali

5.13.4.4 📦 Specifica di input e output (VPL)

Input:

- Riga 1: intero M.
- Riga 2: intero N.
- Righe successive: elementi interi di $f(x, y)$, M righe.

Output:

- Matrice delle magnitudini $|F(u, v)|$ arrotondate, $M \times N$, separate da spazio.

5.13.4.5 📌 Esempi

Input	Output	Osservazione
2 2 1 2 3 4	10 2 4 0	$F(0,0) = 1+2+3+4 = 10$ (DC = somma totale). $F(0,1) = (1-2)+(3-4) = -2 \to \lvert F\rvert = 2$. $F(1,0) = (1+2)-(3+4) = -4 \to \lvert F\rvert = 4$. $F(1,1) = (1-2)-(3-4) = 0$.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-ep0504.html

Figura 5.35: Simulatore EP05_04: DFT 2D manuale

```
%%writefile EP05_04.py
# Codice Python
```

```
Overwriting EP05_04.py
```

```
TestSuite("EP05_04.py").run()
```

```
<IPython.core.display.HTML object>
```

5.13.5 EP05_05 🏆 *Pipeline* JPEG Completo: DCT, Quantizzazione e Ricostruzione

Sei stato incaricato di creare, da zero, un **codec JPEG didattico** in un ambiente embedded, senza alcuna libreria di immagini disponibile — solo operazioni matematiche di base. Il cliente vuole capire esattamente dove la qualità viene persa e dove viene recuperata, blocco per blocco. Questa è la sfida finale del capitolo: integrare **tutto** ciò che è stato studiato — la DCT-II ortonormale, la quantizzazione percettiva e la ricostruzione tramite IDCT — in un unico *pipeline* end-to-end, elaborando un blocco $N \times N$ dall'inizio alla fine, esattamente come fa internamente lo standard JPEG, 8×8 pixel alla volta.

5.13.5.1 📋 Linee Guida di Implementazione

1. **Dimensione del blocco:** Leggere il numero intero N.
2. **Blocco originale:** Leggere la matrice di pixel $f(x, y)$, N righe con N interi in $[0, 255]$.
3. **Tabella di quantizzazione:** Leggere la matrice Q, $N \times N$ interi positivi.
4. **Centratura:** Sottrarre 128 da ogni pixel: $g(x, y) = f(x, y) - 128$.
5. **DCT-II 2D ortonormale:** Calcolare
$$C(u, v) = \alpha(u)\,\alpha(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} g(x, y)\, \cos\left[\frac{\pi(2x+1)u}{2N}\right] \cos\left[\frac{\pi(2y+1)v}{2N}\right]$$
con $\alpha(0) = \sqrt{1/N}$ e $\alpha(k) = \sqrt{2/N}$ per $k > 0$.
6. **Quantizzazione:** $\tilde{C}(u, v) = \text{round}(C(u, v)/Q(u, v))$.
7. **Dequantizzazione:** $C'(u, v) = \tilde{C}(u, v) \times Q(u, v)$.
8. **IDCT-II 2D (inversa ortonormale):** Calcolare $g'(x, y)$ da $C'(u, v)$ usando la trasformata inversa corrispondente (stessa base, sommatoria su u, v).
9. **Inversione della centratura e arrotondamento:** $f'(x, y) = \text{round}(g'(x, y) + 128)$, limitato all'intervallo $[0, 255]$ (*clipping*).
10. **Output:** Visualizzare il blocco ricostruito f', $N \times N$, interi.

5.13.5.2 📌 Vincoli Computazionali

- ***Pipeline* completo obbligatorio:** tutte e sei le fasi (centrare, DCT, quantizzare, dequantizzare, IDCT, invertire) devono essere implementate — saltare la quantizzazione non supera i test, poiché il risultato sarebbe identico all'originale.
- ***Clipping*:** i valori ricostruiti al di fuori di $[0, 255]$ devono essere troncati (0 se negativi, 255 se maggiori di 255).
- **Arrotondamento:** sia nella quantizzazione che nella ricostruzione finale dei pixel, usare l'arrotondamento standard; i casi di test evitano ambiguità `.5`.
- **Base ortonormale:** la normalizzazione $\alpha(u)$ e $\alpha(v)$ deve essere applicata esattamente come specificato — senza di essa, la IDCT non ricostruisce correttamente.

5.13.5.3 🧠 Fondamenti Teorici

Fase	Analoga nel vero standard JPEG	Dove la qualità viene persa
Centratura	Stessa — la DCT presuppone un segnale centrato su zero	Nessuna perdita
DCT-II	Fasi 3–4 del *pipeline* (Tabella 5.7)	Nessuna perdita (trasformazione esatta e reversibile)
Quantizzazione	Fase 5 — divisione per $Q(u, v)$	**Principale fonte di perdita** — i coefficienti ad alta frequenza diventano zero
IDCT	Ricostruzione finale	Ricostruisce esattamente i coefficienti *quantizzati*, non quelli originali

5.13.5.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero N.
- N righe successive: blocco originale $f(x, y)$, interi in $[0, 255]$.
- N righe successive: tabella di quantizzazione Q, interi positivi.

Output:

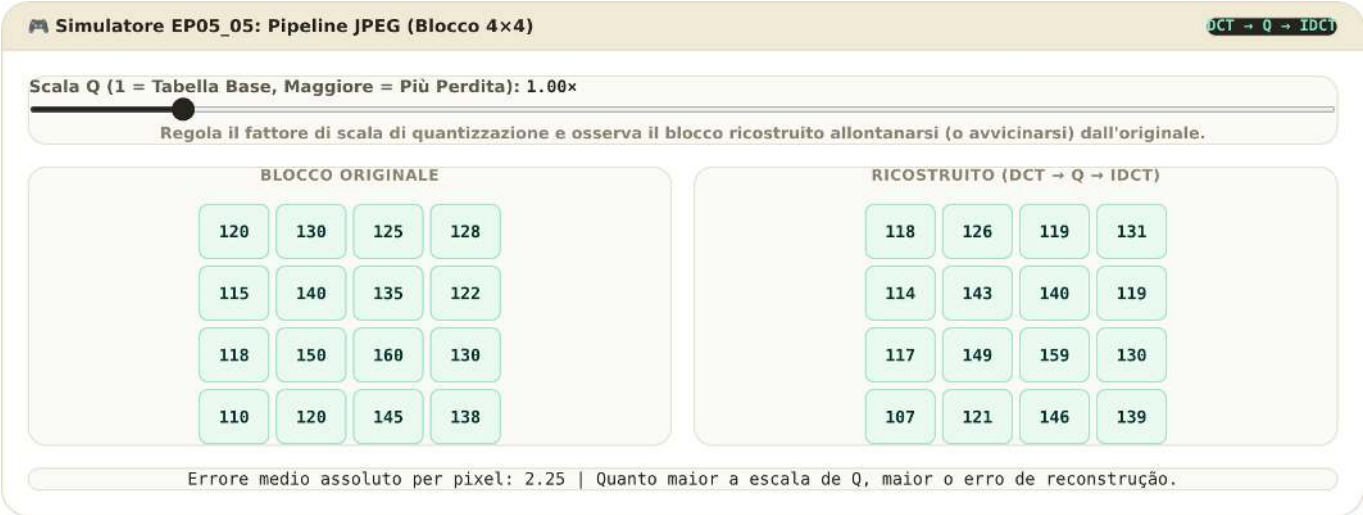
- Blocco ricostruito $f'(x, y)$, $N \times N$, interi in $[0, 255]$, separati da spazi.

5.13.5.5 📌 Esempi

Input	Output	Osservazione
4 120 130 125 128 115 140 135 122 118 150 160 130 110 120 145 138 4 6 8 10 6 8 10 12 8 10 12 16 10 12 16 20	118 126 119 131 114 143 140 119 117 149 159 130 107 121 146 139	Dopo la DCT, una quantizzazione aggressiva sulle alte frequenze (valori grandi di Q nell'angolo in basso a destra) e la ricostruzione tramite IDCT, il blocco risulta **vicino** all'originale, ma non identico — la differenza è il costo della compressione *lossy*.

5.13.5.6 💡 Suggerimento per il Debug

Se il risultato non corrisponde, verifica in questo ordine: (1) i coefficienti DCT grezzi (prima della quantizzazione) — devono ricostruire l'originale **esattamente** tramite IDCT se salti le fasi 6–7; (2) la tabella $\alpha(u)$ — un errore comune è applicare $\sqrt{2/N}$ anche per $u = 0$; (3) l'arrotondamento della quantizzazione, che deve avvenire **prima** di moltiplicare di nuovo per Q.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap05/sim-ep0505.html

Figura 5.36: Simulatore EP05_05: *Pipeline* JPEG completo a blocchi

```
%%writefile EP05_05.py
# Codice Python
```

```
Overwriting EP05_05.py
```

```
TestSuite("EP05_05.py").run()
```

```
<IPython.core.display.HTML object>
```

Parte II

Parte II — Visione Artificiale

Capitolo 6

Analisi dei Documenti e Ispezione Industriale

Executar Colab Abrir GitHub

Nella **Parte I — Elaborazione Digitale delle Immagini (PDI)**, sono state studiate tecniche per la trasformazione e il miglioramento delle immagini, come operazioni morfologiche, filtraggio spaziale, convoluzioni, sogliatura, segmentazione ed elaborazione nel dominio della frequenza.

La **Parte II — Visione Artificiale (VC)** amplia tale ambito trattando l'interpretazione automatica del contenuto visivo, coinvolgendo l'estrazione di informazioni, il riconoscimento di pattern e il processo decisionale a partire dalle immagini.

Questo capitolo presenta tale transizione attraverso due applicazioni rappresentative:

1. **Analisi Automatizzata dei Documenti**, applicata all'elaborazione di moduli, valutazioni e altri documenti strutturati mediante sistemi di riconoscimento ottico dei segni (*Optical Mark Recognition* – OMR);
2. **Ispezione Industriale Automatizzata**, finalizzata al controllo qualità e al rilevamento di difetti nelle linee di produzione.

Tali applicazioni integrano tecniche di rilevamento di strutture geometriche, estrazione di descrittori invarianti, riconoscimento di pattern e classificazione degli oggetti, costituendo la base di numerosi sistemi moderni di ispezione visiva e automazione.

6.1 Obiettivi del Capitolo

Al termine di questo capitolo, lo studente dovrebbe essere in grado di:

- **Valutare l'influenza della pre-elaborazione** sulla qualità del riconoscimento automatico delle informazioni nei documenti;
- **Eseguire il riconoscimento ottico dei caratteri** (OCR) per convertire documenti digitalizzati in testo codificato;
- **Applicare tecniche di elaborazione del linguaggio naturale**, inclusa la traduzione automatica, al testo ottenuto tramite OCR;
- Applicare tecniche di **allineamento e rettifica geometrica dei documenti** utilizzando la Trasformata di Hough e le trasformazioni proiettive;
- **Implementare sistemi di riconoscimento ottico dei marcatori** (OMR) per la lettura automatizzata di valutazioni e moduli;
- **Rilevare e segmentare le regioni di interesse** sulla base di operazioni morfologiche, contorni e proprietà geometriche;
- **Decodificare marcatori bidimensionali e codici a barre**, integrando librerie di Visione Artificiale in *pipeline* di elaborazione documentale;
- **Sviluppare *pipeline* di Visione Artificiale** per l'analisi automatizzata dei documenti.

Questo capitolo segna la transizione dall'**Elaborazione Digitale delle Immagini**, volta alla trasformazione delle immagini, alla **Visione Artificiale**, il cui obiettivo è interpretare il contenuto visivo, estrarre informazioni e supportare processi automatizzati di analisi e presa di decisione.

6.2 Configurazione dell'Ambiente

Gli esempi di questo capitolo utilizzano librerie ampiamente impiegate nella PDI-VC. Il blocco sottostante installa i pacchetti necessari; in ambienti che già li possiedono, l'esecuzione può essere ignorata.

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup()
from morph import mm

# instala più dipendenze oltre a morph.py per questo capitolo
import sys, subprocess, importlib, shutil

def setup_cap06():
    """Installa dipendenze di sistema e Python specifiche del Capitolo 6
    (OCR, lettura PDF, codice a barre)."""

    # 1. Dipendenze di sistema
    if 'google.colab' in sys.modules:
        print("[AMBIENTE] Google Colab. Configurazione delle dipendenze di sistema...")
        subprocess.run(
            "apt-get update && apt-get install -y poppler-utils "
            "libzbar0 tesseract-ocr tesseract-ocr-por",
            shell=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL
        )
    elif shutil.which("tesseract") is None:
        # Ambiente locale senza tesseract: tenta di installare tramite apt-get (richiede
        sudo/root)
        if shutil.which("apt-get"):
            print("[AMBIENTE] Locale. Installazione di tesseract-ocr tramite apt-get (potrebbe
            richiedere password)...")
            resultado = subprocess.run(
                "sudo apt-get update && sudo apt-get install -y tesseract-ocr
                tesseract-ocr-por",
                shell=True
            )
            if resultado.returncode != 0 or shutil.which("tesseract") is None:
                print(
                    "[AVVISO] Impossibile installare automaticamente. "
                    "Installa manualmente: sudo apt install tesseract-ocr tesseract-ocr-por"
                )
        else:
            print(
                "[AVVISO] tesseract non trovato e apt-get non disponibile. "
                "Installa manualmente prima di eseguire le celle OCR."
            )

    # 2. Dipendenze Python (installa solo quelle mancanti)
    pkgs = {
        "cv2": "opencv-python", "skimage": "scikit-image", "numpy": "numpy",
        "pdf2image": "pdf2image", "pandas": "pandas", "tabulate": "tabulate",
```

```
        "PyPDF2": "PyPDF2", "bcrypt": "bcrypt", "pyarrow": "pyarrow",
        "pyzbar": "pyzbar", "pytesseract": "pytesseract", "deep_translator": "deep-translator"
    }
    for mod, pkg in pkgs.items():
        if importlib.util.find_spec(mod) is None:
            resultado_pip = subprocess.run([sys.executable, "-m", "pip", "install", "-q",
pkg])
            if resultado_pip.returncode != 0:
                print(f"[AVVISO] Errore durante l'installazione di {pkg} (necessario per il
                modulo {mod}).")


setup_cap06()

# 3. Import globali della pipeline
import cv2, numpy as np, matplotlib.pyplot as plt
from skimage import io, data, color
```

```
 Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0
[AMBIENTE] Locale. Installazione di tesseract-ocr tramite apt-get (potrebbe richiedere
password)...
[AVVISO] Impossibile installare automaticamente. Installa manualmente: sudo apt install
tesseract-ocr tesseract-ocr-por
```

Oltre a queste librerie, verrà utilizzato il modulo didattico `morph.py`, sviluppato per semplificare le operazioni di lettura, visualizzazione ed elaborazione delle immagini nel corso di questo libro. Il codice seguente ne verifica la disponibilità, esegue il *download* quando necessario e conferma la versione caricata.

```
import os
import urllib.request

if not os.path.exists("morph.py"):
    urllib.request.urlretrieve(
        "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/morph.py",
        "morph.py",
    )

import morph
from morph import mm

print(f"  Ambiente pronto. morph {getattr(morph, '__version__', 'local_file')}")
```

```
 Ambiente pronto. morph 1.1.9
```

6.3 Banche di Immagini per la Sperimentazione

Gli esempi presentati in questa parte del libro utilizzano, quando possibile, documenti digitalizzati, fogli di risposta, codici a barre, *QRCodes* e altre immagini provenienti da applicazioni reali. Per rendere gli esperimenti integralmente riproducibili — anche in ambienti senza accesso a internet o ai file originali del MCTest —, vengono inoltre impiegate immagini pubbliche ampiamente utilizzate nell'insegnamento e nella ricerca in Elaborazione Digitale delle Immagini e Visione Artificiale (PDI-VC).

Perché utilizzare una banca di immagini di *benchmark*?

Immagini come `camera()` e `coins()` sono impiegate da decenni in libri di testo, articoli scientifici e materiali didattici di PDI-VC. Il loro utilizzo offre importanti vantaggi:

- **Riproducibilità:** qualsiasi lettore ottiene esattamente le stesse immagini, indipendentemente dal

computer o dal sistema operativo utilizzato, senza necessità di *download* esterni o di file specifici di questo libro;
- **Comparabilità:** i risultati prodotti possono essere confrontati direttamente con quelli riportati in letteratura, poiché gli stessi set di immagini sono ampiamente adottati come riferimento;
- **Focus sugli algoritmi:** essendo immagini compatte, ben documentate e distribuite senza restrizioni d'uso per fini educativi e scientifici, consentono di concentrare l'attenzione sulle tecniche di elaborazione, riducendo interferenze legate all'acquisizione o alla gestione dei dati.

6.3.1 Immagini Pubbliche con `skimage.data`

Il modulo `skimage.data` fornisce una collezione di immagini di riferimento ampiamente utilizzata in attività di insegnamento, ricerca e validazione di algoritmi in PDI-VC. L'ispezione dell'attributo `data.__all__` mostra che la versione attuale comprende **42 elementi**, inclusi fotografie naturali, documenti digitalizzati, immagini mediche, microscopia, texture, pattern sintetici, modelli tridimensionali e sequenze temporali. È utile notare che alcuni di questi elementi corrispondono a funzioni utilitarie, come `data_dir()` e `download_all()`, e non a immagini propriamente dette.

Poiché l'obiettivo di questo capitolo è presentare applicazioni di analisi documentale e ispezione visiva, la Tabella 6.2 raccoglie una **selezione rappresentativa** delle immagini più rilevanti, organizzata secondo le loro principali applicazioni nella Visione Artificiale. La Figura 6.1 presenta un campione di queste immagini, raggruppate secondo la stessa classificazione adottata nella tabella.

Immagini utilizzate in questo capitolo

Sebbene `skimage.data` fornisca decine di immagini di riferimento, solo quattro sono impiegate direttamente negli esperimenti di questo capitolo. Esse sono state selezionate perché riproducono, in modo controllato, caratteristiche spesso riscontrate in documenti digitalizzati e in sistemi di ispezione visiva industriale. La Tabella 6.1 riassume il ruolo di ciascuna di esse nel corso di questo capitolo.

Tabella 6.1: Immagini di skimage.data utilizzate negli esperimenti di questo capitolo.

Immagine	Applicazione nel capitolo
`data.page()`	Pagina digitalizzata utilizzata negli esperimenti di correzione dell'illuminazione, miglioramento locale (CLAHE) e sogliatura automatica di Otsu.
`data.text()`	Documento contenente testo stampato, impiegato per illustrare segmentazione, estrazione dei contorni e fasi tipiche di OCR e OMR.
`data.coffee()`	Fotografia a colori con variazioni naturali di illuminazione, utilizzata per esemplificare tecniche applicabili a scene reali non documentali.
`data.brick()`	Texture di riferimento impiegata in esempi di ispezione superficiale e rilevamento di difetti tramite analisi della varianza locale.

Tabella 6.2: Selezione di immagini pubbliche rappresentative disponibili nel modulo *skimage.data*, organizzate per ambito di applicazione.

Categoria	Função	Descrição
📄 Documentos & OCR/OMR	`data.page()`	Página digitalizada de documento — normalização de fundo, CLAHE e Otsu.
📄 Documentos & OCR/OMR	`data.text()`	Texto impresso — segmentação e extração de contornos em cenários de OCR/OMR.

Categoria	Função	Descrição
🔵 Segmentação, Morfologia & Contornos	`data.coins()`	Conjunto de moedas — referência clássica para segmentação e *watershed*.
🔵 Segmentação, Morfologia & Contornos	`data.clock()`	Relógio analógico — detecção de formas e contornos.
🔵 Segmentação, Morfologia & Contornos	`data.binary_blobs()`	Blobs binários sintéticos — conectividade e morfologia matemática.
🔵 Segmentação, Morfologia & Contornos	`data.moon()`	Superfície lunar — segmentação de crateras por relevo de intensidade.
Textura & Inspeção Industrial	`data.brick()`	Textura uniforme de tijolos — detecção de defeitos por variância local.
Textura & Inspeção Industrial	`data.checkerboard()`	Padrão xadrez — calibração de câmera e transformações geométricas.
🖼️ Fotografias Clássicas de PDI/VC	`data.camera()`	Fotógrafo com tripé — imagem de referência mais citada na literatura de PDI.
🖼️ Fotografias Clássicas de PDI/VC	`data.astronaut()`	Retrato colorido de astronauta — filtragem e realce em cor.
🖼️ Fotografias Clássicas de PDI/VC	`data.coffee()`	Xícara de café — cena real com variação de iluminação e cor.
🖼️ Fotografias Clássicas de PDI/VC	`data.cat()` / `data.chelsea()`	Fotografias coloridas de gatos — detecção de bordas e realce.
🖼️ Fotografias Clássicas de PDI/VC	`data.horse()`	Silhueta binária de cavalo — descritores de forma e contorno.

```python
import matplotlib.pyplot as plt
from skimage import data

# Immagini ordinate per categoria; il colore del titolo riproduce il colore della categoria
nella tab. precedente
imgs = [
    ("page",         data.page(),         "#2563eb"),   # Documenti & OCR/OMR
    ("text",         data.text(),         "#2563eb"),
    ("coins",        data.coins(),        "#16a34a"),   # Segmentazione & morfologia
    ("binary_blobs", data.binary_blobs(), "#16a34a"),
    ("brick",        data.brick(),        "#ea580c"),   # Texture & ispezione industriale
    ("checkerboard", data.checkerboard(),"#ea580c"),
    ("camera",       data.camera(),       "#7c3aed"),   # Fotografie classiche di PDI/VC
    ("coffee",       data.coffee(),       "#7c3aed"),
]

fig, ax = plt.subplots(2, 4, figsize=(11, 5.5))
for a, (nome, img, cor) in zip(ax.ravel(), imgs):
    a.imshow(img, cmap="gray")
    a.set_title(nome, color=cor, fontweight="bold")
    a.axis("off")
plt.tight_layout()
```

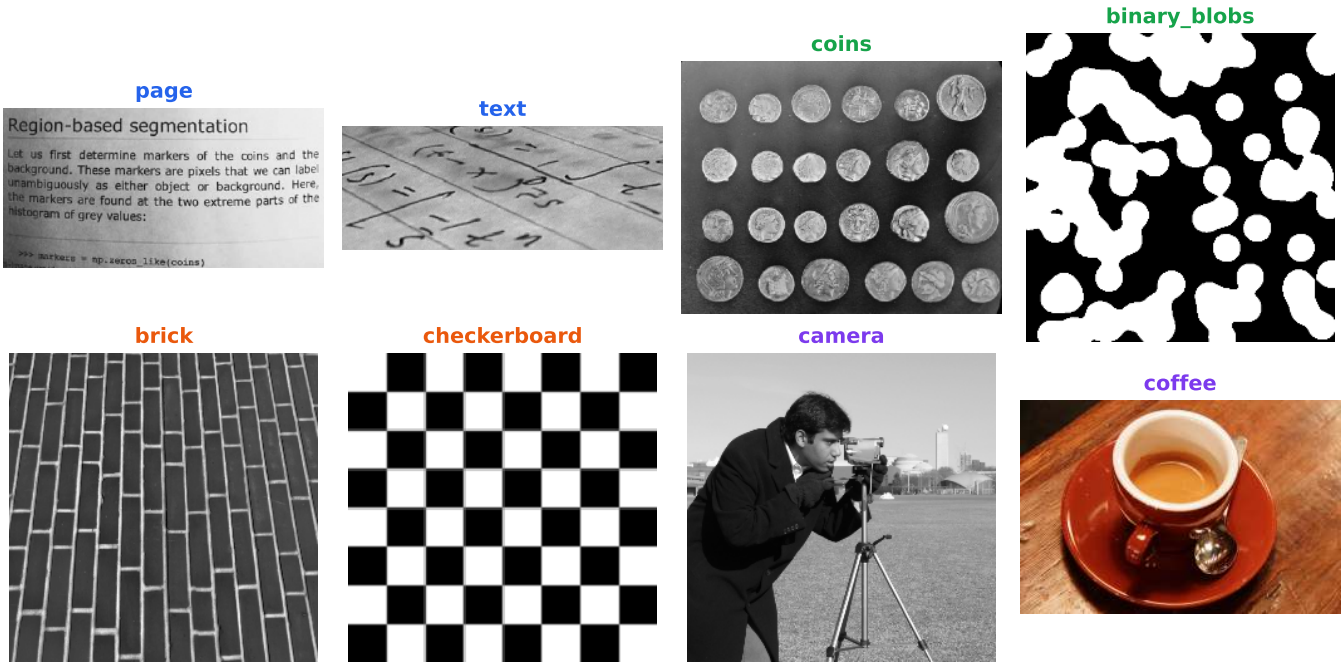


Figura 6.1: Campione di immagini pubbliche di *skimage.data*, raggruppate per area di applicazione.

6.4 Normalizzazione del Fondo e Equalizzazione Locale del Contrasto

La qualità della segmentazione dipende direttamente dalle caratteristiche dell'immagine in ingresso. Nei documenti digitalizzati, variazioni di illuminazione, ombre, regioni sovraesposte e differenze di tonalità della carta riducono il contrasto tra il primo piano e lo sfondo, rendendo difficile l'applicazione di metodi di sogliatura globale, come l'algoritmo di Otsu.

Per minimizzare questi effetti, vengono impiegate due tecniche complementari di pre-elaborazione:

- **Normalizzazione del fondo:** stima la componente a bassa frequenza dell'immagine mediante una forte attenuazione e, successivamente, normalizza l'immagine originale rispetto a tale fondo stimato. Questa procedura riduce i gradienti di illuminazione e compensa le variazioni lente di intensità, preservando le strutture di interesse.
- **CLAHE** (*Contrast Limited Adaptive Histogram Equalization*), presentato nel **Capitolo 4:** suddivide l'immagine in piccole regioni (*tiles*) ed esegue l'equalizzazione dell'istogramma di ciascuna regione in modo indipendente. Il contrasto è limitato per evitare un'amplificazione eccessiva del rumore, rendendo la tecnica particolarmente adatta a immagini con variazioni locali di illuminazione.

La Figura 6.2 confronta queste strategie utilizzando l'immagine `page()` della libreria `skimage.data`. Vengono presentati sei risultati: (a) l'immagine originale; (b) la binarizzazione diretta tramite il metodo di Otsu, utilizzata come riferimento; (c) il fondo stimato tramite filtraggio gaussiano; (d) l'immagine dopo la normalizzazione del fondo; (e) la binarizzazione ottenuta dopo l'applicazione della CLAHE seguita dal metodo di Otsu; e (f) la binarizzazione ottenuta dopo la normalizzazione del fondo seguita dall'applicazione del metodo di Otsu.

Il confronto consente di osservare l'effetto prodotto da ciascuna fase della pre-elaborazione e la sua influenza sulla qualità della segmentazione. In particolare, la normalizzazione del fondo riduce le variazioni globali di illuminazione, mentre la CLAHE aumenta il contrasto locale tra caratteri e sfondo. A seconda delle caratteristiche dell'immagine, l'una o l'altra strategia può produrre risultati superiori, non essendovi una tecnica universalmente più adatta.

```
import cv2
from skimage import data
```

```python
from morph import mm

img = data.page()  # oppure: img = mm.gray(img_final)  - con l'immagine del foglio di prova

#   Metodo 1: Normalizzazione dello sfondo + Otsu
# Stima lo sfondo con un filtro gaussiano di sigma grande (variazioni lente di luce)
# e divide pixel per pixel per annullare il gradiente di illuminazione
bg = cv2.GaussianBlur(img, (0, 0), sigmaX=25)
img_norm = cv2.divide(img, bg, scale=255)
img_norm_otsu = mm.threshold(img_norm)

#   Metodo 2: CLAHE + Otsu
# tileGridSize definisce la dimensione di ciascuna regione locale (tile)
# clipLimit controlla il limite di amplificazione - valori alti aumentano il contrasto
# ma amplificano anche il rumore
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
img_clahe = clahe.apply(img)
img_clahe_otsu = mm.threshold(img_clahe)

#   Metodo 0: Otsu diretto (senza preprocessamento) - riferimento
img_otsu = mm.threshold(img)

mm.show(
    [img,  img_otsu, bg, img_norm, img_clahe_otsu, img_norm_otsu],
    titles=["(a) Originale", "(b) Otsu diretto", "(c) Gaussiano",  \
            "(d) Sfondo normalizzato (img/bg)", "(e) CLAHE + Otsu", \
            "(f) Normalizzazione sfondo + Otsu"],
    cols=3,
    figsize=(14, 8)
)
```

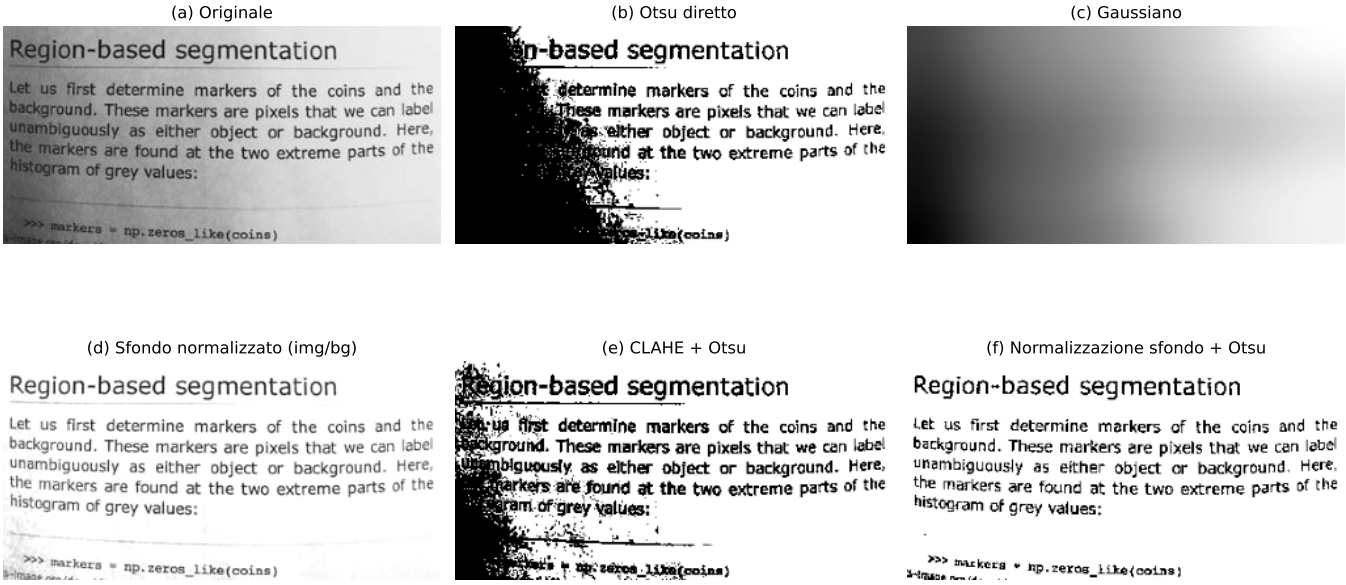


Figura 6.2: Confronto tra strategie di preprocessamento per la binarizzazione dell'immagine del testo: (a) immagine originale; (b) sogliatura diretta con il metodo di Otsu; (c) sfondo stimato mediante filtraggio gaussiano; (d) immagine dopo la normalizzazione dello sfondo (divisione per l'immagine smussata); (e) CLAHE seguito da sogliatura con Otsu; e (f) normalizzazione dello sfondo seguita da sogliatura con Otsu. Le immagini illustrano l'effetto di ciascuna tecnica nella compensazione delle variazioni di illuminazione e nella qualità della segmentazione.

🧠 Perché funziona? — Normalizzazione dello sfondo vs. CLAHE

Normalizzazione dello sfondo: dividendo l'immagine per una versione fortemente attenuata di sé stessa, si eliminano le variazioni lente di luminosità (gradiente di luce, ombra di bordo) senza influire sui dettagli fini — testo, linee, bolle. Il risultato è un'immagine con illuminazione approssimativamente uniforme, dove la soglia globale di Otsu funziona correttamente su tutta la pagina.
CLAHE: un istogramma globale equalizzato "allunga" i toni dell'intera immagine in una volta sola — utile quando l'illuminazione è uniforme, ma problematico quando non lo è. Il CLAHE suddivide l'immagine in piccoli blocchi (*tiles*) e equalizza ciascuno separatamente, con un limite massimo di amplificazione (`clipLimit`) per non amplificare il rumore. È particolarmente efficace per evidenziare regioni localmente sottoesposte, ma non elimina i gradienti globali — per questo, applicarlo dopo la normalizzazione dello sfondo tende a produrre risultati più coerenti.

6.5 Riconoscimento Ottico dei Caratteri (OCR)

Dopo la binarizzazione, la fase successiva dell'elaborazione documentale consiste nella conversione della rappresentazione visiva dei caratteri in testo codificato digitalmente, processo denominato **Riconoscimento Ottico dei Caratteri** (*Optical Character Recognition* — **OCR**).

In generale, un sistema OCR comprende tre fasi:

1. **Segmentazione:** identifica righe, parole e caratteri nell'immagine, utilizzando proiezioni orizzontali e verticali o il rilevamento di componenti connessi.
2. **Estrazione delle caratteristiche:** rappresenta ogni carattere tramite attributi visivi, come bordi, curvature e pattern del tratto.
3. **Riconoscimento:** associa gli attributi estratti al carattere più probabile. I sistemi attuali utilizzano prevalentemente reti neurali ricorrenti (LSTM) o architetture basate su *transformer*.

In questo libro si utilizza il **Tesseract OCR**, accessibile tramite la libreria `pytesseract` (installazione: `pip install pytesseract`). Sviluppato originariamente dalla Hewlett-Packard tra il 1985 e il 1995 e attualmente mantenuto da Google, Tesseract è descritto in Smith (2007) e Smith (2013). Nelle versioni recenti, il riconoscimento testuale è realizzato da reti neurali LSTM.

Le prestazioni dell'OCR dipendono dalla qualità dell'immagine di input. Rumore, basso contrasto, distorsioni geometriche e illuminazione irregolare riducono il tasso di riconoscimento. Per questo motivo, fasi come la rettifica, la normalizzazione dello sfondo e l'equalizzazione adattiva (CLAHE) integrano la pre-elaborazione dell'immagine.

🧠 Il Tesseract necessita di un'immagine binarizzata?

Il Tesseract incorpora internamente una fase di binarizzazione adattiva prima del riconoscimento dei caratteri. Per questo motivo, fornire all'OCR un'immagine precedentemente binarizzata non sempre produce i migliori risultati.
Poiché la sogliatura è un'operazione irreversibile, essa può eliminare variazioni sottili di intensità sui bordi dei caratteri, come l'*antialiasing*, che possono aiutare il meccanismo di riconoscimento. In molti casi, un'immagine in scala di grigi, con buona illuminazione e contrasto, produce una trascrizione più fedele rispetto alla sua versione binarizzata.

Per indagare questo effetto, si confronta il testo estratto da Tesseract a partire da quattro versioni della stessa immagine, presentate nella Figura 6.3: (a) immagine originale; (b) immagine sottoposta a equalizzazione adattiva (CLAHE) seguita dalla sogliatura con il metodo di Otsu; (c) immagine sottoposta a normalizzazione dello sfondo seguita dalla sogliatura con il metodo di Otsu; e (d) immagine sottoposta solo alla normalizzazione dello sfondo, preservando i toni di grigio.

Il confronto tra le versioni (c) e (d) mostra che, in tratti contenenti caratteri visivamente simili, la versione in

scala di grigi (d) ha prodotto una trascrizione più fedele al testo originale rispetto alla versione binarizzata (c). Questo risultato indica che la sogliatura applicata nella pre-elaborazione può eliminare informazioni utili al riconoscimento. Pertanto, sebbene la binarizzazione sia essenziale per diverse operazioni di elaborazione delle immagini, essa non costituisce necessariamente il miglior input per l'OCR. La scelta della tecnica di pre-elaborazione deve considerare la fase successiva del *pipeline* documentale.

```
import pytesseract
import shutil as _sh
if _sh.which("tesseract") is None:
    # Ambiente di build senza Tesseract installato (es.: senza apt/sudo):
    # degrada invece di rompere il render. Su Colab/locale con Tesseract,
    # non cambia nulla.
    _AVISO_OCR = "[Tesseract OCR indisponivel neste ambiente - texto omitido]"
    pytesseract.image_to_string = lambda *a, **k: _AVISO_OCR
from skimage import data
import cv2
from morph import mm

img = data.page()

#   Riutilizzando i risultati della sezione precedente
img_otsu = mm.threshold(img)

bg = cv2.GaussianBlur(img, (0, 0), sigmaX=25)
img_norm = cv2.divide(img, bg, scale=255)       # scala di grigi, senza Otsu
img_norm_otsu = mm.threshold(img_norm)          # binarizzata

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
img_clahe = clahe.apply(img)
img_clahe_otsu = mm.threshold(img_clahe)

#   Configurazione di Tesseract
# --psm 6: assume un unico blocco uniforme di testo (adatto all'immagine `page`)
config = "--psm 6"

texto_original   = pytesseract.image_to_string(img, config=config)
texto_clahe_otsu = pytesseract.image_to_string(img_clahe_otsu, config=config)
texto_norm_otsu  = pytesseract.image_to_string(img_norm_otsu, config=config)
texto_norm_gray  = pytesseract.image_to_string(img_norm, config=config)

for nome, texto in zip(
    ["(a) Original", "(b) CLAHE + Otsu", "(c) Normaliz. fundo + Otsu",
     "(d) Normaliz. fundo (tons de cinza)"],
    [texto_original, texto_clahe_otsu, texto_norm_otsu, texto_norm_gray]
):
    print(f"--- {nome} ---")
    print(texto.strip(), "\n")

mm.show(
    [img, img_clahe_otsu, img_norm_otsu, img_norm],
    titles=["(a) Originale", "(b) CLAHE + Otsu", "(c) Normal. sfondo + Otsu",
            "(d) Normal. sfondo (grigi)"],
    cols=4,
    figsize=(16, 4)
)
```

```
--- (a) Original ---
[Tesseract OCR indisponivel neste ambiente - texto omitido]

--- (b) CLAHE + Otsu ---
```

```
[Tesseract OCR indisponivel neste ambiente - texto omitido]

--- (c) Normaliz. fundo + Otsu ---
[Tesseract OCR indisponivel neste ambiente - texto omitido]

--- (d) Normaliz. fundo (tons de cinza) ---
[Tesseract OCR indisponivel neste ambiente - texto omitido]
```

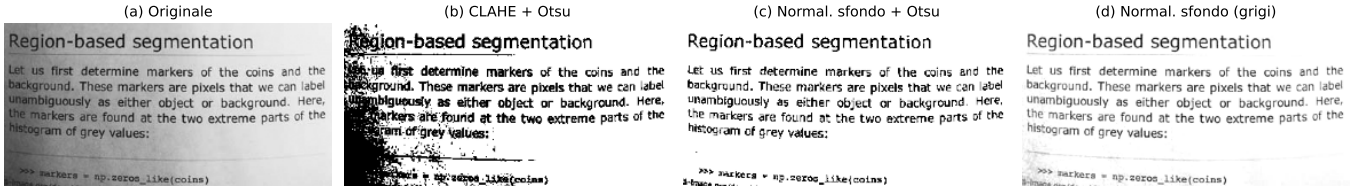


Figura 6.3: Confronto del testo estratto da Tesseract da quattro versioni della stessa immagine: (a) immagine originale; (b) CLAHE seguito da sogliatura di Otsu; (c) normalizzazione dello sfondo seguita da sogliatura di Otsu; e (d) normalizzazione dello sfondo in scala di grigi, senza sogliatura. Il confronto evidenzia che la binarizzazione esterna non sempre favorisce il riconoscimento, poiché Tesseract esegue già internamente la propria binarizzazione adattiva.

Perché il preprocessamento migliora l'OCR?

Le prestazioni dell'OCR dipendono direttamente dalla qualità dell'immagine in ingresso. Basso contrasto, illuminazione non uniforme, rumore e distorsioni geometriche rendono difficile la separazione tra testo e sfondo e aumentano la probabilità di errori di riconoscimento.
Tecniche come la normalizzazione dello sfondo e l'equalizzazione adattiva (CLAHE) correggono i gradienti di illuminazione e migliorano il contrasto locale tra i caratteri e lo sfondo, producendo immagini più adatte al riconoscimento automatico. La sogliatura, d'altro canto, deve essere applicata con cautela: trattandosi di un'operazione irreversibile, può eliminare variazioni sottili di intensità sui bordi dei caratteri — come l'*antialiasing* — che il motore OCR stesso utilizza internamente per risolvere ambiguità tra simboli visivamente simili. Per questo motivo, immagini in scala di grigi, corrette solo per quanto riguarda l'illuminazione, spesso producono trascrizioni più fedeli rispetto alle loro versioni binarizzate.
Nei documenti acquisiti tramite fotocamere di dispositivi mobili, il preprocessamento tende a offrire un maggiore guadagno di prestazioni rispetto ai documenti digitalizzati con *scanner*, nei quali l'illuminazione è solitamente più uniforme.

6.6 Traduzione Automatica del Testo Riconosciuto

Dopo il riconoscimento ottico dei caratteri, il testo ottenuto può essere sottoposto a tecniche di elaborazione del linguaggio naturale, come correzione ortografica, indicizzazione, riassunzione e traduzione automatica.

La traduzione automatica costituisce una fase indipendente dall'OCR. Mentre l'OCR converte i caratteri presenti nell'immagine in testo codificato, la traduzione opera su tale testo nella lingua originale del documento. In questo modo, errori di riconoscimento possono essere propagati alla traduzione, compromettendo la qualità del risultato. I sistemi attuali di traduzione automatica utilizzano prevalentemente architetture neurali basate su meccanismi di attenzione e *transformer* [Bahdanau; Cho; Bengio (2015); Vaswani (2017)].

In questo esempio, si utilizza il testo ottenuto dall'immagine sottoposta solo alla normalizzazione dello sfondo, senza sogliatura (punto **d** della Figura 6.3), poiché presenta la trascrizione più fedele tra le strategie valutate nella sezione precedente.

La traduzione è realizzata tramite la libreria `deep-translator` (installazione: `pip install deep-translator`), che fornisce un'interfaccia per diversi servizi di traduzione automatica, incluso Google

Translate.

```
from deep_translator import GoogleTranslator
import shutil as _sh

# Testo ottenuto dall'OCR dall'immagine con normalizzazione dello sfondo (toni di grigio)
texto_en = texto_norm_gray

if _sh.which("tesseract") is None:
    # Ambiente di build senza Tesseract installato (vedi cella precedente): testo_en
    # è già il segnaposto di OCR non disponibile, non testo reale - salta la traduzione
    # invece di fallire cercando di tradurre una stringa che non è realmente inglese.
    texto_pt = "[Tradução indisponível neste ambiente - Tesseract OCR ausente]"
else:
    texto_pt = GoogleTranslator(source="en", target="pt").translate(texto_en)

print("--- Testo originale generato dall'immagine normalizzata in toni di grigio (OCR, EN)
---")
print(texto_en)

print("\n--- Testo tradotto (PT-BR) ---")
print(texto_pt)

mm.show(
    [img_norm],
    titles=["Immagine con normalizzazione dello sfondo (toni di grigio)"],
    cols=1,
    figsize=(6, 4)
)
```

```
--- Testo originale generato dall'immagine normalizzata in toni di grigio (OCR, EN) ---
[Tesseract OCR indisponivel neste ambiente - texto omitido]

--- Testo tradotto (PT-BR) ---
[Tradução indisponível neste ambiente - Tesseract OCR ausente]
```

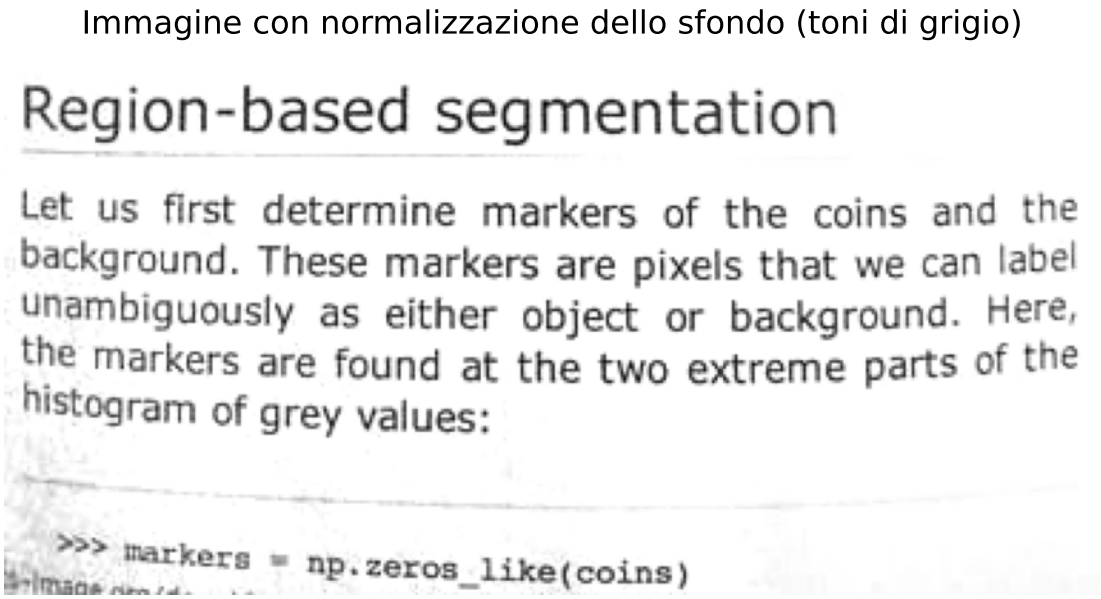


Figura 6.4: Flusso di riconoscimento e traduzione automatica. L'immagine preprocessata con normalizzazione dello sfondo, senza sogliatura, viene utilizzata come input per Tesseract OCR, e il testo riconosciuto viene tradotto dall'inglese al portoghese tramite la libreria *deep-translator*.

🧠 Perché la qualità dell'OCR influenza la traduzione?

La traduzione automatica utilizza come input il testo prodotto dall'OCR. Gli errori di riconoscimento, come caratteri errati, parole incomplete o frammentate, vengono propagati alla fase di traduzione e possono alterare il significato del testo.
Di conseguenza, la qualità della traduzione dipende direttamente dalla fedeltà della trascrizione ottenuta tramite OCR. Come discusso in precedenza, tale fedeltà non è sempre massimizzata da una binarizzazione esterna: le immagini in scala di grigi, corrette solo per quanto riguarda l'illuminazione, possono preservare informazioni rilevanti per la distinzione tra caratteri visivamente simili. Pertanto, la pre-elaborazione dell'immagine — e la scelta adeguata delle sue fasi in funzione del compito successivo — contribuisce a migliorare non solo il riconoscimento dei caratteri, ma anche le prestazioni delle fasi successive di elaborazione del linguaggio naturale, come traduzione, indicizzazione e riassunto.

6.7 Fondamenti di OMR e Ispezione Industriale

Il **Riconoscimento Ottico di Marcature** (*Optical Mark Recognition* — OMR) è una tecnica di Visione Artificiale destinata all'identificazione automatica di marcature in posizioni precedentemente definite di un modulo. Le sue applicazioni includono fogli di risposta, questionari, moduli amministrativi e altri documenti strutturati.

A differenza dell'OCR (*Optical Character Recognition*), che riconosce caratteri e parole, l'OMR determina la presenza, l'assenza o l'intensità delle marcature in regioni precedentemente note. Invece di interpretare il testo, sfrutta proprietà geometriche e statistiche associate alla compilazione di tali regioni.

I moderni sistemi OMR elaborano immagini acquisite da *scanner*, fotocamere o dispositivi mobili, automatizzando attività che in precedenza dipendevano da apparecchiature specializzate.

In generale, un sistema OMR comprende le seguenti fasi:

1. **Acquisizione:** conversione del documento fisico in formato digitale;
2. **Pre-elaborazione:** correzione geometrica, riduzione del rumore e binarizzazione;
3. **Localizzazione delle regioni di interesse:** identificazione delle aree destinate alle marcature;
4. **Analisi delle marcature:** valutazione della compilazione delle regioni candidate;
5. **Interpretazione:** conversione delle marcature in risposte o dati strutturati.

Questi principi si estendono naturalmente all'**Ispezione Industriale Automatizzata**. Nelle linee di produzione, la stessa sequenza — acquisizione, pre-elaborazione, segmentazione, estrazione delle caratteristiche e decisione — viene impiegata per rilevare difetti superficiali, verificare l'integrità dei componenti e misurare dimensioni con precisione subpixel. La differenza risiede nel dominio di applicazione: mentre l'OMR opera su documenti con struttura predefinita, l'ispezione industriale gestisce oggetti le cui variazioni geometriche e radiometriche devono essere modellate in modo più flessibile.

Nelle sezioni successive, entrambe le applicazioni vengono sviluppate attraverso progetti pratici che riproducono fasi tipiche dei sistemi reali.

6.8 Progetti Pratici: Costruzione di un *Pipeline* di Analisi Documentale

I concetti di questo capitolo verranno sviluppati attraverso progetti che riproducono fasi tipiche dei sistemi reali di analisi documentale, introducendo tecniche riutilizzabili in applicazioni di OCR, OMR, ispezione visiva e elaborazione di moduli.

6.8.1 Allineamento Automatico dei Documenti (*OCR/OMR Pre-processing*)

La correzione dell'inclinazione (*deskew*) è una fase fondamentale nell'elaborazione dei documenti. Le rotazioni introdotte durante la digitalizzazione o l'acquisizione compromettono la localizzazione delle regioni di interesse e riducono la precisione delle fasi successive.

In questo progetto verrà sviluppato un sistema per stimare automaticamente l'orientamento predominante del documento e correggerne l'inclinazione. A tal fine, verranno impiegate tecniche classiche di rilevamento dei bordi con l'operatore di Canny e il rilevamento delle linee tramite la Trasformata di Hough. A partire dalle linee identificate, verrà stimato l'angolo di rotazione e applicata una trasformazione affine per produrre una versione allineata del documento.

Poiché moduli e fogli di risposta vengono spesso distribuiti in formato PDF, il *pipeline* inizia con la rasterizzazione di ogni pagina, convertendola in un'immagine matriciale. In questo capitolo, tale fase verrà eseguita con la libreria `pdf2image`, generando immagini PNG con risoluzione di 300 DPI (*dots per inch*). A partire da esse, potranno essere applicate le tecniche di rilevamento dei bordi, Trasformata di Hough, segmentazione, estrazione dei contorni e riconoscimento automatico dei pattern studiate nel corso del capitolo.

```
import os
import urllib.request
from pdf2image import convert_from_path
from skimage import data
import cv2

# Directory dei microdati e dei fogli di risposta dell'esame istituzionale
file_path = "dados/provas_qrcode_EP.pdf"
url_github = (
    "https://raw.githubusercontent.com/fzampirolli/"
    "pdi-vc/master/all/cap06/dados/provas_qrcode_EP.pdf"
)

# Se il file non esiste localmente, viene scaricato automaticamente da GitHub
if not os.path.exists(file_path):
    print(f"[DOWNLOAD] Download del PDF da GitHub: {url_github}")
    try:
        # Assicura che la cartella 'dati' esista prima di salvare
        os.makedirs(os.path.dirname(file_path), exist_ok=True)
        urllib.request.urlretrieve(url_github, file_path)
        print("[DOWNLOAD] PDF scaricato con successo!")
    except Exception as e:
        print(f"[DOWNLOAD] Errore durante il download del file: {e}")

print(f"PDF dei fogli di prova digitalizzati: {file_path}")

if os.path.exists(file_path):
    # Rasterizzazione delle pagine con risoluzione ottimizzata di 300 DPI
    pages = convert_from_path(file_path, dpi=300)
    for i, page in enumerate(pages):
        saida = f"test{i+1:02d}.png"
        page.save(saida)
        print(f"[INGESTIONE] Pagina PDF convertita con successo: {saida}")
else:
    print("[AVVISO] File PDF non trovato nel percorso. Attivazione del fallback tramite
    skimage.data.")
    # Inietta la matrice di testo pubblica per garantire l'esecuzione continua della pipeline
    img_fallback = data.text()
    cv2.imwrite("test02.png", img_fallback)
    print("[INGESTIONE] Immagine di fallback strutturata: test02.png")

# Carica e mostra l'immagine rasterizzata iniziale utilizzando l'ecosistema morph
if os.path.exists('test02.png'):
    img_original = mm.read('test02.png')
else:
    # Fallback definitivo nel caso in cui anche skimage fallisca
    img_original = np.ones((400, 400), dtype=np.uint8) * 255
```

```
mm.show(img_original, figsize=(4, 3))
```

```
PDF dei fogli di prova digitalizzati: dados/provas_qrcode_EP.pdf
[INGESTIONE] Pagina PDF convertita con successo: test01.png
[INGESTIONE] Pagina PDF convertita con successo: test02.png
[INGESTIONE] Pagina PDF convertita con successo: test03.png
```

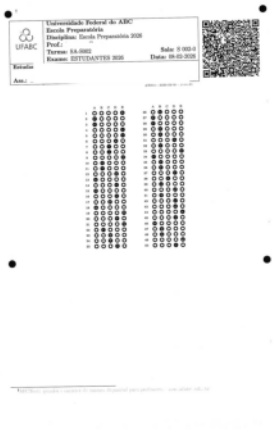


Figura 6.5: *Pipeline* di ingestione dei documenti: rasterizzazione adattiva delle pagine PDF in matrici discrete in formato PNG, mostrando la seconda pagina.

6.8.2 Algoritmo di Rettifica dell'Inclinazione (*Deskew*)

La fase di *deskew* ha come obiettivo stimare e correggere l'inclinazione globale di un documento digitalizzato, allineando il suo contenuto agli assi dell'immagine. La Figura 6.7 presenta il flusso completo di elaborazione, dall'immagine originale fino al risultato dopo la correzione geometrica. In modo complementare, il simulatore della Figura 6.6 consente di visualizzare il funzionamento della Trasformata di Hough e di comprendere come viene stimata l'orientazione predominante.

La procedura è composta da tre fasi principali:

1. rilevamento dei bordi tramite l'operatore di Canny;
2. stima dell'orientazione predominante mediante la Trasformata di Hough Lineare;
3. correzione dell'inclinazione utilizzando una trasformazione affine di rotazione.

Dopo la rettifica, il documento presenta un'orientazione approssimativamente orizzontale, favorendo le fasi successive di segmentazione, etichettatura delle componenti connesse e riconoscimento di caratteri e marcatori.

6.8.3 Modellazione Matematica

Le sottosezioni seguenti formalizzano, in termini matematici, le fasi precedentemente descritte, mettendo in relazione il gradiente dell'immagine, la parametrizzazione delle rette nello spazio di Hough e la matrice di rotazione impiegata nella correzione geometrica.

6.8.3.1 Rilevamento dei Bordi

Inizialmente, l'immagine viene levigata tramite un filtro gaussiano, riducendo l'effetto dei rumori ad alta frequenza che possono generare bordi spurî. I concetti di filtraggio spaziale e convoluzione sono stati presentati nel **Capitolo 3**.

Successivamente, l'operatore di Canny stima il gradiente dell'immagine. Siano $f(x, y)$ l'intensità dell'immagine e α l'angolo di rotazione.

La magnitudine del gradiente è data da

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}.$$

dove:

- $f(x,y)$ rappresenta l'intensità dell'immagine nella posizione (x,y);
- $\frac{\partial f}{\partial x}$ e $\frac{\partial f}{\partial y}$ sono le derivate parziali nelle direzioni orizzontale e verticale;
- $|\nabla f(x,y)|$ è la magnitudine del gradiente.

Dopo il calcolo del gradiente, l'algoritmo applica la soppressione dei non massimi (*non-maximum suppression*) e la sogliatura per isteresi, producendo un'immagine binaria contenente i principali bordi del documento.

6.8.3.2 Trasformata di Hough

L'immagine binaria dei bordi viene elaborata tramite la Trasformata di Hough Lineare, il cui obiettivo è rilevare strutture approssimativamente rettilinee. Invece della rappresentazione cartesiana della retta, $y = ax + b$, si utilizza la rappresentazione in forma normale,

$$\rho = x\cos\theta + y\sin\theta,$$

dove:

- x e y sono le coordinate di un punto appartenente alla retta;
- ρ è la distanza perpendicolare tra la retta e l'origine del sistema di coordinate dell'immagine;
- θ è l'angolo formato tra la normale alla retta e l'asse orizzontale dell'immagine.

In questa rappresentazione, ogni punto di bordo (x,y) genera una curva nello spazio dei parametri (ρ,θ). L'intersezione delle curve prodotte dai punti appartenenti alla stessa retta origina massimi in una matrice bidimensionale denominata **accumulatore**. Pertanto, i picchi dell'accumulatore corrispondono alle rette predominanti dell'immagine, come i bordi del documento, le linee dei moduli o le righe di testo.

Per stimare l'inclinazione globale del documento, si considerano solamente le rette i cui angoli soddisfano

$$-45^\circ \leq \theta \leq 45^\circ.$$

Questa restrizione elimina orientamenti incompatibili con la disposizione attesa del documento e riduce l'influenza di rette verticali o di strutture irrilevanti. Sia $\theta_1, \theta_2, \ldots, \theta_n$ l'insieme degli angoli delle rette selezionate. La stima dell'inclinazione globale si ottiene mediante la mediana,

$$\hat{\theta} = \text{med}\left(\theta_1, \theta_2, \ldots, \theta_n\right),$$

dove:

- θ_i è l'angolo della i-esima retta rilevata dalla Trasformata di Hough;
- n è il numero di rette considerate dopo il filtraggio angolare;
- $\hat{\theta}$ è la stima dell'inclinazione globale del documento.

La mediana viene adottata perché è meno sensibile alla presenza di rilevamenti isolati (*outlier*) rispetto alla media aritmetica, producendo una stima più stabile dell'orientamento predominante.

6.8.3.3 Rotazione Affine

Sia $\hat{\theta}$ l'inclinazione stimata nella fase precedente. La correzione geometrica consiste nell'applicare una trasformazione affine di rotazione attorno al centro dell'immagine, affinché l'orientamento predominante coincida con l'asse orizzontale. Le trasformazioni affini sono state studiate nel **Capitolo 2**, insieme alle operazioni di traslazione, scala, taglio e rotazione.

Sia (x, y) la posizione di un pixel rispetto al centro dell'immagine e (x', y') la sua posizione dopo la rotazione. La trasformazione è descritta da

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R(\alpha) \begin{bmatrix} x \\ y \end{bmatrix},$$

dove

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix},$$

essendo:

- (x, y) le coordinate originali del pixel rispetto al centro dell'immagine;
- (x', y') le coordinate del pixel dopo la rotazione;
- α l'angolo di rotazione applicato per compensare l'inclinazione stimata del documento;
- $R(\alpha)$ la matrice di rotazione.

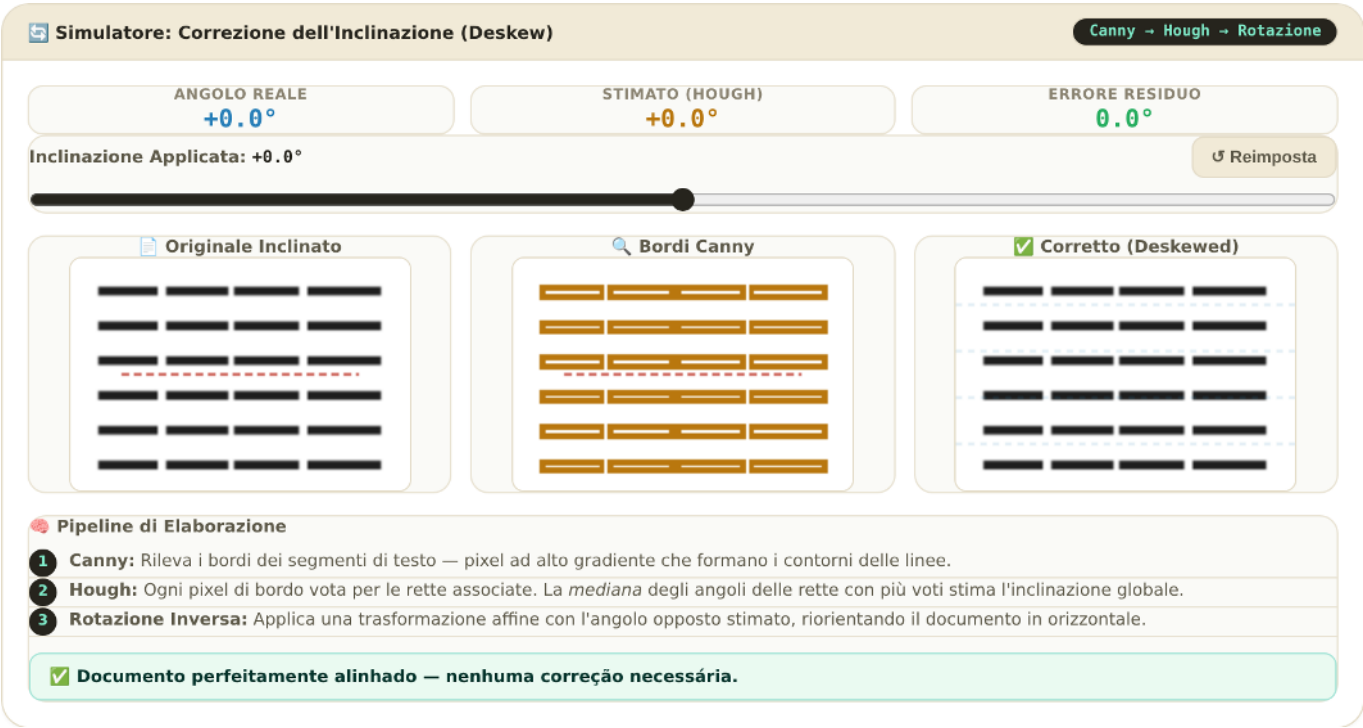
In pratica, l'angolo applicato corrisponde all'opposto dell'inclinazione stimata,

$$\alpha = -\hat{\theta},$$

dove $\hat{\theta}$ rappresenta l'orientamento predominante ottenuto tramite la Trasformata di Hough.

Poiché le coordinate trasformate non sempre coincidono con posizioni intere della griglia di pixel, è necessario ricampionare l'immagine per determinare i nuovi valori di intensità. Nell'implementazione presentata in questo capitolo, la funzione `mm.rotate` esegue questa operazione utilizzando l'interpolazione bicubica, riducendo gli artefatti di ricampionamento e preservando la continuità visiva di bordi e caratteri.

```
def retificar_inclinacao_documento(img):

    gray = mm.gray(img) if img.ndim == 3 else img
    edges = cv2.Canny(cv2.GaussianBlur(gray, (5,5), 0), 50, 150)
    lines = cv2.HoughLines(edges, 1, np.pi/180, 200)

    if lines is None:
        return edges, img

    angulos = []
    for line in lines:
        angulo = np.rad2deg(line[0][1]) - 90
        if -45 < angulo < 45:
            angulos.append(angulo)

    if not angulos:
        return edges, img

    return edges, mm.rotate(img, np.median(angulos), interp="bicubic")

# Esecuzione del pipeline di deskew
img_edges, img_final = retificar_inclinacao_documento(img_original)

# Visualizzazione multipla standardizzata con il formato nativo del libro
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-06-deskew.html

Figura 6.6: Simulatore interattivo di correzione dell'inclinazione (*deskew*): sposta lo *slider* per inclinare il documento e osserva le tre fasi del *pipeline* — immagine inclinata, bordi Canny e risultato corretto.

```
mm.show(
    [img_original, img_edges, img_final],
    titles=["Immagine Originale", "Bordi di Canny", "Documento Rettificato"],
    cols=3,
    figsize=(12, 4)
)
```

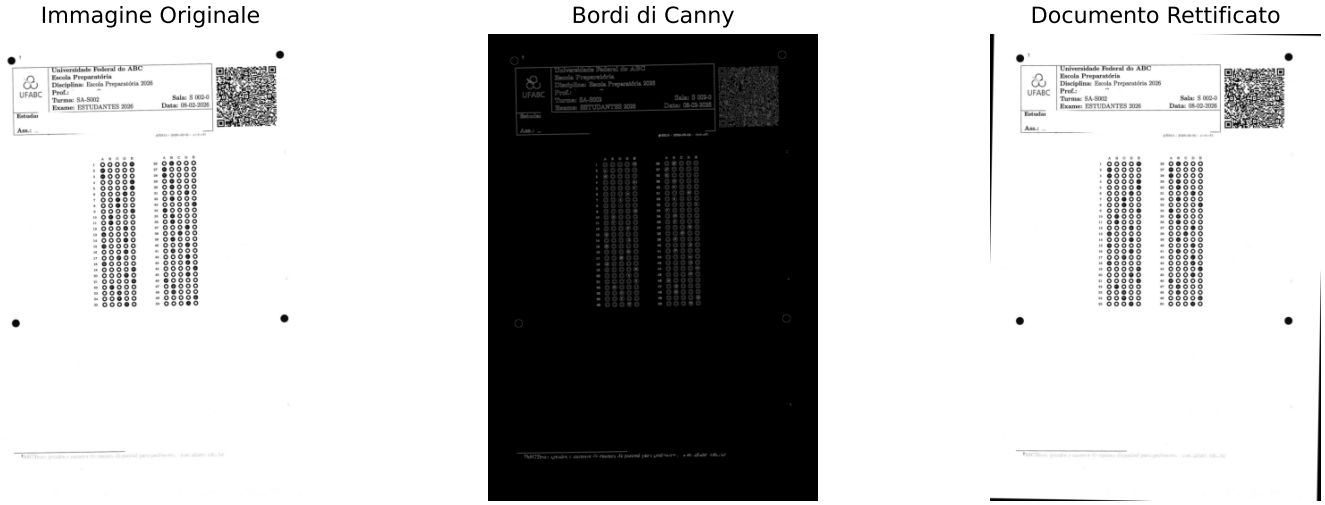


Figura 6.7: *Pipeline* di rettificazione assiale: visualizzazione comparativa tra l'ingresso ruotato originale, la mappa dei gradienti strutturali di Canny e il risultato finale allineato con sfondo normalizzato in bianco.

🧠 Perché funziona? — Trasformata di Hough

Nella Trasformata di Hough, ogni pixel di bordo contribuisce con voti per tutte le rette che possono passare per la sua posizione. Invece di selezionare solo la retta con il maggior numero di voti, l'algoritmo considera tutte le rette la cui quantità di voti supera una soglia minima e calcola i rispettivi angoli. L'inclinazione globale del documento viene quindi stimata mediante la mediana di tali angoli, una misura robusta rispetto ai valori anomali. Così, le rette spurie prodotte da ombre, rumori o altri elementi dell'immagine esercitano scarsa influenza sulla stima finale, purché la maggior parte delle rette rilevate corrisponda ai bordi del documento.

6.8.4 Limiti Pratici

Sebbene presenti buone prestazioni in condizioni usuali di digitalizzazione, il metodo dipende dall'esistenza di strutture lineari sufficientemente definite per essere rilevate dalla Trasformata di Hough, come i bordi della pagina, le linee dei moduli o le righe di testo. La sua precisione può ridursi in immagini a bassa risoluzione, con rumore eccessivo, ombre intense o inclinazioni accentuate. In generale, documenti digitalizzati con una risoluzione prossima a 300 DPI e illuminazione omogenea forniscono risultati adeguati per applicazioni OCR e OMR.

L'implementazione presentata in questo capitolo ha finalità didattica, illustrando i principi della correzione automatica dell'inclinazione tramite il rilevamento dei bordi, la Trasformata di Hough e la rotazione affine. Poiché utilizza solo l'orientamento delle strutture lineari predominanti, il metodo può essere applicato a diversi tipi di documenti, senza dipendere da marcatori specifici.

Nei sistemi reali di analisi documentale, tuttavia, l'allineamento utilizza generalmente marcatori geometrici preventivamente noti. Nel modello di foglio di risposta impiegato dall'ecosistema MCTest, ad esempio, vengono utilizzati quattro dischi neri di riferimento, oltre alle regioni corrispondenti all'intestazione, al *QRCode* e ai riquadri delle risposte. La localizzazione di questi elementi consente di stimare simultaneamente rotazione, scala e traslazione del foglio, rendendo la registrazione meno sensibile alla quantità di testo, all'assenza di linee strutturali e alle variazioni di stampa o digitalizzazione.

Per questo motivo, l'approccio basato sulla Trasformata di Hough viene utilizzato in questo capitolo per introdurre i fondamenti del problema, mentre le fasi successive adottano l'allineamento tramite marcatori geometrici, strategia predominante nei sistemi OMR e nell'analisi documentale.

6.8.5 Rilevamento dei Bordi e dei Contorni

La localizzazione precisa delle regioni di interesse è una fase essenziale nei sistemi OMR. Nel modello di foglio di risposta utilizzato in questo capitolo, l'intestazione e il riquadro delle risposte sono contenuti in un rettangolo virtuale delimitato da quattro dischi neri posizionati agli angoli. L'identificazione di questi marcatori consente di localizzare la regione di interesse e correggere le distorsioni geometriche introdotte durante l'acquisizione dell'immagine.

La procedura si compone di cinque fasi. Inizialmente, si applica una **chiusura morfologica** (dilatazione seguita da erosione), operazione studiata nel **Capitolo 4**, utilizzando un elemento strutturante a disco (`mm.sedisk(33)`). Questa operazione riduce le piccole discontinuità e preserva i dischi di riferimento, rendendoli più omogenei. Successivamente, l'immagine viene invertita (`mm.neg`), così che i dischi vadano a costituire componenti chiare su sfondo scuro.

Nella fase successiva, si applica l'operazione `mm.edgeoff` (**Capitolo 4**), che rimuove i componenti connessi ai bordi dell'immagine, eliminando artefatti come ombre di scansione, segni di taglio e altri oggetti spuri ai margini. I componenti rimanenti vengono quindi analizzati a partire dai loro contorni e filtrati in base a proprietà geometriche, come area e circolarità, per identificare i dischi candidati. L'implementazione di MCTest rende questo processo più robusto selezionando, tra tutti i candidati, i quattro i cui centri formano un rettangolo con larghezza compatibile con quella dell'immagine, riducendo così il verificarsi di falsi positivi.

Infine, i centri dei quattro dischi vengono ordinati spazialmente (in alto a sinistra, in alto a destra, in basso a sinistra e in basso a destra) e utilizzati come punti di controllo in una trasformazione di prospettiva (*perspective warp*). Questa trasformazione raddrizza l'immagine, producendo una rappresentazione allineata e con dimensioni note, adatta alle fasi successive di segmentazione e riconoscimento.

Le principali fasi di questo *pipeline*, dal processamento morfologico fino all'immagine raddrizzata, sono illustrate di seguito.

```
import cv2
import numpy as np
from morph import mm

# img: immagine in scala di grigi del foglio di prova
if img_final.ndim == 2:
    img = img_final
else:
    img = mm.gray(img_final)

# 1. Chiusura morfologica: preserva i dischi scuri, rimuovendo tutto ciò che è più piccolo del
disco
img_close = mm.close(img, mm.sedisk(41))

# 2. Inversione: i dischi scuri diventano componenti chiari su sfondo scuro
img_neg = mm.neg(img_close)

# 3. Rimuove i componenti connessi che toccano il bordo dell'immagine
img_edgeoff = mm.edgeoff(img_neg)

# 4. Estrazione dei contorni esterni
contornos, _ = cv2.findContours(img_edgeoff, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# 5. Filtraggio per area e circolarità, mantenendo solo i 4 dischi
centros = []
for c in contornos:
    area = cv2.contourArea(c)
    perimetro = cv2.arcLength(c, True)
    if area < 50 or perimetro == 0:
        continue
    circularidade = 4 * np.pi * area / (perimetro ** 2)
    if circularidade > 0.8:
        M = cv2.moments(c)
        cx, cy = M["m10"] / M["m00"], M["m01"] / M["m00"]
        centros.append((cx, cy))

# Verifica robusta: interrompe il pipeline con un messaggio chiaro invece di AssertionError
if len(centros) != 4:
    print(f"[AVVISO] Attesi 4 dischi marcatori, trovati {len(centros)}.")
    print("  Verifica se l'immagine è un foglio di risposte MCTest valido")
    print("  o regola i parametri di circolarità e area minima.")
    img_retificada = img  # fallback: preserva l'immagine senza raddrizzamento
else:
    # 6. Ordinamento dei centri: in alto a sinistra, in alto a destra,
    # in basso a sinistra, in basso a destra
    pts = np.array(centros, dtype=np.float32)
    soma = pts.sum(axis=1)
    diff = pts[:, 0] - pts[:, 1]
    tl = pts[np.argmin(soma)]
    br = pts[np.argmax(soma)]
    tr = pts[np.argmax(diff)]
    bl = pts[np.argmin(diff)]
    pts_ordenados = np.array([tl, tr, bl, br], dtype=np.float32)
```

```
    # 7. Raddrizzamento tramite trasformazione prospettica (warp)
    largura, altura = 800, 800
    destino = np.array(
        [[0, 0], [largura, 0], [0, altura], [largura, altura]], dtype=np.float32
    )
    M_persp = cv2.getPerspectiveTransform(pts_ordenados, destino)
    img_retificada = cv2.warpPerspective(img, M_persp, (largura, altura))

    mm.show(
        [img_close, img_edgeoff, img_retificada],
        titles=["Chiusura (sedisk 41)", "edgeoff", "Raddrizzata (warp)"],
        cols=3,
        figsize=(12, 4)
    )

mm.write(img_retificada, "img_beetween_disks.png")
```

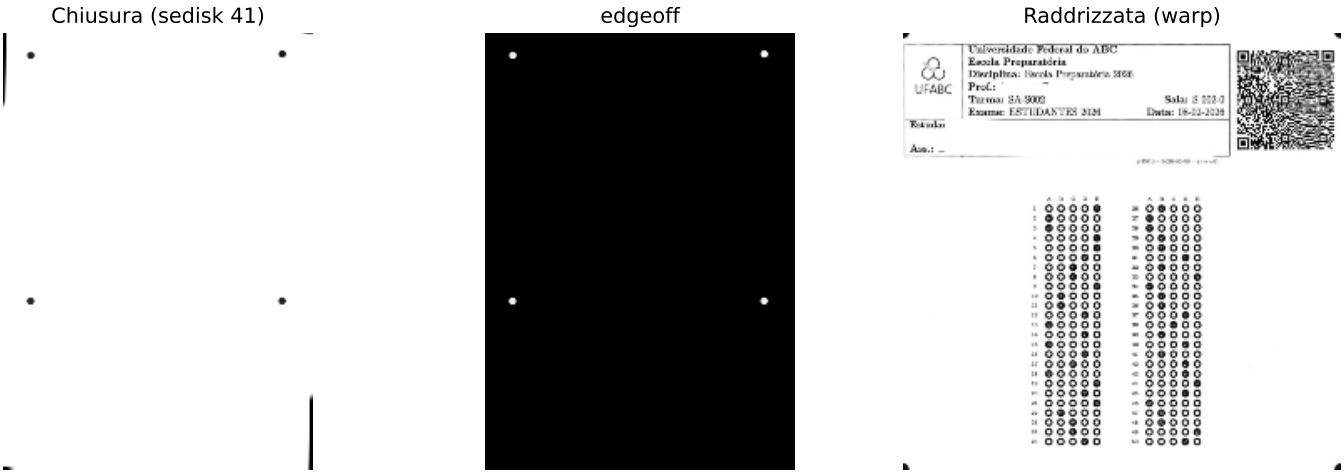
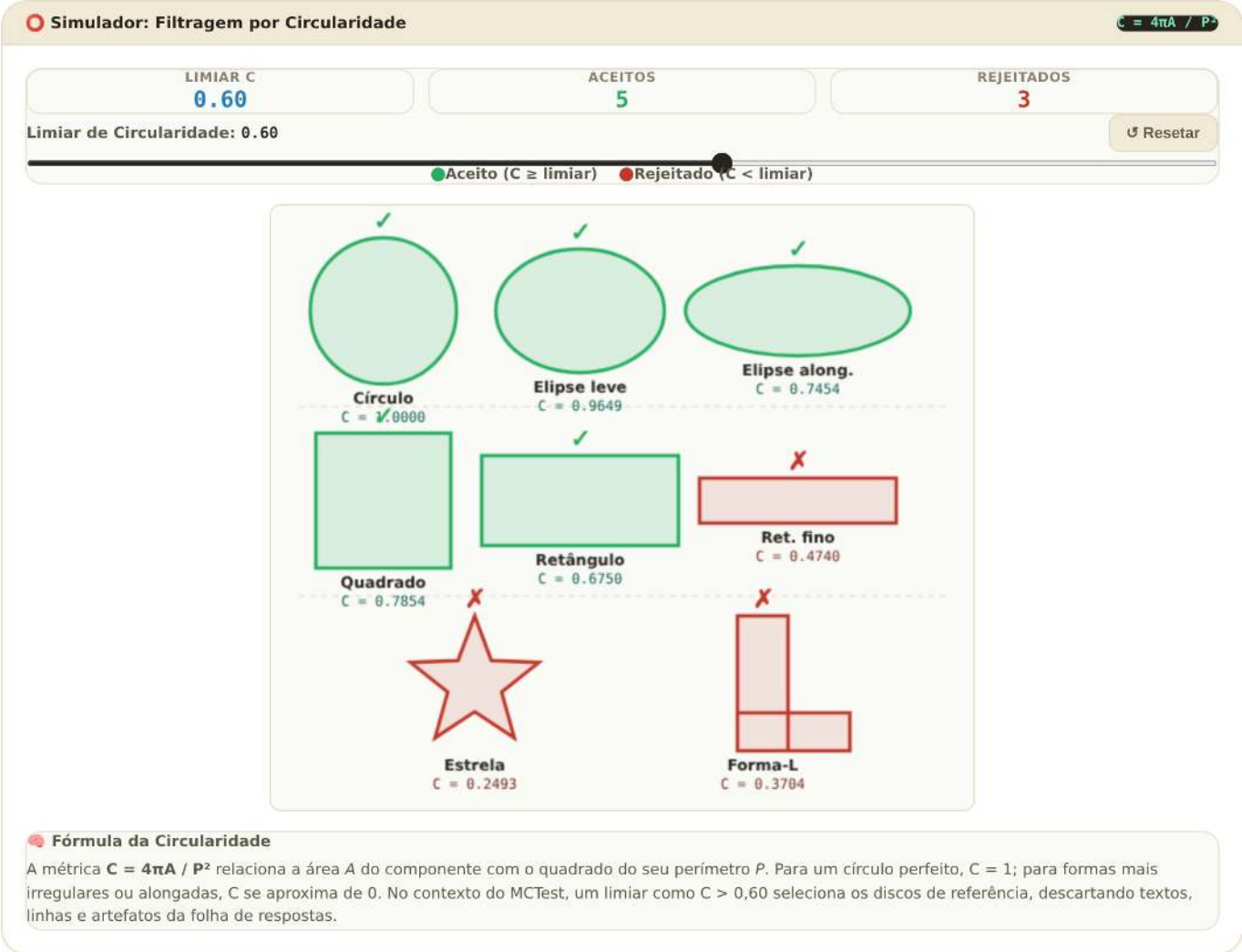


Figura 6.8: Rilevamento dei dischi marcatori, estrazione dei contorni e raddrizzamento tramite trasformazione prospettica.

Perché funziona? — Dalla chiusura morfologica alla rettifica

Chiusura morfologica: la dilatazione seguita dall'erosione riempie piccole discontinuità e smussa i contorni degli oggetti senza alterare significativamente la loro forma complessiva. Utilizzando un elemento strutturante grande (`sedisk(41)`), dettagli sottili, come testi, linee del modulo e piccoli rumori, tendono a essere incorporati nello sfondo durante l'elaborazione, mentre oggetti di scala maggiore, come i dischi di riferimento, rimangono preservati e diventano più omogenei.
Circolarità: dopo l'isolamento dei componenti candidati, la metrica $C = \frac{4\pi A}{P^2}$ quantifica quanto la loro forma si avvicini a un cerchio. Il suo valore è pari a 1 per un cerchio perfetto e diminuisce man mano che il contorno diventa più irregolare. Pertanto, una soglia come $C > 0{,}8$ consente di scartare la maggior parte dei falsi positivi senza ricorrere a modelli di apprendimento. Un simulatore di questa metrica è presentato nella Figura 6.9.
Trasformazione di prospettiva (*perspective warp*): una volta identificati i quattro dischi di riferimento, i loro centri vengono utilizzati come punti di controllo per stimare la trasformazione proiettiva che mappa l'immagine acquisita sul piano del documento. Questa trasformazione corregge le distorsioni introdotte dalla prospettiva durante l'acquisizione dell'immagine, producendo una rappresentazione frontale con dimensioni note e adatta alle fasi successive di segmentazione e riconoscimento.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-06-circularidade.html

Figura 6.9: Simulatore interattivo di filtraggio per circularità: muovi lo *slider* per regolare la soglia C e osserva quali componenti sono accettati (verde) o rifiutati (rosso).

6.8.6 Isolamento, Segmentazione e Decodifica del *QRCode*

Dopo la rettifica geometrica del foglio di risposta, si esegue il rilevamento e la decodifica del *QRCode* presente nel modulo. Questo marcatore memorizza informazioni utilizzate dal sistema di OMR (*Optical Mark Recognition*), come l'identificazione dello studente, il codice della prova e la sua variazione, consentendo il recupero del gabarit corrispondente nel database. Per sicurezza, queste informazioni vengono crittografate prima della generazione del *QRCode*. Pertanto, la sequenza decodificata corrisponde a una *string* esadecimale, la cui interpretazione è eseguita esclusivamente dal sistema MCTest. La procedura è composta da tre fasi: preprocessamento morfologico, isolamento della regione del *QRCode* e decodifica del suo contenuto.

Inizialmente, l'immagine rettificata in scala di grigi viene binarizzata tramite l'operazione `mm.threshold`. Successivamente, si applica un'**apertura morfologica** (erosione seguita da dilatazione), studiata nel **Capitolo 4**, utilizzando un elemento strutturante quadrato (`mm.sebox(2)`). Questa operazione rimuove piccoli rumori e attenua le imperfezioni senza compromettere la struttura del marcatore. Infine, l'immagine viene invertita (`mm.neg`), in modo che il *QRCode* costituisca un componente chiaro su sfondo scuro, facilitando l'estrazione dei suoi contorni.

La localizzazione del *QRCode* viene eseguita tramite l'analisi dei contorni esterni dell'immagine binarizzata. Tra i componenti rilevati, si seleziona quello con area maggiore e geometria approssimativamente quadrata, scartando gli altri elementi stampati sul foglio. Successivamente, la regione corrispondente viene espansa con un piccolo margine di sicurezza, garantendo la conservazione integrale del marcatore.

Il *QRCode* viene quindi estratto direttamente dall'immagine rettificata in scala di grigi, preservandone la qualità radiometrica. Poiché questa regione presenta generalmente dimensioni ridotte, si applica un ridimensionamento con interpolazione cubica, aumentando la risoluzione spaziale e favorendo l'identificazione dei suoi moduli. La lettura viene eseguita dal rilevatore di *QRCode* di OpenCV (`cv2.QRCodeDetector`), che recupera la sequenza di caratteri originariamente codificata.

Il flusso completo di questo processamento, dal preprocessamento morfologico fino alla decodifica del *QRCode*, è illustrato nella Figura 6.10. Il segmento mostrato nell'output corrisponde solo all'inizio della *string* esadecimale crittografata; la sua interpretazione completa viene eseguita internamente da MCTest dopo la decodifica.

```
import cv2
import numpy as np
from morph import mm

# f: immagine raddrizzata convertita al tipo corretto a 8 bit (0-255)
f = img_retificada.astype('uint8')

# 1. Sogliatura: conversione dell'immagine in toni di grigio in binaria
f_thresh = mm.threshold(f)

# 2. Apertura morfologica: elimina piccoli rumori e uniforma il contorno dei blocchi
f_open = mm.open(f_thresh, mm.sebox(2))

# 3. Inversione morfologica: i moduli scuri diventano componenti chiari su sfondo scuro
f_inv = mm.neg(f_open)

# 4. Conversione sicura in uint8 con scala 0-255
img_uint8 = (f_inv.astype(np.uint8) * 255) if f_inv.max() == 1 else f_inv.astype(np.uint8)

# 5. Rilevamento dei contorni esterni
contornos, _ = cv2.findContours(img_uint8, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

if not contornos:
    raise ValueError("Nenhum contorno encontrado. Verifique o limiar ou a imagem de entrada.")

# 6. Filtraggio per il contorno più grande con proporzione approssimativamente quadrata
```

```
#    (aspect ratio tra 0.7 e 1.3 scarta rettangoli allungati del foglio)
def is_square_like(contorno, tol=0.3):
    x, y, w, h = cv2.boundingRect(contorno)
    ratio = w / h if h > 0 else 0
    return (1 - tol) <= ratio <= (1 + tol)

candidatos = [c for c in contornos if is_square_like(c)]

if not candidatos:
    raise ValueError(
        "Nenhum contorno quadrado encontrado. "
        "Verifique se o QRCode está presente na imagem ou ajuste a tolerância."
    )

# Seleziona il candidato quadrato più grande per area di bounding box
maior_contorno = max(candidatos, key=lambda c: cv2.boundingRect(c)[2] *
cv2.boundingRect(c)[3])
x, y, w, h = cv2.boundingRect(maior_contorno)

# 7. Espansione della bounding box con margine di sicurezza (evita troncamento del QRCode)
margem = 5
h_img, w_img = img_uint8.shape[:2]
x1 = max(x - margem, 0)
y1 = max(y - margem, 0)
x2 = min(x + w + margem, w_img)
y2 = min(y + h + margem, h_img)

# 8. Ritaglio della regione di interesse dall'immagine originale (nitida, in grigio)
img_qrcode_final = img_retificada[y1:y2, x1:x2]

# 9. Ingrandimento per risoluzione minima di decodifica (400 px sul lato maggiore)
#    cv2.QRCodeDetector richiede moduli con almeno 3-4 px di larghezza per decodificare
#    in sicurezza; immagini più piccole di ~400 px tendono a fallire.
lado = max(img_qrcode_final.shape[:2])
escala = max(400 / lado, 1.0)
img_para_leitura = cv2.resize(
    img_qrcode_final, None,
    fx=escala, fy=escala,
    interpolation=cv2.INTER_CUBIC
)

# 10. Inizializzazione del rilevatore nativo di QRCode di OpenCV
detector = cv2.QRCodeDetector()

# 11. Rilevamento geometrico e decodifica dei dati testuali
dados, pontos, qrcode_reto = detector.detectAndDecode(img_para_leitura)


# Visualizzazione intermedia: progressione dalla binarizzazione all'isolamento del QRCode
mm.show(
    [f_thresh, f_open, f_inv, img_qrcode_final],
    titles=["1. Sogliatura", "2. Apertura morfologica", "3. Inversione", "Immagine finale"],
    cols=3, figsize=(12, 4)
)

# Validazione e uscita dei metadati estratti
if dados:
    print(f"QRCode decodificato con successo: \n{dados[:50]}...")
else:
    raise ValueError(
        "Falha na decodificação do QRCode. "
```

```
        "Verifique o limiar, as margens da região ou a qualidade da imagem."
    )
```

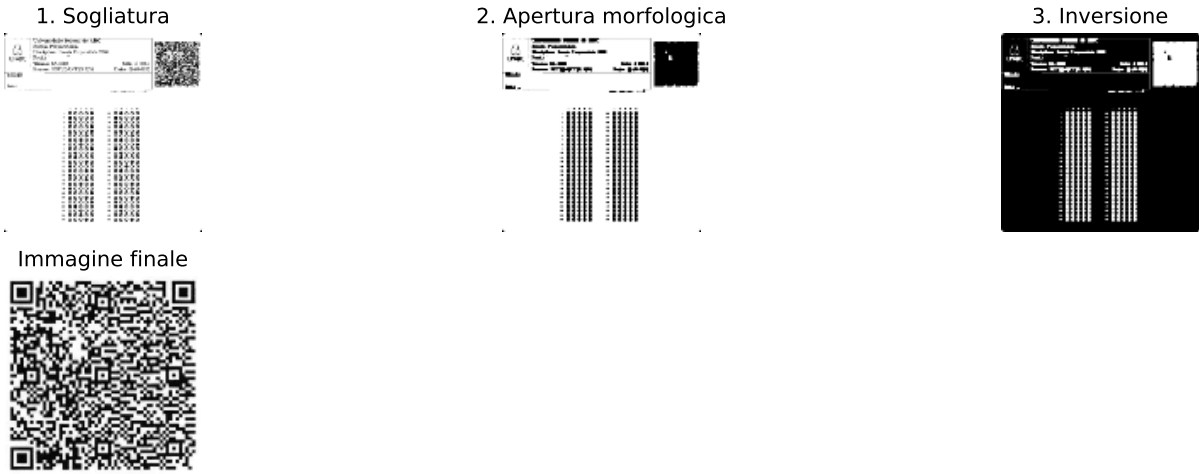


Figura 6.10: *Pipeline* di elaborazione del *QRCode*: sogliatura, apertura morfologica, inversione e ritaglio finale per la decodifica.

```
QRCode decodificato con successo:
325a356b71367266556955646b7233454149624a694f417730...
```

Perché funziona? — Isolamento e decodifica del *QRCode*

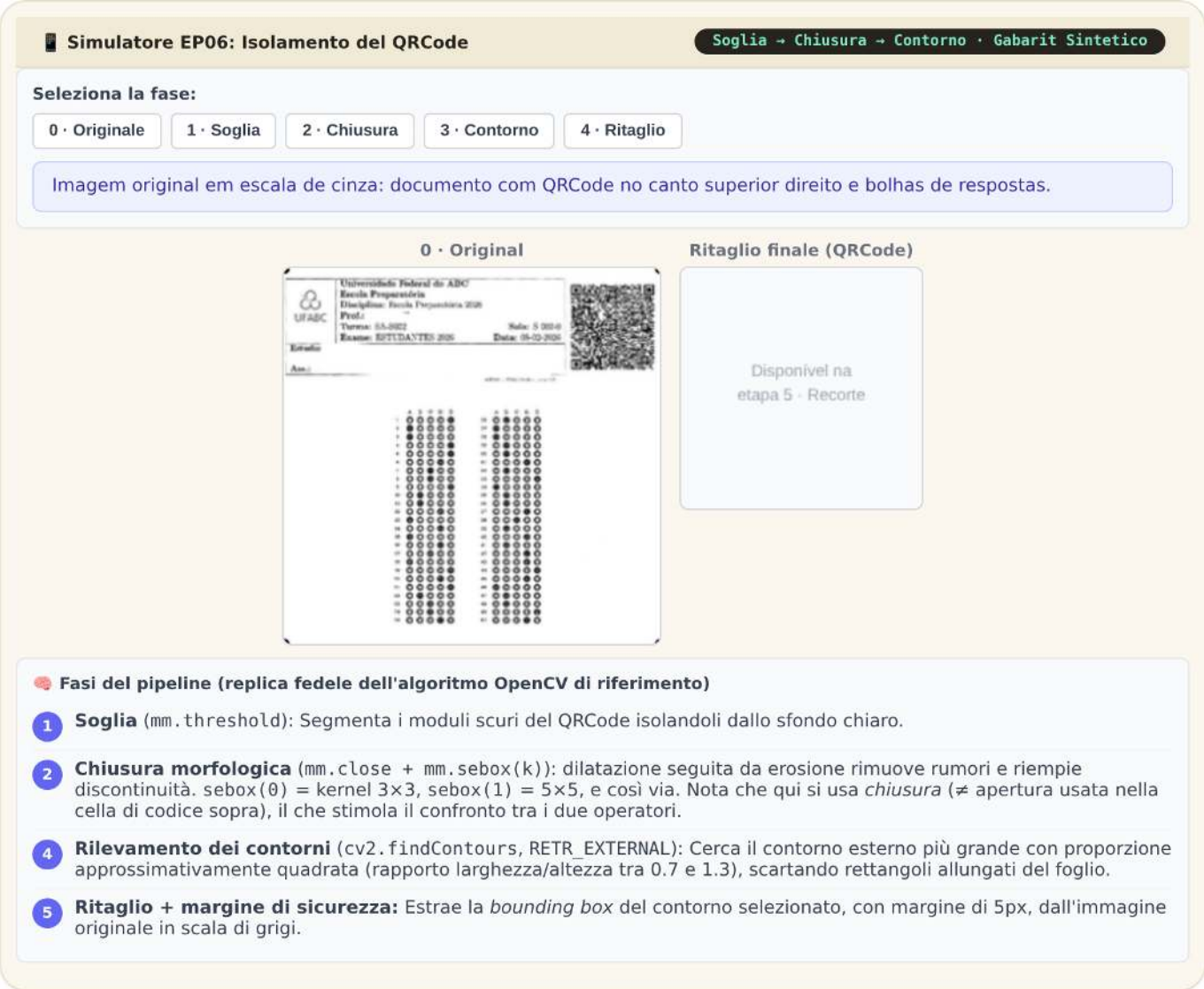
Apertura morfologica: a differenza della chiusura, l'apertura (erosione seguita da dilatazione) rimuove piccoli rumori e protuberanze senza alterare significativamente la geometria degli oggetti più grandi. In questo modo, preserva la struttura del *QRCode* mentre elimina componenti spurie che potrebbero ostacolarne la localizzazione.
Selezione per geometria: il *QRCode* ha una forma approssimativamente quadrata ($w/h \approx 1$). La combinazione di questo criterio con la selezione della componente a maggiore area scarta linee del modulo, testi e altri elementi stampati, consentendo di isolare il marcatore senza l'uso di modelli di apprendimento.
Ridimensionamento prima della decodifica: quando il *QRCode* occupa pochi pixel nell'immagine, i suoi moduli diventano difficili da distinguere. Il ridimensionamento con interpolazione cubica aumenta la risoluzione spaziale della regione di interesse, facilitando l'identificazione dei pattern del codice da parte di `cv2.QRCodeDetector` e rendendo la decodifica più robusta.

6.8.7 Decodifica dei Codici a Barre

Nella sezione precedente, la decodifica dei *codici QR* è stata eseguita utilizzando il rilevatore nativo di OpenCV (`cv2.QRCodeDetector`), che integra in un'unica interfaccia il rilevamento geometrico del simbolo, la sua rettifica e l'estrazione delle informazioni codificate.

Per i codici a barre lineari (1D), come EAN-13, Code 39 e Code 128, un'alternativa ampiamente utilizzata è la libreria `pyzbar`. A differenza di `QRCodeDetector`, essa supporta diverse simbologie di codici a barre e può essere impiegata anche nella lettura dei *codici QR*.

La procedura consiste nel localizzare automaticamente ogni simbolo presente nell'immagine e nell'interpretare la sequenza corrispondente di barre e spazi, producendo la stringa di caratteri codificata. Oltre ai dati decodificati, la libreria fornisce informazioni come la simbologia identificata e la posizione del codice nell'immagine, consentendone la successiva validazione o elaborazione.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-06-qrcode.html

Figura 6.11: Simulatore interattivo del *pipeline* di isolamento del QRCode: naviga tra le fasi di filtraggio, regola l'elemento strutturante della chiusura morfologica e osserva il rilevamento del contorno quadrato sull'immagine originale del modello.

La Figura 6.12 presenta un esempio di codice a barre e il risultato della sua decodifica utilizzando la libreria `pyzbar`.

```
import os
import urllib.request
import numpy as np
from pyzbar.pyzbar import decode
from morph import mm

barcode_path = 'dados/barcode.png'
url_github = (
    "https://raw.githubusercontent.com/fzampirolli/"
    "pdi-vc/master/all/cap06/dados/barcode.png"
)

# Se il file non esiste localmente, lo scarica automaticamente da GitHub
if not os.path.exists(barcode_path):
    print(f"[DOWNLOAD] Scaricando il codice a barre da GitHub: {url_github}")
    try:
        os.makedirs(os.path.dirname(barcode_path), exist_ok=True)
        urllib.request.urlretrieve(url_github, barcode_path)
        print("[DOWNLOAD] Immagine scaricata con successo!")
    except Exception as e:
        print(f"[DOWNLOAD] Errore nel scaricare il file: {e}")

if os.path.exists(barcode_path):
    image = mm.read(barcode_path)
else:
    # Fallback sintetico: genera un modello di barre verticali che simula un Code-128
    print("[AVVISO] File 'dati/codice_a_barre.png' non trovato.")
    print("        Uso di immagine sintetica per la dimostrazione del pipeline.")
    h, w = 100, 400
    img_synth = np.ones((h, w), dtype=np.uint8) * 255
    # Barre scure in posizioni regolari (modello semplificato)
    for x in range(20, w - 20, 8):
        if (x // 8) % 3 != 0:
            img_synth[:, x:x+4] = 0
    image = img_synth

mm.show(image)

# Esegue la decodifica di pyzbar
try:
    barcodes = decode(image)
except ImportError:
    print("[ERRORE] zbar di sistema non trovata. Esegui: !apt-get install -y libzbar0")
    barcodes = []

if barcodes:
    dados_bc = barcodes[0].data.decode("utf-8")
    tipo = barcodes[0].type
    print(f"Codice a barre decodificato con successo [{tipo}]:\n{dados_bc}")
else:
    print("[INFORMAZIONE] Nessun codice a barre rilevato nell'immagine.")
    print(
        "In immagine sintetica è previsto - "
        "sostituiscilo con il file reale per decodificare."
    )
```

Figura 6.12: Decodifica del codice a barre lineare con *pyzbar*: immagine di input e dati estratti.

```
Codice a barre decodificato con successo [EAN13]:
0000000000055
```

6.9 Il MCTest come Caso di Studio: dal Prototipo al Sistema in Produzione

Fino a questo punto, le principali fasi del *pipeline* di elaborazione sono state presentate e analizzate singolarmente, inclusa la correzione dell'inclinazione (*deskew*), la rilevazione dei marker, la rettifica prospettica e la lettura dei *QRCodes*. Sebbene questo approccio faciliti la comprensione di ciascuna tecnica, le applicazioni reali richiedono l'integrazione di queste fasi in un flusso di elaborazione unico e coerente.

Il **MCTest** costituisce un esempio di tale integrazione. Sviluppato presso l'UFABC e reso disponibile come software open source, il sistema è utilizzato dal 2012 per la correzione automatizzata delle valutazioni, offrendo supporto a diversi modelli di fogli di risposta, griglie di correzione individualizzate e generazione automatica di report sulle prestazioni (ZAMPIROLLI, 2023).

A partire da questo punto, l'attenzione non è più rivolta all'implementazione isolata degli algoritmi, bensì all'organizzazione di tali algoritmi in un'applicazione completa. Oltre alla qualità dei metodi di elaborazione delle immagini, un sistema di questa natura deve soddisfare requisiti quali robustezza in diverse condizioni di acquisizione, facilità di manutenzione e capacità di evoluzione verso nuove funzionalità.

Nelle prossime sezioni, verrà analizzato il modulo di Visione Computazionale del MCTest, implementato nel file `CVMCTest.py`. L'obiettivo è mostrare come i concetti presentati nel corso di questo capitolo vengono combinati in un *pipeline* di elaborazione impiegato in un'applicazione reale.

6.9.1 Ottenimento e Preparazione del Modulo

Il file `CVMCTest.py` integra il sistema MCTest e, nella sua versione originale, dipende da modelli, configurazioni e altri componenti del *framework* Django. Poiché tali dipendenze non sono disponibili nell'ambiente utilizzato in questo capitolo, il modulo deve essere adattato per essere eseguito in modo indipendente.

A tale scopo, il file viene ottenuto direttamente dal repository del progetto tramite la libreria `requests`. Successivamente, vengono impiegati comandi `sed` per rimuovere le importazioni e le dipendenze specifiche dell'ambiente Web, producendo una versione autonoma del modulo. Questo adattamento preserva l'implementazione degli algoritmi di Visione Artificiale, consentendone l'esecuzione e l'analisi senza la necessità di installare o configurare l'intera infrastruttura del sistema MCTest.

```python
import requests
CVMCTest = requests.get(
    "https://raw.githubusercontent.com/fzampirolli/mctest/master/exam/CVMCTest.py"
    )
with open('CVMCTest.py', 'w') as writefile:
    writefile.write(CVMCTest.text)
```

Ogni comando `sed` rimuove importazioni specifiche dall'ambiente Django, rendendo il file `CVMCTest.py` utilizzabile in modo indipendente in questo capitolo.

```python
# rimuovi righe con "form django.", ...
!sed --in-place '/from django./d' CVMCTest.py
!sed --in-place '/from exam./d' CVMCTest.py
```

```
!sed --in-place '/from mctest./d' CVMCTest.py
!sed --in-place '/from student./d' CVMCTest.py
!sed --in-place '/from topic./d' CVMCTest.py
!sed --in-place '/from .models import VariationExam/d' CVMCTest.py
```

Perché rimuovere le dipendenze da Django?

Nell'implementazione originale, il file `CVMCTest.py` fa parte di un'applicazione sviluppata con il *framework* Django e, pertanto, importa modelli, configurazioni e altri componenti specifici di tale ambiente. Poiché questi elementi non sono disponibili in questo capitolo, il modulo non può essere importato direttamente.
I comandi `sed` rimuovono soltanto queste dipendenze, senza alterare le routine di Visione Computazionale implementate nel file. In questo modo, il modulo può essere eseguito in maniera indipendente, preservando il comportamento degli algoritmi presentati.
Questa procedura illustra un principio importante dell'ingegneria del software: separare la logica dell'applicazione dall'infrastruttura in cui è inserita, facilitando il riutilizzo, i test e lo studio di componenti specifici.

6.9.2 Estrazione dell'Area delle Risposte

Dopo la lettura del foglio, la funzione `getAnswerArea` esegue automaticamente le fasi di rilevamento dei marcatori di riferimento e di rettifica prospettica presentate nelle sezioni precedenti. Come risultato, si ottiene un'immagine contenente solo la regione destinata alle risposte, allineata e con dimensioni standardizzate.

Questa standardizzazione semplifica le fasi successive di elaborazione, poiché la posizione dei campi di marcatura diventa nota e indipendente dalla posizione originale del foglio durante la scansione.

La Figura 6.13 presenta il foglio delle risposte originale in scala di grigi, mentre la Figura 6.14 mostra la regione delle risposte ottenuta dopo l'applicazione della funzione `getAnswerArea`.

```
import os
from pdf2image import convert_from_path
from skimage import data as skdata
import cv2
from morph import mm

file = "dados/provas_qrcode_EP.pdf"
MYFILES = 'extra02.qrcode'

if os.path.exists(file):
    pages = convert_from_path(file, 200)  # dpi 100=min 500=max
    numPAGES = 0
    for page in pages:
        myfile0 = MYFILES + '_p' + str(numPAGES) + '.png'
        page.save(myfile0)
        numPAGES += 1
        print(f"[ASSUNZIONE] Pagina convertita: {myfile0}")
    pages.clear()
    img_color = mm.read(myfile0)
    img_inicial = mm.gray(img_color)
else:
    print("[AVVISO] File 'dati/prove_qrcode.pdf' non trovato.")
    print("        Uso dell'immagine pubblica skimage.data.page() come sostituto.")
    img_inicial = skdata.page()

mm.show(img_inicial)
```

```
[ASSUNZIONE] Pagina convertita: extra02.qrcode_p0.png
[ASSUNZIONE] Pagina convertita: extra02.qrcode_p1.png
[ASSUNZIONE] Pagina convertita: extra02.qrcode_p2.png
```

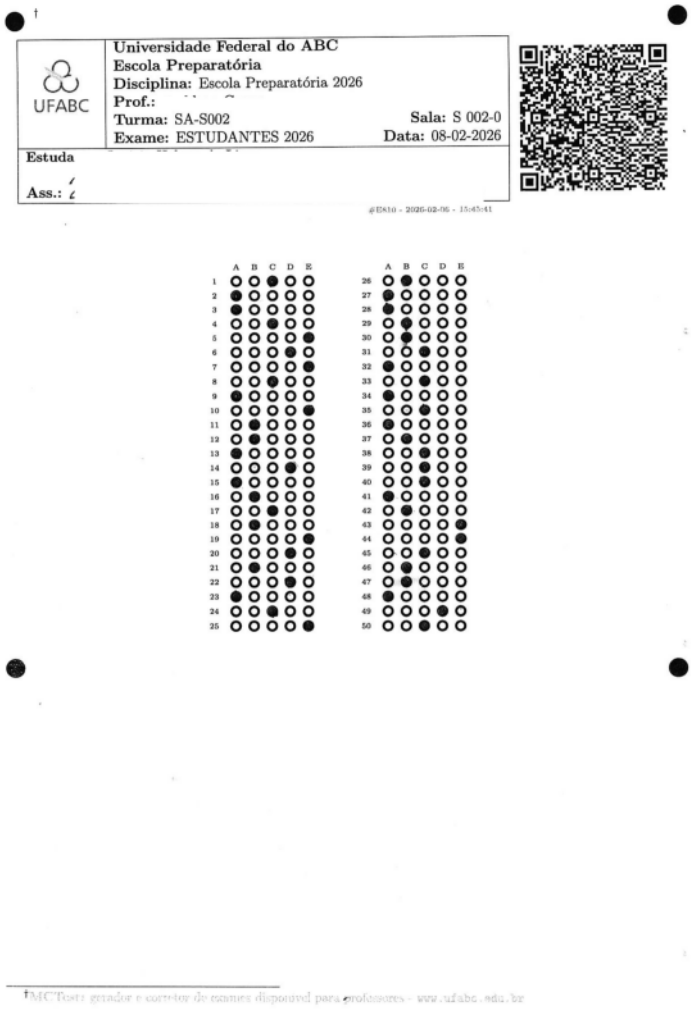


Figura 6.13: Immagine del foglio delle risposte in scala di grigio caricata dal PDF rasterizzato.

```
import CVMCTest
countPage = 0
img_getAnswerArea = CVMCTest.cvMCTest.getAnswerArea(img_inicial, countPage)
mm.show(img_getAnswerArea)
```

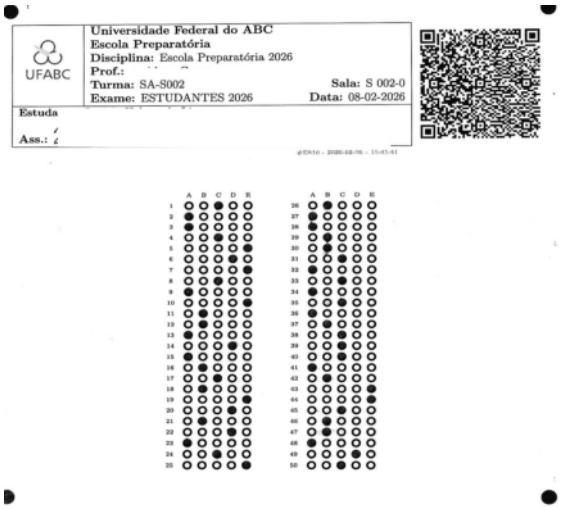


Figura 6.14: Area di risposte estratta da *getAnswerArea*: regione raddrizzata contenente i riquadri di marcatura.

Nota di compatibilità: le versioni recenti di NumPy (2.0) hanno rimosso l'alias `np.int0`. Se `CVMCTest.py` utilizza questo tipo, il comando seguente applica la correzione direttamente al file prima di ricaricarlo:

```
!sed -i 's/box = np.int0(cv2.boxPoints(rect))/box = cv2.boxPoints(rect).astype(np.intp)/' \
    ./CVMCTest.py
```

```
import importlib
import CVMCTest

importlib.reload(CVMCTest)
```

```
<module 'CVMCTest' from '/home/fz/VSCode/pdi-vc/gen/quarto/py.it/cap06/CVMCTest.py'>
```

6.9.3 Segmentazione del *QRCode*

Dopo l'estrazione dell'area delle risposte, il MCTest esegue due fasi preparatorie per la lettura delle marcature: la segmentazione del *QRCode* e la localizzazione dei riquadri che contengono le domande.

La funzione `segmentQRcode` isola la regione dell'immagine corrispondente al *QRCode*, presentata nella Figura 6.15.. Successivamente, tale regione viene elaborata dalla funzione `getQRCode`, responsabile della sua decodifica e dell'estrazione dei metadati della prova.

Qualora il *QRCode* non possa essere decodificato, l'elaborazione del foglio prosegue normalmente. Le risposte dello studente vengono comunque lette e registrate nel file CSV di output; solo le informazioni ottenute dal *QRCode*, come l'identificazione della prova o dello studente, rimangono non disponibili.

```
import CVMCTest
imgQRcode = CVMCTest.cvMCTest.segmentQRcode(img_getAnswerArea, countPage)
mm.show(imgQRcode)
```

Figura 6.15: Regione del *QRCode* isolata da *segmentQRcode* all'interno dell'area delle risposte rettificata.

La funzione `CVMCTest.cvMCTest.getQRCode(img, countPage)` integra le fasi di segmentazione e decodifica del *QRCode*. Internamente, utilizza `CVMCTest.cvMCTest.decodeQRcode(imgQRcode)` per interpretare la *stringa* esadecimale codificata nel simbolo e costruire il dizionario `qr`, oltre a restituire l'indicatore logico `myFlagArea`, che informa se la lettura è stata eseguita con successo.

Il dizionario `qr` raccoglie i metadati della prova utilizzati nelle fasi successive di elaborazione. I suoi campi principali sono:

- **`date`:** identificatore temporale della prova, composto dalla data di generazione e da un *timestamp* interno del MCTest.
- **`idClassroom`, `idExam` e `idStudent`:** identificatori della classe, della prova e dello studente.
- **`term`:** periodo accademico.
- **`stylesheet`:** foglio di stile utilizzato nella generazione del modulo.
- **`var1` a `var5`:** numero di domande per ciascun livello di difficoltà.
- **`text`:** numero di domande a risposta aperta.
- **`answer`:** numero di alternative per domanda.
- **`numquest`:** numero totale di domande.
- **`correct`** e **`dbtext`:** campi compilati durante la correzione, contenenti il gabarit e informazioni aggiuntive.
- **`variations`** e **`variant`:** informazioni sulle versioni della prova.

Questi metadati identificano la prova e lo studente, consentendo di selezionare il gabarit corrispondente e di parametrizzare le fasi successive di lettura e correzione delle risposte.

```
myFlagArea, qr = CVMCTest.cvMCTest.getQRCode(img_inicial, countPage)
myFlagArea, qr
```

```
(True,
 {'date': '260208-1770403541363',
  'idClassroom': '955',
  'idExam': '810',
  'idStudent': '448898',
  'term': '0',
  'stylesheet': '1',
  'var1': '50',
  'var2': '0',
  'var3': '0',
  'var4': '0',
  'var5': '0',
  'text': '0',
  'answer': '5',
  'numquest': 50,
```

```
  'correct': '',
  'dbtext': '',
  'variations': '0',
  'variant': '0'})
```

Questi metadati consentono di identificare la prova, recuperare il corrispondente schema di correzione e parametrizzare le fasi successive dell'elaborazione.

Dopo la decodifica del *QRCode*, il processo torna all'area delle risposte per individuare i riquadri che contengono le marcature dello studente.

6.9.4 Localizzazione dei Riquadri delle Risposte

L'immagine prodotta da `getAnswerArea` contiene l'intera regione utile del foglio, inclusa l'intestazione, dove si trova il *QRCode*, e i riquadri destinati alle risposte. Poiché la lettura delle marcature utilizza solo questi riquadri, la regione corrispondente all'intestazione viene scartata tramite il ritaglio `img_getAnswerArea[300:, :]`.

La Figura 6.16 presenta questa regione di interesse. Sebbene questo ritaglio venga successivamente utilizzato dalla funzione `findSquares` per localizzare i riquadri delle risposte, MCTest esegue inizialmente la lettura del *QRCode*, poiché esso contiene i metadati necessari per identificare la prova e configurare le fasi successive dell'elaborazione.

```
img_getAnswerArea_aux = img_getAnswerArea[300:,:]
mm.show(img_getAnswerArea_aux)
```

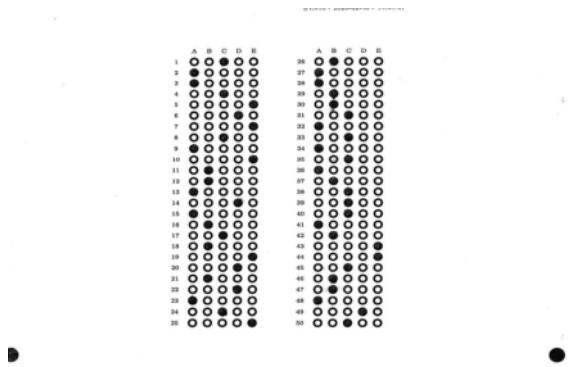


Figura 6.16: Ritaglio inferiore dell'area di risposta, concentrando i riquadri delle bolle da segmentare.

La funzione `findSquares` riceve l'immagine dell'area delle risposte e i metadati memorizzati in `qr`, restituendo le coordinate dei riquadri che delimitano i gruppi di domande.

Ogni elemento di `rectSquares` contiene le coordinate dei vertici in alto a sinistra e in basso a destra di un riquadro di risposte, che saranno utilizzate nella fase di segmentazione delle bolle.

```
rectSquares = CVMCTest.cvMCTest.findSquares(qr,img_getAnswerArea, countPage)
rectSquares
```

```
[[[np.int64(395), np.int64(350)], [np.int64(950), np.int64(516)]],
 [[np.int64(394), np.int64(602)], [np.int64(951), np.int64(767)]]]
```

6.9.5 Lettura Automatica delle Risposte

Noti i metadati della prova (`qr`) e le coordinate dei riquadri delle risposte (`rectSquares`), MCTest identifica automaticamente le alternative selezionate dallo studente.

Il codice seguente integra le fasi presentate in precedenza. Per ciascun riquadro delimitato in `rectSquares`, le funzioni `setColumns` e `setLines` stimano, rispettivamente, il numero di alternative per domanda e il numero di domande a partire dalla distribuzione spaziale delle bolle. Successivamente, `segmentAnswers` determina l'alternativa selezionata in ciascuna domanda e `setAnswersOneLine` raccoglie i risultati di tutti i riquadri nel campo `qr['answers']`.

Nella modalità operativa adottata in questo capitolo, in cui MCTest viene eseguito in modo indipendente dal proprio database, il contenuto di `qr['answers']` viene confrontato con il gabarito memorizzato nella prima pagina del file PDF, corrispondente al modello di prova senza enunciati utilizzato negli esempi.

La Figura 6.17 presenta i riquadri delle risposte elaborati dall'algoritmo, mentre l'output del programma mostra il contenuto finale di `qr['answers']`.

```
testAnswers = []
if myFlagArea:

  imgQ_all = []

  for countSquare in range(len(rectSquares)):
      p1, p2 = rectSquares[countSquare]

      if True:
          imgQi = CVMCTest.cvMCTest.imgAnswers[p1[0]:p2[0], p1[1]:p2[1]]
          [NUM_COLUMNS, img] = CVMCTest.cvMCTest.setColumns(imgQi, countPage, countSquare)
          [NUM_LINES, img] = CVMCTest.cvMCTest.setLines(imgQi, countPage, countSquare)
          NUM_RESPOSTAS = NUM_COLUMNS
          NUM_QUESTOES = NUM_LINES

      imgQiNC = CVMCTest.cvMCTest.imgAnswers[p1[0]:p2[0], p1[1]:p2[1]]
      testAnswers.append(CVMCTest.cvMCTest.segmentAnswers(
          [imgQi, imgQiNC], countPage, countSquare, NUM_QUESTOES, qr

      ))

      imgQ_all.append(imgQiNC)

  qr = CVMCTest.cvMCTest.setAnswarsOneLine(testAnswers, qr)
  # lascia le risposte di ogni quadro su una riga

mm.show(imgQ_all)
print(f"Risposte lette delle {len(qr['answers'].split(","))} domande: \
      \n{qr['answers'][:-19]}...")
```

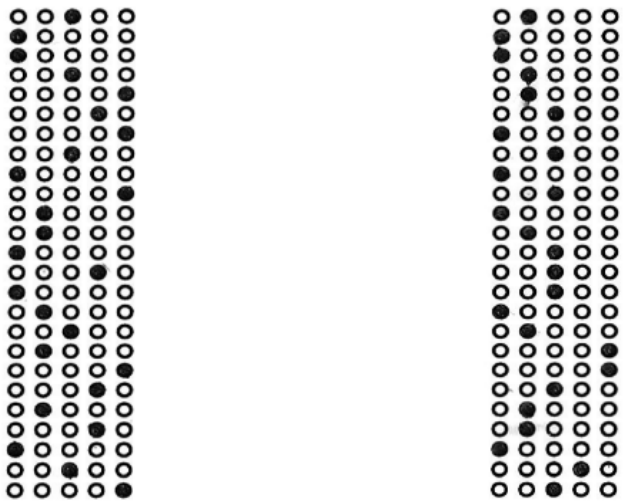


Figura 6.17: Risposte lette automaticamente da MCTest dopo la segmentazione e classificazione di tutte le bolle.

```
Risposte lette delle 50 domande:
C,A,A,C,E,D,E,C,A,E,B,B,A,D,A,B,C,B,E,D,B,D,A,C,E,B,A,A,B,B,C,A,C,A,C,A,B,C,C,C,...
```

Collegando i Punti

Il campo `qr['answers']` rappresenta il risultato finale del *pipeline* di Visione Artificiale presentato in questo capitolo. La sua ottenzione integra tutte le fasi studiate, dalla rasterizzazione del documento e la rettifica geometrica fino all'estrazione dell'area delle risposte, alla decodifica del *QRCode*, alla localizzazione dei riquadri e all'identificazione delle alternative marcate.
Questo *pipeline* illustra la transizione da un prototipo a un sistema in produzione. Nel MCTest, gli algoritmi di Visione Artificiale rimangono essenzialmente gli stessi; le principali differenze si concentrano su aspetti di ingegneria del software, come la gestione delle eccezioni, il supporto a diversi modelli di moduli, l'integrazione con il database, l'interfaccia Web e i meccanismi di audit e manutenzione.
Negli esperimenti di questo capitolo, la prima pagina del file PDF contiene il foglio delle risposte corrette della prova, mentre le pagine successive corrispondono ai fogli delle risposte degli studenti. Dopo l'ottenimento di `qr['answers']`, il MCTest confronta automaticamente le risposte lette con il foglio delle risposte corrette per calcolare il punteggio di ciascuno studente.
Nell'utilizzo completo del sistema, i risultati della correzione vengono consolidati in un file CSV e inviati al docente insieme a un file compresso contenente informazioni ausiliarie per l'audit. Tra questi file vi sono i ritagli delle domande in cui sono state rilevate marcature multiple o altre situazioni che richiedono una revisione manuale. Il docente può quindi ispezionare queste immagini, decidere l'interpretazione più adeguata e, se necessario, aggiornare il file CSV prima dell'importazione definitiva dei voti.

6.10 Ispezione Industriale Automatizzata

L'ispezione visiva automatizzata è un'applicazione della Visione Artificiale in cui immagini di pezzi o prodotti vengono analizzate per verificare il rispetto di criteri di qualità precedentemente definiti. In una linea di produzione, le immagini possono essere acquisite tramite telecamere o altri dispositivi di acquisizione ed elaborate automaticamente per identificare difetti, misurare dimensioni o verificare la presenza di componenti.

La strategia di ispezione dipende dalle caratteristiche del prodotto, dal tipo di difetto di interesse e dalla disponibilità di un'immagine di riferimento. In questo capitolo vengono presentati due approcci classici:

- **Sottrazione di immagini:** confronta l'immagine del pezzo ispezionato con un'immagine di riferimento considerata priva di difetti. Le regioni in cui la differenza di intensità supera una soglia vengono

classificate come possibili difetti. Questo approccio presuppone che le immagini siano geometricamente allineate e siano state acquisite in condizioni di illuminazione simili.

- **Analisi della trama:** utilizza caratteristiche della trama della superficie per identificare regioni il cui aspetto differisce dal modello atteso, senza la necessità di un'immagine di riferimento. Questo approccio è adatto a materiali che presentano una trama approssimativamente omogenea, come tessuti, carte e superfici metalliche.

Nelle prossime sezioni, queste due strategie vengono illustrate mediante esempi costruiti a partire da immagini della libreria `skimage.data`. L'obiettivo è presentare i principi di funzionamento di ciascun approccio in esperimenti che possano essere integralmente riprodotti dal lettore.

6.10.1 Riferimenti e *Dataset* Pubblici

Nelle applicazioni di ispezione industriale, le prestazioni degli algoritmi di rilevamento dei difetti vengono spesso valutate su *dataset* pubblici, che forniscono immagini rappresentative e, in molti casi, annotazioni di riferimento (*ground truth*). In questo capitolo, tuttavia, gli esempi utilizzano immagini sintetiche derivate da `skimage.data` (vedi Figura 6.18), consentendo di riprodurre tutti gli esperimenti senza dipendere da basi di dati esterne.

Per studi più approfonditi e confronti tra algoritmi, si segnalano i seguenti *dataset* pubblici:

- **MVTec *Anomaly Detection Dataset* (MVTec AD):** insieme di immagini di oggetti e texture, contenente campioni senza difetti e con difetti, accompagnate da maschere di segmentazione a livello di pixel per le immagini anomale (BERGMANN, 2019). Disponibile su: https://www.mvtec.com/company/research/datasets/mvtec-ad.
- **Kolektor *Surface-Defect Dataset* (KolektorSDD):** insieme di immagini di componenti industriali con difetti superficiali annotati, utilizzato in studi di rilevamento e segmentazione dei difetti (TABERNIK, 2020). Disponibile su: https://www.vicos.si/resources/kolektorsdd/.
- **NEU *Surface Defect Database*:** insieme di immagini di superfici in acciaio laminato, organizzato in sei categorie di difetti superficiali, spesso impiegato nella valutazione di metodi di classificazione e rilevamento (SONG, 2013). Disponibile su: http://faculty.neu.edu.cn/songkechen/zh_CN/zdylm/263270/list/index.htm.

```
import numpy as np
from skimage import data, color
from morph import mm

# Immagine di riferimento (prodotto senza difetti)
product_color = data.coffee()
product_gray = color.rgb2gray(product_color)

# Inserimento di difetto simulato: graffio scuro di 10×100 px
defect_image = np.copy(product_gray)
defect_image[100:110, 200:300] = 0.1

# Rilevamento per sottrazione e sogliatura
difference = np.abs(product_gray - defect_image)
defect_threshold = 0.15          # regolabile in base all'applicazione
detected_defect = (difference > defect_threshold).astype(np.uint8) * 255

mm.show(
    [product_gray, defect_image, detected_defect],
    titles=["Riferimento", "Con difetto", "Difetto rilevato"],
    cols=3,
    figsize=(12, 4)
)
```

```
status = "Defeito detectado." if detected_defect.any() else "Produto conforme."
print(status)
```

Figura 6.18: Rilevamento di difetti per sottrazione dell'immagine: prodotto di riferimento, immagine con difetto simulato e maschera dell'anomalia rilevata.

```
Defeito detectado.
```

Sottrazione di Immagini: Registrazione Geometrica e Principio di Funzionamento

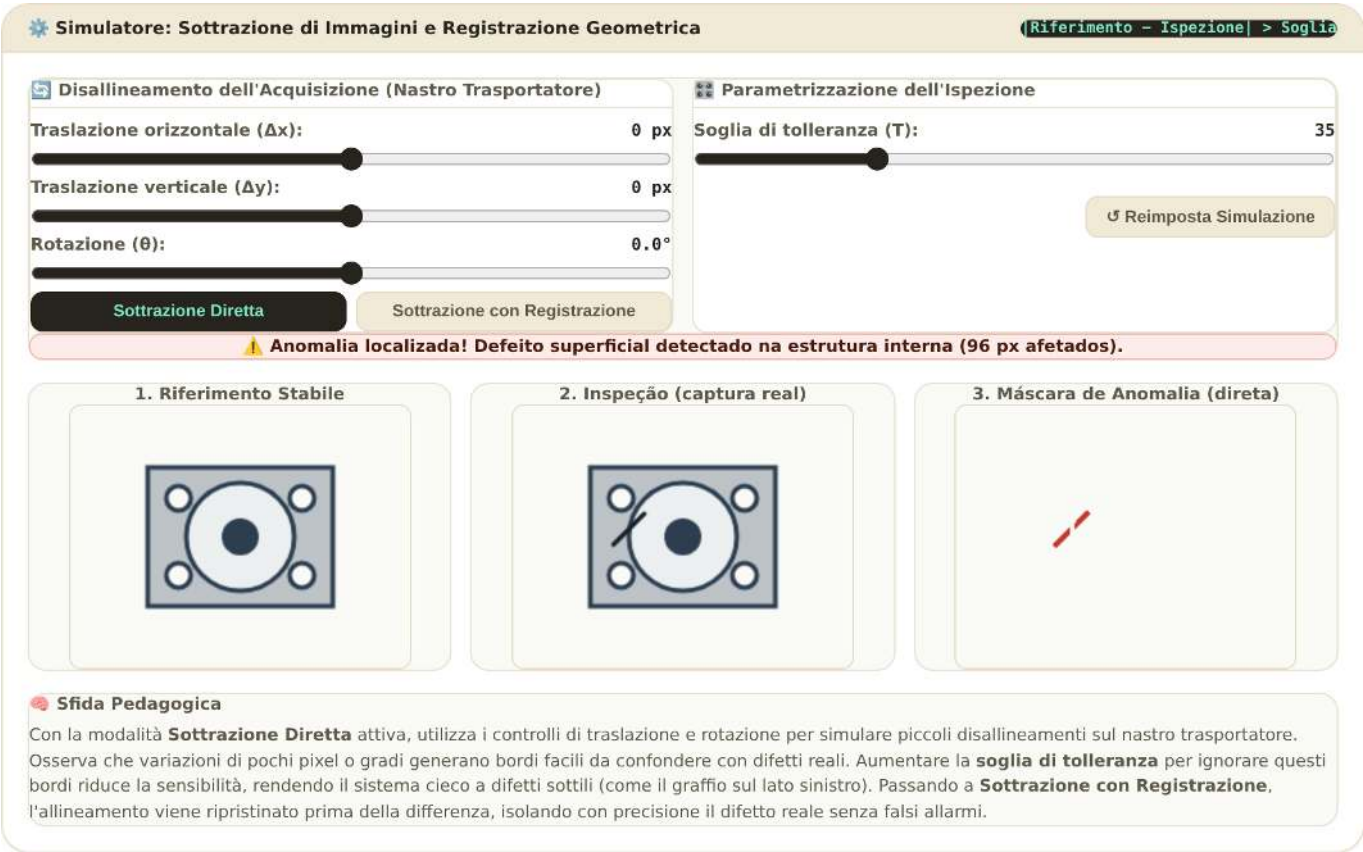
La sottrazione di immagini presuppone che l'immagine di ispezione sia geometricamente allineata all'immagine di riferimento. Differenze di posizionamento, rotazione, scala o prospettiva producono regioni di differenza che possono essere confuse con difetti.
In applicazioni con acquisizione controllata, questo allineamento viene ottenuto durante la cattura tramite maschere meccaniche, nastri trasportatori e telecamere fisse, consentendo di confrontare direttamente immagini successive. Un esempio è l'ispezione di una cassetta degli attrezzi sempre posizionata nella stessa orientazione per verificare l'assenza di un elemento.
Quando questo controllo non è possibile, si impiega la **registrazione di immagini**, che stima una trasformazione geometrica per compensare differenze di traslazione, rotazione, scala e, quando necessario, prospettiva.
Dopo la registrazione, si esegue il confronto pixel per pixel tra le due immagini. Nelle regioni senza variazioni, le differenze di intensità tendono a essere prossime a zero; dove è presente un difetto, emergono differenze locali che possono essere evidenziate tramite sogliatura. L'uso della differenza assoluta consente di rilevare sia difetti più chiari sia più scuri rispetto al riferimento.
Le prestazioni del metodo dipendono principalmente dalla qualità dell'allineamento geometrico e dalla scelta della soglia utilizzata per separare piccole variazioni di acquisizione dalle differenze associate ai difetti.
La Figura 6.19 illustra l'effetto del disallineamento tra le immagini e l'importanza della registrazione geometrica prima dell'applicazione della sottrazione.

6.10.2 Rilevamento di Difetti tramite Analisi della Texture

In applicazioni in cui non è disponibile un'immagine di riferimento, il rilevamento di difetti può basarsi sulle caratteristiche della texture della superficie. In questo caso, si cerca di identificare regioni la cui apparenza differisce dal pattern predominante del materiale.

In questo esempio, si utilizza la **varianza locale** come misura di eterogeneità. Per ogni posizione dell'immagine, si calcola la varianza dei livelli di intensità in un intorno di dimensioni fisse. Le regioni con bassa varianza tendono a presentare una texture più uniforme, mentre alterazioni locali, come graffi, macchie o imperfezioni, possono produrre valori più elevati di tale misura.

La Figura 6.20 illustra questa procedura utilizzando l'immagine `skimage.data.brick()`. Inizialmente, si calcola la mappa della varianza locale tramite una finestra scorrevole. Successivamente, si applica una



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-06-industrial.html

Figura 6.19: Simulatore interattivo di ispezione industriale per sottrazione di immagini: controlla le distorsioni geometriche di disallineamento (registrazione) e la soglia di rilevamento per osservare l'impatto sui falsi positivi.

sogliatura per evidenziare le regioni la cui varianza supera il valore specificato, identificando potenziali aree di interesse per l'ispezione.

Modellazione Matematica

Si consideri un'immagine in scala di grigi rappresentata da

$$f : \Omega \subset \mathbb{Z}^2 \to \mathbb{R},$$

dove Ω è il dominio dell'immagine e $f(x, y)$ rappresenta l'intensità del pixel nelle coordinate (x, y). Nelle immagini a 8 bit, queste intensità appartengono all'intervallo $[0, 255]$. In questo esempio, tuttavia, sono state normalizzate nell'intervallo $[0, 1]$, senza alterare il funzionamento dell'algoritmo.

Per ogni posizione dell'immagine, si considera un intorno quadratico $W_{x,y}$ di dimensione 15×15 pixel.

La media locale è data da

$$\mu(x, y) = \frac{1}{|W_{x,y}|} \sum_{(u,v)\in W_{x,y}} f(u, v),$$

e la varianza locale è calcolata come

$$\sigma^2(x, y) = \frac{1}{|W_{x,y}|} \sum_{(u,v)\in W_{x,y}} f(u, v)^2 - \mu(x, y)^2.$$

Nell'implementazione seguente, queste due medie sono ottenute tramite la funzione `cv2.blur`,

```
mean  = cv2.blur(img, (15,15))
mean2 = cv2.blur(img**2, (15,15))
var   = mean2 - mean**2
```

Successivamente, si calcola la differenza tra le mappe di varianza dell'immagine di riferimento e dell'immagine ispezionata,

$$D(x, y) = \left|\sigma_d^2(x, y) - \sigma_r^2(x, y)\right|,$$

dove $\sigma_r^2(x, y)$ e $\sigma_d^2(x, y)$ sono, rispettivamente, le varianze locali dell'immagine di riferimento e dell'immagine contenente il difetto. Dopo la normalizzazione della mappa $D(x, y)$, si applica una sogliatura per ottenere la maschera delle possibili anomalie.

```
import numpy as np
import cv2
from skimage import data as skdata
from morph import mm

# Immagine di texture uniforme (mattone)
texture = skdata.brick().astype(np.float32) / 255.0

# Inserimento di difetto sintetico: macchia chiara 20×80 px
texture_defect = np.copy(texture)
texture_defect[60:80, 80:160] = 0.95

# Mappa di varianza locale (finestra 15×15)
def variancia_local(img, ksize=15):
    img_f = img.astype(np.float32)
    mean  = cv2.blur(img_f, (ksize, ksize))
    mean2 = cv2.blur(img_f ** 2, (ksize, ksize))
```

```
    return np.clip(mean2 - mean ** 2, 0, None)

var_ref    = variancia_local(texture)
var_defect = variancia_local(texture_defect)
diff_var   = np.abs(var_defect - var_ref)

# Normalizza e soglia
diff_norm = (diff_var / diff_var.max() * 255).astype(np.uint8)
_, mask   = cv2.threshold(diff_norm, 30, 255, cv2.THRESH_BINARY)

mm.show(
    [texture, texture_defect, diff_norm, mask],
    titles=["Texture originale", "Con difetto", "Δ varianza locale", "Anomalia rilevata"],
    cols=4,
    figsize=(16, 4)
)

status = "Defeito de textura detectado." if mask.any() else "Superfície conforme."
print(status)
```

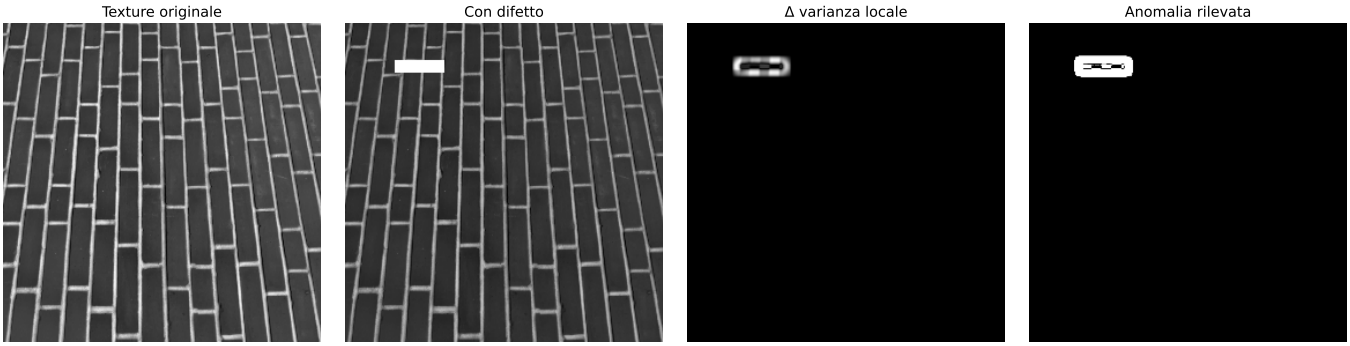


Figura 6.20: Rilevamento dell'eterogeneità della texture: mappa di varianza locale e maschera di anomalia.

```
Defeito de textura detectado.
```

Perché funziona? — Analisi della trama

La varianza locale misura la dispersione delle intensità in un intorno dell'immagine. Nelle regioni in cui la trama rimane uniforme, tale misura tende a variare poco. Quando un difetto modifica il pattern della superficie, anche la distribuzione delle intensità si altera, producendo differenze nella varianza locale.
In questo capitolo, la rilevazione viene eseguita confrontando le mappe di varianza dell'immagine di riferimento e dell'immagine con difetto. Dopo la normalizzazione, si applica una sogliatura per evidenziare le regioni in cui tale differenza supera un valore specificato.
I parametri principali del metodo sono la dimensione della finestra utilizzata nel calcolo della varianza e la soglia impiegata nella segmentazione. Finestre più piccole sono più sensibili ai dettagli fini, mentre finestre più grandi producono mappe più uniformi e possono ridurre la risposta a difetti di piccole dimensioni.

6.11 Riassunto

In questo capitolo sono stati presentati metodi di Visione Artificiale applicati all'analisi di documenti e all'ispezione visiva automatizzata. Le principali tecniche studiate sono state:

- **Pre-elaborazione di documenti:** applicazione di normalizzazione dello sfondo, equalizzazione adattativa (CLAHE) e sogliatura di Otsu per ridurre gli effetti dell'illuminazione non uniforme e migliorare la segmentazione del testo.

- **Riconoscimento ottico dei caratteri (OCR):** conversione di immagini di documenti in testo codificato tramite Tesseract OCR, evidenziando l'influenza della pre-elaborazione sulla qualità del riconoscimento.
- **Traduzione automatica:** applicazione di tecniche di elaborazione del linguaggio naturale per tradurre il testo ottenuto dall'OCR.
- **Rettifica geometrica di documenti:** utilizzo del rilevatore di bordi di Canny, della Trasformata di Hough e di trasformazioni proiettive per correggere la prospettiva di documenti digitalizzati.
- **Localizzazione e rettifica di moduli:** impiego di operazioni morfologiche, analisi dei contorni e trasformazione di prospettiva per identificare marcatori di riferimento ed estrarre automaticamente regioni di interesse.
- **Lettura di codici bidimensionali e unidimensionali:** rilevamento e decodifica di *QRCodes* e codici a barre per l'identificazione automatica di documenti e metadati.
- **Riconoscimento ottico dei marcatori (OMR):** lettura automatizzata di moduli e fogli di risposta, illustrata da un caso di studio del sistema MCTest.
- **Ispezione industriale:** rilevamento di difetti mediante confronto con un'immagine di riferimento e analisi della varianza locale.

Nel corso del capitolo, gli algoritmi sono stati implementati e valutati con immagini della libreria `skimage.data` e con documenti reali, consentendo di riprodurre gli esperimenti presentati.

Prossimi Passi

I metodi presentati in questo capitolo mostrano come tecniche di Elaborazione Digitale delle Immagini, Visione Artificiale ed Elaborazione del Linguaggio Naturale possano essere integrate in *pipeline* per l'analisi automatizzata di documenti.

Nel prossimo capitolo verranno studiate tecniche di **estrazione di caratteristiche** e **riconoscimento di pattern**, con particolare attenzione ai descrittori capaci di rappresentare le immagini tramite attributi numerici per confronto, classificazione e riconoscimento automatico. Questi concetti costituiscono la base per i capitoli dedicati all'apprendimento automatico e all'apprendimento profondo applicati alla Visione Artificiale.

6.12 🤖 Uso del Gemini Notebook come Tutor Complementare

Come supporto allo studio di questo capitolo, si raccomanda l'utilizzo del **Gemini Notebook** come tutor complementare. Lo strumento impiega modelli di intelligenza artificiale per rispondere a domande, elaborare riassunti e spiegare concetti sulla base dei documenti forniti come fonte di consultazione, consentendo allo studente di rivedere il contenuto in modo interattivo.

🎓 Studia con il Tutor Intelligente

🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 06

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato realizzato esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, oppure seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ **Uso Critico delle Risposte**

Le risposte generate dal Gemini Notebook sono prodotte automaticamente da un modello di intelligenza artificiale e possono contenere omissioni o imprecisioni. Per questo motivo, devono essere utilizzate come materiale di supporto e non come sostituto dello studio del capitolo.
Ogni volta che sorgono dubbi, consulta il testo di questo libro, esegui gli esempi presentati e, quando necessario, integra la consultazione con libri, articoli scientifici e altre fonti accademiche affidabili.

6.13 Lista di Esercizi

Gli esercizi seguenti consolidano i concetti presentati in questo capitolo tramite adattamenti, esperimenti ed estensioni degli algoritmi sviluppati nel testo.

1. **(10%)** Indagare l'influenza dell'angolo di inclinazione nella fase di *deskew*. Generare versioni ruotate dell'immagine `skimage.data.page()` per angoli tra -10° e 10°, applicare l'algoritmo presentato nel capitolo e confrontare l'angolo stimato con l'angolo utilizzato nella rotazione. Presentare i risultati in una tabella e discutere la precisione del metodo.

2. **(15%)** Applicare la normalizzazione dello sfondo e il CLAHE (utilizzando almeno tre combinazioni di `clipLimit` e `tileGridSize`) all'immagine `skimage.data.page()` degradata artificialmente con gradiente di illuminazione e ombra laterale. Segmentare ciascuna versione tramite il metodo di Otsu e confrontare i risultati utilizzando il numero di componenti connessi spuri e la metrica IoU rispetto a una maschera di riferimento costruita manualmente.

3. **(15%)** Indagare la sensibilità del filtraggio per circolarità, $C = \frac{4\pi A}{P^2}$, nella rilevazione dei marcatori circolari. Utilizzando il simulatore della Figura 6.9, generare dischi sintetici con rumore geometrico crescente e valutare le soglie $C \in \{0{,}5,\ 0{,}6,\ 0{,}7,\ 0{,}8\}$. Presentare una tabella che correla la soglia al numero di falsi positivi e falsi negativi e discutere il compromesso tra sensibilità e specificità.

4. **(15%)** A partire dai quattro marcatori rilevati, implementare la rettifica prospettica utilizzando `cv2.getPerspectiveTransform` e `cv2.warpPerspective`. Successivamente, perturbare artificialmente le coordinate dei punti di controllo con rumore gaussiano di deviazione standard $\sigma \in \{1, 3, 5\}$ pixel e valutare l'errore di riproiezione ottenuto dopo l'omografia inversa.

5. **(15%)** Adattare il *pipeline* di acquisizione e rettifica sviluppato in questo capitolo per elaborare documenti contenenti codici a barre lineari in sostituzione dei *QRCodes*. Rasterizzare il PDF con `pdf2image` a 300 DPI, applicare il *deskew*, decodificare il simbolo con `pyzbar` e presentare l'immagine rettificata insieme alla sequenza di caratteri ottenuta.

6. **(15%)** Estendere la lettura delle bolle del MCTest per identificare tre situazioni: **OK**, **BIANCO** (nessuna alternativa marcata) e **DOPPIA MARCAZIONE** (due o più alternative al di sopra di una soglia di riempimento). Valutare almeno tre valori di tale soglia, presentare i risultati in un `pandas.DataFrame` e discutere la loro influenza sulla classificazione delle risposte.

7. **(15%)** Costruire un *pipeline* di ispezione industriale combinando sottrazione di immagini e analisi di varianza locale della trama su un insieme di immagini sintetiche contenenti difetti simulati. Per ciascuna immagine, generare una maschera di riferimento (*ground truth*), calcolare la metrica IoU (*Intersection over Union*) dei due approcci per diverse soglie di decisione e presentare i risultati in tabelle e visualizzazioni prodotte con `mm.show`.

8. **(Bonus – 10%)** Implementare manualmente la stima dell'angolo di inclinazione senza utilizzare `cv2.HoughLines` o `cv2.HoughLinesP`. A partire dalla mappa dei bordi ottenuta dal rilevatore di Canny, costruire l'accumulatore della Trasformata di Hough, $\rho = x\cos\theta + y\sin\theta$, per $\theta \in [-45^\circ, 45^\circ]$, identificare i massimi dell'accumulatore e stimare l'inclinazione tramite la mediana delle rette rilevate. Confrontare i risultati con l'implementazione di OpenCV e discutere l'influenza di rette spurie sulla stima finale.

Riferimenti del Capitolo

La fondazione teorica e i casi di studio presentati in questo capitolo si basano sui seguenti riferimenti:

- Gonzalez (2018), per i fondamenti del rilevamento dei bordi, la sogliatura, la segmentazione, le operazioni morfologiche, il riconoscimento ottico dei caratteri e le trasformazioni geometriche applicate all'analisi dei documenti.
- Szeliski (2022), per la Trasformata di Hough, la registrazione e l'allineamento delle immagini, le trasformazioni proiettive (omografie) e i fondamenti dell'ispezione visiva automatizzata.
- Bradski (2008), per l'utilizzo della libreria OpenCV nelle fasi di rilevamento dei bordi, Trasformata di Hough, trasformazioni geometriche, analisi dei contorni e decodifica dei *QRCode*.
- Smith (2007) e Smith (2013), per l'architettura, il funzionamento e l'evoluzione del motore di riconoscimento ottico dei caratteri **Tesseract OCR**, impiegato negli esempi di OCR presentati in questo capitolo.
- Bahdanau (2015) e Vaswani (2017), per i fondamenti della traduzione automatica basata su reti neurali, inclusi i meccanismi di attenzione e le architetture *transformer*.
- Zampirolli (2023), per la descrizione del sistema MCTest, utilizzato come caso di studio di un *pipeline* completo per la lettura e la correzione automatizzata dei fogli di risposta.
- Bergmann (2019), Tabernik (2020) e Song (2013), per i *dataset* pubblici di ispezione industriale **MVTec AD**, **KolektorSDD** e **NEU Surface Defect Database**, utilizzati come riferimento per la valutazione e il confronto degli algoritmi di rilevamento dei difetti.

Executar Colab Abrir GitHub

6.14 💻 Parte Pratica con Esercizi di Programmazione

Gli esercizi di programmazione (EP) di questa sezione completano i concetti presentati lungo il Capitolo 6 attraverso l'implementazione di algoritmi legati all'ispezione industriale e all'analisi dei documenti. L'obiettivo è consolidare i fondamenti studiati, riproducendo, in scala ridotta, le fasi di una *pipeline* tipica della Visione Artificiale.

A differenza dei capitoli precedenti, i cui esercizi enfatizzavano operazioni più direttamente legate ai dati immagine, gli EP di questo capitolo si concentrano sulle **grandezze intermedie** prodotte durante l'elaborazione, come aree, perimetri, circolarità, angoli di rette, gradi di riempimento di bolle, mappe di varianza e mappe di differenza. Questo approccio permette di comprendere e validare ogni fase della *pipeline* in modo indipendente, senza dipendere da librerie specializzate per l'acquisizione di immagini, la rilevazione di marcatori o la decodifica di codici — ad eccezione dell'esercizio di chiusura del capitolo (EP06_08), che introduce intenzionalmente l'uso di OpenCV per la segmentazione e la decodifica reale di un *QRCode*, chiudendo il ciclo tra i concetti teorici e gli strumenti impiegati nella pratica.

Gli esercizi seguono la stessa sequenza concettuale del capitolo, in ordine crescente di complessità. Inizialmente vengono affrontate le metriche di valutazione della segmentazione, utilizzate per quantificare la qualità delle maschere binarie. Successivamente, si studiano criteri geometrici per la selezione dei marcatori, la classificazione delle marcature nei moduli e la stima dell'inclinazione dei documenti tramite la Trasformata di Hough. Nella parte finale, gli esercizi esplorano la normalizzazione dell'illuminazione, il rilevamento dei difetti tramite analisi della trama e l'integrazione tra registrazione geometrica e sottrazione di immagini in una *pipeline* semplificata di ispezione industriale.

Ogni esercizio rappresenta una fase isolata di un sistema reale di Visione Artificiale, consentendo di validare singolarmente concetti che, nelle applicazioni industriali, vengono combinati in un'unica *pipeline* di ispezione.

Legenda della Difficoltà

Livello	Significato	EP
🟢	Molto facile / facile — implementazione di un singolo concetto o algoritmo semplice	EP06_01, EP06_02
🟡	Facile–medio — gestione di più casi o utilizzo di criteri statistici semplici	EP06_03, EP06_04
🟠	Medio — elaborazione matriciale punto a punto	EP06_05
🔴	Difficile — elaborazione matriciale con operazioni di vicinato (finestra scorrevole)	EP06_06
🟣	Molto difficile — integrazione di più fasi di una *pipeline* di Visione Artificiale	EP06_07
⚫	Speciale — uso di una libreria specializzata (`cv2`) per la segmentazione geometrica e la decodifica reale di codici a barre/QRCode	EP06_08

❗ Linee guida per la Risoluzione degli Esercizi di Programmazione

Salvo diversa indicazione, tutti gli esercizi utilizzano la convenzione di coordinate matriciali `[riga][colonna]`, con origine in $(0, 0)$ nell'angolo superiore sinistro dell'immagine.
Quando è necessario un arrotondamento numerico, si deve utilizzare l'arrotondamento standard al numero intero più vicino (*round half away from zero*, con `np.floor(img + 0.5)`). I confronti con soglie (ad esempio, circolarità, varianza, differenza di intensità o grado di riempimento) devono essere considerati **stretti** (`>`), salvo che il testo non specifichi esplicitamente un altro criterio.
Ogni esercizio è stato ideato per enfatizzare un concetto specifico presentato nel capitolo. Si raccomanda di implementare inizialmente la soluzione in modo diretto e, solo dopo la sua validazione, cercare alternative più efficienti o più generali.

🎯 Obiettivo di questo Quaderno

Questo quaderno è stato realizzato per supportare lo sviluppo, la validazione e i test delle soluzioni degli **Esercizi di Programmazione (EP)** in un ambiente interattivo, come Google Colab o Jupyter Notebook. Dopo aver verificato il funzionamento dell'implementazione con i casi di test presentati, il codice può essere inviato a Moodle per la valutazione ufficiale.

Download

Esegui la cella seguente per ottenere i file `morph.py` e `testsuite.py`, utilizzati dagli esercizi di questo capitolo.

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
```

```
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Dopo aver implementato la soluzione, esegui `TestSuite("EP06_01.estensione").run()` in una nuova cella, sostituendo `estensione` con il linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema recupera automaticamente i casi di test dal repository del corso, esegue il programma e presenta il risultato della valutazione.

In Python, è anche possibile testare la soluzione direttamente da una *stringa*, senza la necessità di salvare il codice in un file. A tale scopo, memorizza il programma in una variabile e utilizza il metodo `run_code`:

```
codigo = """
# ... il tuo codice qui ...
"""

TestSuite("EP06_01").run_code(codigo)
```

6.14.1 EP06_01 🟢 Valutazione della Segmentazione tramite IoU (*Intersection over Union*)

Nel corso di questo capitolo, diverse fasi del *pipeline* producono **maschere binarie**, come nella segmentazione di documenti, nella localizzazione di *QRCodes* e nel rilevamento di difetti. Per valutare oggettivamente la qualità di queste segmentazioni, è necessario confrontarle con una maschera di riferimento (*ground truth*).

Una delle metriche più utilizzate a questo scopo è la **IoU** (*Intersection over Union*, o Intersezione su Unione), definita come il rapporto tra l'area di intersezione e l'area di unione di due maschere binarie. Quanto maggiore è il valore della IoU, tanto maggiore è la concordanza tra la segmentazione prodotta dall'algoritmo e il riferimento.

6.14.1.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi L (numero di righe) e C (numero di colonne).
2. **Maschera di riferimento:** Leggere gli $L \times C$ elementi binari (0 o 1) della matrice `ref`.
3. **Maschera predetta:** Leggere gli $L \times C$ elementi binari (0 o 1) della matrice `pred`.
4. **Intersezione:** Contare il numero di posizioni (i, j) per le quali `ref[i][j] = 1` e `pred[i][j] = 1`.
5. **Unione:** Contare il numero di posizioni (i, j) per le quali `ref[i][j] = 1` o `pred[i][j] = 1`.
6. **Caso degenere:** Se l'unione è uguale a 0, definire IoU = 1,0, poiché entrambe le maschere sono vuote.
7. **Calcolo:** Se l'unione è maggiore di zero, calcolare

$$\text{IoU} = \frac{|\text{Intersezione}|}{|\text{Unione}|}.$$

8. **Classificazione:** Determinare la classificazione qualitativa utilizzando il valore di IoU **prima** dell'arrotondamento.
9. **Arrotondamento:** Visualizzare la IoU con quattro cifre decimali.
10. **Output:** Stampare, in quest'ordine, l'intersezione, l'unione, la IoU e la classificazione.

6.14.1.2 📌 Vincoli Computazionali

- Se l'unione è uguale a 0, non deve essere eseguita la divisione; la IoU deve essere definita come 1,0.
- Le fasce di classificazione utilizzano confronti non stretti ($\geq$).
- La classificazione deve essere eseguita utilizzando il valore della IoU a piena precisione, prima dell'arrotondamento per la visualizzazione.

6.14.1.3 🧠 Fondamento Teorico

La IoU è definita da

$$\text{IoU} = \frac{|R \cap P|}{|R \cup P|},$$

dove:

- R rappresenta l'insieme dei pixel appartenenti alla maschera di riferimento;
- P rappresenta l'insieme dei pixel appartenenti alla maschera predetta;
- $|R \cap P|$ corrisponde al numero di pixel appartenenti simultaneamente a entrambe le maschere;
- $|R \cup P|$ corrisponde al numero di pixel appartenenti ad almeno una delle maschere.

Fascia di IoU	Classificazione	Interpretazione
IoU $\geq$ 0,90	`ECCELLENTE`	Concordanza molto elevata tra le maschere.
0,70 $\leq$ IoU $<$ 0,90	`BUONO`	Piccole differenze tra le maschere.
0,50 $\leq$ IoU $<$ 0,70	`ACCETTABILE`	Concordanza parziale tra le maschere.
IoU $<$ 0,50	`SCARSO`	Bassa concordanza tra le maschere.

La IoU dipende solo dalla sovrapposizione tra le maschere e, pertanto, è indipendente dalla dimensione dell'immagine.

6.14.1.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: intero L.
- Riga 2: intero C.
- Prossime L righe: elementi binari (0 o 1) della matrice `ref`.
- Prossime L righe: elementi binari (0 o 1) della matrice `pred`.

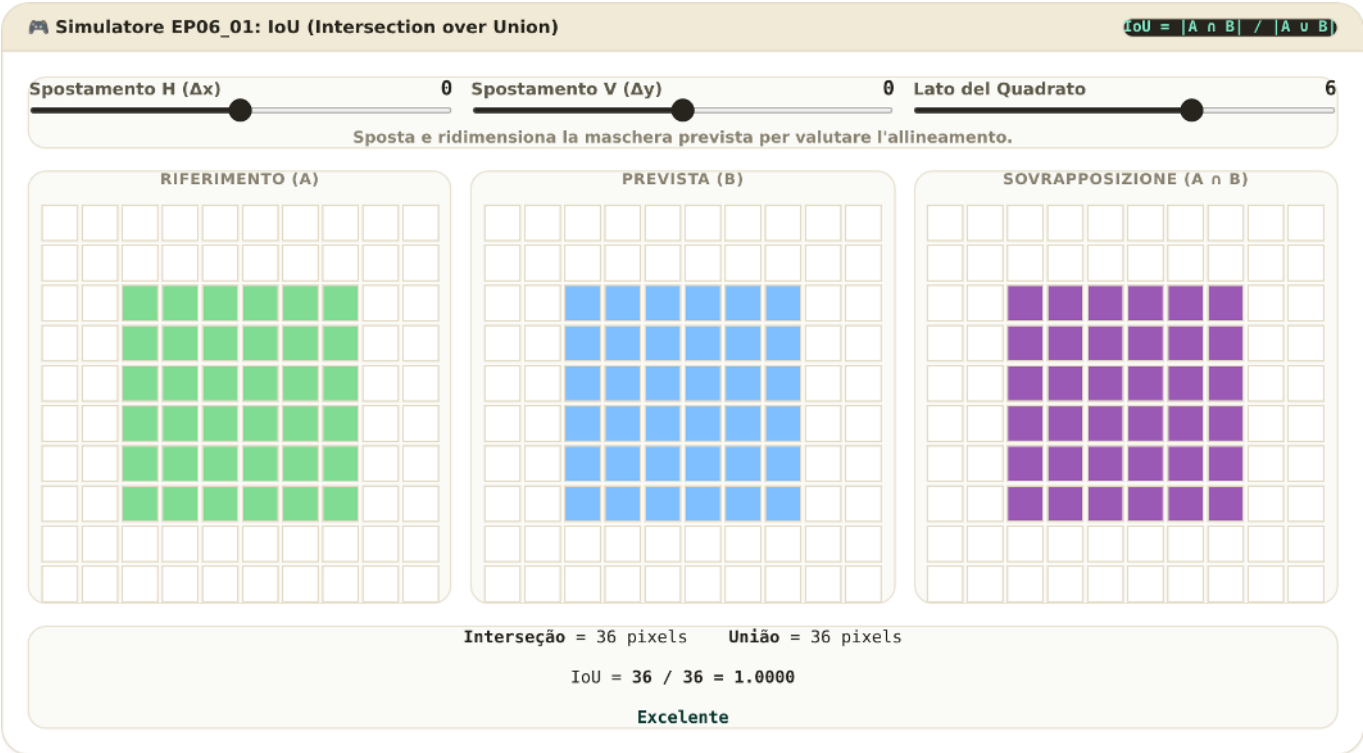
Output:

- Riga 1: `Intersezione: X`
- Riga 2: `Unione: Y`
- Riga 3: `IoU: Z`
- Riga 4: `Classificazione: NOME`

Il valore di `IoU` deve essere stampato con quattro cifre decimali.

6.14.1.5 📌 Esempi

Input	Output	Osservazione
2 2 1 1 0 0 1 0 0 0	Intersezione: 1 Unione: 2 IoU: 0.5000 Classificazione: ACCETTABILE	La metà della regione di riferimento è stata segmentata correttamente.
2 2 0 0 0 0 0 0 0 0	Intersezione: 0 Unione: 0 IoU: 1.0000 Classificazione: ECCELLENTE	Entrambe le maschere sono vuote; per convenzione, IoU = 1,0.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-ep0601.html

Figura 6.21: Simulatore EP06_01: IoU tra maschera di riferimento e maschera predetta

```
%%writefile EP06_01.py
# Codice Python
```

```
Overwriting EP06_01.py
```

```
TestSuite("EP06_01.py").run()
```

```
<IPython.core.display.HTML object>
```

6.14.2 EP06_02 🟢 Filtro di Marcatori per Circolarità

Dopo la segmentazione di un'immagine, è comune che vengano identificati diversi componenti connessi. In applicazioni come la rettifica di documenti, solo alcuni di questi componenti corrispondono ai marcatori

di riferimento utilizzati per l'allineamento dell'immagine. Un criterio spesso impiegato per selezionare tali marcatori è la **circolarità**, che misura quanto la forma di un componente sia vicina a un cerchio.

In questo esercizio, ogni componente è descritto dalla sua area A e dal suo perimetro P. L'obiettivo è calcolarne la circolarità e decidere, in base a una soglia fornita, se il componente debba essere accettato o rifiutato come candidato marcatore.

6.14.2.1 📋 Linee Guida di Implementazione

1. **Quantità:** Leggere l'intero N (numero di candidati) e la soglia di circolarità C_{limiar} (numero reale).
2. **Dati dei candidati:** Per ciascuno degli N candidati, leggere l'area A (intero) e il perimetro P (numero reale).
3. **Circolarità:** Calcolare $C = \frac{4\pi A}{P^2}$, dove:

- A è l'area del componente;
- P è il perimetro del componente;
- C è la circolarità.

4. **Caso degenere:** Se $P = 0$, considerare $C = 0$ e classificare direttamente il candidato come `REJEITADO`.
5. **Classificazione:** Se $C > C_{\text{limiar}}$, classificare il candidato come `ACEITO`; altrimenti, classificarlo come `REJEITADO`.
6. **Arrotondamento:** Visualizzare il valore di C con quattro cifre decimali.
7. **Output:** Per ogni candidato, stampare il valore di C seguito dalla classificazione. Alla fine, stampare il numero totale di candidati accettati.

6.14.2.2 📌 Vincoli Computazionali

- Utilizzare la costante π della libreria standard del linguaggio (ad esempio, `math.pi`), senza approssimazioni.
- Il confronto deve essere effettuato con il valore di C a piena precisione, prima dell'arrotondamento per la visualizzazione.
- Il criterio di accettazione è stretto ($C > C_{\text{limiar}}$).
- Se $P = 0$, la divisione non deve essere eseguita.

6.14.2.3 🧠 Fondamenti Teorici

La circolarità è un descrittore geometrico definito da $C = \frac{4\pi A}{P^2}$, dove:

- A è l'area del componente;
- P è il perimetro del componente;
- C è la circolarità.

Per un cerchio perfetto, $C = 1$. Man mano che la forma diventa più allungata o irregolare, il perimetro cresce più rapidamente dell'area, riducendo il valore di C.

Forma	Circolarità approssimata	Interpretazione
Cerchio	1,0000	Forma circolare.
Quadrato	0,7854	Forma approssimativamente compatta.
Forma allungata o irregolare	$C \ll 1$	Bassa circolarità.
$P = 0$	0 (convenzione adottata)	Contorno degenere.

La circolarità è invariante rispetto a traslazione, rotazione e scala, ed è ampiamente utilizzata per distinguere componenti approssimativamente circolari da altre forme.

6.14.2.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: intero N.
- Riga 2: numero reale C_{limiar}.
- Successive N righe: area A (intero) e perimetro P (reale), separati da spazio.

Output:

- Una riga per ogni candidato, nel formato `C ACEITO` o `C REJEITADO`, con C presentato con quattro cifre decimali.
- Ultima riga: `Total aceitos: X`.

6.14.2.5 📌 Esempi

Input	Output	Osservazione
3 0.6 78 31.4 100 40 50 60	0.9941 ACEITO 0.7854 ACEITO 0.1745 REJEITADO Total aceitos: 2	Candidato approssimativamente circolare, forma compatta e forma allungata.
1 0.9 10 0	0.0000 REJEITADO Total aceitos: 0	Perimetro nullo: contorno degenere.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-ep0602.html

Figura 6.22: Simulatore EP06_02: Filtro di Marcatori per Circolarità

```
%%writefile EP06_02.py
# Codice Python
```

```
Overwriting EP06_02.py
```

```
TestSuite("EP06_02.py").run()
```

```
<IPython.core.display.HTML object>
```

6.14.3 EP06_03 🟡 Classificazione delle Marcature nei Fogli di Risposta (OMR)

Dopo la rettifica del foglio e la segmentazione dei riquadri delle risposte, il MCTest stima, per ogni bolla, un **grado di riempimento**, rappresentato da un valore tra 0 e 100. Sulla base di questi valori, il sistema

deve determinare automaticamente l'alternativa marcata, individuando anche domande in bianco e casi di marcature multiple.

In questo esercizio, implementerai questa fase di decisione del *pipeline* di OMR. La classificazione dipende da una soglia di riempimento: piccole variazioni di questo valore possono alterare il risultato della lettura automatica.

6.14.3.1 📋 Linee Guida di Implementazione

1. **Parametri:** Leggere gli interi Q (numero di domande) e K (numero di alternative per domanda, con $2 \leq K \leq 26$) e la soglia di riempimento Th (numero reale tra 0 e 100).
2. **Gradi di riempimento:** Per ciascuna delle Q domande, leggere i K valori reali corrispondenti alle alternative `A`, `B`, `C`, …, nell'ordine di ingresso.
3. **Conteggio delle marcature:** Per ogni domanda, contare quante alternative hanno un grado di riempimento **strettamente maggiore** di Th.
4. **Classificazione:**
   - Se nessuna alternativa supera Th, classificare la domanda come `BRANCO`.
   - Se esattamente un'alternativa supera Th, stampare la lettera corrispondente (`A`, `B`, `C`, …).
   - Se due o più alternative superano Th, classificare la domanda come `DUPLA_MARCACAO`.
5. **Output per domanda:** Stampare, nell'ordine di lettura, la classificazione di ciascuna domanda.
6. **Totali:** Alla fine, stampare il numero di domande `OK` (marcatura singola), `BRANCO` e `DUPLA_MARCACAO`.

6.14.3.2 📌 Vincoli Computazionali

- **Confronto stretto:** solo i valori maggiori di Th sono considerati marcature valide; i valori esattamente uguali alla soglia non devono essere conteggiati.
- **Lettere delle alternative:** l'indice 0 corrisponde all'alternativa `A`, l'indice 1 all'alternativa `B` e così via.
- **Marcature multiple:** ogni volta che due o più alternative superano la soglia, la classificazione deve essere `DUPLA_MARCACAO`, indipendentemente dai rispettivi gradi di riempimento.

6.14.3.3 🧠 Fondamento Teorico

Situazione	Classificazione	Interpretazione
Esattamente un'alternativa sopra la soglia	Lettera dell'alternativa	Risposta valida
Nessuna alternativa sopra la soglia	`BRANCO`	Domanda senza risposta
Due o più alternative sopra la soglia	`DUPLA_MARCACAO`	Risposta ambigua

La soglia di riempimento controlla la sensibilità dell'algoritmo. Valori molto bassi tendono ad aumentare il numero di `DUPLA_MARCACAO`, mentre valori molto alti possono aumentare la quantità di domande classificate come `BRANCO`.

6.14.3.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Intero Q.
- Riga 2: Intero K.
- Riga 3: Numero reale Th.
- Prossime Q righe: K numeri reali, corrispondenti ai gradi di riempimento delle alternative.
- Riga 1: Interi Q e K.

Uscita:

- Q righe, ciascuna contenente la classificazione della rispettiva domanda.
- Riga finale: `OK: x  BRANCO: y  DUPLA_MARCACAO: z`.

6.14.3.5 📌 Esempi

Ingresso	Uscita	Osservazione
3 4 50 10 85 5 12 20 15 18 22 90 88 10 5	B BRANCO DUPLA_MARCACAO OK: 1 BRANCO: 1 DUPLA_MARCACAO: 1	Nella prima domanda solo `B` supera la soglia; nella seconda nessuna alternativa la supera; nella terza, `A` e `B` superano la soglia.
1 2 50.0 50 50	BRANCO OK: 0 BRANCO: 1 DUPLA_MARCACAO: 0	I valori uguali alla soglia non sono considerati marcature valide.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-ep0603.html

Figura 6.23: Simulatore EP06_03: Classificazione delle Marcature OMR

```
%%writefile EP06_03.py
# Codice Python
```

```
Overwriting EP06_03.py
```

```
TestSuite("EP06_03.py").run()
```

```
<IPython.core.display.HTML object>
```

6.14.4 EP06_04 🟡 Stimatore dell'Inclinazione tramite Mediana Angolare (*Deskew*)

Dopo il rilevamento dei bordi e l'applicazione della Trasformata di Hough, si ottiene un insieme di rette candidate all'orientamento predominante del documento. Ogni retta fornisce una stima dell'angolo di inclinazione, calcolata come

$$\text{angolo} = \text{rad2deg}(\theta) - 90.$$

Tuttavia, non tutte le rette corrispondono alle righe del documento: alcune derivano da rumori, ombre o altri elementi dell'immagine. In questo esercizio, implementerai la fase di stima robusta dell'angolo di inclinazione, filtrando i valori plausibili e calcolandone la mediana.

6.14.4.1 📋 Linee Guida di Implementazione

1. **Quantità:** Leggere l'intero M, corrispondente al numero di angoli stimati.
2. **Angoli:** Leggere gli M valori reali, in gradi.
3. **Filtraggio:** Mantenere solo gli angoli che soddisfano **strettamente** $-45 <$ angolo < 45.
4. **Assenza di candidati:** Se nessun angolo rimane dopo il filtraggio, stampare esattamente `SEM_CORRECAO`.
5. **Mediana:** Nel caso in cui esistano angoli validi:
   - se la quantità è dispari, la mediana è l'elemento centrale della sequenza ordinata;
   - se è pari, la mediana è la media aritmetica dei due elementi centrali.
6. **Uscita:** Stampare la mediana arrotondata a due cifre decimali (arrotondamento standard, *round half away from zero*, con `np.floor(img + 0.5)`).

6.14.4.2 📌 Vincoli Computazionali

- **Intervallo aperto:** angoli uguali a -45 o 45 non devono essere considerati.
- **Precisione:** calcolare la mediana utilizzando i valori originali; l'arrotondamento deve essere effettuato solo in uscita.
- **Caso vuoto:** se non ci sono angoli validi, non deve essere calcolata alcuna mediana.

6.14.4.3 🧠 Fondamenti Teorici

Situazione	Risultato
Maggior parte degli angoli concentrata attorno all'inclinazione reale	La mediana approssima l'orientamento del documento.
Pochi angoli discordanti (*outliers*)	La mediana subisce poca influenza da questi valori.
Angoli al di fuori dell'intervallo $(-45°, 45°)$	Vengono scartati prima del calcolo.
Nessun angolo valido	Non viene applicata alcuna correzione (`SEM_CORRECAO`).

La mediana viene utilizzata perché è più robusta della media in presenza di pochi valori discordanti, producendo una stima più stabile dell'inclinazione predominante del documento.

6.14.4.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Intero M.
- Riga 2: M numeri reali, corrispondenti agli angoli in gradi.

Uscita:

- Una singola riga contenente l'angolo stimato, con due cifre decimali, oppure la parola `SEM_CORRECAO` se nessun angolo è valido.

6.14.4.5 📌 Esempi

Ingresso	Uscita	Osservazione
5 -50 -10.5 2.3 2.3 47	2.30	Solo gli angoli nell'intervallo $(-45, 45)$ sono considerati; la mediana è 2,3.
4 -46 50 45 -45	SEM_CORRECAO	Nessun angolo appartiene all'intervallo aperto $(-45, 45)$.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-ep0604.html

Figura 6.24: Simulatore EP06_04: Stimatore della Pendenza per Mediana Angolare

```
%%writefile EP06_04.py
# Codice Python
```

```
Overwriting EP06_04.py
```

```
TestSuite("EP06_04.py").run()
```

```
<IPython.core.display.HTML object>
```

6.14.5 EP06_05 🟠 Normalizzazione del fondo per divisione (Correzione dell'illuminazione)

Un modulo è stato fotografato in condizioni di illuminazione non uniforme, facendo sì che un lato del foglio appaia più chiaro dell'altro. In queste condizioni, la sogliatura globale di Otsu può produrre risultati insoddisfacenti, poiché una singola soglia non separa adeguatamente testo e sfondo in tutta l'immagine. La soluzione presentata nel capitolo consiste nel **normalizzare lo sfondo**, dividendo l'immagine originale per una versione fortemente smussata di se stessa, che rappresenta l'illuminazione a bassa frequenza.

In questo esercizio, l'immagine originale e lo sfondo smussato (equivalente al risultato di un `cv2.GaussianBlur` con σ elevato) sono già forniti. Il tuo compito è implementare la fase di normalizzazione che produce l'immagine corretta.

6.14.5.1 📋 Linee guida di implementazione

1. **Dimensioni:** Leggere gli interi L (righe) e C (colonne).
2. **Immagine originale:** Leggere i valori interi $L \times C$ della matrice `img` (intensità tra 0 e 255).
3. **Sfondo stimato:** Leggere i valori interi $L \times C$ della matrice `bg` (intensità tra 0 e 255, sempre strettamente maggiori di zero).
4. **Normalizzazione:** Per ogni posizione (i, j), calcolare

$$\text{valore}(i, j) = \frac{\text{img}(i, j)}{\text{bg}(i, j)} \times 255.$$

5. **Arrotondamento:** Arrotondare il risultato all'intero più vicino (*round half away from zero*, con `np.floor(img + 0.5)`).
6. **Saturazione:** Limitare il valore ottenuto all'intervallo $[0, 255]$.
7. **Uscita:** Stampare la matrice `img_norm` risultante.

6.14.5.2 📌 Vincoli computazionali

- **Divisione per zero:** l'input garantisce $\text{bg}(i, j) > 0$ in tutte le posizioni.
- **Ordine delle operazioni:** prima arrotondare, poi applicare la saturazione.
- **Elaborazione indipendente:** ogni pixel deve essere normalizzato singolarmente, senza utilizzare informazioni dai pixel vicini.

6.14.5.3 🧠 Fondamenti teorici

Situazione	Effetto della normalizzazione
$\text{img}(i,j) = \text{bg}(i,j)$	Risultato pari a 255, corrispondente allo sfondo normalizzato.
$\text{img}(i,j) < \text{bg}(i,j)$	Risultato inferiore a 255, preservando le regioni più scure, come il testo.
$\text{img}(i,j) > \text{bg}(i,j)$	Risultato superiore a 255, successivamente saturato.
Sfondo con illuminazione non uniforme	La divisione riduce le variazioni lente dell'illuminazione, rendendo l'immagine più omogenea.

La divisione per lo sfondo stimato riduce gli effetti dell'illuminazione non uniforme e preserva il contrasto tra primo piano e sfondo, facilitando le fasi successive di segmentazione.

6.14.5.4 📦 Specifica di input e output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Prossime L righe: elementi della matrice `img`.
- Prossime L righe: elementi della matrice `bg`.

Output:

- Matrice `img_norm`, con L righe e C colonne, contenente valori interi separati da spazi.

6.14.5.5 📌 Esempi

Input	Output	Osservazione
2 2 60 120 180 40 100 100 200 80	153 255 230 128	I valori superiori a 255 devono essere saturati; $180/200 \times 255 = 229{,}5$ risulta in 230 dopo l'arrotondamento.
1 3 30 60 90 60 60 60	128 255 255	Solo il primo valore rimane al di sotto di 255 dopo la normalizzazione.

```
%%writefile EP06_05.py
# Codice Python
```

```
Overwriting EP06_05.py
```

```
TestSuite("EP06_05.py").run()
```

```
<IPython.core.display.HTML object>
```

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-ep0605.html

Figura 6.25: Simulatore EP06_05: Normalizzazione dello Sfondo per Divisione

6.14.6 EP06_06 🔴 Mappa di Varianza Locale per il Rilevamento di Trama

Una fabbrica tessile deve ispezionare rotoli di tessuto in tempo reale, senza disporre di un'immagine di riferimento — ogni rotolo presenta piccole variazioni naturali. In questa situazione, la strategia presentata nel capitolo consiste nell'analizzare l'**omogeneità locale della trama**: le regioni uniformi presentano una bassa varianza di intensità in piccoli intorni, mentre graffi, macchie e difetti di fabbricazione producono aumenti locali di tale varianza.

In questo esercizio, implementerai il nucleo di questo metodo, calcolando la varianza locale in una finestra scorrevole e generando una maschera binaria che identifica le regioni la cui varianza supera una soglia.

6.14.6.1 📋 Linee Guida di Implementazione

1. **Dimensioni e parametri:** Leggere gli interi L, C, k (dimensione della finestra, sempre dispari) e T (soglia di varianza).
2. **Immagine:** Leggere gli $L \times C$ valori interi della matrice di trama (intensità tra 0 e 255).
3. **Gestione dei bordi:** Quando la finestra supera i limiti dell'immagine, utilizzare la **replicazione del bordo**, cioè ripetere il valore del pixel valido più vicino.
4. **Media locale:** Per ogni posizione (i, j), calcolare
$$\mu(i,j) = \frac{1}{k^2} \sum_{(p,q)\in\text{finestra}} \text{trama}(p,q).$$
5. **Varianza locale:** Calcolare la varianza della popolazione della finestra,
$$\sigma^2(i,j) = \frac{1}{k^2} \sum_{(p,q)\in\text{finestra}} \left(\text{trama}(p,q) - \mu(i,j)\right)^2,$$
o, in modo equivalente,
$$\sigma^2(i,j) = \overline{x^2} - \mu(i,j)^2,$$
dove $\overline{x^2}$ rappresenta la media dei quadrati delle intensità.
6. **Arrotondamento:** Arrotondare la varianza all'intero più vicino (*round half away from zero*, con `np.floor(res_norm + 0.5)`).
7. **Soglia:** Definire $\text{maschera}(i,j) = 1$ se la varianza arrotondata è **strettamente maggiore** di T; altrimenti, definire $\text{maschera}(i,j) = 0$.
8. **Output:** Stampare la maschera binaria risultante.

6.14.6.2 📌 Vincoli Computazionali

- **Replicazione del bordo:** utilizzare il valore del pixel valido più vicino ogni volta che la finestra supera i limiti dell'immagine.
- **Varianza della popolazione:** utilizzare il denominatore k^2, mai $k^2 - 1$.
- **Confronto stretto:** la maschera deve essere calcolata utilizzando la condizione $\sigma^2_{\text{arrotondata}} > T$.
- **Finestra dispari:** il valore di k è sempre dispari, garantendo un pixel centrale.

6.14.6.3 🧠 Fondamento Teorico

Situazione	Varianza locale	Interpretazione
Regione uniforme	Bassa	Intensità simili nell'intorno.
Regione con difetto	Alta	La presenza di intensità distinte aumenta la dispersione dei valori.
Finestra piccola	Maggiore sensibilità a dettagli e rumore	Rileva alterazioni localizzate.
Finestra grande	Risposta più uniforme	Evidenzia difetti più grandi, ma riduce la precisione della loro localizzazione.

La varianza locale misura la dispersione delle intensità in un intorno. Le regioni omogenee presentano una bassa varianza, mentre le alterazioni nella trama aumentano questa misura, consentendo di identificare possibili difetti tramite una semplice sogliatura.

6.14.6.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Intero k (dispari).
- Riga 4: Intero T.
- Prossime L righe: elementi interi della matrice di trama.

Output:

- Maschera binaria (valori 0 o 1), con L righe e C colonne.

6.14.6.5 📌 Esempi

Input	Output	Osservazione
3 3 3 50 10 10 10 10 10 10 10 90 10	0 0 0 1 1 1 1 1 1	Il difetto aumenta la varianza in tutte le finestre che lo contengono.
2 2 3 5 100 100 100 100	0 0 0 0	La trama è uniforme; la varianza è nulla in tutta l'immagine.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-ep0606.html

Figura 6.26: Simulatore EP06_06: Mappa della Varianza Locale per il Rilevamento della Trama

```
%%writefile EP06_06.py
# Codice Python
```

```
Overwriting EP06_06.py
```

```
TestSuite("EP06_06.py").run()
```

```
<IPython.core.display.HTML object>
```

6.14.7 EP06_07 🟣 *Pipeline* di Ispezione Industriale: Registrazione per Traslazione e Sottrazione

In una linea di produzione, una telecamera fissa fotografa ogni pezzo che passa sul nastro trasportatore, confrontandolo con un'immagine di riferimento priva di difetti. Il problema: piccole vibrazioni del nastro spostano il pezzo rispetto alla posizione di riferimento a ogni acquisizione. Se la sottrazione di immagini viene applicata direttamente, senza correzione, lo spostamento di per sé genera già differenze enormi — **falsi positivi** che mascherano i difetti reali.

Questo è l'esercizio più completo del capitolo: devi **prima registrare** (allineare geometricamente) l'immagine acquisita usando uno spostamento noto (dx, dy), fornito da un sensore di posizione del nastro, e **solo successivamente applicare la sottrazione** con sogliatura, esattamente come descritto nella sezione sull'ispezione industriale.

6.14.7.1 📋 Linee Guida di Implementazione

1. **Dimensioni e parametri:** Leggere L, C (dimensioni delle immagini), lo spostamento intero noto dx, dy (che può essere negativo) e la soglia di rilevamento T (intero).
2. **Immagini:** Leggere la matrice di riferimento (`ref`, $L \times C$, senza difetti) e la matrice acquisita (`cap`, $L \times C$, possibilmente spostata e con difetti).
3. **Registrazione per traslazione:** Costruire l'immagine allineata `alin` applicando lo spostamento (dx, dy) ricevuto:

$$\text{alin}(i,j) = \begin{cases} \text{cap}(i+dy,\ j+dx), & \text{se } (i+dy,\ j+dx) \in [0,L) \times [0,C) \\ 0, & \text{altrimenti} \end{cases}$$

4. **Riempimento dei bordi:** Le posizioni che "escono" dall'immagine acquisita dopo lo spostamento ricevono il valore **0** (*zero-padding* — al di fuori del campo visivo della telecamera; **nota che questo esercizio usa zero, diversamente dalla replicazione dei bordi dell'EP06_06**).
5. **Differenza assoluta:** Calcolare, pixel per pixel,

$$\text{diff}(i, j) = |\text{ref}(i, j) - \text{alin}(i, j)|$$

6. **Sogliatura:** Definire $\text{maschera}(i, j) = 1$ se $\text{diff}(i, j) > T$; altrimenti, $\text{maschera}(i, j) = 0$.
7. **Output:** In questo ordine — (a) la matrice `alin` ($L \times C$); (b) la maschera dei difetti ($L \times C$); (c) un'ultima riga con il totale dei pixel classificati come difettosi.

6.14.7.2 📌 Vincoli Computazionali

- ***Zero-padding*, non replicazione:** le posizioni al di fuori dei limiti dell'immagine acquisita, dopo lo spostamento, valgono esattamente 0 — questo è il punto che differenzia maggiormente questo esercizio dall'EP06_06.
- **Confronto stretto:** $\text{diff}(i, j) > T$.
- **Segno di (dx, dy):** lo spostamento può essere positivo o negativo; la formula del passo 3 deve essere applicata letteralmente, senza invertire i segni.
- **Tutti i valori sono interi:** non c'è arrotondamento in questa fase.

6.14.7.3 🧠 Fondamentazione Teorica

Fase omessa	Conseguenza
Saltare la registrazione geometrica	L'intero bordo dell'immagine (introdotto dallo spostamento) viene marcato come "difetto" — falso positivo sistematico
Registrazione con (dx, dy) errato	Il pezzo e il riferimento rimangono disallineati; la sottrazione rileva contorni spostati, non difetti reali
Soglia T troppo bassa	Il rumore di acquisizione (variazioni di 1–2 livelli di grigio) viene confuso con un difetto
Soglia T troppo alta	I difetti sottili non vengono rilevati

La registrazione geometrica e la sottrazione sono fasi complementari: la prima garantisce che entrambe le immagini rappresentino esattamente la stessa scena nello stesso riferimento spaziale; la seconda isola ciò che è realmente cambiato tra di esse — idealmente, solo i difetti.

6.14.7.4 📦 Specifica di Input e Output (VPL)

Input:

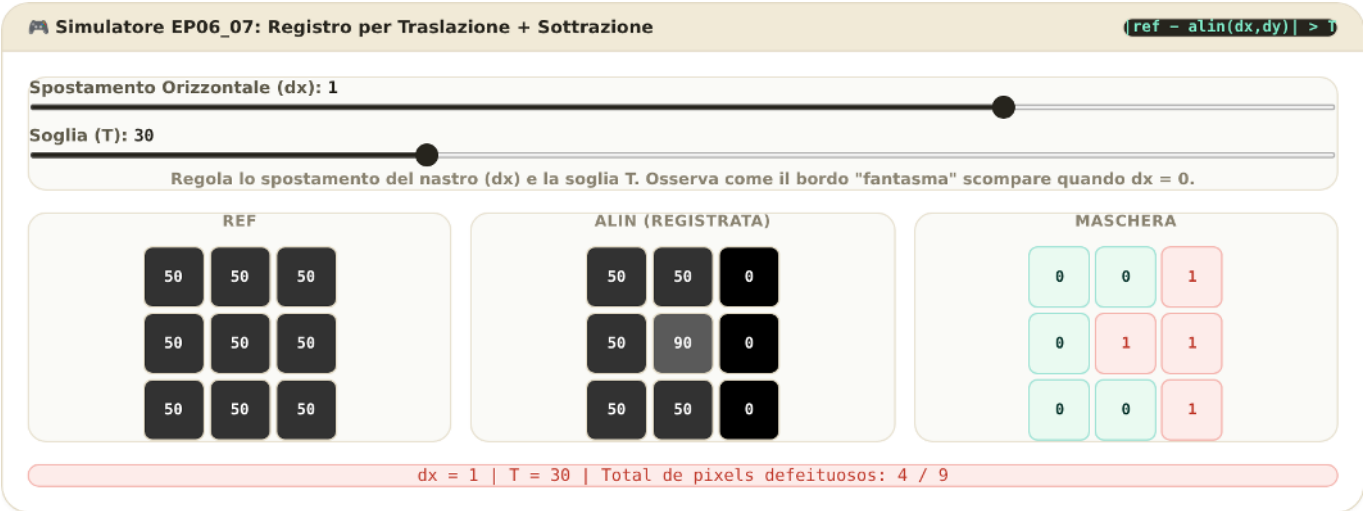
- Riga 1: Intero L.
- Riga 2: Intero C.
- Riga 3: Due interi dx e dy, separati da uno spazio.
- Riga 4: Intero T.
- Prossime L righe: elementi interi della matrice `ref`.
- Prossime L righe: elementi interi della matrice `cap`.

Output:

- L righe con la matrice `alin`.
- L righe con la maschera dei difetti (0/1).
- Ultima riga: `Totale di pixel difettosi: X`.

6.14.7.5 📌 Esempi

Input	Output	Osservazione
3 3 1 0 30 50 50 50 50 50 50 50 50 50 0 50 50 0 50 90 0 50 50	50 50 0 50 90 0 50 50 0 0 0 1 0 1 1 0 0 1 Totale di pixel difettosi: 4	$dx = 1$ sposta la lettura di una colonna verso destra; l'ultima colonna di `alin` rimane senza corrispondenza (diventa 0) e viene sistematicamente marcata; anche il difetto reale (90) viene rilevato.
2 2 0 0 20 10 10 10 10 10 10 10 60	10 10 10 60 0 0 0 1 Totale di pixel difettosi: 1	Senza spostamento ($dx = dy = 0$): `alin` è identica a `cap`; solo il difetto reale (60) viene rilevato.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-ep0607.html

Figura 6.27: Simulatore EP06_07: *Pipeline* di Ispezione — Registrazione per Traslazione e Sottrazione

```
%%writefile EP06_07.py
# Codice Python
```

```
Overwriting EP06_07.py
```

```
TestSuite("EP06_07.py").run()
```

```
<IPython.core.display.HTML object>
```

6.14.8 EP06_08 ⚫ Segmentazione e Decodifica Reale di *QRCode* con OpenCV

Negli esercizi precedenti, le grandezze intermedie del *pipeline* di elaborazione delle immagini — come aree, perimetri, varianze e spostamenti — sono state fornite direttamente o calcolate a partire da matrici numeriche,

senza la necessità di librerie specializzate di Visione Artificiale. In questo esercizio conclusivo del capitolo, questa restrizione viene rimossa intenzionalmente: verrà utilizzata la libreria **OpenCV** (`cv2`) per localizzare e decodificare un *QRCode* reale presente in una scena.

La proposta riproduce un flusso semplificato di sistemi impiegati nell'ispezione visiva, nell'automazione industriale e nella lettura automatica di documenti. Per mantenere l'input dei dati accessibile al contesto educativo, il caricamento dell'immagine sarà integrato nella libreria didattica `morph`, tramite la funzione `mm.readImg`.

La scena è fornita nel formato **PGM ASCII (P2)** e contiene un singolo *QRCode* valido, oltre a diversi **oggetti distrattori**, come rettangoli, regioni di rumore texturizzato e blocchi isolati. La segmentazione basata esclusivamente su proprietà geometriche — come area e forma approssimativamente quadrata — è necessaria per ridurre lo spazio di ricerca, ma non è sufficiente per identificare il codice corretto. La conferma finale sarà effettuata esclusivamente tramite il tentativo di decodifica utilizzando `cv2.QRCodeDetector`, una procedura compatibile con applicazioni reali di riconoscimento automatico.

6.14.8.1 📋 Linee Guida di Implementazione

1. **Lettura delle dimensioni e dei parametri**

   Leggere, in questo ordine, tramite l'input standard:

   - una riga contenente il numero di righe L;
   - una riga contenente il numero di colonne C;
   - una riga contenente i quattro parametri dell'algoritmo separati da spazio:
     - soglia di binarizzazione T (intero);
     - area minima A_{min} (intero);
     - tolleranza di aspetto tol (reale);
     - margine M (intero, in pixel).

2. **Caricamento dell'immagine**

   Utilizzare la funzione didattica `f = mm.readImg(L, C)` per leggere i valori $L \times C$ dell'immagine in scala di grigi, ottenendo un *array* NumPy di tipo `uint8`.

3. **Binarizzazione**

   Applicare una sogliatura binaria invertita utilizzando la soglia T. Ogni pixel dell'immagine originale con intensità strettamente maggiore di T deve essere convertito a 255, mentre i rimanenti devono assumere il valore 0.

4. **Rilevamento dei contorni**

   Estrarre i componenti connessi esterni utilizzando `cv2.findContours(...)` con i parametri:

   - `cv2.RETR_EXTERNAL`;
   - `cv2.CHAIN_APPROX_SIMPLE`.

5. **Filtraggio geometrico**

   Per ogni contorno trovato:

   - calcolare il rettangolo delimitatore `(x, y, w, h)` tramite `cv2.boundingRect`;
   - mantenere solo i candidati che soddisfano simultaneamente:

     Area minima

     $$w \times h > A_{\text{min}}$$

     Rapporto d'aspetto

$$\left|\frac{w}{h} - 1\right| \leq \text{tol}$$

6. **Ordinamento dei candidati**

   Ordinare i candidati per area del rettangolo delimitatore

   $$w \times h$$

   in ordine decrescente.

   In caso di parità, preservare l'ordine originariamente restituito da `cv2.findContours`.

7. **Verifica tramite decodifica**

   Per ogni candidato, seguendo l'ordine stabilito:

   - espandere il rettangolo di M pixel nelle quattro direzioni;
   - limitare gli indici per rimanere all'interno dell'immagine;
   - estrarre il ritaglio direttamente dall'immagine originale `f`;
   - applicare `cv2.QRCodeDetector().detectAndDecode(...)` su tale ritaglio.

8. **Criterio di arresto**

   Interrompere immediatamente l'elaborazione quando il primo candidato produce una *stringa* decodificata non vuota.

9. **Caso non trovato**

   Se nessun candidato viene decodificato con successo, stampare esattamente: `QRCODE_NAO_ENCONTRATO`

10. **Output (caso trovato)**

    Stampare due righe.

    Prima riga: `riga colonna altezza larghezza` utilizzando il rettangolo delimitatore **originale**, prima dell'espansione tramite il margine M.

    Seconda riga: `testo_decodificato`

6.14.8.2 📌 Vincoli Computazionali

- Utilizzare funzioni OpenCV per eseguire la binarizzazione, il rilevamento dei contorni, il calcolo del rettangolo delimitatore e la decodifica del QRCode.
- Il filtraggio geometrico deve avvenire obbligatoriamente prima della fase di decodifica.
- Utilizzare esclusivamente la soglia fissa T fornita in input. Non è consentito utilizzare metodi automatici di sogliatura, come Otsu o sogliatura adattativa.
- Garantire che i ritagli inviati al decodificatore rimangano all'interno dei limiti dell'immagine.

6.14.8.3 🧠 Fondamenti Teorici

Fase	Ruolo nel *pipeline*	Conseguenza se omessa
Filtraggio geometrico	Riduce lo spazio di ricerca selezionando solo regioni compatibili con la geometria attesa di un QRCode.	Il decodificatore elaborerebbe tutti i contorni, inclusi rumori e oggetti distrattori.
Decodifica	Conferma semanticamente se il candidato contiene un QRCode valido.	Oggetti geometricamente simili potrebbero essere classificati erroneamente come QRCode.

Fase	Ruolo nel *pipeline*	Conseguenza se omessa
Margine M	Preserva la *zona di silenzio* attorno al codice, facilitandone il rilevamento.	L'assenza di tale margine può impedire l'allineamento e la corretta lettura del codice.

Questo esercizio integra concetti studiati durante il capitolo in un unico *pipeline* di Visione Artificiale. La segmentazione riduce l'insieme delle regioni candidate tramite caratteristiche geometriche, mentre la fase di decodifica valida il contenuto della regione utilizzando un algoritmo specializzato di riconoscimento.

6.14.8.4 📦 Specifica di Input e Output (VPL)

Struttura di Input

```
L
C
T A_min tol M
[matrice dell'immagine]
```

Struttura di Output (Successo)

```
riga colonna altezza larghezza
testo_decodificato
```

Struttura di Output (Fallimento)

```
QRCODE_NAO_ENCONTRATO
```

6.14.8.5 📌 File di Riferimento (.pgm)

A scopo di validazione, debug locale e analisi di matrici reali di pixel, i file immagine generati nel formato ASCII P2 sono disponibili nella directory del progetto. Puoi utilizzarli per testare con i decoder del tuo cellulare l'aderenza del tuo codice (salvare i *.pgm localmente per visualizzarli):

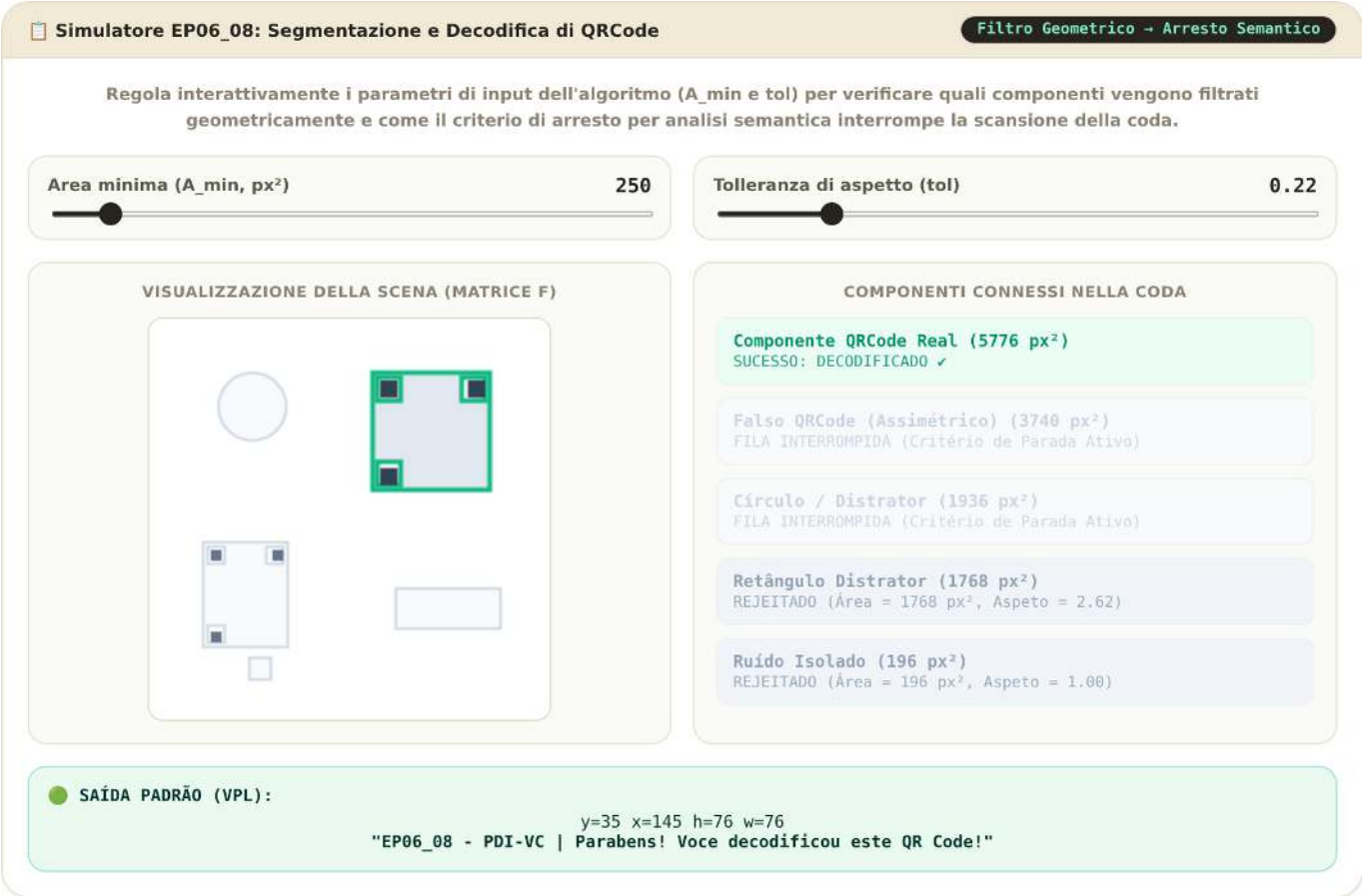
- 📥 **Caso 1: Pattern Normale** – Contiene un singolo codice perfettamente centralizzato con semplici distrattori geometrici alla periferia.
- 📥 **Caso 2: Scenario Complesso** – Presenta una maggiore densità di rumore texturizzato e molteplici candidati distrattori che mettono alla prova i limiti del filtraggio per aspetto.
- 📥 **Caso 3: Messaggio Esteso** – Contiene un QRCode strutturato a partire da una stringa di caratteri di lunghezza maggiore, generando una maggiore densità di moduli interni.
- 📥 **Caso 4: Geometria Compatta** – Valuta il comportamento del *pipeline* in condizioni ottimizzate di contrasto e posizionamento limite.
- 📥 **Caso 5: Scenario di Esclusione** – Immagine composta puramente da elementi distrattori di alta area, progettata per validare il comportamento di fallimento controllato del programma.

```
%%writefile EP06_08.py
# Codice Python
```

```
Overwriting EP06_08.py
```

```
TestSuite("EP06_08.py").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap06/sim-ep0608.html

Figura 6.28: Simulatore EP06_08: Segmentazione Geometrica + Verifica tramite Decodifica di QRCode

Capitolo 7

Classificazione delle Immagini e Riconoscimento di Pattern

Executar Colab Abrir GitHub

Nel **Capitolo 6**, la transizione dalla Parte I alla Parte II è stata presentata attraverso due applicazioni che richiedevano già decisioni automatizzate: il riconoscimento dei segni nei fogli di risposta (OMR) e il rilevamento dei difetti nell'ispezione industriale. In entrambi i casi, tuttavia, le decisioni dipendevano da regole geometriche e soglie definite manualmente, come determinare se un disco fosse sufficientemente circolare o se una regione fosse abbastanza scura.

Questo capitolo formalizza il problema più generale alla base di queste applicazioni: dato un insieme di esempi etichettati, come addestrare un sistema per **classificare automaticamente** nuove immagini o regioni di interesse? Questa questione è al centro del **Riconoscimento di Pattern**, disciplina che fonda gran parte dei compiti moderni della Visione Artificiale, dalla classificazione delle immagini al rilevamento degli oggetti e alla segmentazione semantica, esplorati nei prossimi capitoli.

Verranno studiati i principali **descrittori classici dell'immagine** (colore, tessitura e forma/gradiente) e il classificatore **k-Nearest Neighbors** (*k-Nearest Neighbors* — k-NN), scelto per la sua semplicità concettuale e per evidenziare, in modo diretto, la relazione tra lo spazio delle caratteristiche, le metriche di distanza e le frontiere di decisione — concetti che rimangono centrali anche nei classificatori basati su reti neurali profonde, studiati nel capitolo finale di questa parte.

7.1 Obiettivi del Capitolo

Al termine di questo capitolo, lo studente dovrebbe essere in grado di:

- **Comprendere il *pipeline* classico di riconoscimento dei pattern**: acquisizione, pre-elaborazione, estrazione dei descrittori, classificazione e valutazione;
- **Estrarre e interpretare descrittori classici** di colore, tessitura (*Local Binary Patterns* — LBP) e forma/gradiente (*Histogram of Oriented Gradients* — HOG);
- **Implementare e addestrare un classificatore k-NN** per compiti di classificazione delle immagini;
- **Valutare i classificatori** attraverso metriche come accuratezza, matrice di confusione, precisione e recall;
- **Analizzare l'effetto del parametro k** e della dimensionalità dello spazio delle caratteristiche sulle prestazioni del classificatore;
- **Riconoscere i limiti dei descrittori artigianali** (*hand-crafted features*) e comprendere la motivazione per la transizione, nei prossimi capitoli, verso descrittori appresi automaticamente.

7.2 Configurazione dell'Ambiente

Gli esempi di questo capitolo utilizzano librerie ampiamente impiegate nell'Elaborazione Digitale delle Immagini, nella Visione Artificiale e nell'Apprendimento Automatico. Il blocco seguente installa i pacchetti necessari; in ambienti che già li possiedono, l'esecuzione può essere ignorata.

```python
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup()
from morph import mm

import importlib
import subprocess
import sys

def setup_cap07():
    """Installa le librerie mancanti necessarie per questo capitolo
    (visione computazionale e apprendimento automatico)."""
    pacotes = {
        "cv2": "opencv-python",
        "skimage": "scikit-image",
        "numpy": "numpy",
        "sklearn": "scikit-learn",
        "matplotlib": "matplotlib",
        "pandas": "pandas",
        "seaborn": "seaborn",
        "tabulate": "tabulate",
        "kaleido": "kaleido",
    }
    for modulo, pacote in pacotes.items():
        if importlib.util.find_spec(modulo) is None:
            resultado = subprocess.run(
                [sys.executable, "-m", "pip", "install", "-q", pacote]
            )
            if resultado.returncode != 0:
                print(f"[AVVISO] Impossibile installare {pacote} (necessario per il modulo
                {modulo}).")


setup_cap07()

# ==========================================================
# Librerie
# ==========================================================

# Calcolo scientifico
import numpy as np
import pandas as pd

# Visualizzazione
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import seaborn as sns

# Visione computazionale
import cv2
from skimage import data as skdata
from skimage.feature import hog, local_binary_pattern

# Apprendimento automatico
from sklearn.datasets import load_digits, make_classification
from sklearn.model_selection import cross_val_score, train_test_split
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score,
    precision_score,
    recall_score,
)
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0
```

7.3 Un Problema Concreto: Classificazione della Frutta

Prima di presentare i fondamenti teorici, si consideri il seguente problema, che verrà utilizzato come esempio lungo tutto questo capitolo per illustrare i principali concetti del riconoscimento di pattern.

Lo scenario: Una fattoria automatizzata utilizza un sistema di Visione Artificiale per separare mele, banane e arance sulle linee di confezionamento.

La sfida: I frutti arrivano sul nastro in diverse posizioni e orientamenti, in condizioni di illuminazione che possono variare. Inoltre, foglie, ombre e piccole occlusioni possono rendere difficile la loro identificazione. Come sviluppare un sistema in grado di classificarli correttamente?

Un possibile approccio:

1. Estrarre descrittori che rappresentino caratteristiche rilevanti dei frutti:
   - **Colore:** distribuzione predominante dei colori;
   - **Texture:** differenze nella superficie della buccia;
   - **Forma:** caratteristiche geometriche del contorno.
2. Addestrare un classificatore utilizzando esempi precedentemente etichettati.
3. Utilizzare il modello addestrato per classificare automaticamente nuovi frutti.

La Figura 7.1 illustra, in modo concettuale, come diversi frutti possano essere rappresentati in uno spazio delle caratteristiche tridimensionale.

> **Rifletti prima di continuare**
>
> Se ogni frutto fosse rappresentato solo dai valori di intensità dei suoi pixel, sarebbe possibile distinguerli in modo affidabile? Che tipi di informazioni potrebbero essere estratti dall'immagine per facilitare questo compito?

```
np.random.seed(42)

centros = {
    "Maçã":    [0.8, 0.2, 0.9],
    "Banana":  [0.3, 0.1, 0.2],
    "Laranja": [0.9, 0.8, 0.8],
}

fig = plt.figure(figsize=(5, 5))
ax = fig.add_subplot(projection="3d")

for fruta, centro, cor, marcador in zip(
```

```
    centros,
    centros.values(),
    ["red", "gold", "orange"],
    ["o", "s", "^"],
):
    X = np.clip(np.random.normal(centro, 0.08, (70, 3)), 0, 1)
    ax.scatter(X[:, 0], X[:, 1], X[:, 2],
               c=cor, marker=marcador, s=35,
               alpha=0.7, label=fruta)

ax.set(
    xlim=(0,1), ylim=(0,1), zlim=(0,1),
    xlabel="Intensidade de cor",
    ylabel="Textura",
    zlabel="Forma",
    title="Espaço de Características"
)
ax.view_init(elev=25, azim=-60)
ax.legend(title="Frutas")
ax.zaxis.labelpad = 0.01

plt.tight_layout()
plt.show()
```

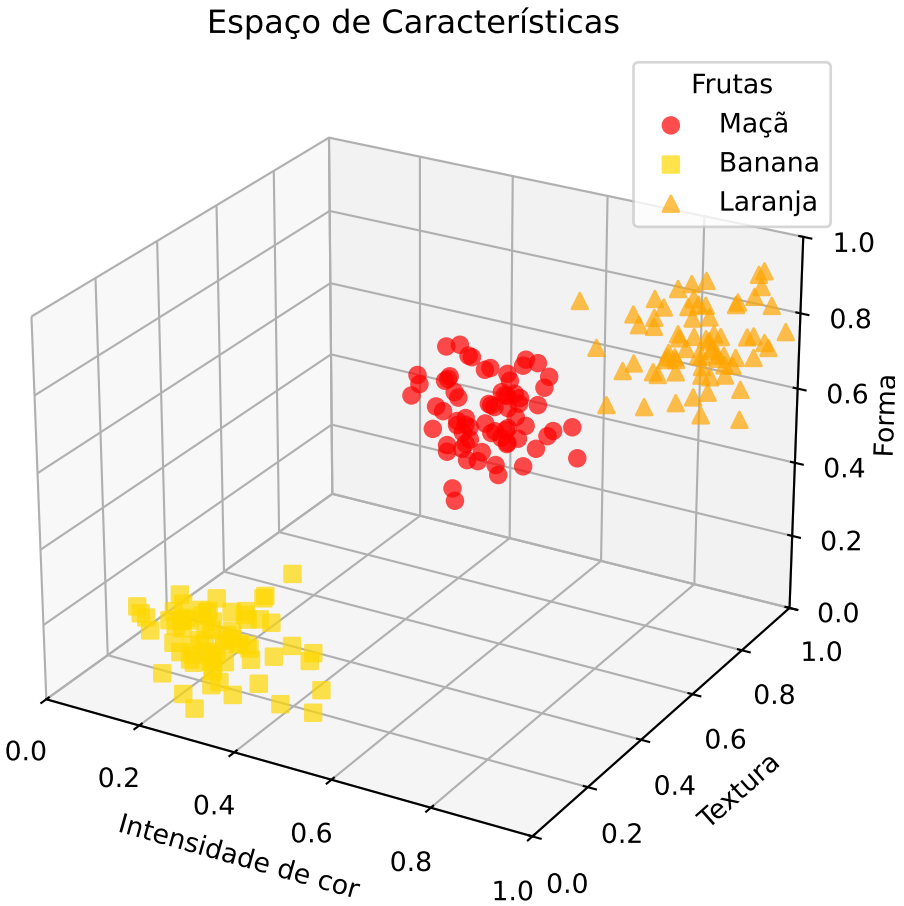


Figura 7.1: Esempio motivazionale: diversi frutti che formano raggruppamenti distinti in uno spazio delle caratteristiche.

7.4 Panoramica del Capitolo: Il *Pipeline* Classico di Classificazione delle Immagini

Prima di procedere, è utile presentare una visione integrata di ciò che verrà studiato. La Figura 8.1 mostra il flusso generale di un sistema classico di classificazione delle immagini, dall'immagine di input fino alla fase di assegnazione dell'etichetta finale. Nelle prossime sezioni, ciascuna delle fasi di questo processo verrà studiata in dettaglio.

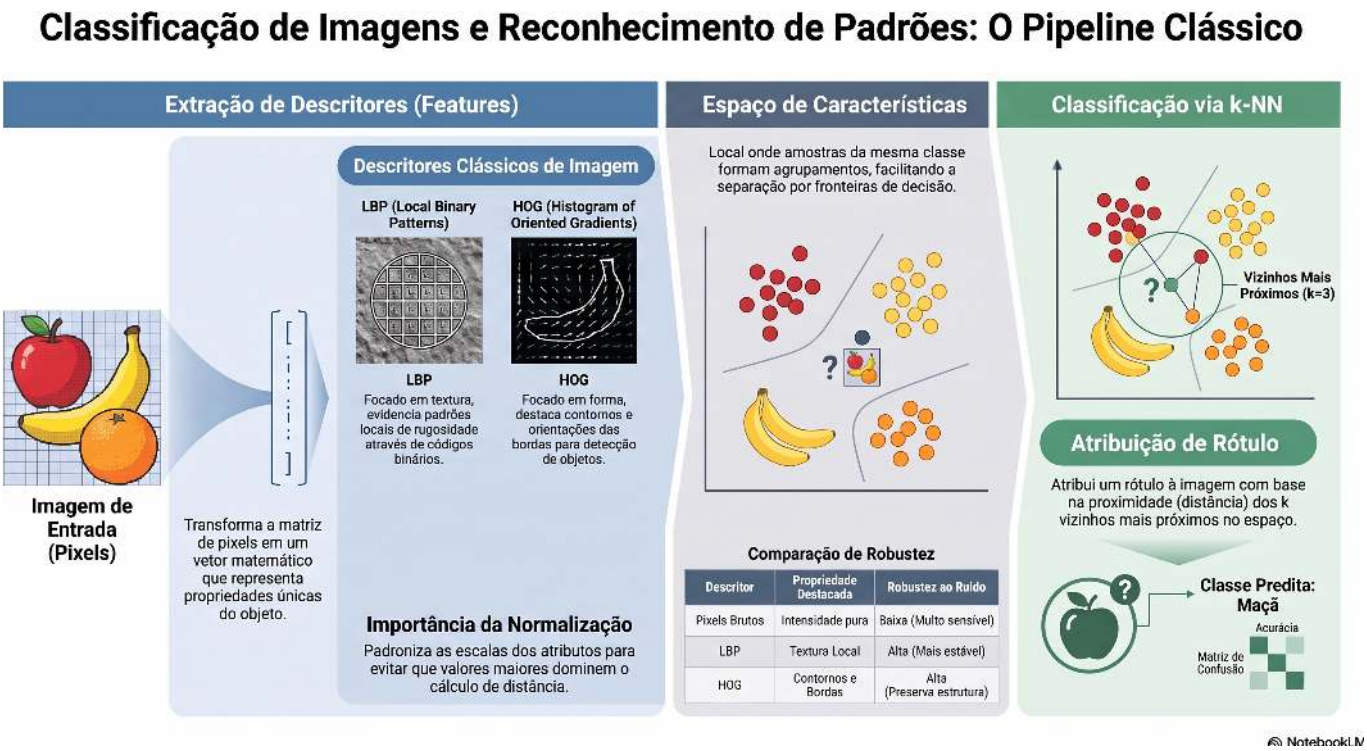


Figura 7.2: Panoramica del *pipeline* classico di classificazione delle immagini: estrazione dei descrittori (LBP, HOG), formazione dello spazio delle caratteristiche e classificazione tramite k-NN. Non si applica ai modelli di Deep Learning (CNN, YOLO), che apprendono *feature end-to-end* direttamente dai pixel. **Fonte:** elaborato con l'ausilio di Gemini Notebook ({GOOGLE}, 2025).

> **ℹ Ambito di questo capitolo**
>
> In questo capitolo, l'attenzione ricade esclusivamente sul classificatore k-NN, per la sua semplicità didattica e per illustrare in modo intuitivo il concetto di spazio delle caratteristiche. Altri classificatori tradizionali ampiamente utilizzati nel Riconoscimento di Pattern — come Alberi Decisionali, Regole di Classificazione e Macchine a Vettori di Supporto (SVM) — sono discussi in profondità in Quilici-gonzalez (2014), in particolare nella sua 2ª edizione, attualmente in produzione (QUILICI-GONZALEZ, 2026).

7.5 Fondamenti del Riconoscimento di Pattern

Un sistema di **riconoscimento di pattern** ha come obiettivo assegnare una categoria (etichetta) a un'osservazione — un'immagine intera, una regione di interesse o un segnale — basandosi su esempi precedentemente etichettati. In generale, questo processo è organizzato nelle seguenti fasi:

1. **Acquisizione:** ottenimento dell'immagine o del segnale da classificare;
2. **Pre-elaborazione:** normalizzazione, rimozione del rumore, correzione geometrica o dell'illuminazione — fasi già studiate nei capitoli precedenti;
3. **Estrazione di descrittori (*feature*):** trasformazione dell'immagine in un **vettore di caratteristiche** di dimensione fissa, che rappresenta le proprietà rilevanti per il compito di classificazione;

4. **Classificazione:** applicazione di un modello che associa il vettore di caratteristiche a una classe;
5. **Valutazione:** analisi delle prestazioni del modello su un insieme di dati indipendente da quello utilizzato per l'addestramento.

L'insieme di tutti i vettori di caratteristiche possibili costituisce lo **spazio delle caratteristiche** (*feature space*). Un buon descrittore produce rappresentazioni che avvicinano, in questo spazio, osservazioni della stessa classe e allontanano osservazioni di classi distinte. Questa proprietà favorisce metodi di classificazione basati sulla prossimità, come il **k-NN**, e avvantaggia anche diversi altri classificatori.

La Figura 7.1 illustra questo concetto in modo schematico: ogni frutto è rappresentato da un punto in uno spazio delle caratteristiche a tre dimensioni (colore, consistenza e forma). Sebbene questo spazio sia solo una semplificazione didattica, esso mostra come i campioni della stessa classe tendano a formare raggruppamenti, mentre classi diverse occupano regioni distinte, facilitando il compito di classificazione.

7.6 Estrazione di Descrittori Classici

Prima della popolarizzazione delle reti neurali profonde, i descrittori di immagine erano, nella maggior parte dei casi, progettati manualmente da esperti (*hand-crafted features*), sulla base di proprietà statistiche o geometriche note. Tre famiglie classiche sono particolarmente rilevanti:

- **Descrittori di colore:** istogrammi di intensità o di tinta, che catturano la distribuzione dei valori cromatici di una regione, già introdotti nel **Capitolo 3** tramite la funzione `mm.hist`;
- **Descrittori di texture:** catturano pattern locali di ripetizione, rugosità o orientamento, come il *Local Binary Patterns* (LBP), studiato in seguito;
- **Descrittori di forma/gradiente:** descrivono la distribuzione dei bordi e delle orientazioni del gradiente, come l'*Histogram of Oriented Gradients* (HOG), ampiamente impiegato nel rilevamento di persone e altri oggetti.

Per confrontare l'informazione catturata da ciascun approccio, la Figura 7.3 mostra come diverse tecniche "vedono" la stessa immagine.

```
# Caricare immagine di esempio
imagem = skdata.camera()

# Applicare descrittori
lbp_img = local_binary_pattern(
    imagem,
    P=8,
    R=1,
    method="uniform"
    )

# Convertire l'LBP in RGB solo per facilitare la visualizzazione
lbp_norm = (lbp_img - lbp_img.min()) / (lbp_img.max() - lbp_img.min() + 1e-8)
lbp_rgb = (cm.nipy_spectral(lbp_norm)[..., :3] * 255).astype("uint8")

hog_features, hog_img = hog(
    imagem,
    orientations=9,
    pixels_per_cell=(8, 8),
    cells_per_block=(2, 2),
    visualize=True,
)

# Visualizzazione standard
mm.show(
    [imagem, lbp_rgb, hog_img],
    titles=[
        "Immagine Originale\n(come la vede l'uomo)",
```

```
        "LBP: Texture\n(ogni colore = un codice LBP)",
        "HOG: Gradient e Contorni\n(regioni chiare = maggiore intensità)",
    ],
    cols=3,
    figsize=(12, 4),
)

print("Osserva come ogni descrittore evidenzia proprietà diverse:")
print("• LBP: evidenzia pattern locali di texture.")
print("• HOG: evidenzia contorni e orientamenti dei bordi.")
print("• Immagine originale: contiene solo i valori di intensità.")
```

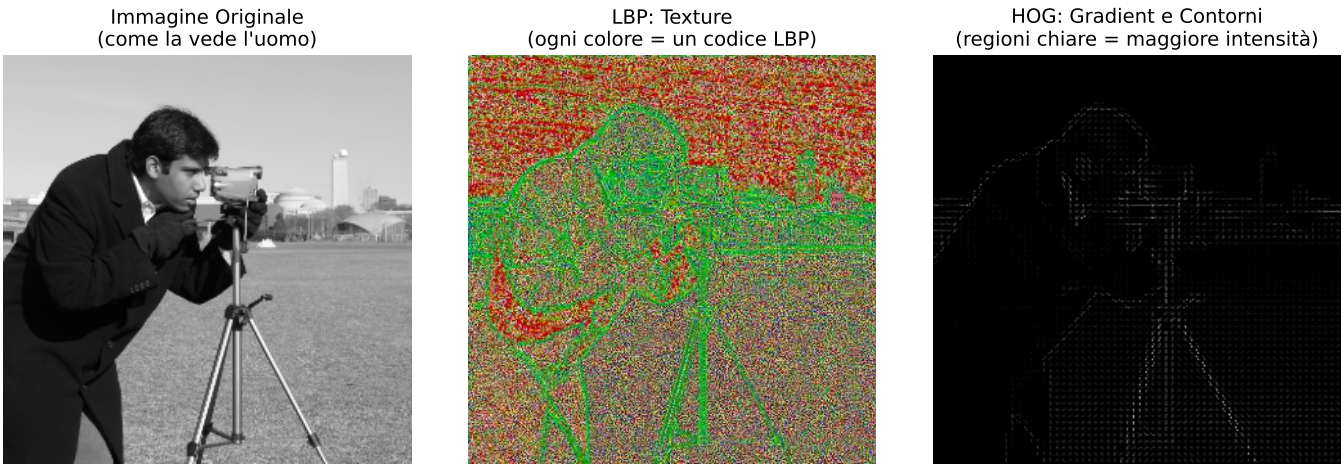


Figura 7.3: Confronto visivo di diversi descrittori applicati alla stessa immagine. Ogni descrittore rivela aspetti distinti della scena.

```
Osserva come ogni descrittore evidenzia proprietà diverse:
• LBP: evidenzia pattern locali di texture.
• HOG: evidenzia contorni e orientamenti dei bordi.
• Immagine originale: contiene solo i valori di intensità.
```

7.6.1 *Local Binary Patterns* (LBP)

L'LBP è un descrittore di tessitura che codifica, per ogni pixel centrale g_c, la relazione tra la sua intensità e quella dei P vicini disposti in un intorno circolare di raggio R:

$$\mathrm{LBP}_{P,R}(x_c, y_c) = \sum_{p=0}^{P-1} s(g_p - g_c)\, 2^p, \qquad s(z) = \begin{cases} 1, & z \geq 0 \\ 0, & z < 0 \end{cases}$$

dove:

- (x_c, y_c) sono le coordinate del pixel centrale;
- g_c è l'intensità del pixel centrale;
- g_p è l'intensità del p-esimo pixel vicino;
- P è il numero di vicini considerati;
- R è il raggio dell'intorno circolare;
- p è l'indice del vicino, con $p = 0, 1, \ldots, P-1$;
- $s(z)$ è la funzione soglia definita nell'equazione, dove $z = g_p - g_c$; essa assume valore 1 quando $z \geq 0$ e 0 quando $z < 0$;
- 2^p corrisponde al peso binario associato al p-esimo vicino.

Il codice LBP ottenuto descrive il pattern locale di contrasto attorno al pixel. L'istogramma di questi codici forma un vettore di caratteristiche compatto per rappresentare la tessitura dell'immagine (Figura 7.4). In questo capitolo si utilizza la variante **uniforme**, che raggruppa i pattern non uniformi in un'unica categoria, riducendo la dimensionalità e aumentando la robustezza del descrittore.

```
plt.figure(figsize=(6, 4))

plt.hist(
    lbp_img.ravel(),
    bins=np.arange(-0.5, lbp_img.max() + 1.5, 1),
    density=True,
    edgecolor="black",
)

plt.title("Histograma dos códigos LBP")
plt.xlabel("Código LBP")
plt.ylabel("Frequência relativa")
plt.xticks(range(int(lbp_img.max()) + 1))
plt.grid(axis="y", alpha=0.3)

plt.tight_layout()
plt.show()
```

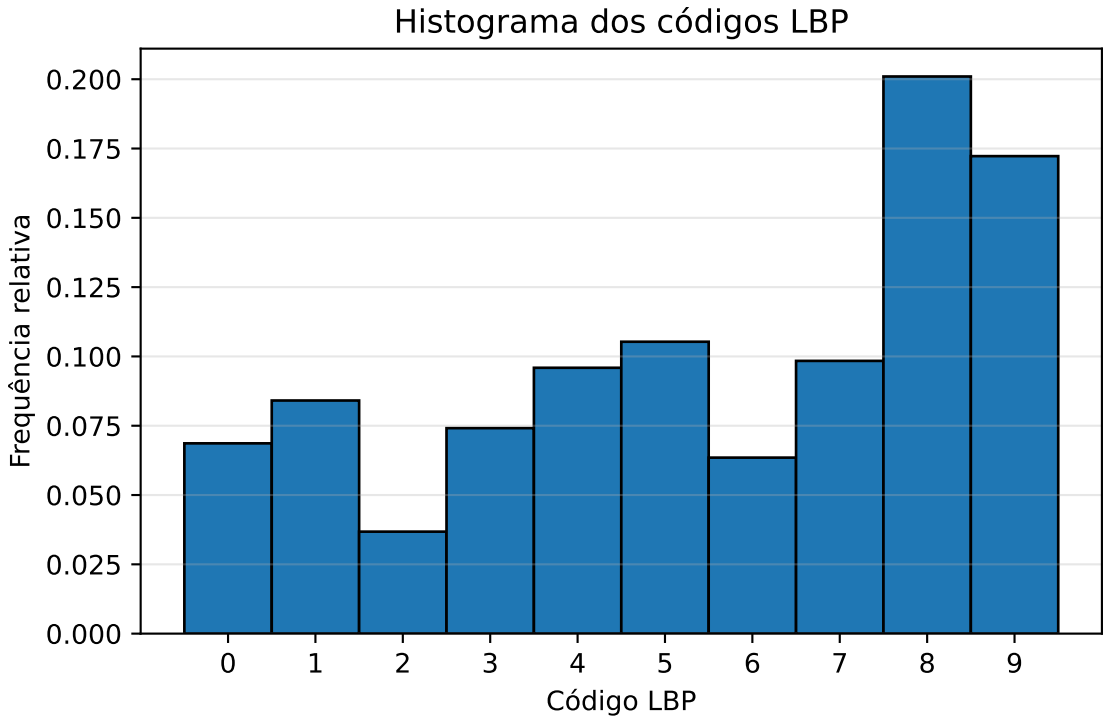


Figura 7.4: Istogramma dei codici LBP dell'immagine del cameraman. Ogni barra rappresenta la frequenza relativa di un codice LBP, formando il vettore di caratteristiche utilizzato per descrivere la texture.

Funzione `local_binary_pattern`

L'implementazione utilizzata in questo capitolo è fornita dalla libreria `scikit-image`:

```
local_binary_pattern(
    imagem,
    P=8,
    R=1,
    method="uniform"
)
```

dove:

- `image`: immagine in scala di grigi;
- `P`: numero di vicini equidistanti nell'intorno circolare;
- `R`: raggio dell'intorno, in pixel;
- `method`: strategia di codifica. In questo capitolo si utilizza il valore `"uniform"`.

L'equazione presentata in precedenza descrive l'**LBP originale**. Nell' implementazione adottata in questo capitolo, l'opzione `method="uniform"` calcola inizialmente tale codice e, successivamente, rimappa i pattern non uniformi in un'unica categoria, riducendo la dimensionalità del descrittore e rendendolo più robusto a piccole variazioni locali.

La Figura 7.3 presenta la rappresentazione visiva dell'LBP, mentre la Figura 7.4 mostra l'istogramma dei codici LBP utilizzato come vettore di caratteristiche.

Il **Progetto Pratico 2** (sezione **Confronto di Descrittori per la Classificazione di Trame**) impiega l'LBP nella classificazione di diversi tipi di trama sintetica.

7.6.2 *Histogram of Oriented Gradients* (HOG)

L'HOG è un descrittore che rappresenta la forma di un oggetto tramite la distribuzione delle orientazioni del gradiente locale. Come nell'operatore di Canny (**Capitolo 6**), si calcola inizialmente il gradiente:

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}, \qquad \theta(x,y) = \operatorname{atan2}\left(\frac{\partial f}{\partial y}, \frac{\partial f}{\partial x}\right).$$

dove:

- $f(x,y)$ è l'intensità dell'immagine nel pixel (x,y);
- $\frac{\partial f}{\partial x}$ e $\frac{\partial f}{\partial y}$ sono, rispettivamente, le derivate parziali dell'immagine nelle direzioni orizzontale e verticale;
- $|\nabla f(x,y)|$ è la magnitudine del vettore gradiente nel pixel (x,y), che indica l'intensità della variazione locale dell'immagine;
- $\theta(x,y)$ è l'orientazione del vettore gradiente nel pixel (x,y), calcolata tramite la funzione atan2, il cui risultato appartiene all'intervallo $(-\pi,\pi]$.

Sebbene $\theta(x,y)$, come calcolata dalla funzione atan2, appartenga all'intervallo $(-\pi,\pi]$, l'implementazione standard dell'HOG utilizza il **gradiente non firmato** (*unsigned*): orientazioni opposte (ad esempio, 0 e π) vengono trattate come equivalenti, e gli angoli vengono mappati nell'intervallo $[0,\pi)$ prima della costruzione dell'istogramma. Questa scelta rende il descrittore invariante alla direzione del contrasto (ad esempio, un bordo chiaro-scuro e un bordo scuro-chiaro producono la stessa orientazione).

L'immagine viene quindi suddivisa in **celle** (*cells*). Per ciascuna cella, si costruisce un istogramma delle orientazioni del gradiente, ponderato dalla magnitudine corrispondente. La concatenazione degli istogrammi di tutte le celle forma il vettore di caratteristiche HOG, che rappresenta la distribuzione spaziale delle orientazioni del gradiente e cattura informazioni sulla forma e sui contorni dell'oggetto (Figura 7.5).

```
n = 100

plt.figure(figsize=(8, 3))
plt.bar(
    range(n),
```

```
    hog_features[:n],
    width=0.9
)

plt.title("Primeiros componentes do vetor HOG")
plt.xlabel(f"Índice do componente (0-{n-1}, de um total de {hog_features.shape[0]})")
plt.ylabel("Valor normalizado")
plt.grid(axis="y", alpha=0.3)

plt.tight_layout()
plt.show()
```

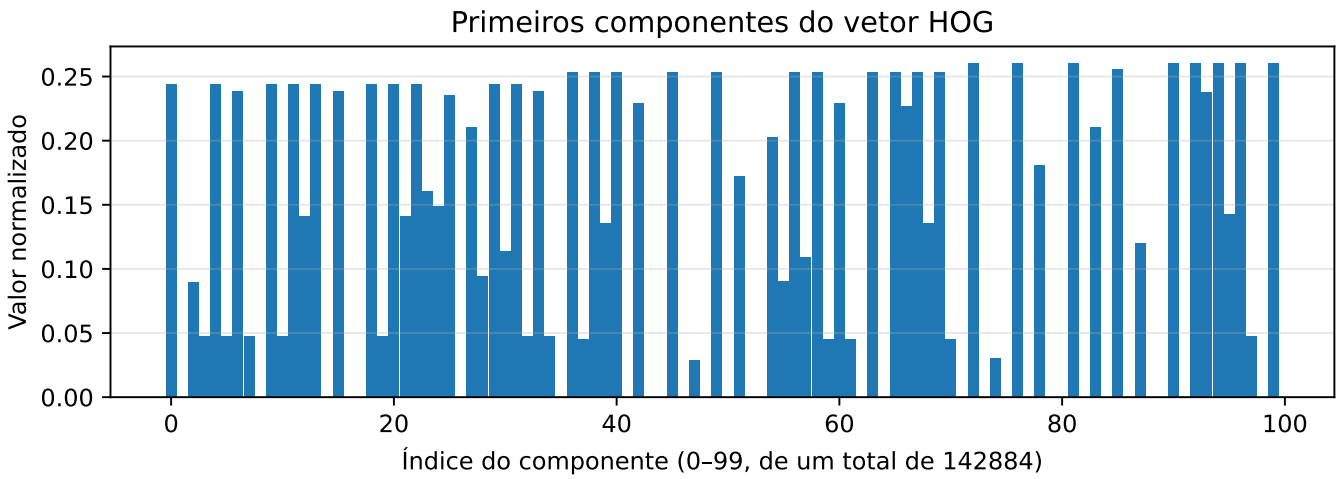


Figura 7.5: Primi 100 componenti del vettore di caratteristiche HOG.

Funzione `hog`

L'estrazione del descrittore HOG è realizzata dalla funzione:

```
hog(
    image,
    orientations=9,
    pixels_per_cell=(8, 8),
    cells_per_block=(2, 2),
    visualize=True,
)
```

I parametri principali sono:

- `image`: immagine di input;
- `orientations`: numero di suddivisioni angolari dell'istogramma delle orientazioni in ogni cella;
- `pixels_per_cell`: dimensione, in pixel, di ogni cella in cui viene calcolato l'istogramma;
- `cells_per_block`: numero di celle utilizzate per la normalizzazione del descrittore;
- `visualize`: quando `True`, restituisce anche un'immagine che illustra i gradienti utilizzati dall'HOG.

L'equazione presentata in precedenza descrive il calcolo della magnitudine e dell'orientazione del gradiente, che costituiscono la base del descrittore HOG. Nell'implementazione adottata in questo capitolo, la funzione `hog()` utilizza queste informazioni per costruire istogrammi delle orientazioni in ogni cella dell'immagine e, successivamente, esegue la normalizzazione a blocchi (`cells_per_block`), riducendo la sensibilità del descrittore alle variazioni di illuminazione e contrasto.

La Figura 7.3 presenta la rappresentazione visiva dell'HOG, mentre la Figura 7.5 illustra le prime componenti del vettore di caratteristiche estratto dall'immagine.

Il **Progetto Pratico 1** (sezione **Classificazione di Cifre Manoscritte con k-NN**) confronta le prestazioni dei descrittori HOG con l'uso diretto delle intensità dei pixel come vettore di caratteristiche.

7.6.3 L'Impatto della Scala e la Normalizzazione delle Caratteristiche

Il classificatore k-NN prende le sue decisioni basandosi sulla distanza tra i vettori di caratteristiche. Per questo motivo, la scala di ciascuna caratteristica influenza direttamente il risultato della classificazione. Se una variabile presenta valori molto maggiori rispetto alle altre (ad esempio, un' intensità di colore che varia da 0 a 255, mentre un indice di circolarità varia da 0 a 1), essa tende a dominare il calcolo della distanza, riducendo l'influenza degli altri descrittori.

Per evitare questo problema, si applica una fase di **normalizzazione delle caratteristiche**, generalmente tramite la standardizzazione (*Z-score standardization*). In questa procedura, ciascuna caratteristica assume media pari a zero e deviazione standard pari a uno, rendendo comparabili grandezze originariamente misurate su scale diverse.

La standardizzazione viene effettuata tramite la trasformazione

$$z = \frac{x - \mu}{\sigma},$$

in cui:

- x è il valore originale della caratteristica;
- μ è la media di tale caratteristica calcolata sull'insieme di addestramento;
- σ è la deviazione standard della caratteristica;
- z è il valore standardizzato.

Dopo questa trasformazione, tutte le caratteristiche possiedono media pari a zero e deviazione standard pari a uno, consentendo loro di contribuire in modo equilibrato al calcolo delle distanze.

Classe `StandardScaler`

La standardizzazione utilizzata in questo capitolo viene effettuata tramite la classe `StandardScaler`, della libreria `scikit-learn`:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_norm = scaler.fit_transform(X)
```

in cui:

- `StandardScaler()`: crea l'oggetto responsabile della standardizzazione;
- `fit_transform(X)`: calcola la media e la deviazione standard di ciascuna caratteristica dell'insieme `X` e restituisce la matrice standardizzata.

In pratica, il metodo `fit_transform()` esegue due fasi: prima (`fit`), stima la media (μ) e la deviazione standard (σ) di ogni caratteristica; successivamente (`transform`), applica la trasformazione di standardizzazione presentata in precedenza a tutti i valori della matrice di input.

La Figura 7.6 mostra l'effetto della normalizzazione. Visivamente, la distribuzione dei punti rimane la stessa; ciò che cambia è la scala degli assi. Senza normalizzazione, la caratteristica di maggiore magnitudine domina il calcolo delle distanze tra i campioni. Dopo la standardizzazione, tutte le caratteristiche contribuiscono in modo equilibrato al calcolo delle distanze utilizzate dal classificatore k-NN.

```
# Dati sintetici con scale molto diverse
np.random.seed(42)
X_demo = np.random.randn(20, 2) * [100, 1]
y_demo = np.array([0] * 10 + [1] * 10)
```

```python
print("Effetto della normalizzazione:")
print("  Caratteristica 1: scala   100")
print("  Caratteristica 2: scala   1")
print("\nSenza normalizzazione, la prima caratteristica domina il calcolo delle distanze.")
print("Con la normalizzazione, entrambe contribuiscono in modo equilibrato.")
print("\nLa normalizzazione è essenziale quando le caratteristiche hanno scale diverse.")
fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Senza normalizzazione
axes[0].scatter(
    X_demo[y_demo == 0, 0], X_demo[y_demo == 0, 1],
    c="blue", label="Classe 0"
)
axes[0].scatter(
    X_demo[y_demo == 1, 0], X_demo[y_demo == 1, 1],
    c="red", label="Classe 1"
)
axes[0].set_title("Sem Normalização\n(escalas diferentes)")
axes[0].set_xlabel("Característica 1 (escala 100)")
axes[0].set_ylabel("Característica 2 (escala 1)")
axes[0].legend()

# Con normalizzazione
scaler = StandardScaler()
X_norm = scaler.fit_transform(X_demo)

axes[1].scatter(
    X_norm[y_demo == 0, 0], X_norm[y_demo == 0, 1],
    c="blue", label="Classe 0"
)
axes[1].scatter(
    X_norm[y_demo == 1, 0], X_norm[y_demo == 1, 1],
    c="red", label="Classe 1"
)
axes[1].set_title("Com Normalização\n(características balanceadas)")
axes[1].set_xlabel("Característica 1")
axes[1].set_ylabel("Característica 2")
axes[1].legend()

plt.tight_layout()
plt.show()
```

```
Effetto della normalizzazione:
  Caratteristica 1: scala   100
  Caratteristica 2: scala   1

Senza normalizzazione, la prima caratteristica domina il calcolo delle distanze.
Con la normalizzazione, entrambe contribuiscono in modo equilibrato.

La normalizzazione è essenziale quando le caratteristiche hanno scale diverse.
```

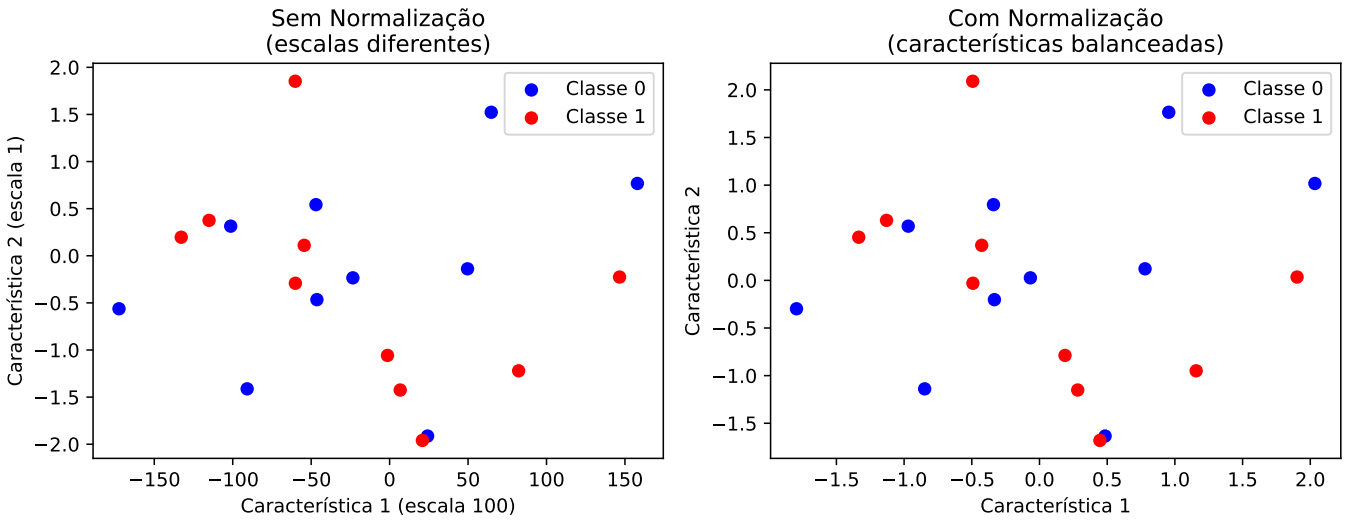


Figura 7.6: Importanza della normalizzazione delle caratteristiche per il classificatore k-NN.

7.7 📌 Mappa Concettuale

Fino a qui sono stati presentati i descrittori classici (LBP, HOG, pixel grezzi) e il modo in cui organizzano i campioni in uno spazio delle caratteristiche. La Figura 7.7 sintetizza questo percorso e colloca tali fasi all'interno del flusso generale di un sistema classico di classificazione delle immagini, indicando anche le fasi successive — classificazione (k-NN) e valutazione dei risultati — che verranno formalizzate nelle sezioni seguenti.

7.8 Descrittori nella Pratica

Dopo aver conosciuto i principali descrittori classici, è naturale chiedersi come essi influenzino le prestazioni di un classificatore in situazioni vicine a quelle riscontrabili nella pratica.

In questa sezione, si confronta l'uso di tre diverse rappresentazioni delle stesse immagini: intensità dei pixel, descrittori LBP e descrittori HOG. Per rendere l'esperimento più realistico, si aggiunge rumore sintetico ai dati, con intensità diverse per ciascun descrittore — una forma semplificata di simulare il fatto che, nella pratica, diverse rappresentazioni tollerano in modo diseguale le imperfezioni dell'acquisizione (rumore del sensore, piccole variazioni di posizione, ecc.).

La Figura 7.8 presenta le matrici di confusione ottenute per ciascun descrittore, consentendo di identificare in quali classi si verificano i principali errori di classificazione. L'interpretazione di queste matrici è stata introdotta nel **Capitolo 1**, quando sono stati presentati i concetti di **Vero Positivo (VP)**, **Falso Positivo (FP)**, **Vero Negativo (VN)** e **Falso Negativo (FN)**. Questi concetti sono stati approfonditi negli **EP 01_02** (metriche di classificazione) e **01_03** (*mean Average Precision* – mAP), disponibili su:

- https://fzampirolli.github.io/pdi-vc/eps/py.pt/EP01_02.html
- https://fzampirolli.github.io/pdi-vc/eps/py.pt/EP01_03.html

In questo capitolo, le matrici di confusione sono impiegate per analizzare come diversi descrittori influenzino le prestazioni del classificatore.

Successivamente, la Figura 7.9 riassume l'accuratezza globale ottenuta da ciascun descrittore.

I risultati mostrano che le prestazioni del classificatore dipendono direttamente dalla rappresentazione scelta per descrivere le immagini. Mentre l'uso diretto delle intensità dei pixel è più sensibile alle degradazioni introdotte, i descrittori LBP e HOG preservano meglio le informazioni rilevanti per la classificazione, risultando in maggiori prestazioni in questo scenario. È importante sottolineare che i livelli di rumore applicati a ciascun descrittore sono stati scelti solo a scopo didattico, in modo da illustrare il principio generale secondo

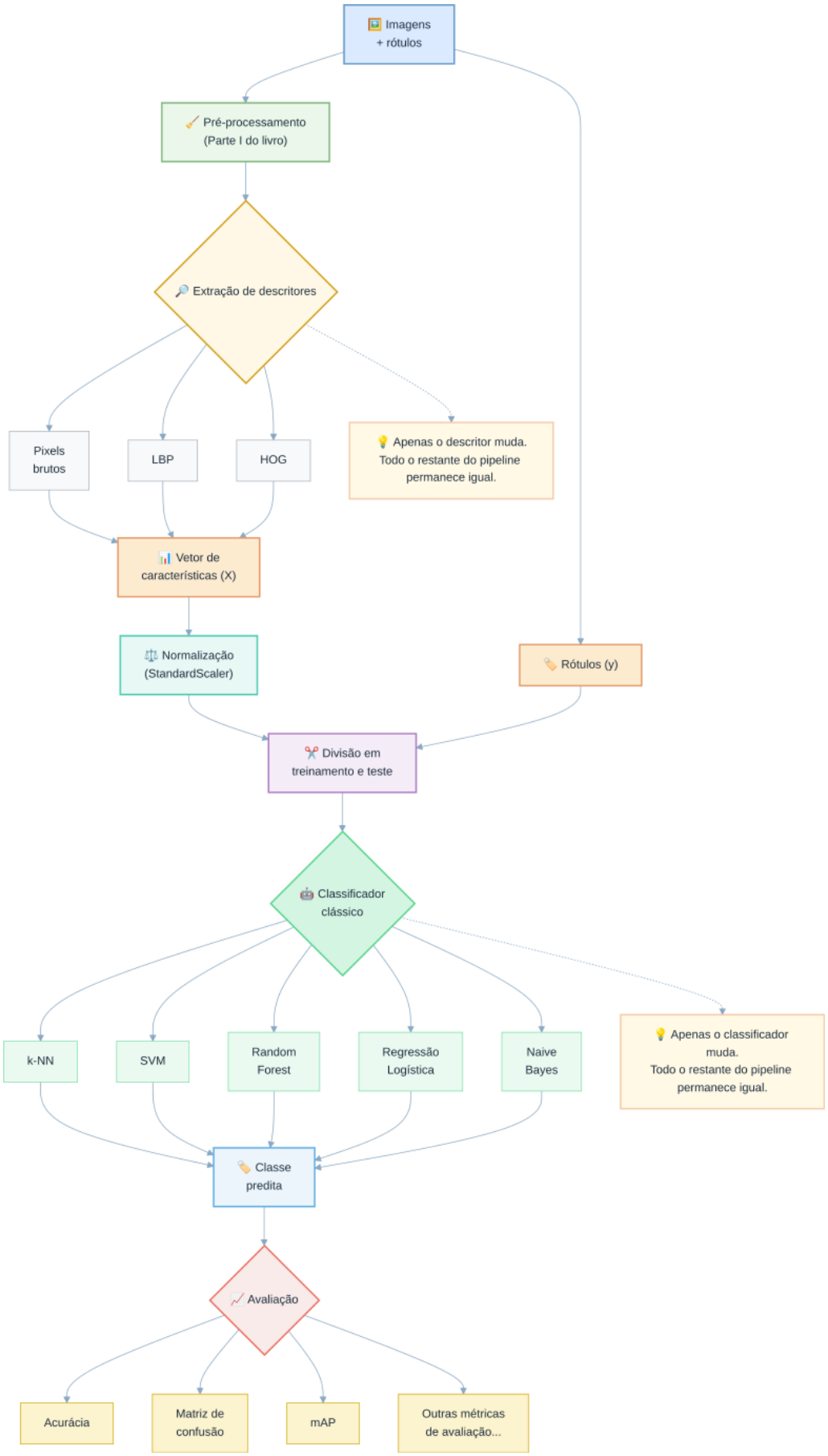


Figura 7.7: Mappa concettuale del processo di classificazione delle immagini utilizzando descrittori classici (LBP, HOG, pixel grezzi) e classificatore tradizionale (k-NN). Non si applica a modelli di Apprendimento Profondo (CNN, YOLO), che apprendono *feature end-to-end* direttamente dai pixel.

cui descrittori più elaborati *possono* essere più robusti alle degradazioni — il che non significa che questa relazione si verifichi sempre, come dimostrerà il caso di studio della prossima sezione.

Come viene eseguito l'esperimento

Poiché l'obiettivo di questa sezione è confrontare esclusivamente l'effetto dei descrittori, viene generato un insieme di dati sintetico semplice: tre nubi di punti gaussiani, centrati sugli stessi valori di "colore, texture e forma" già utilizzati nella Figura 7.1 — lo stesso schema impiegato fin dall'inizio del capitolo per rappresentare le tre classi di frutta.

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

```
centri = {
    "Mela":     [0.8, 0.2, 0.9],
    "Banana":   [0.3, 0.1, 0.2],
    "Arancia":  [0.9, 0.8, 0.8],
}

n_per_classe = 100
X = np.vstack([
    np.random.normal(centro, 0.12, (n_per_classe, 3))
    for centro in centri.values()
])
y = np.repeat(list(centri.keys()), n_per_classe)
```

dove:

- `centri`: dizionario con il punto medio di ciascuna classe nello spazio delle caratteristiche (colore, texture, forma);
- `n_per_classe`: numero di campioni generati per classe;
- `np.random.normal(centro, 0.12, (n_per_classe, 3))`: genera `n_per_classe` campioni attorno a ciascun centro, con deviazione standard 0,12 in ogni dimensione;
- `np.repeat(list(centri.keys()), n_per_classe)`: genera il vettore delle etichette corrispondenti, nello stesso ordine dei centri.

Successivamente, si utilizza il flusso di addestramento e valutazione:

- `train_test_split(X, y, test_size=0.3)`: divide i dati in addestramento (70%) e test (30%);
- `KNeighborsClassifier(n_neighbors=5)`: crea un classificatore k-NN con $k = 5$ vicini;
- `fit(X_train, y_train)`: adatta il modello ai dati di addestramento;
- `predict(X_test)`: classifica i campioni di test;
- `accuracy_score(y_test, y_pred)`: calcola l'accuratezza;
- `confusion_matrix(y_test, y_pred)`: genera la matrice di confusione.

In questo esperimento, il set di addestramento, il classificatore e il metodo di valutazione rimangono esattamente gli stessi. L'unica differenza tra gli esperimenti è la rappresentazione utilizzata per ciascuna immagine (pixel grezzi, LBP o HOG), consentendo di valutare esclusivamente l'influenza del descrittore sulle prestazioni del classificatore.

```
classes = ["Maçã", "Banana", "Laranja"]

np.random.seed(42)

# Stessi centri di classe (colore, consistenza, forma) utilizzati nel
# esempio precedente, ora riutilizzati per generare i dati
# sintetici di addestramento e test di questo esperimento.
centros = {
```

```
    "Maçã":    [0.8, 0.2, 0.9],
    "Banana":  [0.3, 0.1, 0.2],
    "Laranja": [0.9, 0.8, 0.8],
}

n_por_classe = 100
X = np.vstack([
    np.random.normal(centro, 0.12, (n_por_classe, 3))
    for centro in centros.values()
])
y = np.repeat(list(centros.keys()), n_por_classe)

# Simulare descrittori con diversi livelli di sensibilità al rumore.
# Maggiore è il rumore aggiunto, peggiore tende a essere la rappresentazione.
descritores = {
    "Pixels Brutos": X + 0.5 * np.random.randn(*X.shape),
    "LBP":           X + 0.3 * np.random.randn(*X.shape),
    "HOG":           X + 0.2 * np.random.randn(*X.shape),
}

# Creare un'unica figura con 3 sottotrame affiancate per le matrici
fig, axes = plt.subplots(1, 3, figsize=(14, 4))
resultados = {}

for idx, (nome, Xd) in enumerate(descritores.items()):
    X_train, X_test, y_train, y_test = train_test_split(Xd, y, test_size=0.3, random_state=42)

    knn = KNeighborsClassifier(n_neighbors=5)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    resultados[nome] = acc

    # Matrice di confusione nel sottotrama corrispondente

    cm = confusion_matrix(y_test, y_pred, labels=classes)

    sns.heatmap(
        cm,
        annot=True,
        fmt="d",
        cmap="Blues",
        cbar=False,
        xticklabels=classes,
        yticklabels=classes,
        ax=axes[idx],
    )

    axes[idx].set_title(
        f"{nome}\nAcurácia: {acc:.3f}",
        fontsize=11,
        fontweight="bold"
    )
    axes[idx].set_xlabel("Classe Predita")
    axes[idx].set_ylabel("Classe Real")


plt.tight_layout()
plt.show()
```

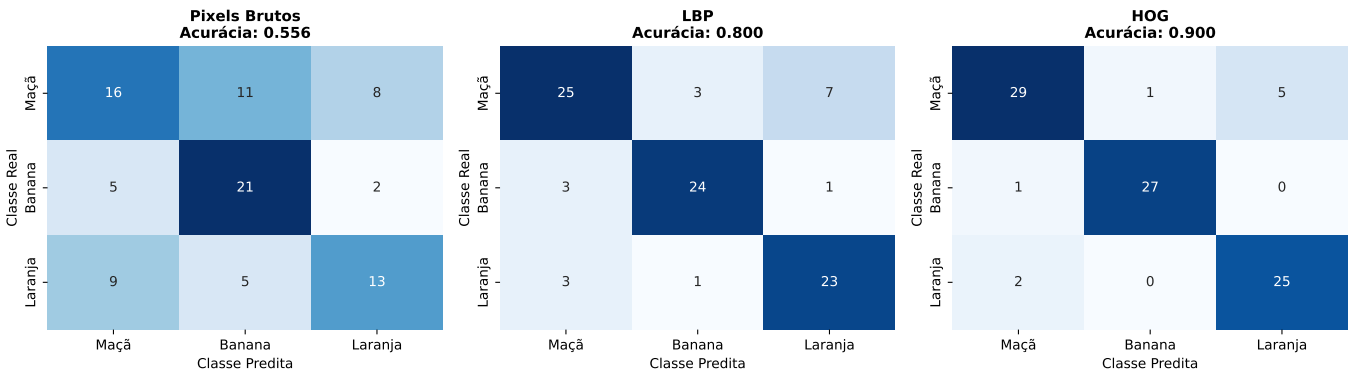


Figura 7.8: Matrici di confusione ottenute dal classificatore k-NN utilizzando tre descrittori diversi. Le righe rappresentano la classe reale (Mela, Banana e Arancia) e le colonne la classe predetta. Maggiore è la concentrazione di valori sulla diagonale principale, migliore è la prestazione del descrittore.

```
# Confronto visivo in una figura isolata
plt.figure(figsize=(6, 3.5))
nomes = list(resultados.keys())
acuracia = list(resultados.values())
colors = ['#6366f1', '#f97316', '#22c55e']

bars = plt.bar(nomes, acuracia, color=colors, width=0.5)
plt.ylabel('Acurácia Global')
plt.title('Desempenho Geral dos Descritores sob Ruído Realista', fontsize=12,
fontweight='bold')
plt.ylim(0.5, 1.0)
plt.grid(axis='y', linestyle='--', alpha=0.5)

# Aggiungere i valori sopra le barre usando il round standard per la visualizzazione
for bar, val in zip(bars, acuracia):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.01,
             f'{val:.3f}', ha='center', fontweight='bold', fontsize=10)

plt.tight_layout()
plt.show()

print("Analisi dei risultati:")
print("- Pixel Grezzi: Sensibili a variazioni locali di illuminazione e rumore.")
print("- LBP: Buona tolleranza per variazioni monotoniche dell'illuminazione globale.")
print("- HOG: Eccellente per contorni e forme stabili sotto piccole fluttuazioni
geometriche.")
```

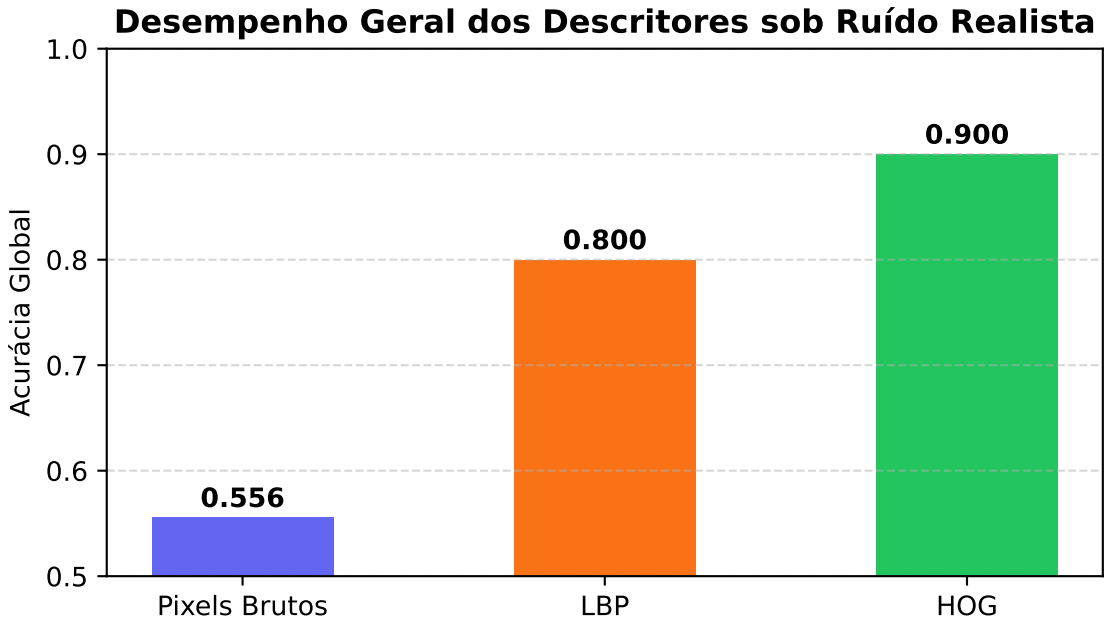


Figura 7.9: Confronto dettagliato dell'accuratezza globale in uno scenario realistico. Si noti come i descrittori estratti strutturalmente (LBP e HOG) superino l'uso delle intensità pure dei pixel grezzi.

```
Analisi dei risultati:
- Pixel Grezzi: Sensibili a variazioni locali di illuminazione e rumore.
- LBP: Buona tolleranza per variazioni monotoniche dell'illuminazione globale.
- HOG: Eccellente per contorni e forme stabili sotto piccole fluttuazioni geometriche.
```

7.9 Un Problema Concreto: Simulando Descrittori di Frutta

Riprendendo il problema di classificazione della frutta presentato all'inizio del capitolo, ogni immagine può essere rappresentata da un vettore di caratteristiche (*feature vector*) ottenuto tramite l'estrazione di descrittori di colore, texture e forma. La Tabella 7.1 presenta alcuni descrittori frequentemente utilizzati nelle applicazioni di Visione Computazionale, incluse tecniche introdotte nel Capitolo 3 e in questo capitolo.

Tabella 7.1: Insieme di descrittori cromatici, testurali e geometrici utilizzati per rappresentare immagini di frutta.

Caratteristica	Descrizione
R, G, B	intensità media dei canali rosso, verde e blu
NC	intensità media in livelli di grigio (*grayscale*)
LBP	descrittore di texture (*Local Binary Pattern*)
HOG	descrittore di forma (*Histogram of Oriented Gradients*)
Area	numero di pixel dell'oggetto
Perimetro	lunghezza del contorno
Circolarità	misura di quanto è circolare l'oggetto
Rapporto larghezza/altezza	proporzione tra larghezza e altezza della regione

In questo esempio, ogni immagine è rappresentata dal vettore

$$X = (R, G, B, NC, \text{LBP}, \text{HOG}, \text{Area}, \text{Perimetro}, \text{Circolarità}, \text{Rapporto}).$$

Semplificazione adottata in questa tabella

Nella pratica, LBP e HOG non sono valori scalari, ma istogrammi con decine o centinaia di componenti. In questa sezione, ciascuno di essi è rappresentato da un unico valore solo per semplificare la presentazione. Nelle applicazioni reali, queste posizioni sarebbero sostituite dalle componenti complete dei rispettivi istogrammi.

Nelle applicazioni reali, non tutti i descrittori contribuiscono in egual misura a distinguere le classi. Alcuni forniscono informazioni più rilevanti, mentre altri possono essere ridondanti o poco discriminativi.

Per riprodurre questo scenario in modo controllato, si utilizzerà `make_classification()`, dalla libreria `scikit-learn`. La funzione genera un insieme di dati sintetico le cui caratteristiche possono essere interpretate come descrittori di immagini, consentendo di definire quante di esse saranno informative per la classificazione.

In questo esempio, vengono generate dieci caratteristiche sintetiche, delle quali solo sette (`n_informative=7`) partecipano alla separazione tra le tre classi. Le restanti simulano attributi poco informativi o ridondanti. La Figura 7.10 presenta una rappresentazione concettuale di tale processo.

Prima di introdurre l'algoritmo che sarà studiato in dettaglio in questo capitolo, è opportuna un'osservazione: per identificare, tra le dieci caratteristiche sintetiche, quali siano più discriminative — e quindi selezionarne due per la visualizzazione 2D —, si utilizza un *Random Forest* solo come strumento ausiliario di diagnosi. Il KNN, focus di questo capitolo, viene presentato di seguito.

```
# Configurazione per la riproduzione
np.random.seed(42)

# Generazione di dati sintetici con caratteristiche controllate
X, y = make_classification(
    n_samples=300,
    n_features=10,
    n_informative=7,
    n_redundant=2,
    n_repeated=1,        # Una caratteristica è copia di un'altra
    n_classes=3,
    n_clusters_per_class=1,
    random_state=42,
)

# Creare figura con due sottotrame
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

# Sottotrama 1: Visualizzazione delle classi in 2D (usando due caratteristiche informative)
# Identificare quali caratteristiche sono più informative
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)
importancias = rf.feature_importances_
caracteristicas_informativas = np.argsort(importancias)[-2:]  # Le due più importanti

cores = {0: "#e74c3c", 1: "#f1c40f", 2: "#e67e22"}  # Mela, Banana, Arancia
rotulos = {0: "Maçã", 1: "Banana", 2: "Laranja"}

for classe in range(3):
    idx = y == classe
    ax1.scatter(X[idx, caracteristicas_informativas[0]],
                X[idx, caracteristicas_informativas[1]],
                c=cores[classe], label=rotulos[classe],
                alpha=0.6, s=50, edgecolors="white", linewidth=0.5)
```

```
ax1.set_xlabel(f"Característica {caracteristicas_informativas[0]+1} (informativa)",
fontsize=11)
ax1.set_ylabel(f"Característica {caracteristicas_informativas[1]+1} (informativa)",
fontsize=11)
ax1.set_title("Classes no Espaço de Características\n(2 características informativas)",
             fontsize=12, fontweight="bold")
ax1.legend(loc="upper right")
ax1.grid(alpha=0.3)

# Sottotrama 2: Importanza delle caratteristiche
bars = ax2.bar(range(1, 11), importancias, color="#4a90d9", alpha=0.7)
ax2.set_xlabel("Índice da Característica", fontsize=11)
ax2.set_ylabel("Importância", fontsize=11)
ax2.set_title("Importância de cada Característica\npara a Classificação",
             fontsize=12, fontweight="bold")
ax2.set_xticks(range(1, 11))
ax2.grid(axis="y", alpha=0.3)

# Colorare le barre per evidenziare le caratteristiche
cores_barras = ["#e74c3c" if i < 7 else "#95a5a6" for i in range(10)]
for bar, cor in zip(bars, cores_barras):
    bar.set_color(cor)

# Aggiungere legenda
from matplotlib.patches import Patch
legenda_elements = [
    Patch(facecolor="#e74c3c", label="Características Informativas (7)"),
    Patch(facecolor="#95a5a6", label="Características Redundantes (3)")
]
ax2.legend(handles=legenda_elements, loc="upper right")

# Annotare il numero di caratteristiche informative
ax2.axhline(y=0.15, color="red", linestyle="--", alpha=0.3)
ax2.text(0.5, 0.17, "Limiar de importância", fontsize=9, color="red", alpha=0.7)

plt.tight_layout()
plt.show()

print("\n  Analisi dei dati generati:")
print(f"  • Totale dei campioni: {X.shape[0]}")
print(f"  • Numero di caratteristiche: {X.shape[1]}")
print(f"  • Caratteristiche informative: 7 (colonne da 1 a 7 del grafico)")
print(f"  • Caratteristiche ridondanti: 2 (colonne 8 e 9)")
print(f"  • Caratteristiche ripetute: 1 (colonna 10)")
print(f"  • Distribuzione delle classi: {np.bincount(y)}")
```

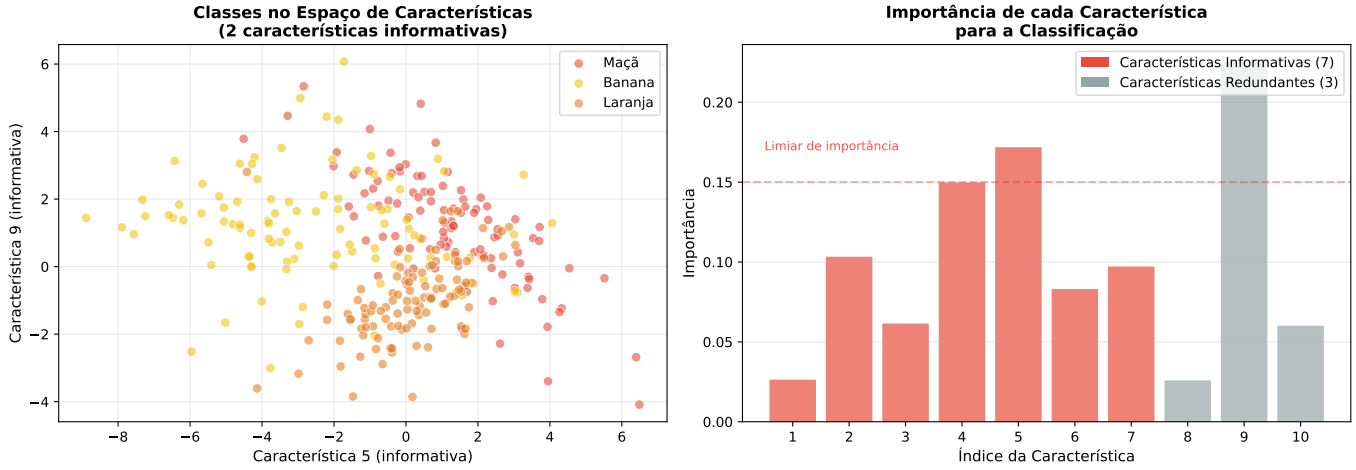


Figura 7.10: Illustrazione del processo di generazione di dati sintetici con *make_classification*. A sinistra, visualizzazione delle tre classi in uno spazio bidimensionale formato da due caratteristiche informative. A destra, importanza relativa di ciascuna caratteristica per la classificazione, evidenziando che solo 7 delle 10 caratteristiche sono effettivamente discriminative, mentre le altre sono ridondanti (2) o ripetute (1).

```
Analisi dei dati generati:
• Totale dei campioni: 300
• Numero di caratteristiche: 10
• Caratteristiche informative: 7 (colonne da 1 a 7 del grafico)
• Caratteristiche ridondanti: 2 (colonne 8 e 9)
• Caratteristiche ripetute: 1 (colonna 10)
• Distribuzione delle classi: [101  98 101]
```

7.10 Classificatore k-NN: Come Funziona Internamente

Il **k-*Nearest Neighbors* (k-NN)** è uno degli algoritmi di classificazione più semplici e intuitivi dell'Apprendimento Automatico. A differenza di molti classificatori, non costruisce esplicitamente un modello durante la fase di addestramento. Invece, memorizza i campioni etichettati e, quando un nuovo campione deve essere classificato, cerca quelli che più gli somigliano.

Il principio dell'algoritmo si basa sull'ipotesi che campioni con caratteristiche simili tendano ad appartenere alla stessa classe. Per quantificare questa vicinanza, il k-NN utilizza una misura di distanza tra i vettori di caratteristiche.

Come esempio, si consideri la Tabella 7.2, che presenta una versione semplificata del problema di classificazione della frutta utilizzando solo due caratteristiche: intensità del colore e circolarità, entrambe normalizzate nell'intervallo da 0 a 1.

Tabella 7.2: Esempio semplificato di classificazione della frutta utilizzando due caratteristiche normalizzate.

Campione	Colore	Circolarità	Classe
Frutta 1	0,82	0,88	Mela
Frutta 2	0,30	0,20	Banana
Frutta 3	0,88	0,85	Mela
Frutta ? (test)	0,80	0,90	?

Osservando solo queste due caratteristiche, si nota che la frutta di test è molto più vicina ai campioni etichettati come **Mela** che al campione etichettato come **Banana**. Nella sezione successiva, questa nozione intuitiva di

vicinanza sarà formalizzata tramite una metrica di distanza, utilizzata dall'algoritmo per identificare i vicini più prossimi e decidere la classe del nuovo campione.

7.10.1 Metrica di Distanza

La prossimità tra due campioni è normalmente quantificata dalla **distanza euclidea**, definita da

$$d(x, x_i) = \|x - x_i\|_2 = \sqrt{\sum_{j=1}^{n} (x_j - x_{i,j})^2},$$

dove:

- x è il campione di test;
- x_i è un campione del set di addestramento;
- n è il numero di caratteristiche;
- x_j e $x_{i,j}$ rappresentano la j-esima caratteristica.

Nell'implementazione di questo capitolo, x corrisponde a una riga di `X_test` e x_i a una riga di `X_train`. Il metodo `predict()` calcola automaticamente la distanza tra x e tutti i campioni di addestramento.

Nell'esempio della Tabella 7.2:

$$d(\text{teste}, \text{Frutta 1}) \approx 0{,}028, \qquad d(\text{teste}, \text{Frutta 2}) \approx 0{,}860, \qquad d(\text{teste}, \text{Frutta 3}) \approx 0{,}094.$$

Poiché le distanze minori corrispondono alla Frutta 1 e alla Frutta 3, questi campioni saranno utilizzati nella fase di decisione.

7.10.2 Regola di Decisione

Dopo aver ordinato le distanze, l'algoritmo seleziona i k vicini più prossimi. Sia $N_k(x)$ questo insieme. La classe prevista è data da

$$\hat{y} = \text{moda}\{\, y_i : x_i \in N_k(x) \,\},$$

dove y_i è l'etichetta del campione x_i e $\hat{y}$ è la classe assegnata al campione di test.

Nell'esempio, per $k = 3$, i vicini sono Frutto 1 (Mela), Frutto 3 (Mela) e Frutto 2 (Banana). Poiché **Mela** riceve due voti, questa è la classe prevista.

Classe `KNeighborsClassifier`

In questo capitolo, l'algoritmo viene implementato con la classe `KNeighborsClassifier`, della libreria `scikit-learn`:

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

dove:

- `KNeighborsClassifier(n_neighbors=3)`: definisce il valore di k;
- `fit(X_train, y_train)`: memorizza i campioni di addestramento (`X_train`) e le loro etichette (`y_train`);
- `predict(X_test)`: restituisce le classi previste per i campioni di `X_test`.

Internamente, `predict()` esegue le fasi descritte in precedenza: calcola le distanze, identifica i k vicini più prossimi e determina la classe per votazione a maggioranza.

La Figura 7.11 illustra questa procedura in un insieme bidimensionale. La figura evidenzia i vicini utilizzati nella classificazione, mentre la console presenta le fasi dell'algoritmo: calcolo delle distanze, ordinamento, selezione dei vicini, votazione e previsione della classe.

```
def knn_passo_a_passo(X, y, x, k=3):
    """Esegue le cinque fasi dell'algoritmo k-NN.
    Parametri: X (addestramento), y (etichette), x (test) e k (numero di vicini).
    """
    # 1. Distanze
    dist = [(np.linalg.norm(x-xi), yi, i) for i, (xi, yi) in enumerate(zip(X, y))]
    print(f"1. Distanze calcolate: {len(dist)}")

    # 2. Ordinamento
    dist.sort(key=lambda t: t[0])
    print("2. Distanze ordinate")

    # 3. Selezione
    vizinhos = dist[:k]
    print(f"3. {k} vicini più prossimi:")
    for d, c, _ in vizinhos:
        print(f"   {d:.4f} → {c}")

    # 4. Votazione
    votos = {}
    for _, c, _ in vizinhos:
        votos[c] = votos.get(c, 0) + 1
    print("4. Voti:", votos)

    # 5. Decisione
    classe = max(votos, key=votos.get)
    print("5. Classe prevista:", classe)

    return classe, vizinhos


# Dati di esempio
np.random.seed(4)
X = np.r_[np.random.randn(15,2)+[2,2],
          np.random.randn(15,2)+[-2,-2]]
y = np.array(["Classe A"]*15 + ["Classe B"]*15)
x = np.array([0.5,0.5])

classe, vizinhos = knn_passo_a_passo(X, y, x)

# Visualizzazione
plt.figure(figsize=(5,5))

for c, rotulo in [("Classe A","Classe A"), ("Classe B","Classe B")]:
    P = X[y==c]
    plt.scatter(P[:,0], P[:,1], s=80, label=rotulo)

plt.scatter(*x, marker="*", s=220, edgecolors="black", label="Teste")

for _, _, i in vizinhos:
    plt.scatter(*X[i], s=220, facecolors="none", edgecolors="black", linewidths=2)
    plt.plot([x[0], X[i,0]], [x[1], X[i,1]], "--", lw=1)
```

```
plt.xlabel("Característica 1")
plt.ylabel("Característica 2")
plt.title(f"k-NN ($k=3$): classe predita = {classe}")
plt.legend()
plt.grid(alpha=.3)
plt.axis("equal")
plt.tight_layout()
plt.show()
```

```
1. Distanze calcolate: 30
2. Distanze ordinate
3. 3 vicini più prossimi:
   1.0850 → Classe A
   1.6379 → Classe B
   1.6382 → Classe A
4. Voti: {np.str_('Classe A'): 2, np.str_('Classe B'): 1}
5. Classe prevista: Classe A
```

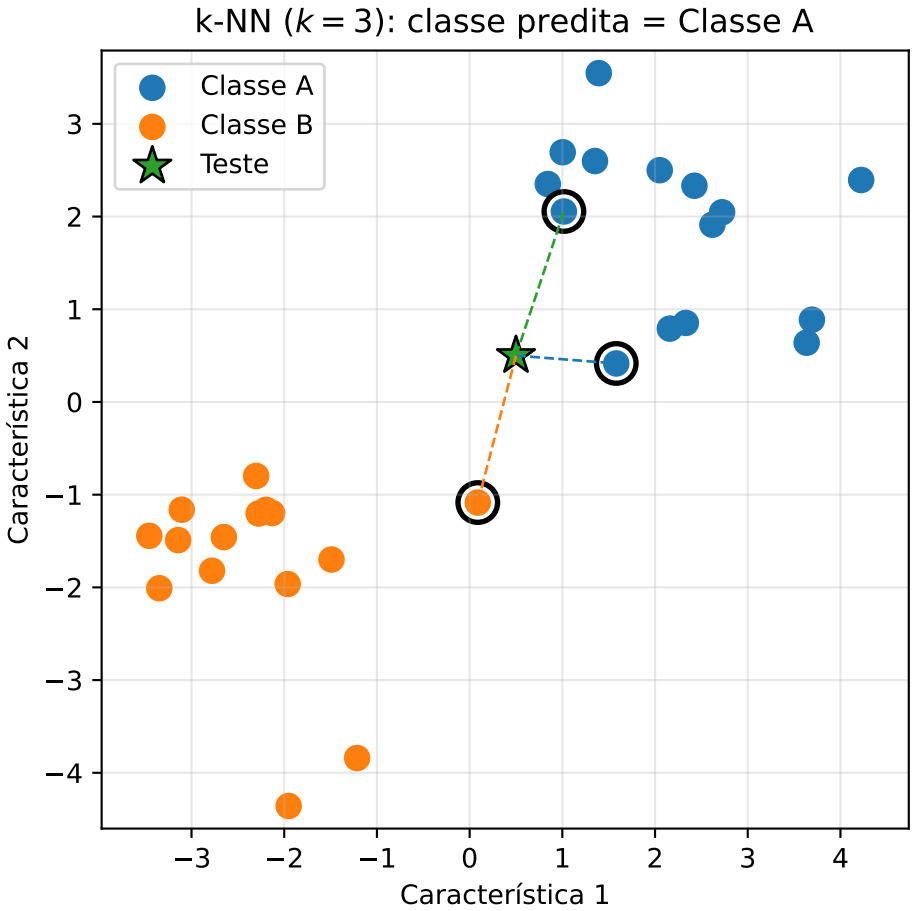


Figura 7.11: Classificazione di un nuovo campione tramite l'algoritmo k-NN. Il punto di test (stella) viene classificato in base ai tre vicini più prossimi, evidenziati da cerchi.

7.10.3 Il Ruolo del Parametro k

Il parametro k determina quanti vicini partecipano alla decisione di classificazione.

- **Valori piccoli di** k (ad esempio, $k = 1$) rendono il classificatore più sensibile a rumori e variazioni locali, producendo frontiere di decisione più irregolari e favorendo l'*overfitting*.
- **Valori maggiori di** k producono frontiere di decisione più morbide, ma possono ridurre la sensibilità a strutture locali, favorendo l'*underfitting*.

In problemi con due classi, è comune utilizzare valori dispari di k per ridurre l'occorrenza di pareggi.

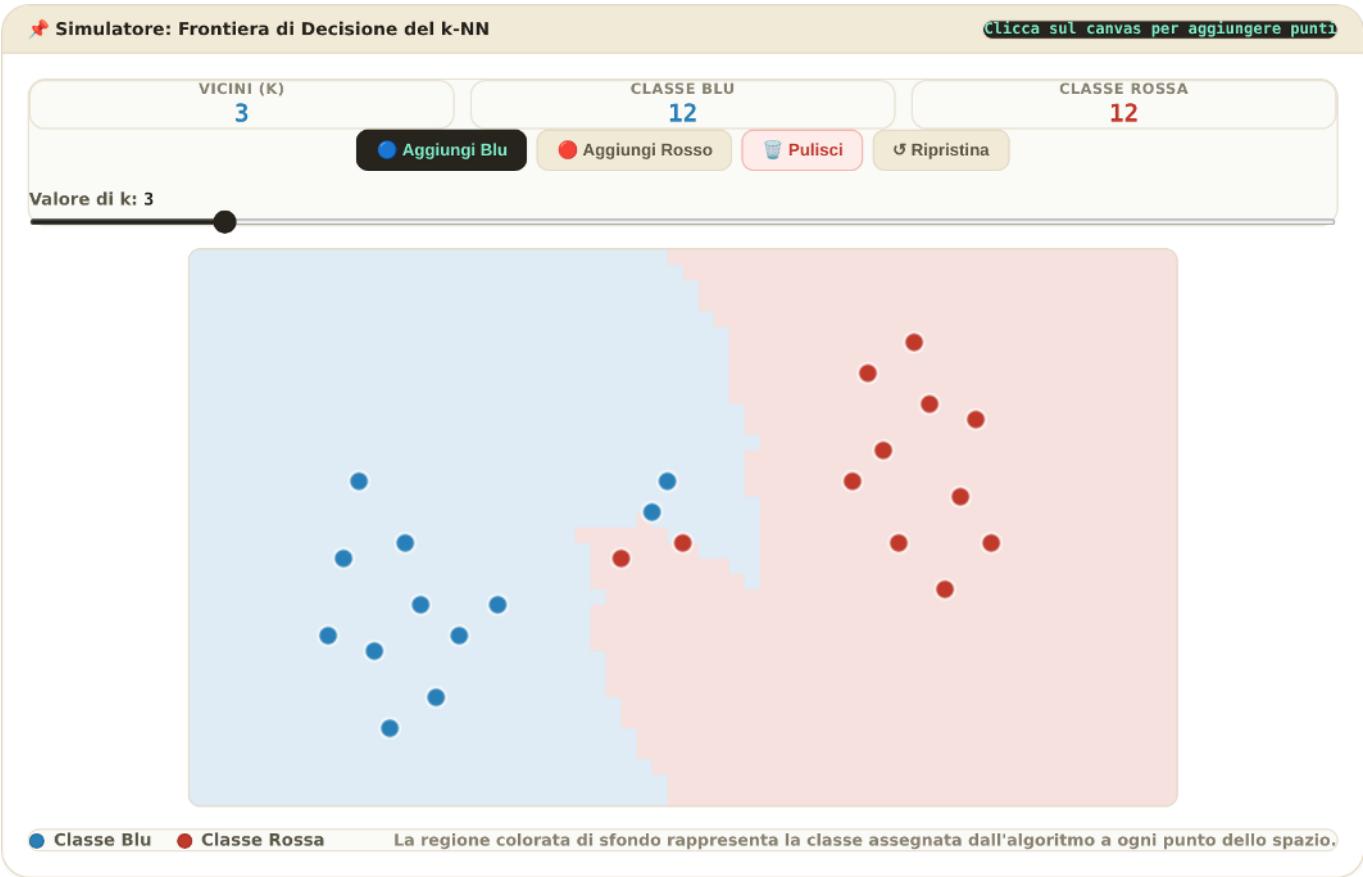
Un altro aspetto importante è la **maledizione della dimensionalità** (*curse of dimensionality*). Man mano che il numero di caratteristiche aumenta, le distanze tra i campioni tendono a diventare più simili, rendendo difficile l'identificazione di vicini realmente rappresentativi.

Riepilogo

L'algoritmo k-NN può essere riassunto in tre fasi:

1. estrarre il vettore di caratteristiche del nuovo campione;
2. identificare i k vicini più prossimi;
3. classificare il campione tramite la classe più frequente tra questi vicini.

Il simulatore della Figura 7.12 consente di esplorare visivamente l'effetto del parametro k sulla frontiera di decisione.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap07/sim-07-knn.html

Figura 7.12: Simulatore interattivo della frontiera di decisione del k-NN: aggiungi punti di addestramento e regola il valore di k per osservare l'effetto sulla regione di decisione.

7.11 Progetto Pratico 1: Classificazione di Cifre Scritte a Mano con k-NN

Le sezioni precedenti hanno presentato l'algoritmo k-NN attraverso un esempio semplificato di classificazione della frutta, utilizzando solo due caratteristiche. Di seguito, lo stesso algoritmo viene applicato a un insieme di dati di immagini, in cui ciascun campione è rappresentato da un vettore di dimensione maggiore.

Come caso di studio, si utilizza la base pubblica `load_digits`, messa a disposizione dalla libreria `scikit-learn`. Questo insieme di dati contiene 1797 immagini di cifre scritte a mano appartenenti alle classi da 0 a 9, ciascuna con una risoluzione di 8×8 pixel in scala di grigi. Ogni immagine è rappresentata da un vettore con 64 caratteristiche, corrispondenti alle intensità dei pixel, e ogni vettore possiede un'etichetta che indica la cifra corrispondente.

La base `load_digits` è messa a disposizione dalla libreria `scikit-learn` e viene utilizzata in questo capitolo per illustrare l'applicazione dell'algoritmo k-NN. Oltre a essere disponibile direttamente in `scikit-learn`, essa non richiede fasi aggiuntive di acquisizione e preparazione dei dati, consentendo di concentrare l'attenzione sull'implementazione e sulla valutazione del classificatore.

La Figura 7.13 presenta un campione delle immagini della base di dati.

```
digits = load_digits()
print(f"Totale dei campioni: {digits.data.shape[0]}, dimensione del vettore:
{digits.data.shape[1]}")
print(f"Classi: {[int(i) for i in sorted(set(digits.target))]}")

n_amostras = 16
imgs = list(digits.images[:n_amostras])
imgs_titles = [str(label) for label in digits.target[:n_amostras]]
mm.show(imgs, titles=imgs_titles, cols=8, figsize=(12, 4))
```

```
Totale dei campioni: 1797, dimensione del vettore: 64
Classi: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

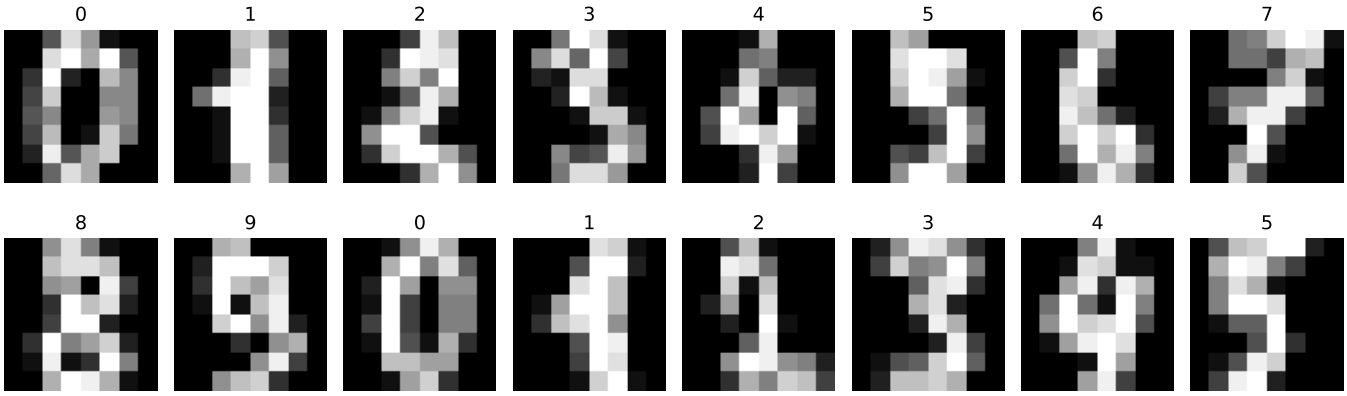


Figura 7.13: Campione di cifre scritte a mano dalla base *load_digits*, utilizzato come caso di studio di classificazione.

7.11.1 Classificazione con Vettori di Intensità

In questo primo esperimento, ogni immagine di dimensione 8×8 è rappresentata direttamente dalle intensità dei suoi 64 pixel, senza l'estrazione di descrittori aggiuntivi. Pertanto, ogni campione corrisponde a un vettore di 64 caratteristiche, utilizzato come input del classificatore k-NN.

Successivamente, il set di dati viene suddiviso in sottoinsiemi di training e test, preservando la proporzione delle dieci classi tramite il parametro `stratify=y`. Il classificatore viene addestrato con $k = 3$ e valutato sul set di test utilizzando l'accuratezza e la matrice di confusione presentata nella Figura 7.14..

```
X, y = digits.data, digits.target

X_treino, X_teste, y_treino, y_teste = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

n_neighbors = 3
knn_pixels = KNeighborsClassifier(n_neighbors=n_neighbors)
knn_pixels.fit(X_treino, y_treino)
pred_pixels = knn_pixels.predict(X_teste)

acc_pixels = accuracy_score(y_teste, pred_pixels)
```

```
print(f"Accuratezza (vettori di intensità, k={n_neighbors}): {acc_pixels:.4f}")

cm = confusion_matrix(y_teste, pred_pixels)

plt.figure(figsize=(5,4))
plt.imshow(cm, cmap="Blues")

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(
            j, i, cm[i, j],
            ha="center", va="center",
            color="white" if cm[i, j] > cm.max()/2 else "black",
            fontsize=9
        )

plt.title("Matriz de Confusão - Pixels Brutos")
plt.xlabel("Classe Predita")
plt.ylabel("Classe Real")
plt.xticks(range(10))
plt.yticks(range(10))
plt.colorbar(fraction=0.046)
plt.tight_layout()
plt.show()
```

```
Accuratezza (vettori di intensità, k=3): 0.9870
```

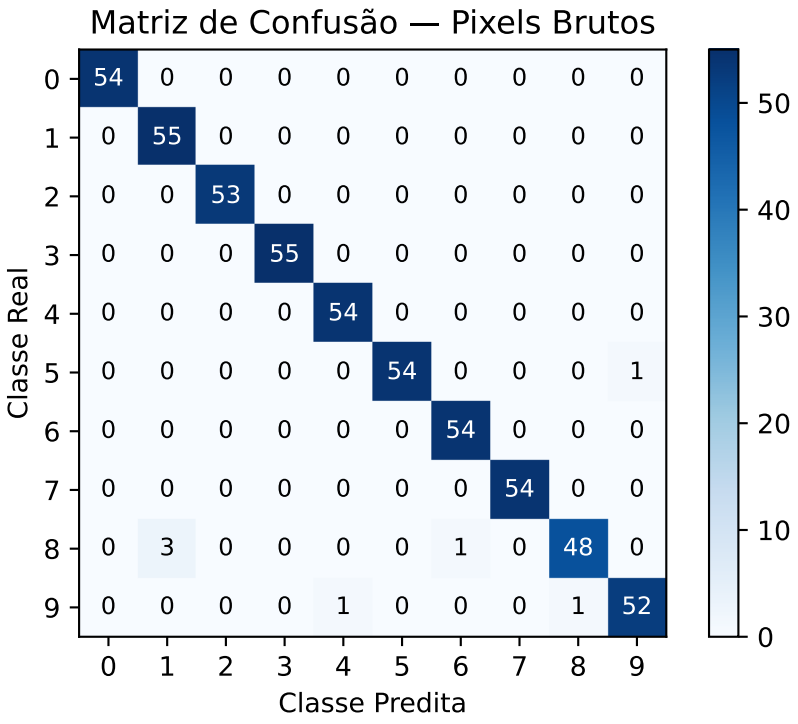


Figura 7.14: Matrice di confusione del classificatore k-NN addestrato con vettori di intensità grezzi (pixel).

7.11.2 Classificazione con descrittori HOG

Nell'esperimento precedente, ogni immagine è stata rappresentata direttamente dalle intensità dei suoi pixel. In questa sezione, tale rappresentazione viene sostituita da descrittori HOG (*Histogram of Oriented Gradients*), che codificano informazioni sulla distribuzione delle orientazioni dei gradienti dell'immagine.

Si mantengono la stessa suddivisione dei dati, lo stesso classificatore k-NN e lo stesso protocollo di valutazione, modificando solo la rappresentazione delle immagini. La Figura 7.15 confronta i risultati ottenuti con vettori di intensità e con descrittori HOG.

```
descritores_hog = np.array([
    hog(img, orientations=8, pixels_per_cell=(4, 4), cells_per_block=(1, 1))
    for img in digits.images
])
print(f"Dimensione del vettore HOG: {descritores_hog.shape[1]}")

Xh_treino, Xh_teste, yh_treino, yh_teste = train_test_split(
    descritores_hog, y, test_size=0.3, random_state=42, stratify=y
)

n_neighbors = 3
knn_hog = KNeighborsClassifier(n_neighbors=n_neighbors)
knn_hog.fit(Xh_treino, yh_treino)
pred_hog = knn_hog.predict(Xh_teste)
acc_hog = accuracy_score(yh_teste, pred_hog)
print(f"Accuratezza (descrittore HOG, k={n_neighbors}): {acc_hog:.4f}")

plt.figure(figsize=(4, 3))
plt.bar(["Pixels brutos", "HOG"], [acc_pixels, acc_hog], color=["#6366f1", "#f97316"])
plt.ylim(0, 1.1)  # Aumenta il limite superiore per dare spazio
plt.ylabel("Acurácia")
plt.title("Comparação de Descritores")
for i, v in enumerate([acc_pixels, acc_hog]):
    plt.text(i, v + 0.02, f"{v:.3f}", ha="center")  # Aumenta lo spostamento verticale
plt.tight_layout()
```

```
Dimensione del vettore HOG: 32
Accuratezza (descrittore HOG, k=3): 0.7593
```

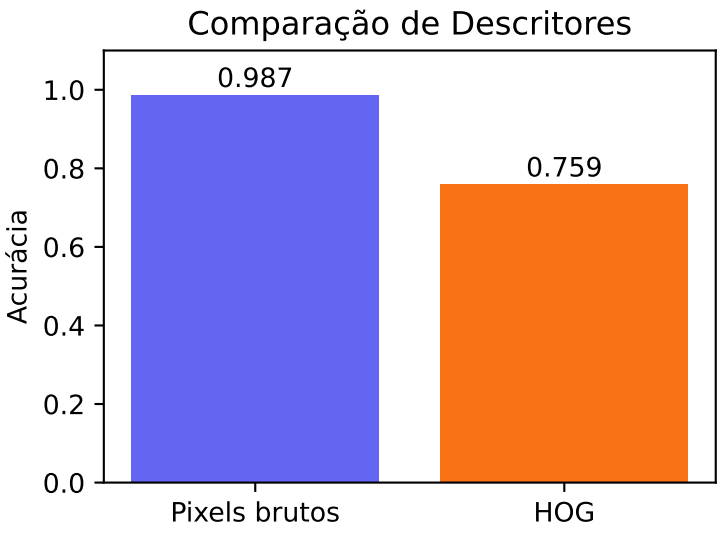


Figura 7.15: Confronto di accuratezza tra descrittori di pixel grezzi e HOG per il classificatore k-NN (k=3) sulla base di cifre.

Perché accade questo?

Nella Figura 7.15, il classificatore addestrato con vettori di **intensità dei pixel** raggiunge una maggiore accuratezza (0.987) rispetto a quello basato su descrittori **HOG** (0.759). Questo risultato è legato alle caratteristiche della base `load_digits`.
Le immagini hanno una risoluzione di soli 8×8 pixel, sono approssimativamente centrate e presentano poca variazione di illuminazione, scala e orientamento. In questo scenario, le intensità dei pixel preservano praticamente tutta l'informazione necessaria per distinguere le classi. Al contrario, l'HOG riassume l'immagine in istogrammi di orientamenti dei gradienti, riducendo parte del dettaglio spaziale disponibile nei pixel originali.
Questa riduzione delle informazioni può rendere difficile la separazione di cifre visivamente simili, come 3 e 8 oppure 4 e 9, specialmente quando la risoluzione dell'immagine è bassa.
In problemi con immagini a più alta risoluzione o soggette a variazioni di illuminazione, posizione, scala o piccole deformazioni, descrittori come l'HOG tendono a rappresentare meglio la struttura locale dell'immagine rispetto ai valori individuali dei pixel. Pertanto, questo esperimento illustra un principio importante dell'Apprendimento Automatico: **la rappresentazione dei dati deve essere scelta in base alle caratteristiche del problema e non alla complessità del descrittore.**

7.12 Valutazione dei Classificatori

Nelle sezioni precedenti, la qualità del classificatore è stata analizzata mediante l'accuratezza e la matrice di confusione. In questa sezione, questi strumenti vengono integrati con metriche utilizzate nella valutazione dei modelli e con una procedura per selezionare il valore del parametro k.

L'accuratezza corrisponde alla proporzione di campioni classificati correttamente. Sebbene sia una misura semplice e ampiamente utilizzata, essa può risultare insufficiente quando le classi presentano distribuzioni molto sbilanciate.

A partire dalla matrice di confusione — introdotta nel **Capitolo 1** e utilizzata nel corso di questo capitolo — possono essere calcolate metriche per classe, come precisione e richiamo:

$$\text{Precisione} = \frac{VP}{VP + FP}, \qquad \text{Richiamo} = \frac{VP}{VP + FN},$$

dove VP, FP e FN rappresentano, rispettivamente, il numero di veri positivi, falsi positivi e falsi negativi della classe analizzata. La precisione quantifica la proporzione di predizioni positive corrette, mentre il richiamo misura la capacità del classificatore di identificare gli esempi appartenenti alla classe.

7.12.1 Scelta di k tramite Validazione Incrociata

Negli esperimenti precedenti, si è adottato $k = 3$ per illustrare il funzionamento dell'algoritmo. Tuttavia, questo parametro influenza direttamente le prestazioni del classificatore e, nella pratica, deve essere selezionato a partire dai dati.

Un approccio ampiamente utilizzato è la **validazione incrociata** (*cross-validation*), in cui il set di addestramento viene suddiviso in partizioni successive per stimare le prestazioni del modello su dati non utilizzati durante l'addestramento.

Il codice seguente calcola l'accuratezza media ottenuta tramite validazione incrociata di cinque partizioni (*5-fold cross-validation*) per diversi valori di k. La Figura 7.16 presenta i risultati, consentendo di identificare la regione in cui il classificatore raggiunge le migliori prestazioni.

```
valores_k = range(1, 16)
acuracias_medias = []
```

```
for k in valores_k:
    modelo = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(modelo, X, y, cv=5)
    acuracias_medias.append(scores.mean())

melhor_k = list(valores_k)[int(np.argmax(acuracias_medias))]
print(f"Miglior valore di k trovato: {melhor_k} (precisione
media={max(acuracias_medias):.4f})")

plt.figure(figsize=(6, 4))
plt.plot(list(valores_k), acuracias_medias, marker="o", color="#4f46e5")
plt.axvline(melhor_k, color="#f97316", linestyle="--", label=f"melhor k = {melhor_k}")
plt.xlabel("k")
plt.ylabel("Acurácia média (validação cruzada)")
plt.title("Seleção de k por Validação Cruzada")
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
```

```
Miglior valore di k trovato: 2 (precisione media=0.9672)
```

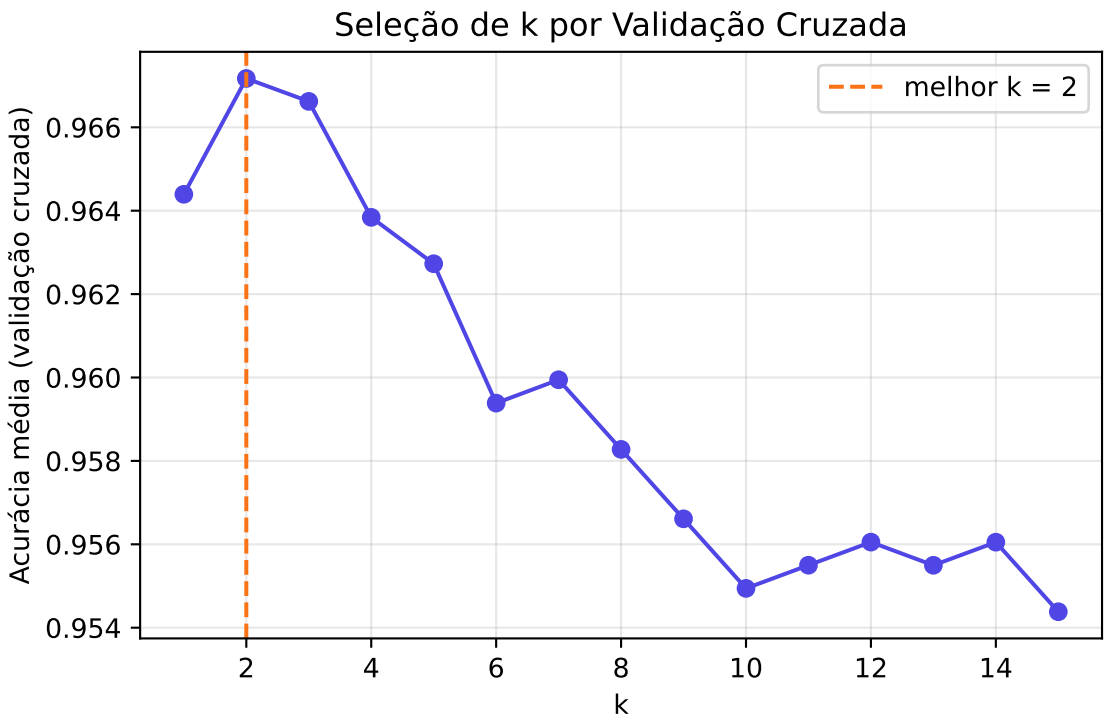


Figura 7.16: Precisione media per cross-validazione (5 pieghe) in funzione del parametro k, per il dataset di cifre con vettori di intensità grezzi.

7.12.2 Il Compromesso tra Bias e Varianza: Diagnosi di *Overfitting* e *Underfitting*

L'iperparametro k influenza la complessità del confine di decisione del classificatore k-NN e, di conseguenza, la sua capacità di generalizzazione. In termini generali, valori piccoli di k rendono il modello più sensibile ai campioni di addestramento, mentre valori più grandi producono confini di decisione più morbidi.

Questi comportamenti sono associati al compromesso tra **bias** (*bias*) e **varianza** (*variance*). Valori molto piccoli di k tendono ad aumentare il rischio di **sovradattamento** (*overfitting*), soprattutto in insiemi di dati rumorosi, mentre valori molto grandi possono portare al **sottodattamento** (*underfitting*), riducendo la capacità del modello di catturare le strutture locali dei dati.

Mentre la Figura 7.16 ha presentato solo l'accuratezza media ottenuta tramite validazione incrociata, la Figura 7.17 confronta le accuratezze di addestramento e di test per diversi valori di k. Le regioni evidenziate nel grafico rappresentano il comportamento atteso dell'algoritmo: maggiore rischio di sovradattamento per valori piccoli di k, una regione intermedia che spesso produce un buon equilibrio tra bias e varianza, e maggiore rischio di sottodattamento per valori elevati di k.

Tuttavia, queste regioni devono essere interpretate solo come un riferimento concettuale. Il comportamento osservato dipende dalle caratteristiche dell'insieme di dati. Nella base `load_digits`, ad esempio, le immagini presentano poca variabilità e una buona separazione tra le classi, così che valori piccoli di k possono mostrare prestazioni simili — o addirittura superiori — agli altri, senza evidenziare un sovradattamento significativo.

```
k_values = range(1, 16)
train_acc, test_acc = [], []

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k).fit(X_treino, y_treino)
    train_acc.append(accuracy_score(y_treino, knn.predict(X_treino)))
    test_acc.append(accuracy_score(y_teste, knn.predict(X_teste)))

plt.figure(figsize=(9,5))
plt.plot(k_values, train_acc, "o-", lw=2, label="Treinamento")
plt.plot(k_values, test_acc,  "s-", lw=2, label="Teste")

plt.axvspan(1, 3,  color="#fca5a5", alpha=.25, label="Maior risco de overfitting")
plt.axvspan(3,11,  color="#86efac", alpha=.25, label="Compromisso entre viés e variância")
plt.axvspan(11,15, color="#93c5fd", alpha=.25, label="Maior risco de underfitting")

plt.xlabel("Número de vizinhos ($k$)")
plt.ylabel("Acurácia")
plt.xticks(k_values)
plt.grid(alpha=.3)
plt.legend(loc="upper right")
plt.tight_layout()
plt.show()

maior_acc = max(test_acc)
melhores_k = [k for k, a in zip(k_values, test_acc) if np.isclose(a, maior_acc)]

print("Interpretazione")
print("- Valori piccoli di k: maggiore rischio di overfitting.")
print("- Valori intermedi: migliore compromesso tra bias e varianza.")
print("- Valori grandi di k: maggiore rischio di underfitting.")
print("\nLe regioni colorate rappresentano tendenze generali;")
print("il comportamento osservato dipende dal set di dati.")
print(f"\nMaggiore precisione sul test: {maior_acc:.3f}")
print(f"Valori di k che hanno raggiunto questa precisione: {melhores_k}")
```

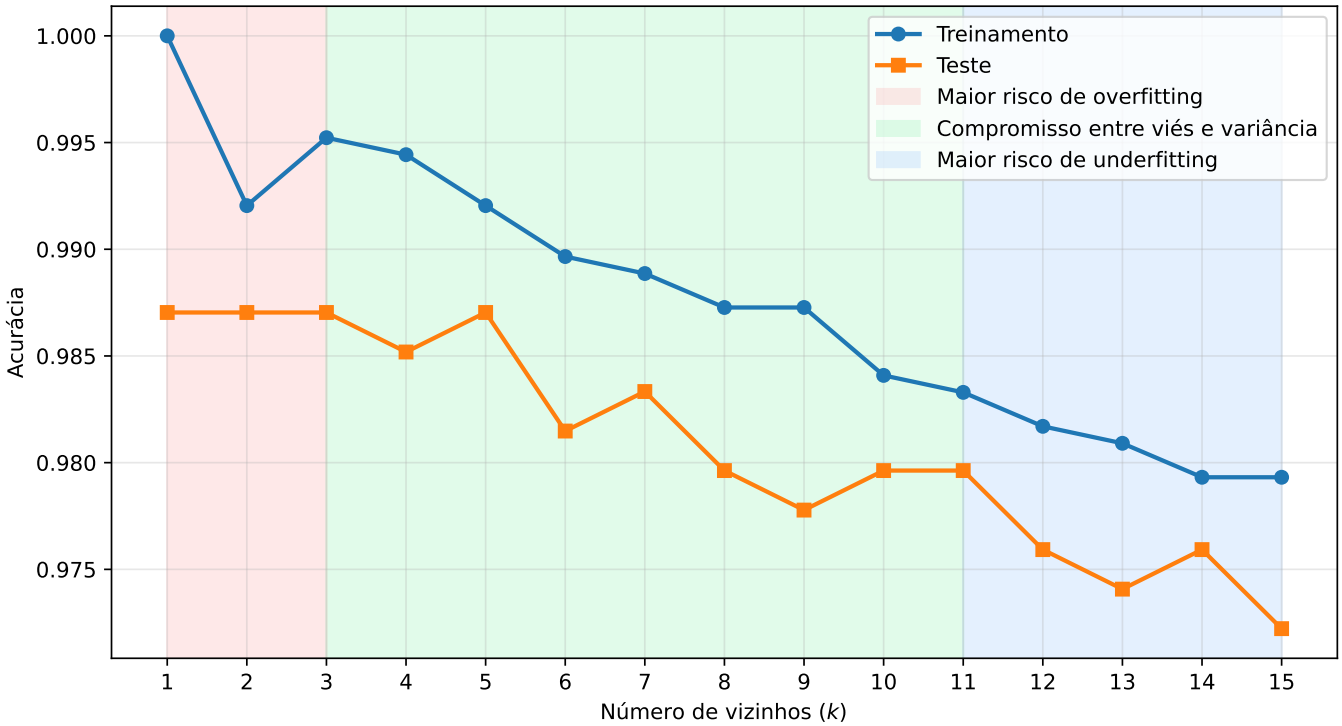


Figura 7.17: Precisione sui set di training e test per diversi valori di k. Le regioni colorate rappresentano, in modo concettuale, tendenze di comportamento del classificatore: maggiore rischio di overfitting (rosso), compromesso tra bias e varianza (verde) e maggiore rischio di underfitting (blu).

```
Interpretazione
- Valori piccoli di k: maggiore rischio di overfitting.
- Valori intermedi: migliore compromesso tra bias e varianza.
- Valori grandi di k: maggiore rischio di underfitting.

Le regioni colorate rappresentano tendenze generali;
il comportamento osservato dipende dal set di dati.

Maggiore precisione sul test: 0.987
Valori di k che hanno raggiunto questa precisione: [1, 2, 3, 5]
```

7.13 Progetto Pratico 2: Confronto di Descrittori per la Classificazione di Trame

Nel **Capitolo 6**, la varianza locale è stata utilizzata come descrittore di trama per **rilevare** anomalie su superfici industriali, distinguendo campioni **conformi** e **difettosi**. In questo progetto, il problema viene riformulato come un compito di **classificazione multiclasse**, in cui diverse rappresentazioni dell'immagine vengono utilizzate come input per un classificatore.

Verranno considerati tre tipi di descrittori: le **intensità dei pixel**, il **Local Binary Patterns (LBP)** e l'**Histogram of Oriented Gradients (HOG)**. Per ogni rappresentazione, verrà estratto un vettore di caratteristiche che servirà da input per l'algoritmo dei k vicini più prossimi (k-NN). Alla fine, verranno confrontate le accuratezze ottenute da ciascun descrittore nelle condizioni definite per questo esperimento.

La valutazione sarà effettuata mediante **validazione incrociata stratificata a cinque partizioni (*5-fold stratified cross-validation*)**. In questa procedura, il set di dati viene suddiviso in cinque sottoinsiemi preservando la proporzione tra le classi. In ogni iterazione, una partizione viene utilizzata per il test e le quattro rimanenti per l'addestramento, ripetendo il processo fino a quando tutte le partizioni non siano state utilizzate come set di test. Al termine delle cinque esecuzioni, vengono calcolate l'accuratezza media e la

deviazione standard per ciascun descrittore.

Il set di dati è composto da tre classi di trame sintetiche: **granulare**, ottenuta da rumore gaussiano smussato; **a strisce**, formata da pattern sinusoidali periodici; e **macchiata**, composta da regioni circolari sovrapposte. Per introdurre variabilità tra i campioni, tutte le immagini ricevono una perturbazione tramite rumore gaussiano di bassa intensità. La Figura 7.18 presenta esempi delle tre classi utilizzate nell'esperimento.

```
rng = np.random.default_rng(42)

def gerar_textura(classe, tamanho=64, ruido=0.10):
    """Genera una texture sintetica 64x64 appartenente a una delle tre classi."""
    if classe == "granular":
        escala = rng.uniform(0.14, 0.22)
        img = rng.normal(0.5, escala, (tamanho, tamanho))
        img = cv2.GaussianBlur(img.astype(np.float32), (3, 3), 0)

    elif classe == "listrada":
        n_periodos = rng.uniform(4, 8)
        amplitude = rng.uniform(0.22, 0.38)
        eixo_x = np.linspace(0, n_periodos * np.pi, tamanho)
        base = 0.5 + amplitude * np.sin(eixo_x)
        img = np.tile(base, (tamanho, 1)).astype(np.float32)
        img += rng.normal(0, 0.09, (tamanho, tamanho)).astype(np.float32)

    elif classe == "manchada":
        img = np.full((tamanho, tamanho), 0.5, dtype=np.float32)
        n_manchas = rng.integers(5, 11)
        for _ in range(n_manchas):
            cx, cy = rng.integers(0, tamanho, 2)
            raio = int(rng.integers(3, 11))
            intensidade = float(rng.uniform(0.15, 0.9))
            cv2.circle(img, (int(cx), int(cy)), raio, intensidade, -1)
        img = cv2.GaussianBlur(img, (5, 5), 0)

    else:
        raise ValueError(f"Classe desconhecida: {classe}")

    img = img + rng.normal(0, ruido, (tamanho, tamanho)).astype(np.float32)
    img = np.clip(img, 0, 1)
    return (img * 255).astype(np.uint8)

classes_textura = ["granular", "listrada", "manchada"]
amostras = [gerar_textura(c) for c in classes_textura]

mm.show(amostras, titles=classes_textura, cols=3, figsize=(9, 3))
```

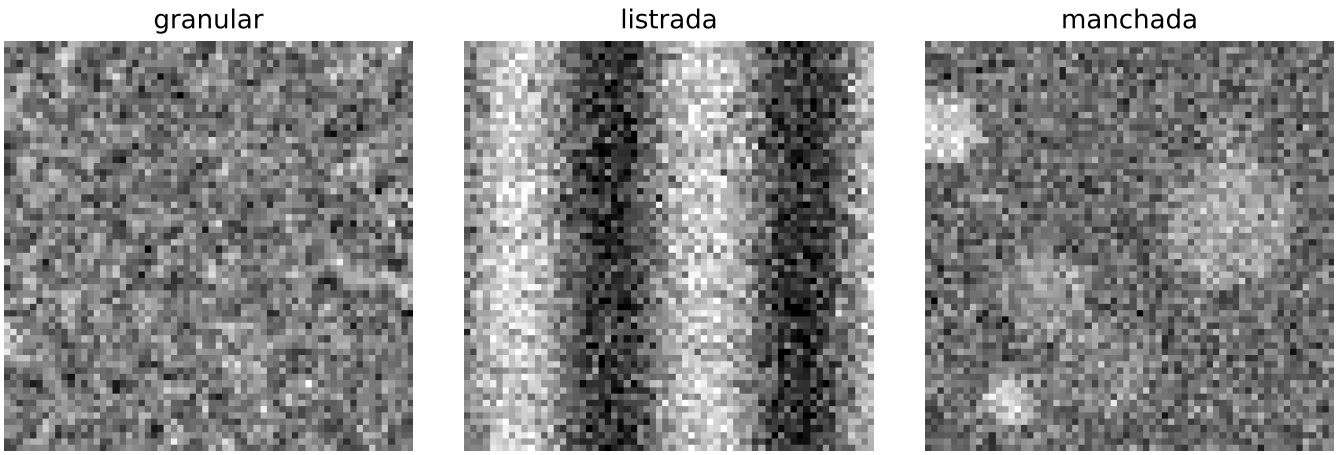


Figura 7.18: Campioni sintetici delle tre classi di texture utilizzate nell'esperimento di classificazione, generati con rumore gaussiano e variabilità intra-classe.

7.13.1 *Pipeline* di estrazione delle caratteristiche e valutazione comparativa

Per confrontare le prestazioni di diverse forme di rappresentazione delle immagini, è stato generato un dataset bilanciato contenente 60 campioni per ciascuna classe. Da questo dataset sono stati estratti tre tipi di vettori di caratteristiche, ciascuno rappresentante aspetti distinti dell'informazione visiva:

1. **Pixel grezzi**: vettore ottenuto dall'appiattimento (*flattening*) della matrice delle intensità dell'immagine, risultante in un vettore di $64 \times 64 = 4096$ attributi;
2. **Istogramma LBP uniforme**: istogramma normalizzato delle frequenze dei pattern locali prodotti dall'operatore LBP uniforme, composto da 10 attributi;
3. **Descrittore HOG**: vettore formato da istogrammi di gradienti orientati, che rappresentano la distribuzione spaziale delle orientazioni dei bordi, per un totale di 128 attributi.

Poiché questi descrittori presentano scale e dimensionalità distinte, i vettori di caratteristiche vengono standardizzati utilizzando lo `StandardScaler`, in modo che ciascun attributo abbia media zero e deviazione standard unitaria. Questa fase evita che attributi con maggiore ampiezza influenzino in modo sproporzionato il calcolo delle distanze euclidee impiegato dal classificatore.

La valutazione viene effettuata utilizzando l'algoritmo k-NN con $k = 5$, sotto lo stesso protocollo di validazione incrociata stratificata in cinque partizioni descritto nella sezione precedente. L'accuratezza media ottenuta lungo le cinque esecuzioni, vedi Figura 7.19, fornisce una stima più stabile delle prestazioni del classificatore, riducendo la dipendenza da una singola suddivisione tra addestramento e test.

```
rng = np.random.default_rng(42)

# 1. Generazione del database
X_bruto, X_lbp, X_hog, y_textura = [], [], [], []

for classe in classes_textura:
    for _ in range(60):
        img = gerar_textura(classe, ruido=0.10)

        # Estrazione 1: Pixel grezzi
        X_bruto.append(img.ravel())

        # Estrazione 2: Istogramma LBP uniforme
        lbp = local_binary_pattern(img, P=8, R=1, method="uniform")
        hist_lbp, _ = np.histogram(lbp, bins=10, range=(0, 10), density=True)
        X_lbp.append(hist_lbp)
```

```
        # Estrazione 3: Descrittore HOG
        feat_hog = hog(img, orientations=8, pixels_per_cell=(16, 16), cells_per_block=(1, 1))
        X_hog.append(feat_hog)

        y_textura.append(classe)

y_textura = np.array(y_textura)
descritores = {
    "Pixels Brutos": np.array(X_bruto),
    "LBP (Textura)": np.array(X_lbp),
    "HOG (Forma)": np.array(X_hog)
}

# 2. Valutazione statistica tramite 5-fold cross-validation
resultados_media = {}
resultados_desvio = {}

knn = KNeighborsClassifier(n_neighbors=5)
scaler = StandardScaler()

for nome, X_dados in descritores.items():
    X_norm = scaler.fit_transform(X_dados)
    scores = cross_val_score(knn, X_norm, y_textura, cv=5, scoring="accuracy")
    resultados_media[nome] = scores.mean()
    resultados_desvio[nome] = scores.std()
    print(f"{nome:15s} -> Accuratezza Media: {scores.mean():.4f} (± {scores.std():.4f})")

# 3. Tracciamento del grafico comparativo formale
plt.figure(figsize=(7, 4.5))
nomes_desc = list(resultados_media.keys())
medias = list(resultados_media.values())
desvios = list(resultados_desvio.values())

bars = plt.bar(nomes_desc, medias, yerr=desvios, capsize=6,
               color=["#6366f1", "#9333ea", "#f97316"],
               width=0.45, edgecolor="black", alpha=0.85)
plt.ylabel("Acurácia Média (5-Fold CV)", fontsize=11)
plt.title("Análise Comparativa de Descritores para Classificação de Texturas",
          fontsize=12, fontweight="bold")
plt.ylim(0.3, 1.1)
plt.grid(axis="y", linestyle="--", alpha=0.5)

for bar in bars:
    h = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2,
             h + 0.03, f"{h:.3f}", ha="center", fontweight="bold")

plt.tight_layout()
plt.show()
```

```
Pixels Brutos   -> Accuratezza Media: 0.6889 (± 0.0478)
LBP (Textura)   -> Accuratezza Media: 0.7611 (± 0.0648)
HOG (Forma)     -> Accuratezza Media: 0.4889 (± 0.0624)
```

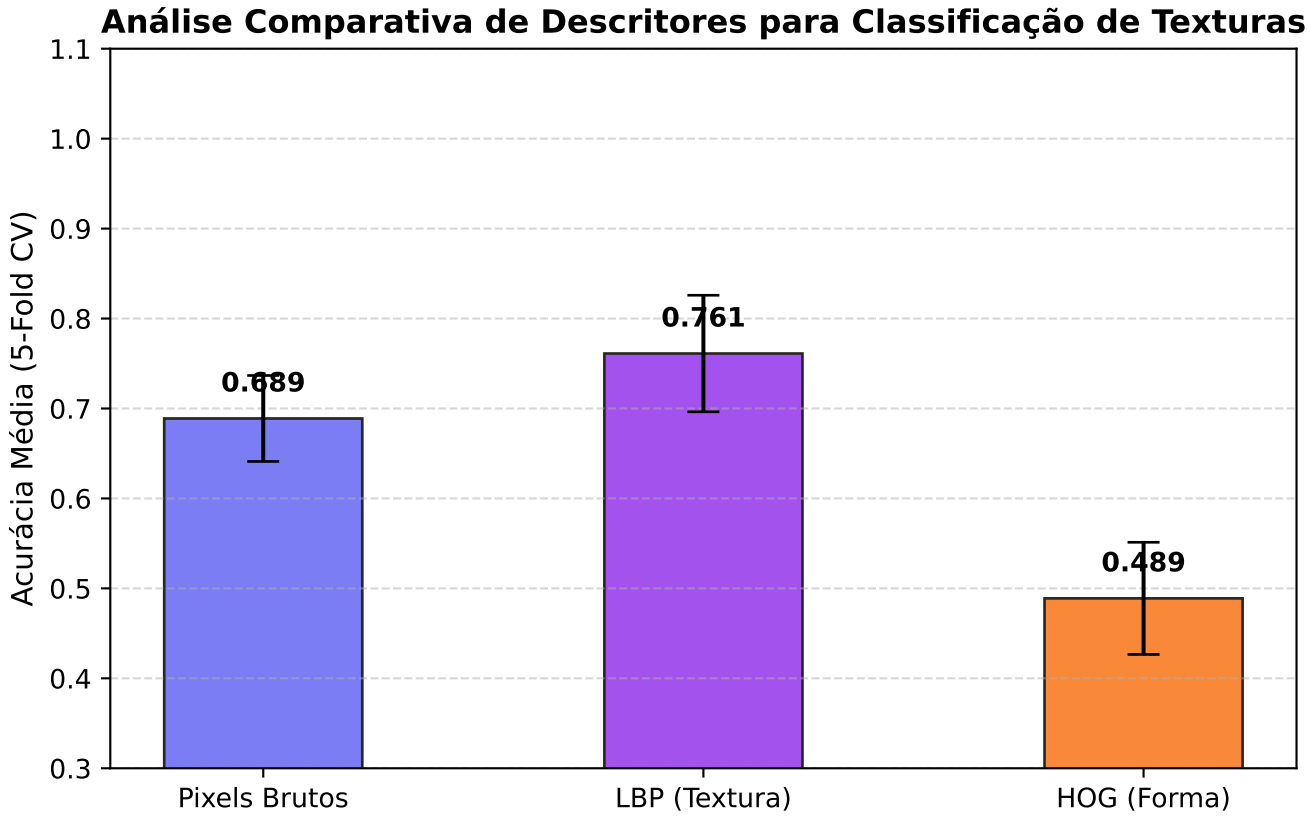


Figura 7.19: Acurácia média obtida via validação cruzada (5-fold) para os descritores de Pixels Brutos, LBP e HOG aplicados à base de texturas sintéticas.

Perché funziona? — LBP come rappresentazione delle trame

Il descrittore LBP rappresenta la trama di un'immagine tramite un istogramma normalizzato che conta la frequenza dei pattern locali di intensità. Invece di memorizzare direttamente i valori dei pixel o le loro posizioni, questa rappresentazione riassume la distribuzione delle microstrutture presenti nell'immagine, producendo un vettore di caratteristiche compatto.

In questo esperimento sono stati confrontati tre tipi di descrittori: pixel grezzi, LBP e HOG. I vettori formati dai **pixel grezzi** preservano tutte le intensità dell'immagine, ma incorporano anche variazioni derivanti da rumore e piccoli spostamenti spaziali, il che può rendere difficile il confronto tra campioni tramite la distanza euclidea.

Il descrittore **HOG** rappresenta la distribuzione delle orientazioni dei gradienti, essendo adatto a descrivere forme e contorni. Poiché le immagini utilizzate in questo progetto differiscono principalmente per le proprietà di trama, e non per la presenza di contorni ben definiti, questa rappresentazione tende a catturare meno informazioni discriminative rispetto all'LBP.

L'**LBP**, invece, è stato sviluppato specificamente per caratterizzare pattern locali di trama. Il suo istogramma descrive la frequenza delle microstrutture presenti nell'immagine, indipendentemente dalla loro posizione esatta, rendendo la rappresentazione meno sensibile a piccole variazioni spaziali e a cambiamenti monotonici dell'illuminazione.

Sebbene gli istogrammi delle diverse classi presentino distribuzioni distinte, il rumore gaussiano introdotto nella generazione delle immagini aumenta la variabilità tra campioni della stessa classe e può produrre regioni di sovrapposizione nello spazio delle caratteristiche. Di conseguenza, alcune trame possono essere confuse dal classificatore. Ciò nonostante, quando le caratteristiche rilevanti per distinguere le classi sono associate ai pattern locali di trama, ci si aspetta che descrittori progettati per questo scopo, come l'LBP, producano rappresentazioni più informative rispetto a quelle basate solo sulle intensità dei pixel o sulle orientazioni dei gradienti.

7.13.2 Diagnóstico Fino del Classificatore: Precisione, Recall e F1-Score

L'accuratezza riassume le prestazioni del classificatore in un unico valore, ma non indica come queste prestazioni siano distribuite tra le diverse classi. Per un'analisi più dettagliata, si utilizzano metriche calcolate individualmente per ciascuna classe.

La **precisione** (*precision*) misura la proporzione di campioni classificati come appartenenti a una classe che vi appartengono effettivamente. Il **recall** (*recall*) misura la proporzione di campioni della classe che sono stati correttamente identificati dal classificatore. Il **F1-score** corrisponde alla media armonica tra precisione e recall, fornendo un indicatore che bilancia entrambe le misure.

Il report riporta anche il **supporto** (*support*), ovvero il numero di campioni di ciascuna classe presenti nel set di test. Questa informazione è importante per contestualizzare le metriche, poiché i risultati ottenuti su pochi campioni tendono a presentare una maggiore variabilità.

La Figura 7.20 presenta queste metriche per le tre classi di texture. Insieme, esse consentono di identificare differenze di prestazioni che non sono evidenti solo attraverso l'accuratezza. Ad esempio, una classe può presentare alta precisione e un recall inferiore, indicando che il classificatore commette pochi falsi positivi, ma non riesce a identificare parte dei campioni che appartengono effettivamente a quella classe. Questo tipo di analisi aiuta a comprendere i limiti del modello e a individuare possibili strategie per il suo miglioramento.

```
X_lbp_data = np.array(X_lbp)
y_textura_data = np.array(y_textura)

# Eseguire uno split train/test per il report dettagliato
Xt_treino, Xt_teste, yt_treino, yt_teste = train_test_split(
    X_lbp_data, y_textura_data, test_size=0.3, random_state=42, stratify=y_textura_data
)

# Scalare i dati
scaler = StandardScaler()
Xt_treino_scaled = scaler.fit_transform(Xt_treino)
Xt_teste_scaled = scaler.transform(Xt_teste)

# Addestrare il classificatore k-NN
knn_textura = KNeighborsClassifier(n_neighbors=5)
knn_textura.fit(Xt_treino_scaled, yt_treino)

y_pred = knn_textura.predict(Xt_teste_scaled)

report = classification_report(
    yt_teste,
    y_pred,
    target_names=classes_textura,
    output_dict=True
)

print("=== REPORT DI CLASSIFICAZIONE DETTAGLIATO ===")
print(f"{'Classe':<12} {'Precisão':>10} {'Revocação':>12} {'F1-score':>10} {'Suporte':>10}")
for classe in classes_textura:
    r = report[classe]
    print(f"{classe:<12} {r['precision']:>10.2f} {r['recall']:>12.2f} "
          f"{r['f1-score']:>10.2f} {r['support']:>10.0f}")

precision = precision_score(yt_teste, y_pred, average=None, labels=classes_textura)
recall = recall_score(yt_teste, y_pred, average=None, labels=classes_textura)
f1 = f1_score(yt_teste, y_pred, average=None, labels=classes_textura)

fig, ax = plt.subplots(figsize=(10, 5))
x = np.arange(len(classes_textura))
```

```
width = 0.25

bars1 = ax.bar(x - width, precision, width, label='Precisão', color='#6366f1', alpha=0.8)
bars2 = ax.bar(x, recall, width, label='Revocação', color='#f97316', alpha=0.8)
bars3 = ax.bar(x + width, f1, width, label='F1-Score', color='#22c55e', alpha=0.8)

ax.set_xlabel('Classe', fontsize=12)
ax.set_ylabel('Score', fontsize=12)
ax.set_title('Métricas por Classe - Classificação de Texturas', fontsize=14,
fontweight='bold')
ax.set_xticks(x)
ax.set_xticklabels(classes_textura)
ax.legend(loc='upper right')
ax.set_ylim(0, 1.35)
ax.grid(axis='y', alpha=0.3)

for bars in [bars1, bars2, bars3]:
    for bar in bars:
        height = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2., height + 0.02,
                f'{height:.2f}', ha='center', va='bottom', fontsize=9)

plt.tight_layout()
plt.show()

print("Interpretazione delle metriche:")
print("- Precisione: tra i campioni classificati come appartenenti alla classe,",
      "quanti erano corretti?")
print("- Recall: tra i campioni che appartengono realmente alla classe, quanti",
      "sono stati identificati?")
print("- F1-Score: media armonica tra precisione e recall.")
print("- Supporto: numero di campioni reali di ciascuna classe presenti nel set di test.")
print("\nIl supporto non misura le prestazioni; informa solo su quanti esempi di ciascuna",
      "classe sono \nstati utilizzati nella valutazione.")
```

```
=== REPORT DI CLASSIFICAZIONE DETTAGLIATO ===
Classe          Precisão    Revocação   F1-score    Suporte
granular            0.67        0.78       0.72         18
listrada            0.55        0.61       0.58         18
manchada            1.00        0.72       0.84         18
```

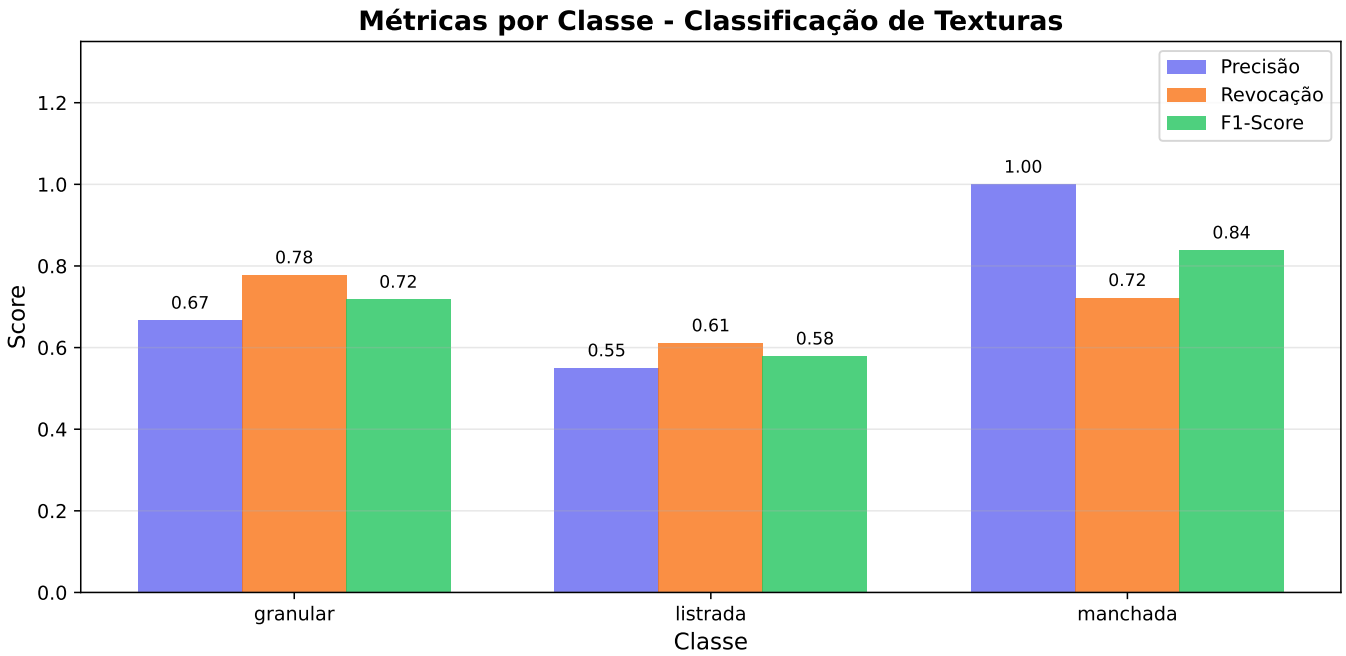


Figura 7.20: Métricas di valutazione dettagliate per il classificatore k-NN con descrittori LBP

```
Interpretazione delle metriche:
- Precisione: tra i campioni classificati come appartenenti alla classe, quanti erano
corretti?
- Recall: tra i campioni che appartengono realmente alla classe, quanti sono stati
identificati?
- F1-Score: media armonica tra precisione e recall.
- Supporto: numero di campioni reali di ciascuna classe presenti nel set di test.

Il supporto non misura le prestazioni; informa solo su quanti esempi di ciascuna classe sono
stati utilizzati nella valutazione.
```

7.14 Limiti dei Descrittori Artigianali

Gli esperimenti di questo capitolo mostrano che i descrittori classici possono essere piuttosto efficaci in compiti di classificazione, ma presentano anche importanti limitazioni:

- **Specificità:** ogni descrittore è stato sviluppato per rappresentare un determinato tipo di informazione, come colore, tessitura o forma. Pertanto, un descrittore adeguato per un compito potrebbe non essere il più appropriato per un altro.
- **Dipendenza dagli iperparametri:** le prestazioni di descrittori come LBP e HOG dipendono dalla scelta di parametri, come il raggio di vicinanza, il numero di punti campionati, la dimensione della cella e il numero di orientamenti, che devono essere regolati in base all'applicazione.
- **Rappresentazione limitata:** i descrittori di colore, tessitura e gradiente catturano proprietà di basso livello dell'immagine, ma non rappresentano direttamente concetti semantici più complessi, come oggetti o scene.
- **Maledizione della dimensionalità:** descrittori molto estesi possono ridurre l'efficacia di classificatori basati sulla distanza, come il k-NN.

Queste limitazioni motivano l'evoluzione delle tecniche studiate nei prossimi capitoli. Il **Capitolo 8** presenta metodi classici per il rilevamento e la corrispondenza di caratteristiche nelle immagini, mentre il **Capitolo 9** introduce le **Reti Neurali Convoluzionali**, capaci di apprendere automaticamente rappresentazioni adeguate per ciascun compito a partire dai dati.

7.15 Riassunto

In questo capitolo sono stati presentati i fondamenti del riconoscimento di pattern applicato alle immagini. I principali concetti studiati sono stati:

- ***Pipeline* di riconoscimento di pattern:** acquisizione, pre-processing, estrazione di descrittori, classificazione e valutazione.
- **Descrittori classici:** descrittori di colore, LBP per la texture e HOG per la forma, utilizzati per rappresentare diverse caratteristiche delle immagini.
- **Normalizzazione delle caratteristiche:** standardizzazione (*Z-score*) per evitare che attributi di maggiore magnitudine dominino il calcolo delle distanze.
- **Classificatore k-NN:** classificazione basata sui k vicini più prossimi nello spazio delle caratteristiche.
- **Scelta del parametro k:** influenza del valore di k sulle prestazioni del classificatore e uso della validazione incrociata per la sua selezione.
- **Valutazione dei classificatori:** accuratezza, matrice di confusione, precisione, recall e F1-score come metriche complementari di prestazione.
- **Limiti dei descrittori artigianali:** specificità, dipendenza dagli iperparametri e difficoltà nel rappresentare informazioni di alto livello.

I concetti sono stati illustrati mediante esperimenti con la base pubblica `load_digits`, texture sintetiche generate a scopo didattico e dati simulati di descrittori di frutta.

7.16 🤖 Uso del Gemini Notebook come Tutor

In questa edizione, il **Gemini Notebook** viene presentato come uno strumento di supporto allo studio. Il sistema utilizza esclusivamente i documenti messi a disposizione dall'autore come fonte di conoscenza, consentendo di esplorare i concetti del capitolo tramite domande, riepiloghi e spiegazioni correlate al materiale studiato.

🎓 Studia con il Tutor Intelligente

🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 07

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato realizzato esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, oppure segui il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ **Avviso sui Contenuti Generati dall'IA**

Le risposte fornite dal Gemini Notebook possono contenere imprecisioni o omissioni. Quando necessario, conferma le informazioni utilizzando il materiale di questo capitolo e altre fonti accademiche affidabili. L'esecuzione degli esempi pratici presentati lungo il testo rimane il modo migliore per consolidare i concetti studiati.

7.17 Lista di Esercizi

Gli esercizi seguenti esplorano ed estendono i concetti presentati in questo capitolo tramite adattamenti degli algoritmi implementati, analisi sperimentali e confronti tra diversi approcci.

1. **(10%)** Implementare un descrittore di colore (istogramma RGB o HSV, con almeno 16 *bin* per canale) per le tre classi di frutti simulati in Figura 7.1.. Addestrare un classificatore k-NN con questo descrittore,

confrontarne l'accuratezza con quella ottenuta dai descrittori LBP e HOG (Figura 7.9) e discutere in quali situazioni l'informazione cromatica è maggiormente discriminante.

2. **(15%)** Studiare l'effetto della normalizzazione delle caratteristiche (*Z-score*) sulle prestazioni del k-NN in uno spazio di attributi eterogeneo, combinando descrittori di colore, LBP e HOG in un unico vettore. Confrontare i risultati ottenuti con e senza normalizzazione per almeno tre valori di k.

3. **(15%)** Riprodurre l'analisi di *overfitting* e *underfitting* della Figura 7.17 variando la dimensione del set di addestramento (ad esempio, 20%, 50% e 80% del database `load_digits`). Discutere come la quantità di esempi influenzi la scelta del valore di k.

4. **(15%)** Estendere il Progetto Pratico 2 aggiungendo una quarta classe sintetica di trama. Valutare precisione, richiamo e F1-score per ciascuna classe, seguendo lo schema della Figura 7.20, e analizzare l'impatto della nuova classe sulla matrice di confusione.

5. **(15%)** Implementare manualmente il classificatore k-NN, senza utilizzare `sklearn`:

   `sklearn.neighbors.KNeighborsClassifier`,

   completando la funzione `knn_passo_a_passo` presentata nel capitolo. Confrontare l'accuratezza e il tempo di esecuzione dell'implementazione manuale con quella del `scikit-learn` su dataset di dimensioni crescenti e collegare i risultati alla maledizione della dimensionalità.

6. **(15%)** Studiare l'influenza dei parametri `orientations`, `pixels_per_cell` e `cells_per_block` del descrittore HOG sul database `load_digits`. Valutare almeno quattro combinazioni di parametri e discutere il compromesso tra dimensionalità del descrittore e prestazioni del classificatore.

7. **(15%)** Valutare l'influenza dei parametri P (numero di vicini) e R (raggio) del descrittore LBP nella classificazione delle trame sintetiche del Progetto Pratico 2, considerando $P \in \{4, 8, 16\}$ e $R \in \{1, 2, 3\}$. Analizzare come tali parametri influenzino la capacità discriminativa del descrittore.

8. **(Bonus – 10%)** Implementare manualmente la validazione incrociata *k-fold* per il classificatore k-NN sul database `load_digits`, senza utilizzare `cross_val_score`, e confrontare i risultati con quelli ottenuti dall'implementazione del `scikit-learn` presentata in Figura 7.16..

Riferimenti del Capitolo

La base teorica e gli esperimenti presentati in questo capitolo si fondano sui seguenti riferimenti:

- Gonzalez (2018), per i fondamenti degli descrittori statistici di texture e delle operazioni di preprocessing applicate all'estrazione delle caratteristiche.
- Szeliski (2022), per la presentazione del pipeline classico di riconoscimento di pattern, dell'estrazione di descrittori e della valutazione dei classificatori in Visione Computazionale.
- Duda (2001), per i fondamenti teorici del riconoscimento di pattern, del classificatore k-NN e della relazione tra bias e varianza.
- Cover (1967), per la formulazione originale dell'algoritmo dei k vicini più prossimi.
- Ojala (2002), per la formulazione del descrittore *Local Binary Patterns* (LBP) e della sua variante uniforme, utilizzata in questo capitolo.
- Dalal (2005), per la formulazione del descrittore *Histogram of Oriented Gradients* (HOG), impiegato nella rappresentazione di forma e contorno.
- Pedregosa (2011), per l'implementazione del classificatore k-NN, delle metriche di valutazione e della validazione incrociata nella libreria `scikit-learn`.
- Quilici-gonzalez (2014), per la presentazione didattica di classificatori tradizionali di Riconoscimento di Pattern, come Alberi di Decisione, Regole di Classificazione e Macchine a Vettori di Supporto (SVM), complementari al classificatore k-NN esplorato in questo capitolo.
- Quilici-gonzalez (2026), per l'aggiornamento e l'ampliamento di questi contenuti nella sua 2ª edizione, attualmente in produzione.

Executar Colab Abrir GitHub

7.18 💻 Parte Pratica con Esercizi di Programmazione

La presente lista di esercizi di programmazione (EP) consolida le formulazioni teoriche presentate nel corso del Capitolo 7 — Classificazione di Immagini e Riconoscimento di Pattern — attraverso un percorso pratico applicato. Diversamente dalla manipolazione diretta dei pixel dei capitoli precedenti, gli EP di questo capitolo lavorano con le **grandezze intermedie** di una *pipeline* reale di riconoscimento di pattern — vettori di caratteristiche, distanze, etichette previste e reali, codici binari locali e istogrammi di orientamento — consentendo di validare manualmente ogni fase del ragionamento senza dipendere da librerie esterne di apprendimento automatico.

L'incatenamento degli esercizi riproduce il flusso concettuale del capitolo: si inizia con l'implementazione manuale della regola di decisione del classificatore **k-NN** su un piccolo spazio di caratteristiche; successivamente, si rivisita, sotto l'ottica della normalizzazione delle caratteristiche, il classificatore implementato nel primo esercizio della lista; si prosegue con il calcolo delle metriche di **valutazione** (matrice di confusione, precisione e richiamo) a partire da etichette previste e reali; si continua con la codifica manuale del descrittore di texture **LBP** da un intorno 3×3; si approfondisce il calcolo dell'istogramma delle orientazioni del descrittore **HOG** per una singola cella; si avanza, quindi, verso l'integrazione di **estrazione di descrittori**, **classificazione k-NN** e **valutazione multi-classe** in una *pipeline* completa di riconoscimento di texture; e si conclude con l'applicazione di questa stessa *pipeline* su un'**immagine reale** (formato PGM), in cui il descrittore LBP viene calcolato direttamente sui pixel di un mosaico di texture.

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EPs)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, esegui `TestSuite("EP07_01.estensione").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola il voto automaticamente.

Per testare il codice Python direttamente, senza salvare un file, usa `run_code(codice)` passando il codice come *stringa* in una variabile `codice`:

```
codice = """
# ... il tuo codice qui ...
"""
TestSuite("EP07_01").run_code(codice)
```

Riepilogo dei Metodi di `morph.py` (Cap. 7)

La libreria `morph.py` mette a disposizione due versioni per la maggior parte degli algoritmi: una **didattica** (metodi che terminano in `0`), implementata passo dopo passo in NumPy, e un'altra **classica**, basata sulle librerie `scikit-learn` e `scikit-image`. Le implementazioni didattiche vengono utilizzate negli **Esercizi di Programmazione (EP)**, poiché non dipendono da librerie esterne e vengono eseguite entro il limite di memoria dell'ambiente **VPL** di Moodle. Le versioni classiche, invece, sono più efficienti e indicate per esperimenti in ambienti come Colab e Jupyter Notebook, ma normalmente **non possono essere utilizzate negli EP** di Moodle, poiché la libreria `scikit-learn` supera la memoria disponibile nel VPL.

1. **Lettura dei dati (`readClasses`, `readDataset`, `readTrain`, `readTest`)**
   Standardizzano l'input dei set di addestramento e di test, restituendo le matrici delle caratteristiche (X) e i vettori delle etichette (y).
2. **Classificazione (`knn0` / `knn`)**
   Implementano l'algoritmo dei **k-nearest neighbors (k-NN)** per la classificazione binaria e multiclasse, utilizzando la distanza Euclidea o di Manhattan.
3. **Normalizzazione (`zscore0` / `zscore`)**
   Applicano la normalizzazione *z-score* agli attributi, riducendo le differenze di scala prima della classificazione.
4. **Valutazione (`confusion0` / `confusion`)**
   Calcolano la matrice di confusione e metriche come accuratezza, precisione e richiamo, sia per problemi binari che multiclasse.
5. **Descrittore di texture (`lbp0` / `lbp`)**
   Calcolano il ***Local Binary Pattern*** **(LBP)**, consentendo di ottenere la mappa LBP, il codice di un pixel o l'istogramma di una regione dell'immagine.
6. **Descrittore di forma (`hog0` / `hog`)**
   Calcolano l'***Histogram of Oriented Gradients*** **(HOG)**, producendo istogrammi delle orientazioni dei gradienti per rappresentare informazioni su forma e contorno.

7.18.1 EP07_01 🟢 Classificatore k-NN Passo dopo Passo

Il `KNeighborsClassifier` di `scikit-learn`, utilizzato nel corso del capitolo, nasconde dietro una singola chiamata (`.fit` / `.predict`) una regola decisionale piuttosto semplice: per ogni nuova osservazione, calcolare la distanza rispetto a tutti gli esempi di addestramento, selezionare i k più vicini e votare per la classe di maggioranza tra di essi.

Prima di fare affidamento sulla libreria, ti è stato affidato il compito di implementare questa regola da zero, per uno spazio delle caratteristiche bidimensionale, esattamente come fa internamente il simulatore interattivo della frontiera decisionale del capitolo a ogni clic dell'utente.

7.18.1.1 📋 Linee Guida di Implementazione

1. **Quantità e parametro:** Leggere l'intero N (numero di esempi di addestramento) e l'intero dispari k (numero di vicini).
2. **Esempi di addestramento:** Per ciascuno degli N esempi, leggere tre valori: le coordinate x e y (reali) e l'etichetta r (intero, 0 o 1).
3. **Query:** Leggere l'intero Q (numero di punti di query) e successivamente le coordinate x_q, y_q (reali) di ciascuna query.

4. **Distanza:** Per ogni query, calcolare la distanza euclidea rispetto a **tutti** gli esempi di addestramento:

$$d(x_q, x_i) = \sqrt{(x_q - x_i)^2 + (y_q - y_i)^2}.$$

5. **Selezione dei vicini:** Ordinare gli esempi per distanza crescente e selezionare i primi k. In caso di **parità di distanza** al confine del k-esimo vicino, risolvere a favore dell'esempio letto **per primo** nell'input (ordine di lettura stabile).
6. **Votazione di maggioranza:** Contare i voti di ciascuna classe tra i k vicini selezionati. In caso di **pareggio nella votazione** (possibile solo quando k è pari, cosa che non dovrebbe verificarsi per la direttiva del punto 1, ma gestire in modo difensivo), assegnare la classe del vicino più prossimo tra le classi in parità.
7. **Output:** Per ogni query, nell'ordine di input, stampare la classe prevista. Alla fine, stampare il totale delle query classificate come classe `1`.

7.18.1.2 📌 Vincoli Computazionali

- **Metrica fissa:** utilizzare esclusivamente la distanza euclidea (non la *distanza al quadrato*) per l'ordinamento, sebbene il risultato del confronto sia lo stesso.
- **k sempre dispari:** l'input garantisce k dispari e $k \leq N$; ciononostante, implementare il pareggio del punto 6 per robustezza.
- **Stabilità:** nell'ordinamento per distanza, preservare l'ordine relativo degli esempi con la stessa distanza (ordinamento stabile).

7.18.1.3 🧠 Fondamenti Teorici

Elemento	Ruolo nel k-NN
Spazio delle caratteristiche	Insieme di tutti i vettori (x, y) possibili
Distanza euclidea	Misura di similarità tra osservazioni
k piccolo	Frontiera irregolare, alta varianza
k grande	Frontiera regolare, alto bias
Votazione di maggioranza	Regola decisionale $\hat{y} = \text{moda}\{y_i : x_i \in N_k(x)\}$

7.18.1.4 📦 Specifica di Input e Output (VPL)

Input:

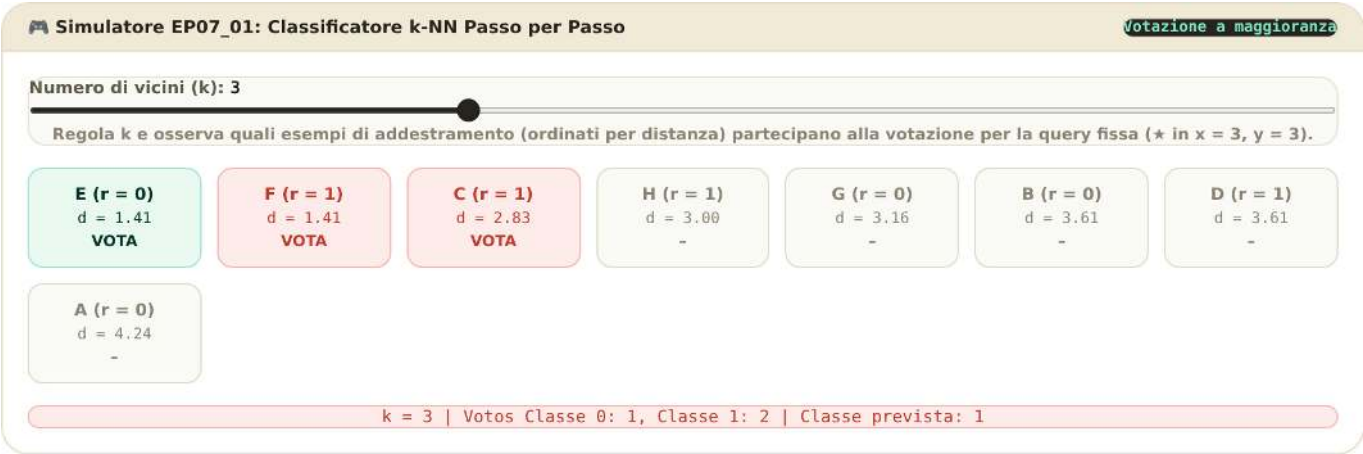
- Riga 1: Interi N e k, separati da spazio.
- Prossime N righe: tre valori per riga — x, y (reali) e r (intero $\in \{0, 1\}$), separati da spazio.
- Riga successiva: intero Q.
- Prossime Q righe: due valori per riga — x_q, y_q (reali), separati da spazio.

Output:

- Q righe, ciascuna con la classe prevista (`0` o `1`) per la rispettiva query, nell'ordine di input.
- Ultima riga: `Totale classe 1: X`.

7.18.1.5 📌 Esempi

Input	Output	Osservazione
4 3 0 0 0 1 0 0 5 5 1 6 5 1 1 1 1	0 Totale classe 1: 0	Query vicina al gruppo di classe 0.
4 1 0 0 0 1 0 0 5 5 1 6 5 1 2 0.9 0.1 5.5 5.1	0 1 Totale classe 1: 1	Con $k = 1$, ogni query eredita la classe del vicino più prossimo.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap07/sim-ep0701.html

Figura 7.21: Simulatore EP07_01: Classificatore k-NN Passo a Passo

```
%%writefile EP07_01.py
# Codice Python
```

```
Overwriting EP07_01.py
```

```
TestSuite("EP07_01.py").run()
```

```
<IPython.core.display.HTML object>
```

7.18.2 EP07_02 🟡 Normalizzazione *Z-score* e Robustezza del k-NN a Scale Diverse

Questo esercizio riprende il classificatore implementato nell'**EP07_01**, questa volta sotto la prospettiva discussa nella sezione *L'Impatto della Scala e la Normalizzazione delle Caratteristiche* del capitolo: il k-NN decide in base alla distanza tra vettori, per cui una caratteristica misurata su una scala molto più ampia rispetto alle altre tende a **dominare** il calcolo della distanza, anche quando non è la più rilevante per separare le classi.

Un sistema di ispezione registra, per ogni pezzo, la sua **area** (in pixel, che può arrivare a centinaia o migliaia) e la sua **circolarità** (sempre tra 0 e 1). Ti è stato affidato il compito di classificare nuovi pezzi tramite k-NN

in due modi — con e senza la standardizzazione *Z-score* presentata nel capitolo — e di riportare in quali casi i due approcci **divergono**.

7.18.2.1 📋 Linee Guida di Implementazione

1. **Quantità e parametro:** Leggere l'intero N (numero di esempi di addestramento) e l'intero dispari k.
2. **Esempi di addestramento:** Per ciascuno degli N esempi, leggere tre valori: l'area x_1 (reale), la circolarità x_2 (reale) e l'etichetta r (intero, 0 o 1).
3. **Query:** Leggere l'intero Q e, successivamente, le coordinate x_1, x_2 di ciascuna query.
4. **Classificazione senza normalizzazione:** Per ogni query, classificarla tramite k-NN direttamente su (x_1, x_2), con distanza euclidea e le stesse regole di pareggio dell'EP07_01 (ordine di lettura per distanze a pari merito; vicino più prossimo tra classi a pari merito nella votazione).
5. **Parametri di normalizzazione:** Calcolare la media μ_j e la deviazione standard **popolazionale** σ_j (divisione per N, non per $N-1$ — la stessa convenzione adottata dalla classe `StandardScaler`) di ciascuna caratteristica $j \in \{1, 2\}$, **esclusivamente sul set di addestramento**.
6. **Standardizzazione:** Trasformare ciascuna caratteristica di addestramento e di query tramite
$$z_j = \frac{x_j - \mu_j}{\sigma_j}.$$
Se $\sigma_j = 0$ (caratteristica costante nell'addestramento), definire $z_j = 0$ per tutti i campioni di quella caratteristica, evitando la divisione per zero.
7. **Classificazione con normalizzazione:** Ripetere la classificazione k-NN del punto 4, ora sui vettori standardizzati (z_1, z_2), con le stesse regole di pareggio.
8. **Output:** Per ogni query, nell'ordine di input, stampare le due classi previste. Alla fine, stampare il numero di query in cui le due classificazioni **divergono**.

7.18.2.2 📌 Vincoli Computazionali

- **Adattamento solo sul training:** μ_j e σ_j sono calcolati unicamente a partire dal set di addestramento e riapplicati alle query — mai ricalcolati a partire da esse. Questa pratica evita la **dispersione dei dati** (*data leakage*), menzionata nella sezione sulla normalizzazione del capitolo.
- **Deviazione standard popolazionale:** utilizzare $\sigma_j = \sqrt{\frac{1}{N}\sum_i (x_{i,j} - \mu_j)^2}$, e non la versione campionaria (divisione per $N-1$).
- **Caratteristica costante:** trattare $\sigma_j = 0$ come caso speciale (punto 6); non deve verificarsi un errore di divisione per zero.
- **Regole di pareggio:** riutilizzare esattamente le convenzioni dell'EP07_01, sia nella selezione dei k vicini che nella votazione a maggioranza.

7.18.2.3 🧠 Fondamenti Teorici

Elemento	Ruolo
Standardizzazione *Z-score*	Riscalare ogni caratteristica a media 0 e deviazione standard 1, rendendo scale eterogenee comparabili
Adattamento (*fit*) solo sul training	Garantisce che la valutazione sulle query rifletta solo ciò che il modello ha appreso nell'addestramento
Distanza euclidea senza normalizzazione	Dominata dalla caratteristica con maggiore ampiezza — qui, l'area
Predizione divergente	Evidenzia che la scala delle caratteristiche, e non solo l'algoritmo o i dati, può determinare il confine decisionale del k-NN

Questo esercizio sottolinea, in modo controllato, la ragione per cui lo `StandardScaler` viene applicato prima del k-NN nel corso del capitolo: senza questo passaggio, caratteristiche come la circolarità — anche

se altamente discriminative — possono essere praticamente ignorate dal classificatore di fronte a una caratteristica come l'area con ampiezza centinaia di volte maggiore.

7.18.2.4 📦 Specifica di Input e Output (VPL)

Input:

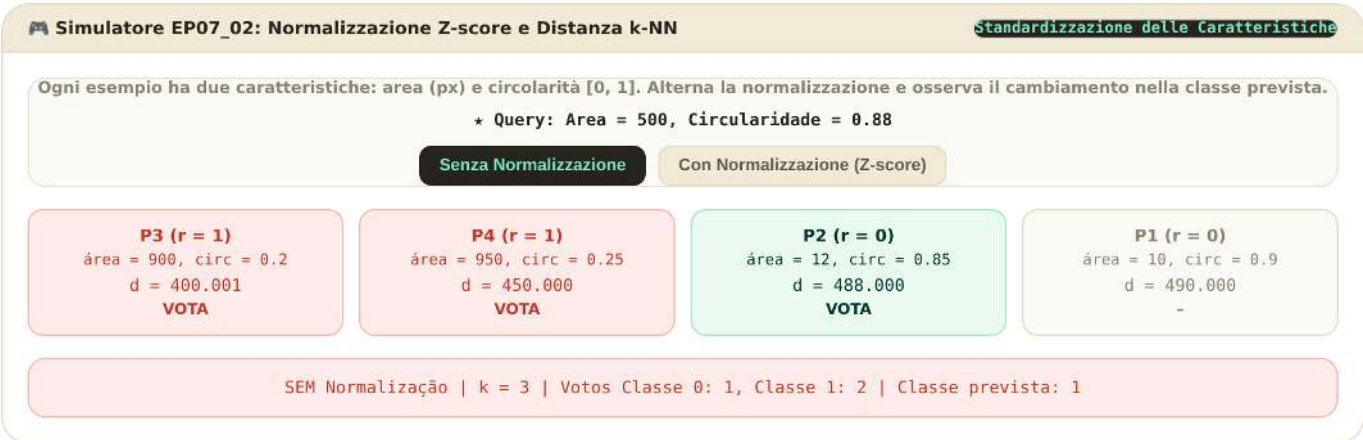
- Riga 1: Interi N e k, separati da spazio.
- Prossime N righe: tre valori per riga — x_1, x_2 (reali) e r (intero $\in \{0, 1\}$), separati da spazio.
- Prossima riga: intero Q.
- Prossime Q righe: due valori per riga — x_1, x_2 (reali) della query, separati da spazio.

Output:

- Q righe, nel formato `SemNorm=<0|1> ComNorm=<0|1>`, nell'ordine di input delle query.
- Ultima riga: `Divergiu: <int>`.

7.18.2.5 📌 Esempi

Input	Output	Osservazione
4 3 10 0.9 0 12 0.85 0 900 0.2 1 950 0.25 1 1 500 0.88	SemNorm=1 ComNorm=0 Divergiu: 1	Senza normalizzazione, l'area (scala di centinaia) domina la distanza e la query viene classificata come classe `1`. Dopo la standardizzazione, la circolarità — molto più vicina ai campioni di classe `0` — inizia a pesare in modo confrontabile, e la predizione cambia a `0`.
2 1 0 0.5 0 100 0.5 1 1 60 0.5	SemNorm=1 ComNorm=1 Divergiu: 0	La circolarità è costante nell'addestramento ($\sigma_2 = 0$); secondo la regola del punto 6, $z_2 = 0$ per tutti i campioni, e la classificazione dipende solo dall'area in entrambi i casi.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap07/sim-ep0702.html

Figura 7.22: Simulatore EP07_02: Effetto della Normalizzazione *Z-score* sulla Distanza k-NN

```
%%writefile EP07_02.py
# Codice Python
```

```
Overwriting EP07_02.py
```

```
TestSuite("EP07_02.py").run()
```

```
<IPython.core.display.HTML object>
```

7.18.3 EP07_03 🟡 Valutazione tramite Matrice di Confusione

Un classificatore binario della qualità della saldatura è stato addestrato e testato su una linea di produzione. Per ogni pezzo ispezionato, il sistema ha registrato l'etichetta **reale** (ottenuta da un esperto) e l'etichetta **prevista** dal classificatore, dove `1` rappresenta "difettoso" e `0` rappresenta "conforme".

La direzione qualità vuole conoscere non solo l'accuratezza del sistema, ma anche la sua **precisione** (quando il sistema segnala un difetto, con quale frequenza ha ragione?) e il suo **richiamo** (di tutti i pezzi realmente difettosi, quanti il sistema è riuscito a identificare?) — la distinzione discussa nella sezione sulla valutazione dei classificatori del capitolo.

7.18.3.1 📋 Linee Guida di Implementazione

1. **Quantità:** Leggere l'intero N (numero di pezzi ispezionati).
2. **Dati di ciascun pezzo:** Per ciascuno degli N pezzi, leggere due interi — l'etichetta reale y e l'etichetta prevista $\hat{y}$ (entrambe $\in \{0, 1\}$).
3. **Matrice di confusione:** Considerando la classe `1` (difettoso) come **positiva**, contare:
   - VP (Vero Positivo): $y = 1$ e $\hat{y} = 1$;
   - FP (Falso Positivo): $y = 0$ e $\hat{y} = 1$;
   - FN (Falso Negativo): $y = 1$ e $\hat{y} = 0$;
   - VN (Vero Negativo): $y = 0$ e $\hat{y} = 0$.
4. **Metriche:** Calcolare
$$\text{Accuratezza} = \frac{VP + VN}{N}, \quad \text{Precisione} = \frac{VP}{VP + FP}, \quad \text{Richiamo} = \frac{VP}{VP + FN}.$$
5. **Casi degeneri:** Se $VP + FP = 0$ (nessuna previsione positiva), stampare `Precisao: indefinida`. Se $VP + FN = 0$ (nessun caso positivo reale), stampare `Revocacao: indefinida`.
6. **Arrotondamento:** Tutte le metriche numeriche devono essere arrotondate a 4 cifre decimali (*round half away from zero*) solo nella visualizzazione.

7.18.3.2 📌 Vincoli Computazionali

- **Convenzione di classe positiva fissa:** la classe `1` è sempre la classe positiva in questo esercizio, indipendentemente dalla sua frequenza relativa.
- **Protezione dalla divisione per zero:** implementare i casi degeneri del punto 5 prima di eseguire la divisione.
- **Ordine di uscita:** seguire esattamente l'ordine specificato nella sezione di uscita, anche nei casi degeneri.

7.18.3.3 🧠 Fondamento Teorico

Metrica	Domanda a cui risponde	Sensibile allo squilibrio?
Accuratezza	Quale frazione di pezzi è stata classificata correttamente?	Sì — può mascherare errori nella classe minoritaria
Precisione	Dei pezzi segnalati come difettosi, quanti lo sono realmente?	Penalizza i falsi positivi
Richiamo	Dei pezzi realmente difettosi, quanti sono stati rilevati?	Penalizza i falsi negativi

In un contesto industriale, un **richiamo** basso è spesso più grave di una **precisione** bassa: lasciar passare un pezzo difettoso (falso negativo) tende a essere più costoso che ispezionare manualmente un pezzo buono segnalato per errore (falso positivo).

7.18.3.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero N.
- Successive N righe: due interi per riga — y e $\hat{y}$, separati da spazio.

Output (in questo ordine esatto):

```
VP=<int> FP=<int> FN=<int> VN=<int>
Acuracia: <valore o metrica indefinita>
Precisao: <valore o indefinita>
Revocacao: <valore o indefinita>
```

7.18.3.5 📌 Esempi

Input	Output	Osservazione
4 1 1 0 1 1 0 0 0	VP=1 FP=1 FN=1 VN=1 Acuracia: 0.5000 Precisao: 0.5000 Revocacao: 0.5000	Un errore di ogni tipo.
3 0 0 0 0 0 0	VP=0 FP=0 FN=0 VN=3 Acuracia: 1.0000 Precisao: indefinida Revocacao: indefinida	Nessun caso positivo reale né previsto.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap07/sim-ep0703.html

Figura 7.23: Simulatore EP07_03: Precisione x Richiamo

```
%%writefile EP07_03.py
# Codice Python
```

```
Overwriting EP07_03.py
```

```
TestSuite("EP07_03.py").run()
```

```
<IPython.core.display.HTML object>
```

7.18.4 EP07_04 🟠 Codifica Manuale del Descrittore LBP

La funzione `local_binary_pattern` di `scikit-image`, utilizzata nel progetto di classificazione delle trame, calcola automaticamente il codice LBP di ogni pixel di un'immagine. Prima di utilizzarla come una scatola nera, ti è stato affidato il compito di implementare manualmente il calcolo del codice LBP classico ($P = 8$, $R = 1$) per il pixel centrale di un intorno 3×3, esattamente come definito nell'equazione del capitolo.

Oltre al codice, il sistema di ispezione delle trame deve anche sapere se quel pattern è **uniforme** — un pattern è uniforme quando il numero di transizioni ($0 \to 1$ o $1 \to 0$) percorrendo gli 8 bit **circolarmente** (tornando dall'ultimo bit al primo) è **al massimo 2**, proprietà sfruttata dalla variante *uniforme* del LBP menzionata nel capitolo.

7.18.4.1 📋 Linee Guida di Implementazione

1. **Quantità:** Leggere l'intero T (numero di intorni da elaborare).
2. **Dati di ogni intorno:** Per ciascuno dei T intorni, leggere una matrice 3×3 di interi (intensità), fornita in 3 righe di 3 valori ciascuna. Il pixel centrale è la posizione `[1][1]`.
3. **Ordine dei vicini:** Percorrere gli 8 vicini in senso **orario**, partendo dall'angolo in alto a sinistra, nel seguente ordine di posizioni `[riga][colonna]`: `[0][0]`, `[0][1]`, `[0][2]`, `[1][2]`, `[2][2]`, `[2][1]`, `[2][0]`, `[1][0]`. Questo è l'indice $p = 0, 1, \dots, 7$ dell'equazione del LBP.
4. **Funzione soglia:** Per ogni vicino p con intensità g_p e centro g_c, calcolare $s(g_p - g_c)$, che vale `1` se $g_p \geq g_c$ e `0` altrimenti.
5. **Codice LBP:** Calcolare
$$\text{LBP} = \sum_{p=0}^{7} s(g_p - g_c)\, 2^p.$$
6. **Transizioni:** Considerando la sequenza circolare di bit $s_0, s_1, \dots, s_7$ (nell'ordine del punto 3), contare quante coppie consecutive **adiacenti nella sequenza circolare** (inclusa la coppia s_7, s_0) differiscono tra loro.
7. **Classificazione:** Se il numero di transizioni è ≤ 2, classificare come `UNIFORME`; altrimenti, `NAO_UNIFORME`.
8. **Uscita:** Per ogni intorno, nell'ordine di ingresso, stampare il codice LBP (intero decimale, 0–255), il numero di transizioni e la classificazione.

7.18.4.2 📌 Vincoli Computazionali

- **Ordine fisso dei vicini:** l'ordine del punto 3 è obbligatorio — invertirlo produce un codice numericamente diverso, anche se rappresenta lo stesso pattern visivo.
- **Confronto non stretto:** $s(z) = 1$ quando $z \geq 0$ (il capitolo stesso definisce l'uguaglianza come inclusa nel caso `1`).
- **Conteggio circolare:** non dimenticare la coppia che chiude il ciclo (s_7 con s_0); ignorare questa coppia è un errore comune che classifica erroneamente i pattern uniformi.

7.18.4.3 🧠 Fondamento Teorico

Pattern (bit $s_0 \dots s_7$)	Transizioni	Interpretazione
`00000000` o `11111111`	0	Regione omogenea (macchia chiara o scura)
`00001111`	2	Bordo semplice tra due regioni
`01010101`	8	Trama a contrasto alternato — non uniforme

I pattern uniformi si concentrano in regioni di trama liscia o bordi semplici; i pattern non uniformi tendono a corrispondere a rumore ad alta frequenza. Per questo motivo, l'istogramma LBP *uniforme*, utilizzato

nel progetto di classificazione delle trame, raggruppa tutti i pattern non uniformi in un unico contenitore, riducendo la dimensionalità del descrittore.

7.18.4.4 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

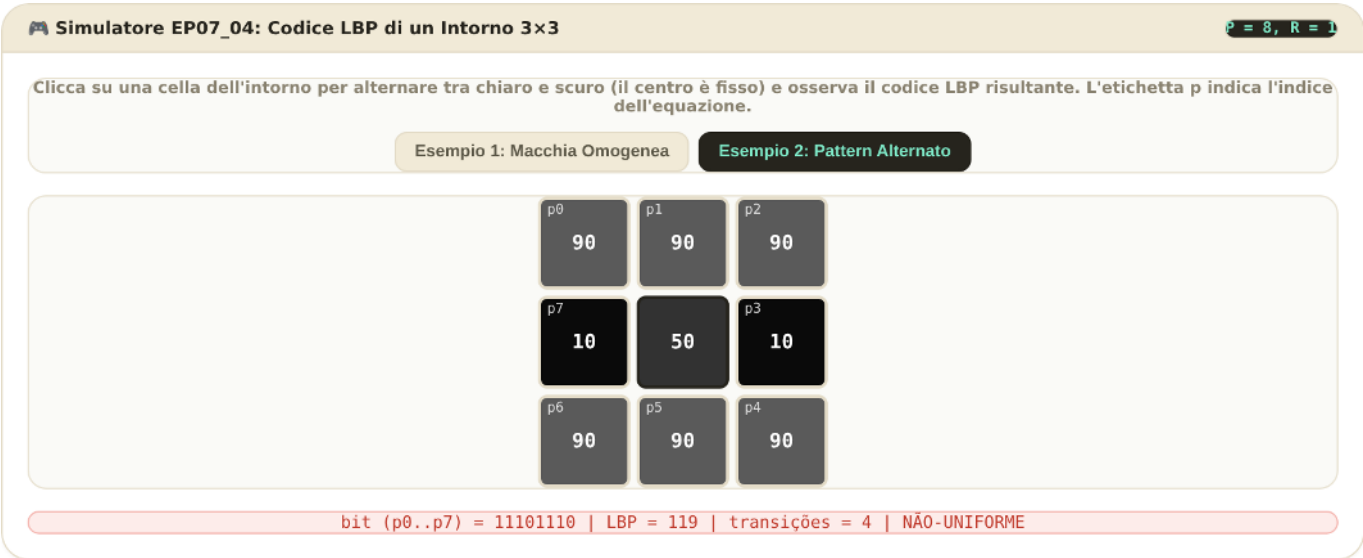
- Riga 1: Intero T.
- Per ogni intorno: 3 righe con 3 interi ciascuna (matrice 3×3).

Uscita:

- T righe, nel formato `LBP=<int> transicoes=<int> <UNIFORME|NAO_UNIFORME>`.

7.18.4.5 📌 Esempi

Ingresso	Uscita	Osservazione
1 10 10 10 10 50 10 10 10 10	LBP=0 transicoes=0 UNIFORME	Centro è il più chiaro; tutti i vicini generano bit 0.
1 90 90 90 10 50 10 90 90 90	LBP=119 transicoes=4 NAO_UNIFORME	Vicini chiari e scuri alternati nell'intorno.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap07/sim-ep0704.html

Figura 7.24: Simulatore EP07_04: Codice LBP di un vicinato 3×3

```
%%writefile EP07_04.py
# Codice Python
```

```
Overwriting EP07_04.py
```

```
TestSuite("EP07_04.py").run()
```

```
<IPython.core.display.HTML object>
```

7.18.5 EP07_05 🔴 Istogramma delle Orientazioni di una Cella HOG

La funzione `hog` di `scikit-image`, impiegata nel progetto di classificazione delle cifre, divide l'immagine in piccole **celle** e, per ciascuna, costruisce un istogramma delle orientazioni del gradiente ponderato per la magnitudine — esattamente il passaggio centrale descritto nella sezione sul descrittore HOG del capitolo.

Sei stato incaricato di implementare questo calcolo per una singola cella, a partire dai valori di magnitudine e orientazione del gradiente **già calcolati** per ogni pixel della cella (tralasciando il calcolo delle derivate parziali).

7.18.5.1 📋 Linee Guida di Implementazione

1. **Dimensioni:** Leggere gli interi n (la cella ha $n \times n$ pixel) e B (numero di contenitori dell'istogramma).
2. **Magnitudini:** Leggere n righe con n valori reali ciascuna, che rappresentano $|\nabla f(x, y)|$ per ogni pixel della cella.
3. **Orientazioni:** Leggere altre n righe con n valori reali ciascuna, che rappresentano $\theta(x, y)$ in **gradi**, già convertiti nell'intervallo **non orientato** $[0^\circ, 180^\circ)$, come convenzionalmente utilizzato dal HOG.
4. **Contenitori:** I B contenitori coprono $[0^\circ, 180^\circ)$ in fasce uguali di larghezza $180/B$ gradi. Un pixel con orientazione θ appartiene al contenitore $\lfloor \theta/(180/B) \rfloor$; se questo indice è uguale a B (possibile solo quando θ è esattamente 180°, cosa che non dovrebbe verificarsi secondo la direttiva del punto 3), utilizzare il contenitore $B - 1$.
5. **Istogramma grezzo:** Per ogni pixel, accumulare la sua **magnitudine** (non il suo conteggio) nel contenitore corrispondente:
$$H[b] = \sum_{(x,y)\,:\,\mathrm{bin}(\theta(x,y))=b} |\nabla f(x, y)|.$$
6. **Normalizzazione L2:** Dopo aver costruito H, normalizzarlo per ottenere $\hat{H}$:
$$\hat{H}[b] = \frac{H[b]}{\sqrt{\sum_{j=0}^{B-1} H[j]^2 + \epsilon}}, \qquad \epsilon = 10^{-6}.$$
7. **Output:** Stampare l'istogramma grezzo H (arrotondato a 2 cifre decimali) su una riga, seguito dall'istogramma normalizzato $\hat{H}$ (arrotondato a 4 cifre decimali) su un'altra riga, entrambi con i B valori separati da spazi, nell'ordine dei contenitori.

7.18.5.2 📌 Vincoli Computazionali

- ***Binning* non orientato:** l'intervallo delle orientazioni è $[0, 180)$, non $[0, 360)$ — i gradienti in direzioni opposte (differenza di 180°) contribuiscono allo **stesso** contenitore, convenzione standard del HOG per il rilevamento di oggetti.
- **Accumulo per magnitudine, non per conteggio:** l'istogramma pondera ogni pixel per la sua magnitudine del gradiente, non conta semplicemente quanti pixel cadono in ciascun contenitore.
- **Costante di stabilizzazione:** l'$\epsilon = 10^{-6}$ al denominatore della normalizzazione evita la divisione per zero quando la cella è completamente omogenea (tutte le magnitudini nulle).

7.18.5.3 📐 Da dove provengono le matrici di ingresso

Prima di questo EP, ogni pixel (x, y) dell'immagine passa attraverso:

$$G_x = f(x + 1, y) - f(x - 1, y), \qquad G_y = f(x, y + 1) - f(x, y - 1)$$

$$|\nabla f| = \sqrt{G_x^2 + G_y^2}, \qquad \theta_{\text{segnalato}} = \mathrm{atan2}(G_y, G_x)$$

Poiché il HOG ignora la polarità del contrasto, l'angolo viene raddoppiato nell'intervallo non orientato:

$$\theta = \theta_{\text{segnalato}} \bmod 180^\circ$$

Ripetendo questo per tutti i pixel di una cella $n \times n$, si ottengono le due matrici di ingresso di questo esercizio: **magnitudini** $|\nabla f|$ e **orientazioni** $\theta \in [0^\circ, 180^\circ)$.

7.18.5.4 🧠 Fondamenti Teorici

Fase	Ruolo
Magnitudine del gradiente	Pondera il contributo di ogni pixel — i bordi forti pesano più del rumore debole
Orientazione non orientata	Rende il descrittore invariante alla polarità del contrasto (chiaro→scuro vs. scuro→chiaro)
Istogramma per cella	Riassume la distribuzione locale dei bordi in un vettore compatto
Normalizzazione L2	Riduce la sensibilità del descrittore alle variazioni globali di illuminazione e contrasto

La concatenazione degli istogrammi normalizzati di tutte le celle dell'immagine — non implementata in questo esercizio — forma il vettore di caratteristiche HOG completo, utilizzato come ingresso del classificatore k-NN nel progetto del capitolo.

7.18.5.5 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Interi n e B.
- Prossime n righe: n magnitudini reali ciascuna.
- Prossime n righe: n orientazioni reali (gradi, $[0, 180)$) ciascuna.

Uscita:

- Riga 1: i B valori dell'istogramma grezzo, arrotondati a 2 cifre decimali.
- Riga 2: i B valori dell'istogramma normalizzato, arrotondati a 4 cifre decimali.

7.18.5.6 📌 Esempi

Ingresso	Uscita	Osservazione
2 2 1.0 2.0 3.0 4.0 10 100 170 20	5.00 5.00 0.7071 0.7071	Contenitore di larghezza 90°: $[0, 90)$ e $[90, 180)$; le magnitudini 1 e 4 cadono nel contenitore 0, 2 e 3 nel contenitore 1.
2 4 0.0 0.0 0.0 0.0 0 0 0 0	0.00 0.00 0.00 0.00 0.0000 0.0000 0.0000 0.0000	Cella omogenea: ϵ evita la divisione per zero.

```
%%writefile EP07_05.py
# Codice Python
```

🎮 **Simulatore EP07_05: Istogramma delle orientazioni di una cella** — cella 3×3 fissa

Regola B e osserva come la **matrice delle orientazioni** (indipendente da quella delle magnitudini) viene mappata nei contenitori tramite `bin = floor(θ / (180/B))`, e come le magnitudini vengono sommate in ciascun bin.

Numero di contenitori (B) 2

📄 **Input (esattamente come il programma legge da stdin)**

```
3 2
1.0 2.0 0.5
3.0 4.0 1.5
0.8 2.5 3.2
10 100 45
170 20 95
60 150 5
```

🔢 **Matrice delle magnitudini |∇f|**

1.0	2.0	0.5
3.0	4.0	1.5
0.8	2.5	3.2

📐 **Matrice delle orientazioni θ (gradi) — colorata per bin**

10° ⌊·⌋=0.11 ⌊·⌋=bin 0	100° ⌊·⌋=1.11 ⌊·⌋=bin 1	45° ⌊·⌋=0.50 ⌊·⌋=bin 0
170° ⌊·⌋=1.89 ⌊·⌋=bin 1	20° ⌊·⌋=0.22 ⌊·⌋=bin 0	95° ⌊·⌋=1.06 ⌊·⌋=bin 1
60° ⌊·⌋=0.67 ⌊·⌋=bin 0	150° ⌊·⌋=1.67 ⌊·⌋=bin 1	5° ⌊·⌋=0.06 ⌊·⌋=bin 0

📏 **Dove ciascun θ cade sul righello [0°, 180°) — bin = floor(θ / larghezza)**

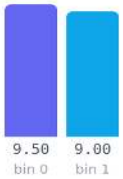
📊 **Intervalli di ciascun contenitore (larghezza = 180° / B)**

- bin 0: [0.0°, 90.0°)
- bin 1: [90.0°, 180.0°)

🧩 **Ogni pixel: magnitudine + orientazione → bin**

mag=1.0 θ=10° → bin 0	mag=2.0 θ=100° → bin 1	mag=0.5 θ=45° → bin 0
mag=3.0 θ=170° → bin 1	mag=4.0 θ=20° → bin 0	mag=1.5 θ=95° → bin 1
mag=0.8 θ=60° → bin 0	mag=2.5 θ=150° → bin 1	mag=3.2 θ=5° → bin 0

Istogramma grezzo (H) | Istogramma normalizzato (Ĥ)



🧮 **Calcolo passo passo (floor della divisione + somma delle magnitudini per bin)**

H[0] = 1.00 (θ=10°→⌊0.11⌋=0) + 0.50 (θ=45°→⌊0.50⌋=0) + 4.00 (θ=20°→⌊0.22⌋=0) + 0.80 (θ=60°→⌊0.67⌋=0) + 3.20 (θ=5°→⌊0.06⌋=0) = **9.50**

H[1] = 2.00 (θ=100°→⌊1.11⌋=1) + 3.00 (θ=170°→⌊1.89⌋=1) + 1.50 (θ=95°→⌊1.06⌋=1) + 2.50 (θ=150°→⌊1.67⌋=1) = **9.00**

H=[9.50, 9.00] | Ĥ=[0.7260, 0.6877]

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap07/sim-ep0705.html

Figura 7.25: Simulatore EP07_05: Istogramma HOG di una Cella (mappatura degli angoli ai bins)

```
Overwriting EP07_05.py
```

```
TestSuite("EP07_05.py").run()
```

```
<IPython.core.display.HTML object>
```

7.18.6 EP07_06 🟣 *Pipeline* Completo: Descrittori + k-NN + Valutazione Multi-Classe

Questo esercizio integra le tre fasi centrali del capitolo in un unico *pipeline*, riproducendo in miniatura il **Progetto Pratico 2** (classificazione di trame sintetiche tramite LBP): un insieme di istogrammi di descrittori **già estratti** (come se fossero istogrammi LBP) viene utilizzato per addestrare un classificatore k-NN, che a sua volta viene valutato su un insieme di test indipendente mediante una matrice di confusione multi-classe.

A differenza dell'EP07_01, qui lo spazio delle caratteristiche ha dimensione arbitraria H (la dimensione dell'istogramma), esistono più di due classi e la metrica di distanza è un parametro di input — consentendo di riprodurre l'esperimento di confronto delle metriche discusso nel capitolo.

7.18.6.1 📋 Linee Guida di Implementazione

1. **Classi:** Leggere l'intero C (numero di classi) seguito da C nomi di classe (*stringhe* senza spazi), nell'ordine in cui devono apparire nella matrice di confusione.
2. **Configurazione:** Leggere l'intero H (dimensione degli istogrammi), la *stringa* M (metrica: `euclidiana` o `manhattan`) e l'intero dispari k.
3. **Addestramento:** Leggere l'intero N e, successivamente, N righe, ciascuna contenente il nome della classe seguito da H valori reali (l'istogramma del descrittore).
4. **Test:** Leggere l'intero Q e, successivamente, Q righe, ciascuna contenente il nome della classe **reale** seguito da H valori reali (l'istogramma del descrittore del campione di test).
5. **Distanza:** Per ogni campione di test, calcolare la distanza da ogni esempio di addestramento utilizzando la metrica M:

$$d_{\text{euclidiana}}(u, v) = \sqrt{\sum_{j=1}^{H} (u_j - v_j)^2}, \qquad d_{\text{manhattan}}(u, v) = \sum_{j=1}^{H} |u_j - v_j|.$$

6. **Classificazione k-NN:** Selezionare i k esempi di addestramento più vicini (in caso di parità di distanza, dare la precedenza all'ordine di lettura, come nell'EP07_01) e classificare in base alla classe maggioritaria tra di essi. In caso di **pareggio di voti** tra due o più classi, scegliere quella che appare **per prima** nella lista di classi del punto 1.
7. **Matrice di confusione:** Costruire una matrice $C \times C$ in cui la riga corrisponde alla classe reale e la colonna alla classe prevista, seguendo l'ordine delle classi del punto 1.
8. **Accuratezza:** Calcolare l'accuratezza globale come rapporto tra i successi e Q.
9. **Output:** Per ogni campione di test, nell'ordine di input, stampare la classe prevista. Successivamente, stampare la matrice di confusione (una riga per classe reale, valori separati da spazi, nell'ordine delle classi). Infine, stampare l'accuratezza arrotondata a 4 cifre decimali.

7.18.6.2 📌 Vincoli Computazionali

- **Metrica selezionabile:** implementare entrambe le distanze; la metrica M definisce quale viene utilizzata per l'intera esecuzione (non è possibile mescolare metriche nella stessa chiamata).
- **Pareggio di voti deterministico:** il criterio del punto 6 (ordine della lista delle classi) deve essere seguito anche quando il pareggio coinvolge più di due classi.
- **Indipendenza tra addestramento e test:** non è necessario verificare che i campioni di test non appaiano nell'addestramento — si assume che l'input sia valido.

7.18.6.3 🧠 Fondamenti Teorici

Fase dell'esercizio	Fase corrispondente nel capitolo
Istogrammi di addestramento/test già estratti	`descritor_lbp` applicato alle trame sintetiche
Distanza euclidea o Manhattan	Parametro `metric` del `KNeighborsClassifier`
Votazione maggioritaria con k vicini	`KNeighborsClassifier.predict`
Matrice di confusione $C \times C$	`confusion_matrix` di `scikit-learn`
Accuratezza globale	`accuracy_score` di `scikit-learn`

Questo esercizio evidenzia, in modo controllato, un risultato discusso nel capitolo: la **scelta della metrica di distanza** e del **valore di** k può modificare la classe prevista per lo stesso campione, anche mantenendo fisso il descrittore utilizzato — rafforzando l'idea che, nel riconoscimento di pattern classico, il descrittore, la metrica e il classificatore formano un sistema interdipendente, e non componenti isolate.

7.18.6.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: intero C seguito da C nomi di classe.
- Riga 2: intero H, *stringa* M e intero k.
- Riga 3: intero N.
- Prossime N righe di addestramento: nome della classe seguito da H reali.
- Riga successiva: intero Q.
- Prossime Q righe di test: nome della classe reale seguito da H reali.

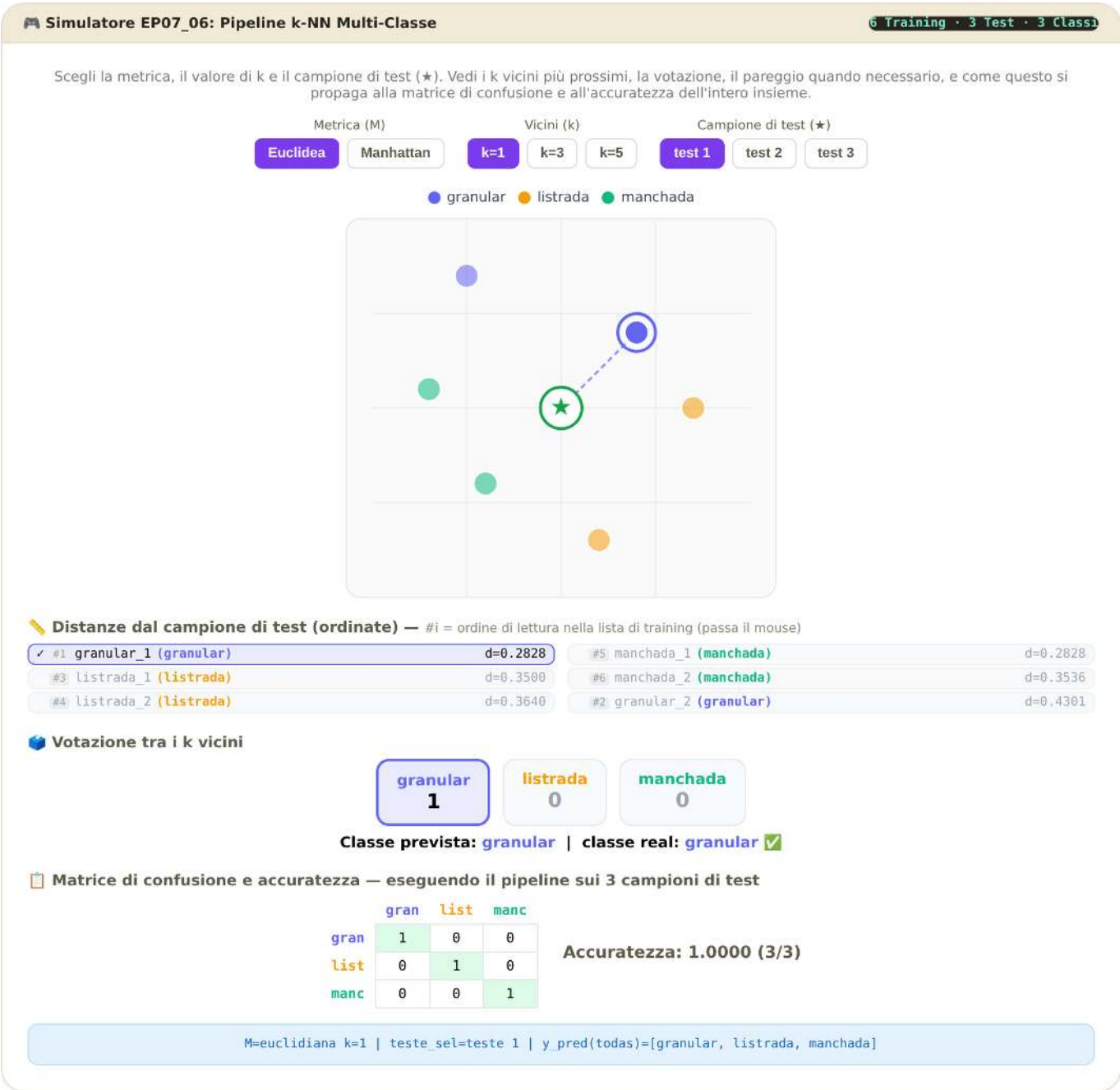
Output:

- Q righe con la classe prevista per ogni campione di test, nell'ordine di input.
- C righe con la matrice di confusione (una riga per classe reale).
- Ultima riga: `Acuracia: <valore>`.

7.18.6.5 📌 Esempi

Input (riassunto)	Output	Osservazione
2 granular listrada 2 euclidiana 1 4 granular 0.9 0.1 granular 0.8 0.2 listrada 0.1 0.9 listrada 0.2 0.8 2 granular 0.85 0.15 listrada 0.15 0.85	granular listrada 1 0 0 1 Acuracia: 1.0000	Con $k = 1$, ogni test viene classificato dal vicino di addestramento più prossimo.

> **Nota**
>
> Questo simulatore utilizza un insieme semplificato di **3 classi** (`granulare`, `a strisce`, `maculata`) su punti 2D fittizi, unicamente per illustrare il *pipeline* di votazione, spareggio e matrice di confusione del k-NN. Nel **EP07_07**, applicherai questa stessa logica a un mosaico di un'immagine reale, che introduce una quarta classe (`a scacchi`) e sostituisce i punti 2D con istogrammi LBP estratti direttamente dai pixel dell'immagine.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap07/sim-ep0706.html

Figura 7.26: Simulatore EP07_06: *Pipeline* k-NN Multi-Classe (votazione, spareggio e matrice di confusione)

```
%%writefile EP07_06.py
# Codice Python
```

```
Overwriting EP07_06.py
```

```
TestSuite("EP07_06.py").run()
```

```
<IPython.core.display.HTML object>
```

7.18.7 EP07_07 ⚫ Classificazione Reale di un Mosaico di Trame tramite LBP + k-NN

Negli esercizi precedenti, il descrittore LBP (**EP07_04**) e il classificatore k-NN multiclasse (**EP07_06**) sono stati studiati separatamente, sempre a partire da dati già forniti in input — vicinanze 3×3 isolate o istogrammi precedentemente estratti. In questo esercizio conclusivo del capitolo, il programma dovrà **leggere un'immagine reale**, nel formato **PGM ASCII (P2)**, calcolare il descrittore LBP direttamente dai pixel e, successivamente, classificare ogni regione tramite il k-NN, riproducendo, in scala ridotta, il flusso completo di un sistema di riconoscimento delle trame. Questo approccio anticipa anche l'idea di **classificazione per mosaico di regioni**, legata alla segmentazione semantica studiata in un capitolo successivo.

Il simulatore interattivo dell'**EP07_06** utilizzava solo tre classi (`granulare`, `a righe` e `maculata`) rappresentate da punti bidimensionali fittizi. In questo esercizio, si aggiunge una quarta classe, **a scacchi**, e i punti vengono sostituiti da istogrammi LBP estratti da un'immagine reale.

L'immagine di input è un **mosaico** formato da una griglia $G \times G$ di blocchi quadrati di $S \times S$ pixel. Ogni blocco contiene un campione di una delle quattro classi di trama sintetica del capitolo: **granulare**, **a righe**, **maculata** o **a scacchi** (motivo a scacchiera con intensità alternate). Come negli altri esercizi del libro, il caricamento dell'immagine viene eseguito dalla funzione didattica `mm.readImg`.

> 💡 Perché un mosaico unico, e non più immagini?
>
> L'input riunisce i $G \times G$ campioni di trama in un unico file **PGM**, solo per semplificare la lettura dei dati ed evitare l'apertura di più file. Per l'algoritmo, ciò non modifica l'elaborazione: ogni blocco viene trattato in modo indipendente, come se fosse un'immagine isolata. L'unica eccezione è l'**esclusione del bordo** (punto 4 di seguito).

7.18.7.1 📋 Linee Guida di Implementazione

1. **Lettura delle dimensioni dell'immagine**

   Leggere, tramite l'input standard, due righe contenenti, rispettivamente, il numero di righe L e il numero di colonne C del mosaico (entrambi multipli della dimensione del blocco S, con $L = C$).

2. **Caricamento dell'immagine**

   Utilizzare la funzione didattica

   ```
   f = mm.readImg(L, C)
   ```

   per leggere i valori di intensità $L \times C$ (toni di grigio, `uint8`) del mosaico.

3. **Parametri della griglia**

   Leggere l'intero G (numero di blocchi per lato) e l'intero S (dimensione del lato di ogni blocco, in pixel), soddisfacendo $L = C = G \times S$.

4. **Calcolo del codice LBP per pixel**

   Per ogni pixel **interno** dell'immagine (cioè che non si trova sul bordo globale di `f` — riga o colonna 0 o $L - 1/C - 1$), calcolare il codice LBP con $P = 8$ vicini e raggio $R = 1$, percorrendo i vicini in senso

orario a partire dall'angolo superiore sinistro, esattamente come nell'EP07_04: `[riga-1][colonna-1]`, `[riga-1][colonna]`, `[riga-1][colonna+1]`, `[riga][colonna+1]`, `[riga+1][colonna+1]`, `[riga+1][colonna]`, `[riga+1][colonna-1]`, `[riga][colonna-1]`.

I pixel sul bordo globale dell'immagine **non** hanno una vicinanza completa e devono essere **ignorati** (non contribuiscono a nessun istogramma). Questo include i pixel di bordo che cadono all'interno di un blocco (l'esclusione è sempre relativa al bordo dell'intera immagine, non al bordo di ogni singolo blocco).

5. **Istogramma LBP uniforme per blocco (10 contenitori)**

   Per ogni blocco (i, j) della griglia $(i, j = 0, \dots, G-1)$, accumulare, tra i suoi pixel validi (punto 4), un istogramma $H^{(i,j)}$ di 10 contenitori:

   - Considerando la sequenza circolare di bit $s_0, \dots, s_7$ del pixel (stessa regola di transizioni dell'EP07_04): se il numero di transizioni è ≤ 2 (pattern **uniforme**), il pixel contribuisce al contenitore $\text{popcount}(s_0, \dots, s_7) \in \{0, \dots, 8\}$ (numero di bit uguali a `1`);
   - In caso contrario (pattern **non uniforme**), il pixel contribuisce al contenitore 9.

   Alla fine, normalizzare l'istogramma di ogni blocco dividendolo per il numero di pixel validi in esso contenuti, ottenendo $\hat{H}^{(i,j)}$, con $\sum_{b=0}^{9} \hat{H}^{(i,j)}[b] = 1$.

6. **Prototipi di addestramento**

   Leggere l'intero Ncl (numero di classi) seguito da Ncl nomi di classe (ordine che definisce la matrice di confusione e il criterio di parità per la votazione, come nell'EP07_06); successivamente, leggere la stringa M (metrica: `euclidiana` o `manhattan`) e l'intero dispari k; infine, leggere l'intero N (numero di prototipi) e, per ciascuno, il nome della classe seguito da 10 valori reali (istogramma prototipo già normalizzato).

7. **Classificazione k-NN di ogni blocco**

   Per ogni blocco, calcolare la distanza di $\hat{H}^{(i,j)}$ da ciascuno degli N prototipi, usando la metrica M (stesse formule dell'EP07_06). Selezionare i k prototipi più vicini (criterio di parità per la distanza basato sull'ordine di lettura dei prototipi) e classificare tramite la classe maggioritaria (criterio di parità per la votazione basato sull'ordine delle classi del punto 6).

8. **Etichette reali e valutazione**

   Leggere, in un'unica riga, i $G \times G$ nomi di classe **reali** di ogni blocco, in ordine di lettura per riga della griglia (blocco $(0, 0)$, $(0, 1)$, ..., $(0, G-1)$, $(1, 0)$, ...). Costruire la matrice di confusione $Ncl \times Ncl$ (riga = classe reale, colonna = classe prevista) e calcolare l'accuratezza globale.

9. **Output**

   Stampare, per ogni blocco (nello stesso ordine di lettura delle etichette reali del punto 8), la classe prevista. Successivamente, stampare la matrice di confusione (una riga per classe reale, nell'ordine del punto 6). Infine, stampare l'accuratezza, arrotondata a 4 cifre decimali.

7.18.7.2 📌 Vincoli Computazionali

- **Descrittore fisso:** $P = 8$, $R = 1$ e 10 contenitori (come da punto 5) sono fissi in questo esercizio — non vengono letti dall'input.
- **Esclusione del bordo globale, non del blocco:** un pixel sul confine tra due blocchi, ma all'interno dell'immagine, è valido e contribuisce normalmente all'istogramma del blocco a cui appartiene.
- **Ordine di lettura come criterio di parità:** sia il criterio di parità per la distanza (punto 7) che quello per la votazione (punto 7) seguono esattamente le stesse convenzioni dell'EP07_01 e dell'EP07_06.
- **Prototipi come input, non appresi:** a differenza del Progetto Pratico 2, gli istogrammi di addestramento vengono forniti direttamente nell'input; il programma non deve generare trame sintetiche.

7.18.7.3 🧠 Fondamenti Teorici

Fase dell'esercizio	Fase corrispondente nel capitolo
Lettura dell'immagine tramite `mm.readImg`	Acquisizione dell'immagine nel *pipeline* di riconoscimento di pattern
Codice LBP per pixel (EP07_04)	`local_binary_pattern(immagine, P=8, R=1, method="uniform")`
Istogramma di 10 contenitori per blocco	Funzione `descrittor_lbp` del Progetto Pratico 2 (`bins=10`, `range=(0, P+2)`)
Classificazione k-NN con metrica selezionabile (EP07_06)	`KNeighborsClassifier` addestrato su `X_texture`
Matrice di confusione $Ncl \times Ncl$ e accuratezza	`confusion_matrix` e `accuracy_score` su `yt_test`

Questo esercizio evidenzia, con pixel reali invece di valori sintetici, una limitazione discussa nella sezione finale del capitolo: classi di trama visivamente distinte per un osservatore umano — come **granulare** e **maculata** — possono produrre istogrammi LBP simili quando la vicinanza considerata è piccola ($R = 1$), poiché entrambe presentano un'alta frequenza di pattern non uniformi alla scala di un singolo pixel. La classe **a scacchi**, invece, avendo bordi regolari e ripetitivi, tende a essere separata con maggiore facilità. Ci si aspetta che la matrice di confusione prodotta rifletta esattamente questo pattern di confusione parziale.

7.18.7.4 📦 Specifica di Input e Output (VPL)

Input:

```
L
C
[matrice L x C dell'immagine]
G S
Ncl nome_classe_1 ... nome_classe_Ncl
M k
N
nome_classe h0 h1 ... h9      (ripetuta N volte)
etichetta(0,0) etichetta(0,1) ... etichetta(G-1,G-1)
```

Output:

- $G \times G$ righe con la classe prevista per ogni blocco, nell'ordine di lettura della griglia.
- Ncl righe con la matrice di confusione (una riga per classe reale, valori separati da spazi).
- Ultima riga: `Acuracia: <valore>`.

7.18.7.5 📌 Esempio (verifica manuale)

Per verificare l'implementazione del descrittore prima di testarla su un mosaico completo, si consideri un'immagine 6×6 **omogenea**, con tutti i pixel di intensità 100, trattata come un unico blocco ($G = 1$, $S = 6$). Poiché ogni pixel interno ha gli 8 vicini con intensità uguale a quella del centro ($g_p \geq g_c$ in tutti i casi), tutti i bit s_p valgono `1`, il numero di transizioni è 0 (uniforme) e il contenitore è popcount(11111111) = 8. L'istogramma dell'unico blocco è, quindi, `0 0 0 0 0 0 0 0 1 0`.

Input (riepilogo)	Output	Osservazione
6 6 [36 valori uguali a 100] 1 6 2 uniforme altro euclidiana 1 2 uniforme 0 0 0 0 0 0 0 0 1 0 altro 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 uniforme	uniforme 1 0 0 0 Acuracia: 1.0000	La distanza del blocco al prototipo `uniforme` è esattamente 0; la classe `altro` non appare nell'etichetta reale, quindi la sua riga nella matrice di confusione è nulla.

7.18.7.6 📌 File di Riferimento (.pgm)

Per il debug locale, due mosaici di test nel formato ASCII P2 sono messi a disposizione (allegati a questa consegna; quando li si integra nel repository del capitolo, salvarli in `all/cap07/dati/EP07/`):

- 📥 **Caso 1 — Mosaico semplice (`Caso1_Mosaico_Simples.pgm`)**: griglia 2×2 di blocchi di 24×24 pixel, un campione di ciascuna delle quattro classi, con basso rumore — utile per validare la lettura dell'immagine e la logica di classificazione in uno scenario controllato.
- 📥 **Caso 2 — Mosaico misto (`Caso2_Mosaico_Misto.pgm`)**: griglia 3×3 di blocchi di 16×16 pixel, con classi ripetute e maggiore variabilità — scenario in cui la confusione tra **granulare** e **maculata** discussa nei Fondamenti Teorici tende a manifestarsi.

La Figura 7.27 mostra i due mosaici, per un'ispezione visiva prima dell'implementazione.

```
import os
import urllib.request
import numpy as np

def garantir_e_baixar_arquivo(nome_arquivo):
    diretorio_local = "dados/EP07"
    caminho_local = os.path.join(diretorio_local, nome_arquivo)

    # Creare la directory locale se non esiste
    if not os.path.exists(diretorio_local):
        os.makedirs(diretorio_local)

    # Se il file non esiste localmente, lo scarica dal repository remoto
    if not os.path.exists(caminho_local):
        url_base = "https://raw.githubusercontent.com/fzampirolli/"
        url_base += "pdi-vc/master/all/cap07/dados/EP07"
        url_arquivo = f"{url_base}/{nome_arquivo}"
        print(f"Scaricamento {nome_arquivo} da GitHub...")
        try:
            urllib.request.urlretrieve(url_arquivo, caminho_local)
        except Exception as e:
            raise IOError(f"Erro ao baixar {nome_arquivo} do GitHub. ",
                          "Verifique a conexão ou a URL. Detalhes: {e}")

    return caminho_local

def ler_pgm_p2(caminho):
    with open(caminho) as f:
        linhas = [l for l in f.read().split() if l]
    assert linhas[0] == "P2"
    C, L = int(linhas[1]), int(linhas[2])
```

```
    maxv = int(linhas[3])
    valores = list(map(int, linhas[4:4 + L * C]))
    return np.array(valores, dtype=np.uint8).reshape(L, C)

# Garantisce il download e ottiene il percorso corretto
arq_caso1 = garantir_e_baixar_arquivo("Caso1_Mosaico_Simples.pgm")
arq_caso2 = garantir_e_baixar_arquivo("Caso2_Mosaico_Misto.pgm")

# Legge le matrici PGM
caso1 = ler_pgm_p2(arq_caso1)
caso2 = ler_pgm_p2(arq_caso2)

mm.show(
    [caso1, caso2],
    titles=[
        "Caso 1: Mosaico Semplice\n(blocchi 2x2, 1 campione/classe)",
        "Caso 2: Mosaico Misto\n(blocchi 3x3, classi ripetute)",
    ],
    cols=2,
    figsize=(8, 4),
)
```

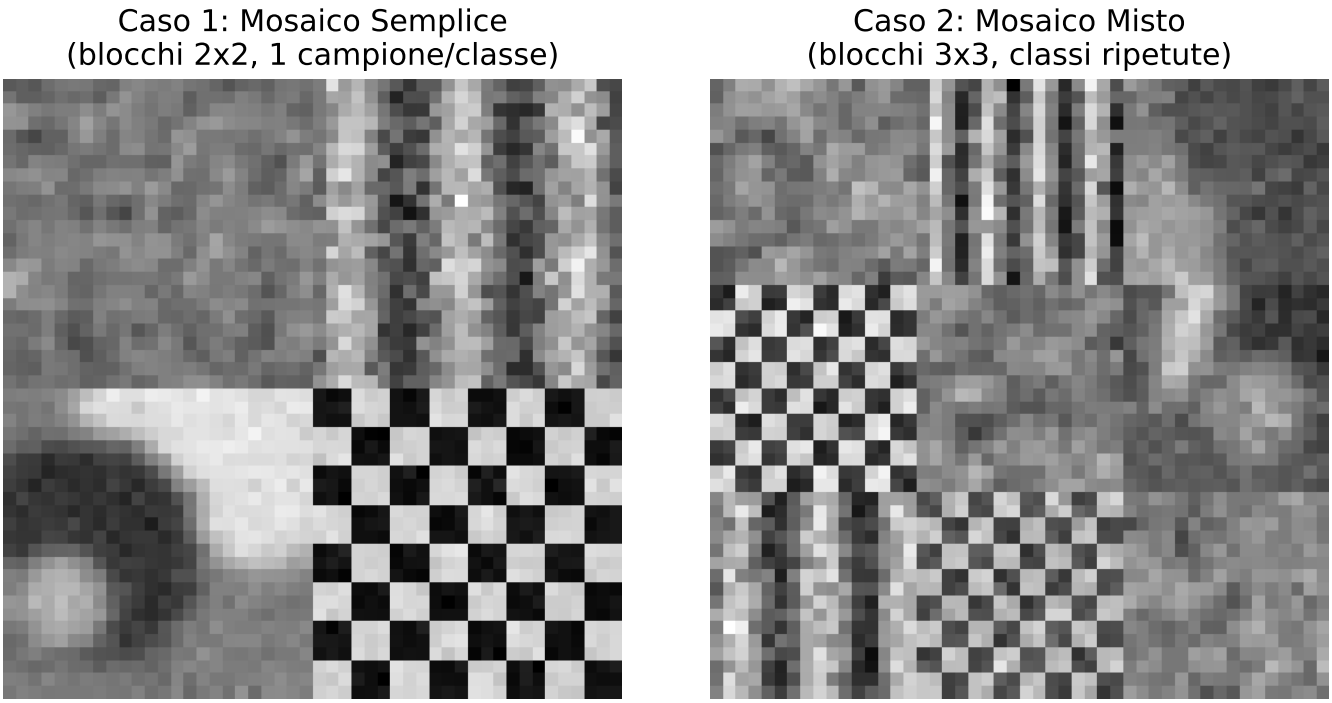


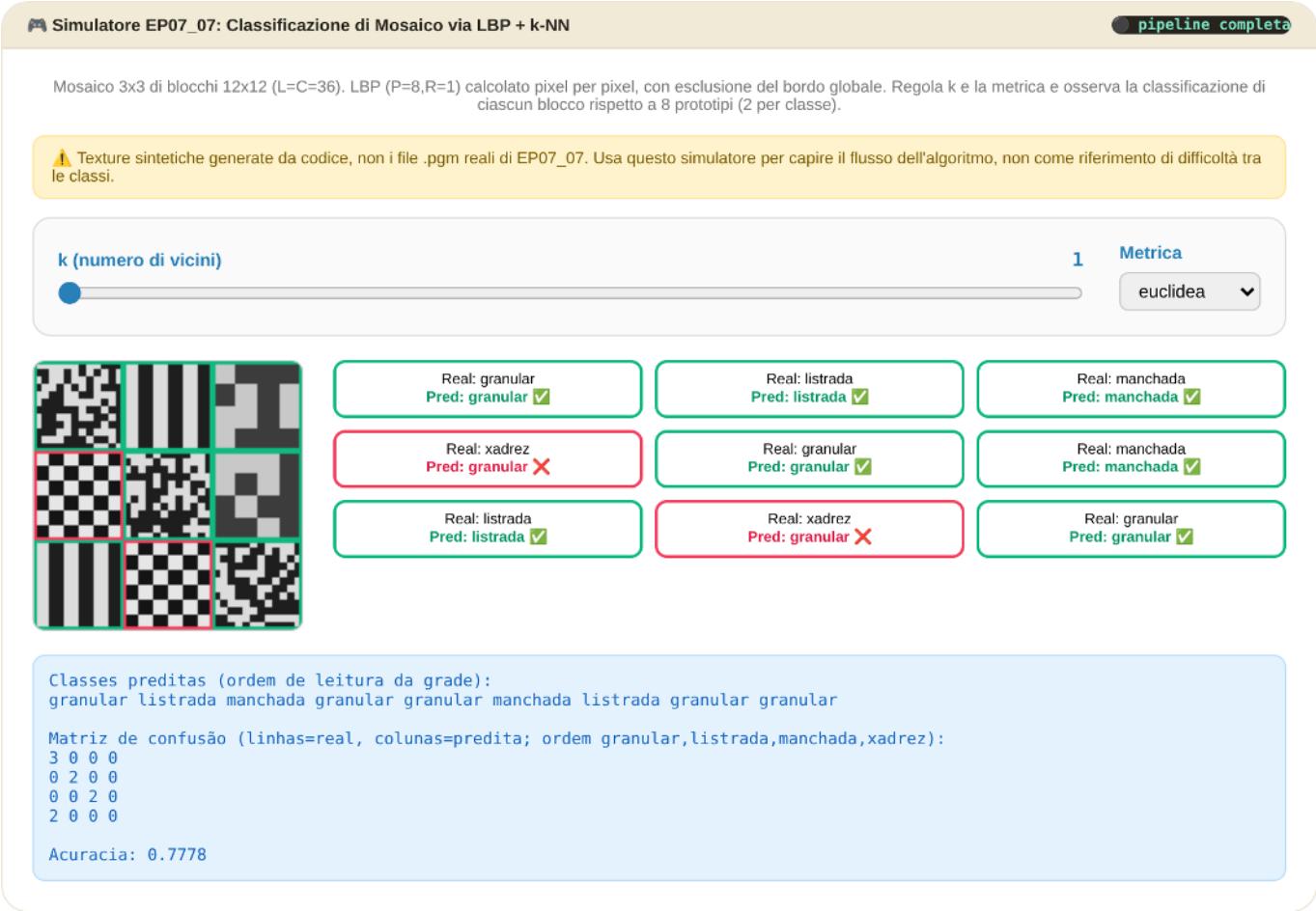
Figura 7.27: Mosaici di riferimento (formato PGM ASCII) usati nell'EP07_07. Caso 1: griglia 2x2 con un campione di ogni classe. Caso 2: griglia 3x3 con classi ripetute e maggiore variabilità.

```
%%writefile EP07_07.py
# Codice Python
```

```
Overwriting EP07_07.py
```

```
TestSuite("EP07_07.py").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap07/sim-ep0707.html

Figura 7.28: Simulatore EP07_07: Classificazione di un Mosaico di Trame tramite LBP + k-NN

Capitolo 8

Comprendere le Scene: Correspondenza delle Caratteristiche, Rilevamento e Segmentazione

Executar Colab Abrir GitHub

Nel **Capitolo 7** è stato studiato come rappresentare le immagini mediante descrittori e utilizzare tali rappresentazioni per compiti di classificazione. In questo capitolo, il problema viene ampliato: oltre a riconoscere pattern, diventa necessario stabilire corrispondenze tra diverse immagini, localizzare automaticamente oggetti di interesse e interpretare l'organizzazione spaziale di una scena.

Questi problemi costituiscono alcuni dei principali compiti della Visione Artificiale e rappresentano una fase naturale successiva alla classificazione delle immagini. Per risolverli, verranno presentati metodi classici di corrispondenza delle caratteristiche, rilevamento degli oggetti e segmentazione delle immagini, nonché una panoramica degli approcci moderni basati sul *Deep Learning*, preparando la transizione al **Capitolo 9**.

8.1 Obiettivi del Capitolo

Al termine di questo capitolo, lo studente dovrebbe essere in grado di:

- **Stabilire corrispondenze tra immagini** utilizzando rilevatori e descrittori locali, stimando trasformazioni geometriche tramite omografie per la registrazione e la correzione prospettica;
- **Localizzare oggetti di interesse** nelle immagini utilizzando metodi classici di rilevamento e valutare i risultati tramite metriche come la ***Intersection over Union*** **(IoU)** e la tecnica di ***Non-Maximum Suppression*** **(NMS)**;
- **Distinguere i paradigmi di segmentazione** — semantica, per istanze e panottica — comprendendo che la segmentazione panottica unifica la segmentazione semantica e quella per istanze, fornendo una descrizione più completa della scena, e applicare metodi classici di segmentazione;
- **Estrarre descrittori geometrici e topologici** dagli oggetti segmentati ed esportarli come annotazioni strutturate (CSV o formato YOLO), validando la qualità delle annotazioni tramite la metrica IoU;
- **Mettere in relazione i metodi classici di corrispondenza, rilevamento e segmentazione** con gli approcci moderni basati sul *Deep Learning*, studiati nel **Capitolo 9**.

La Figura 8.1 presenta una panoramica dei principali concetti e delle relazioni tra gli argomenti trattati in questo capitolo, fungendo da mappa concettuale per guidare la lettura.

8.2 Configurazione dell'Ambiente

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")
```

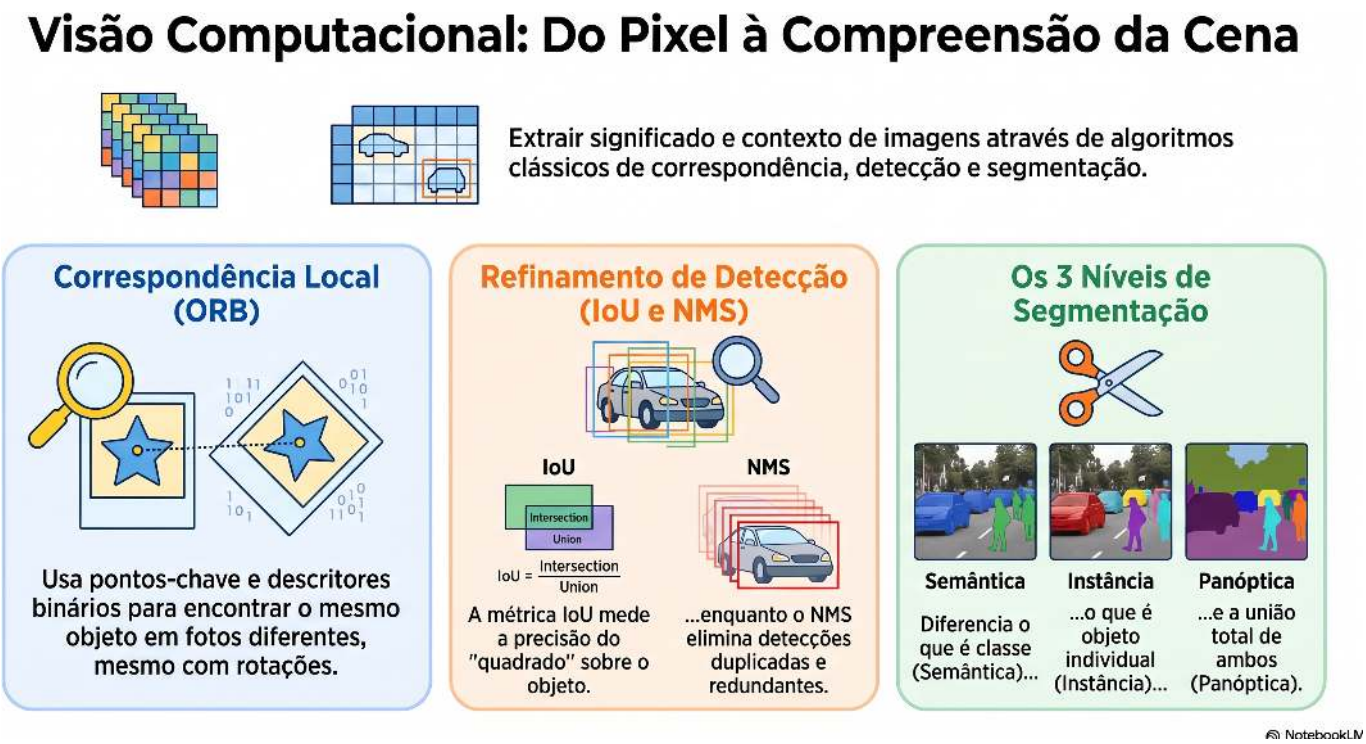


Figura 8.1: Panoramica dei principali concetti trattati in questo capitolo, inclusi la corrispondenza delle caratteristiche, il rilevamento degli oggetti, la segmentazione delle immagini e le loro relazioni con gli approcci moderni basati sul *Deep Learning*. **Fonte:** elaborato con l'ausilio di Gemini Notebook ({GOOGLE}, 2025).

```
import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite


import importlib
import subprocess
import sys

def setup_cap08():
    """Installa le librerie mancanti necessarie per questo capitolo."""
    pacotes = {
        "cv2": "opencv-python",
        "skimage": "scikit-image",
        "numpy": "numpy",
        "sklearn": "scikit-learn",
        "matplotlib": "matplotlib",
    }
    for mod, pkg in pacotes.items():
        if importlib.util.find_spec(mod) is None:
            resultado = subprocess.run(
                [sys.executable, "-m", "pip", "install", "-q", pkg]
            )
            if resultado.returncode != 0:
                print(f"[AVVISO] Impossibile installare {pkg} (necessaria per il modulo
                {mod}).")


setup_cap08()
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from skimage import data as skdata
from skimage.filters import threshold_otsu
from skimage.measure import label
from skimage.morphology import remove_small_objects, opening, disk
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

8.3 Rilevatori e Descrittori Locali di Caratteristiche

Nel capitolo precedente, ogni immagine è stata rappresentata da un singolo vettore di caratteristiche utilizzato per la classificazione. In molte applicazioni, tuttavia, è necessario confrontare solo parti dell'immagine, stabilendo corrispondenze tra regioni osservate in istanti, posizioni o punti di vista diversi. Questo compito richiede una rappresentazione locale dell'immagine, in grado di identificare strutture sufficientemente distintive da essere ritrovate in altre immagini.

Questo processo viene realizzato in due fasi complementari. Inizialmente, un **rilevatore** identifica punti di interesse (*keypoints*), generalmente associati a angoli o regioni con variazioni significative di intensità. Successivamente, un **descrittore** rappresenta numericamente il vicinato di ciascun punto rilevato, consentendo di confrontare regioni corrispondenti tra diverse immagini.

Una volta ottenute le coppie *(punto, descrittore)*, la corrispondenza (*matching*) consiste nel trovare, per ogni descrittore di un'immagine, il descrittore più simile nell'altra. Queste corrispondenze costituiscono la base di numerose applicazioni, come la registrazione di immagini, la ricostruzione tridimensionale, la navigazione visiva e la realtà aumentata.

8.3.1 L'Algoritmo ORB (*Oriented FAST and Rotated BRIEF*)

In questo capitolo verrà utilizzato l'**ORB**, un rilevatore e descrittore locale che combina efficienza computazionale e robustezza alle rotazioni. L'algoritmo riunisce tre componenti principali:

- **FAST** (*Features from Accelerated Segment Test*), responsabile della rilevazione dei punti chiave;
- **BRIEF** (*Binary Robust Independent Elementary Features*), responsabile della costruzione del descrittore binario;
- un meccanismo di **stima dell'orientamento del vicinato**, che rende il descrittore approssimativamente invariante alla rotazione.

Il rilevatore **FAST** scorre tutti i pixel dell'immagine. Per ogni pixel candidato, analizza un cerchio di **16 pixel** attorno ad esso. Se un insieme di pixel consecutivi presenta un'intensità significativamente maggiore o minore rispetto all'intensità del pixel centrale, quel pixel viene considerato un **punto chiave** (*keypoint*). Successivamente, i candidati troppo vicini vengono filtrati, conservando solo i più rappresentativi.

Dopo la rilevazione dei punti chiave, il descrittore **BRIEF** viene calcolato in un vicinato attorno a ciascun punto chiave, e non solo sui 16 pixel utilizzati dal FAST. In questa regione, un **modello di campionamento**, formato da un insieme fisso di coppie di punti (x, y) distribuiti in una finestra attorno al punto chiave, viene utilizzato per effettuare confronti di intensità secondo la Equazione 8.1. Nell'ORB, l'**orientamento predominante del vicinato** viene stimato a partire dalla distribuzione delle intensità in quella regione, e il modello di campionamento viene ruotato in base a tale orientamento. Inoltre, l'ORB utilizza una versione ottimizzata del BRIEF, denominata **rBRIEF** (*Rotated BRIEF*), nella quale le coppie di punti vengono selezionate per produrre descrittori più discriminatori e con bassa correlazione tra i loro bit.

$$\tau(p; x, y) = \begin{cases} 1, & \text{se } I(x) < I(y), \\ 0, & \text{altrimenti,} \end{cases} \tag{8.1}$$

dove:

- p è il punto chiave rilevato dal FAST;
- x e y sono due punti del vicinato di p, scelti dal modello di campionamento del BRIEF;
- $I(\cdot)$ rappresenta l'intensità di un pixel;
- $\tau(p; x, y)$ è il risultato del confronto binario tra i punti x e y.

Ogni confronto genera un bit del descrittore. La concatenazione di tutti questi confronti forma il descrittore binario associato al punto chiave.

Poiché il descrittore è binario, la similarità tra due punti viene misurata mediante la **distanza di Hamming**, corrispondente al numero di bit differenti tra due descrittori. Questa metrica può essere calcolata in modo molto efficiente tramite operazioni logiche sui bit, rendendo l'ORB adatto ad applicazioni in tempo reale.

8.3.2 Esplorando il Simulatore dell'ORB

La Figura 8.3 illustra, in modo interattivo, come viene costruito il descrittore **ORB**. Ogni segmento rappresenta una delle **32 coppie di punti** (x, y) utilizzate nella Equazione 8.1. Il colore del segmento indica il risultato del confronto delle intensità: **verde** quando $I(x) < I(y)$ (bit pari a 1) e **rosso** in caso contrario (bit pari a 0). La freccia **gialla** rappresenta l'orientamento predominante del vicinato, stimato a partire dal centroide di intensità. Il pattern di campionamento del BRIEF viene ruotato in base a questo orientamento, rendendo il descrittore approssimativamente invariante alla rotazione.

Inizialmente, imposta il rumore a zero e confronta la sequenza di bit con lo *slider* a 0° e poi a 15° (annotando il valore del campo "Firma del Descrittore Binario" in ciascun caso):

```
0°  : 00000000000000100000011111100010
15° : 00000000000000100000011111100010
```

In questo esempio, entrambi i descrittori sono identici, e il pannello "Dist. di Hamming" conferma una distanza pari a zero — evidenza che la stima dell'orientamento sta compensando correttamente la rotazione dell'immagine.

Ora clicca su "Aggiungi Rumore" e ripeti l'esperimento. Poiché il rumore viene generato casualmente a ogni esecuzione, i valori seguenti sono solo un esempio — i tuoi saranno diversi, ma dovrebbero presentare una distanza di Hamming di magnitudo simile (tipicamente tra 6 e 14 bit, su un totale di 32):

```
0°  : 01000000001000110110011111101010
15° : 10110000001011100100111111100010
```

Il rumore altera parte dei confronti di intensità, modificando alcuni bit del descrittore. La differenza tra due descrittori viene misurata dalla **distanza di Hamming**, corrispondente al numero di posizioni in cui i bit differiscono. Per i descrittori sopra riportati, tale distanza è pari a **10**.

Un modo efficiente per calcolare questa distanza in Python consiste nell'applicare l'operazione **XOR** (`^`), che identifica i bit differenti, seguita dal metodo `bit_count()`, che conta quanti bit pari a 1 sono presenti nel risultato.

```python
def hamming(a: int, b: int) -> int:
    return (a ^ b).bit_count()
a = 0b01000000001000110110011111101010
b = 0b10110000001011100100111111100010
hamming(a, b)
```

```
10
```

La Figura 8.2 confronta questa implementazione con una versione basata sul confronto dei caratteri e con l'implementazione ottimizzata di OpenCV (`cv2.NORM_HAMMING`).

```
import random, timeit, cv2, numpy as np, pandas as pd
import matplotlib.pyplot as plt

BITS, N = 256, 100_000
A = [''.join(random.choice('01') for _ in range(BITS)) for _ in range(N)]
B = [''.join(random.choice('01') for _ in range(BITS)) for _ in range(N)]
Ai, Bi = map(lambda L: [int(x,2) for x in L], (A,B))
Acv = np.array([[int(s[i:i+8],2) for i in range(0,BITS,8)] for s in A], np.uint8)
Bcv = np.array([[int(s[i:i+8],2) for i in range(0,BITS,8)] for s in B], np.uint8)

H = [
    ("Strings", lambda: sum(sum(x!=y for x,y in zip(a,b)) for a,b in zip(A,B))),
    ("XOR+bit_count()", lambda: sum((a^b).bit_count() for a,b in zip(Ai,Bi))),
    ("OpenCV", lambda: sum(cv2.norm(a,b,cv2.NORM_HAMMING) for a,b in zip(Acv,Bcv)))
]

df = pd.DataFrame(
    [(n, timeit.timeit(f, number=1)) for n,f in H],
    columns=["Método","Tempo (s)"]
)
df["Speedup"] = (df.iloc[0,1]/df["Tempo (s)"]).round(1)

print(df)

plt.figure(figsize=(6,3))
plt.bar(df["Método"], df["Tempo (s)"])
plt.ylabel("Tempo (s)")
plt.show()
```

```
            Método  Tempo (s)  Speedup
0          Strings   1.890271      1.0
1  XOR+bit_count()   0.013022    145.2
2           OpenCV   0.215497      8.8
```

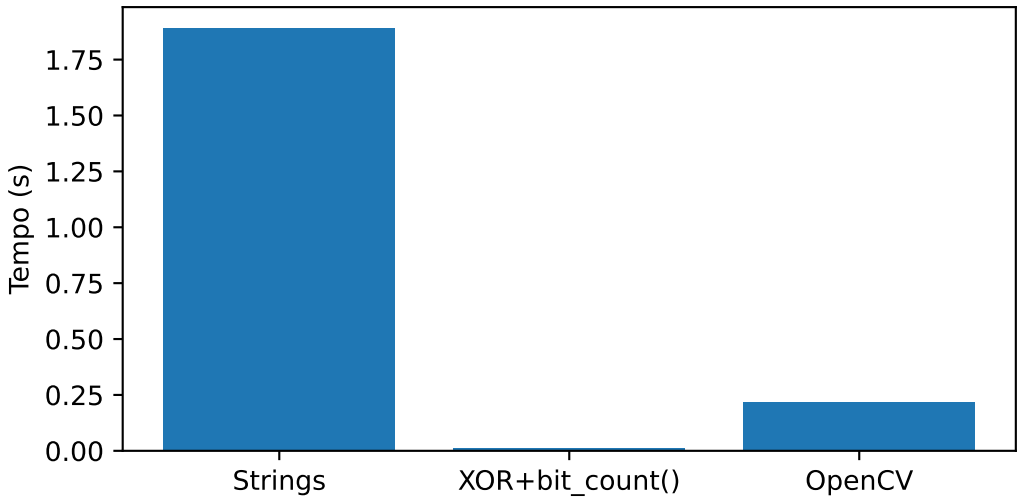


Figura 8.2: Confronto delle prestazioni di diverse implementazioni della distanza di Hamming.

8.4 Corrispondenza delle Caratteristiche e Omografia con ORB

Per illustrare il processo di corrispondenza delle caratteristiche, si utilizza una scena sintetica ottenuta dalla rotazione di 20° dell'immagine originale. Questa trasformazione simula una seconda acquisizione della stessa

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-08-orb.html

Figura 8.3: Simulatore interattivo del descrittore ORB: esplora la logica di rotazione e costruzione del descrittore binario. Modifica la rotazione per osservare come il pattern di campionamento dei test binari del BRIEF (linee verdi e rosse) si orienta dinamicamente per garantire l'invarianza angolare.

scena da un altro punto di vista. La Figura 8.4 presenta l'immagine di riferimento e la sua versione ruotata, che saranno utilizzate nelle fasi successive.

```
caminho_local = "dados/fig-08-Gemini-faces3.png"
url_imagem = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/all/cap08/"
url_imagem += caminho_local

if not os.path.exists(caminho_local):
    mm.write(mm.read(url_imagem), caminho_local)

img_original = mm.read(caminho_local)
img_cena = mm.rotate(img_original, angle=20)

mm.show([img_original, img_cena],
        titles=["Original", "Rotação (20°)"],
        cols=2, figsize=(8, 4))
```

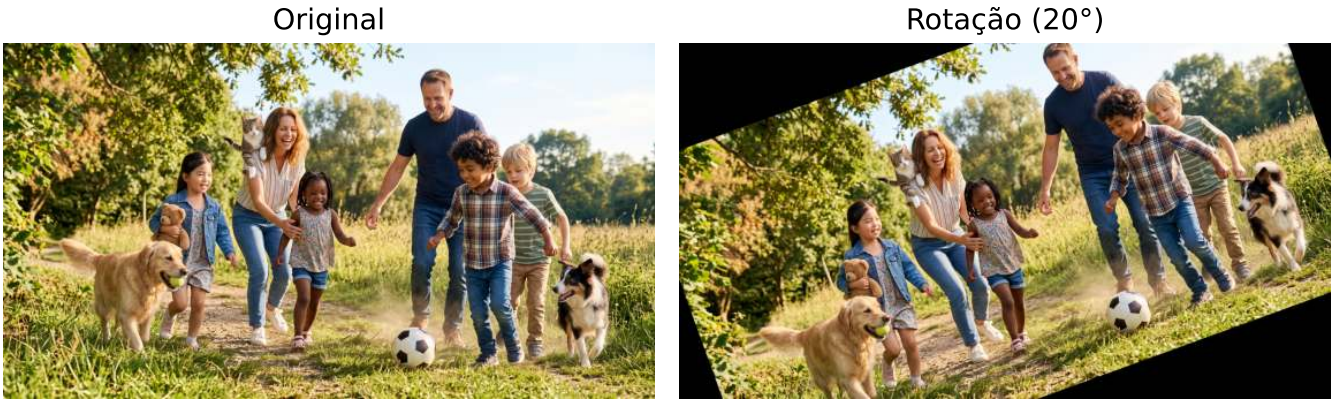


Figura 8.4: Imagem original gerada pelo Gemini e versão rotacionada (20°), simulando uma mudança de ponto de vista.

8.4.1 Rilevamento e Corrispondenza delle Caratteristiche con ORB

Con le due immagini disponibili, **ORB** rileva i punti chiave e calcola i loro descrittori binari. Successivamente, il **BFMatcher** della libreria `cv2` stabilisce le corrispondenze tra i descrittori utilizzando la distanza di Hamming e la verifica incrociata (*cross-check*). Infine, le corrispondenze vengono ordinate dalla distanza di Hamming minore a quella maggiore, dando priorità alle coppie potenzialmente più affidabili. Nel codice seguente, che genera la Figura 8.5, si evidenziano i seguenti passaggi:

1. `cv2.ORB_create(nfeatures=500)` istanzia il rilevatore ORB, limitando la ricerca ai **500 punti chiave** più rappresentativi di ciascuna immagine. Questa restrizione riduce il costo computazionale ed evita la selezione di punti poco distintivi.
2. `orb.detectAndCompute(...)` esegue, in un'unica chiamata, il rilevamento dei punti chiave tramite **FAST** e il calcolo dei descrittori tramite **BRIEF orientato**, restituendo l'elenco dei punti chiave (`kp`) e i loro descrittori binari a **256 bit** (`des`).
3. `cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)` crea un comparatore a forza bruta (*Brute-Force Matcher*), che utilizza la **distanza di Hamming** — la stessa metrica esplorata nella Figura 8.3 — per confrontare ciascun descrittore dell'immagine originale con tutti i descrittori dell'immagine ruotata. Il parametro `crossCheck=True` conserva solo le coppie in cui la migliore corrispondenza è reciproca, cioè quando il miglior corrispondente di A è B e, simultaneamente, il miglior corrispondente di B è A. Questo criterio elimina gran parte delle corrispondenze ambigue.
4. `matches = sorted(...)` ordina le corrispondenze dalla distanza di Hamming minore a quella maggiore. Minore è questa distanza, maggiore è la similarità tra i descrittori e, di conseguenza, maggiore è la probabilità che la corrispondenza sia corretta.

La Figura 8.5 presenta solo le cinque corrispondenze con la distanza di Hamming minore. Sebbene queste coppie siano le più promettenti, non esiste ancora alcuna restrizione geometrica tra i punti corrispondenti. Di conseguenza, alcune connessioni possono rappresentare false corrispondenze (*false matches*), giustificando l'uso del **RANSAC** nella fase successiva per identificare solo le corrispondenze geometricamente coerenti.

```
orb = cv2.ORB_create(nfeatures=500)
kp1, des1 = orb.detectAndCompute(img_original, None)
kp2, des2 = orb.detectAndCompute(img_cena, None)
print(f"Punti di interesse rilevati: {len(kp1)} (originale), {len(kp2)} (scena)")

bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = sorted(
    bf.match(des1, des2),
    key=lambda m: m.distance
)
print(f"Corrispondenze trovate: {len(matches)}")

def draw_matches_destacado(img1, kp1, img2, kp2, matches, espessura=2, raio_ponto=4,
                           seed=42, cor_fixa=None):
    """Disegna le immagini affiancate con linee che collegano i punti corrispondenti.
    Se cor_fixa=None, ogni corrispondenza riceve un colore casuale (facilita distinguere
    collegamenti individuali). Se cor_fixa è definita (es: verde), tutte le linee usano lo
    stesso colore - utile per evidenziare un sottoinsieme specifico, come gli inlier del
RANSAC.
    """
    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]
    h = max(h1, h2)
    canvas = np.zeros((h, w1 + w2, 3), dtype=np.uint8)
    canvas[:h1, :w1] = cv2.cvtColor(img1, cv2.COLOR_GRAY2BGR) if img1.ndim == 2 else img1
    canvas[:h2, w1:w1+w2] = cv2.cvtColor(img2, cv2.COLOR_GRAY2BGR) if img2.ndim == 2 else img2

    rng = np.random.RandomState(seed)  # seed fissa = colori riproducibili a ogni esecuzione

    for m in matches:
        pt1 = tuple(np.round(kp1[m.queryIdx].pt).astype(int))
        pt2 = tuple(np.round(kp2[m.trainIdx].pt).astype(int) + np.array([w1, 0]))

        cor = cor_fixa if cor_fixa is not None else \
            tuple(int(c) for c in rng.randint(60, 256, size=3))

        cv2.line(canvas, pt1, pt2, cor, espessura, lineType=cv2.LINE_AA)
        cv2.circle(canvas, pt1, raio_ponto, cor, -1, lineType=cv2.LINE_AA)
        cv2.circle(canvas, pt2, raio_ponto, cor, -1, lineType=cv2.LINE_AA)

    return canvas

# prova le 5 peggiori: matches[-5:]
img_matches = draw_matches_destacado(
    img_original, kp1, img_cena, kp2, matches[:5],
    espessura=10, raio_ponto=15
)
mm.show([img_matches], titles=["Top 5 Corrispondenze ORB"], cols=1, figsize=(10, 5))
```

```
Punti di interesse rilevati: 500 (originale), 500 (scena)
Corrispondenze trovate: 300
```

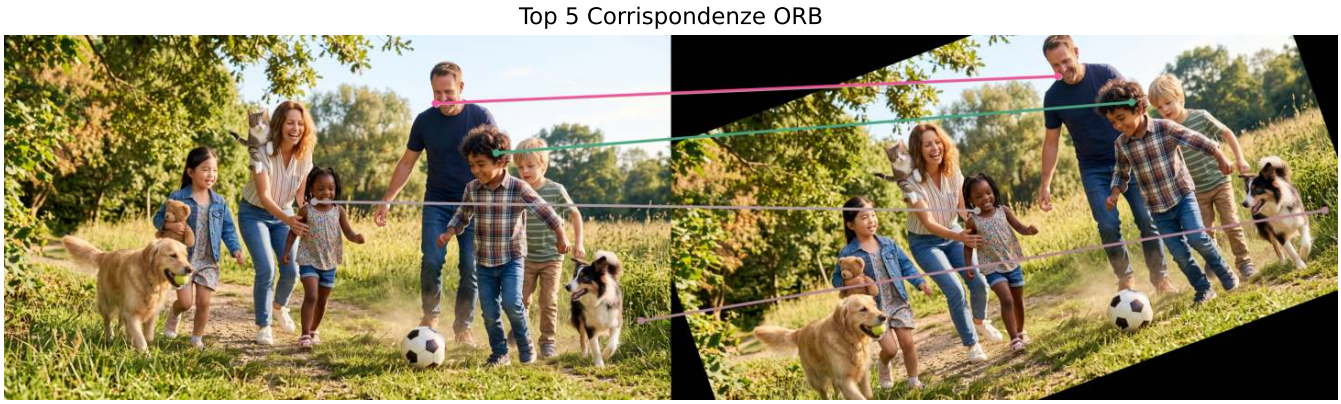


Figura 8.5: Le 5 migliori corrispondenze delle caratteristiche ORB tra l'immagine originale e la scena sintetica, prima del filtraggio con RANSAC.

La funzione `draw_matches_destacado()` ha esclusivamente finalità di visualizzazione: posiziona le immagini fianco a fianco e disegna linee tra le coppie corrispondenti, senza interferire nella stima delle corrispondenze.

8.5 Modellazione Matematica: Omografia e RANSAC

8.5.1 Omografia

Negli Esercizi 10 e 11 del Capitolo 2, le funzioni `cv2.getPerspectiveTransform` e `cv2.warpPerspective` sono state utilizzate per correggere la prospettiva delle immagini a partire da **quattro coppie di punti corrispondenti** forniti manualmente. In questo capitolo, tali corrispondenze vengono ottenute automaticamente tramite ORB, rendendo possibile stimare la trasformazione tra due immagini senza intervento dell'utente.

Matematicamente, questa trasformazione è descritta da un'**omografia**, rappresentata da una matrice 3×3 che mette in relazione le coordinate di uno stesso piano osservato da diversi punti di vista:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

Dopo la normalizzazione delle coordinate omogenee, si ottiene il punto corrispondente

$$\left(\frac{x'}{w'}, \frac{y'}{w'} \right).$$

Poiché l'omografia è definita a meno di un fattore di scala, essa possiede **otto gradi di libertà**. Di conseguenza, sono necessarie almeno **quattro coppie di punti corrispondenti** per stimarne i parametri.

Nella pratica, tuttavia, le corrispondenze generate automaticamente da ORB possono contenere associazioni errate (*outlier*). Per stimare l'omografia in modo affidabile anche in presenza di questi errori, si utilizza l'algoritmo **RANSAC**, presentato nella sezione successiva.

8.5.2 RANSAC

Il **RANSAC** (*Random Sample Consensus*) è un algoritmo di stima robusta in grado di adattare un modello geometrico anche in presenza di osservazioni errate (*outlier*). In questo capitolo, il modello di interesse è una **omografia**, stimata a partire dalle corrispondenze prodotte dall'ORB.

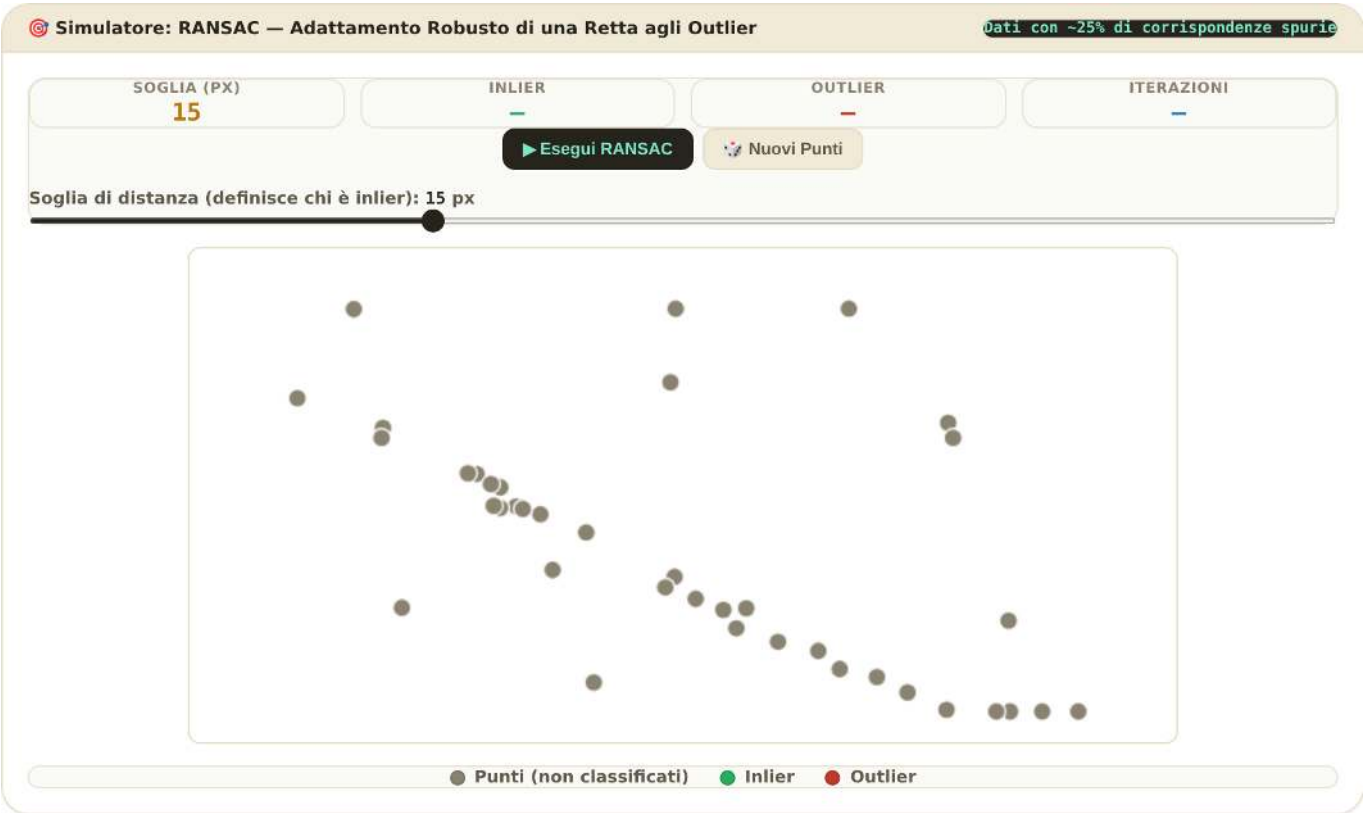
In ogni iterazione, l'algoritmo:

1. seleziona casualmente un piccolo sottoinsieme di corrispondenze (quattro coppie di punti, nel caso dell'omografia);
2. stima un'omografia candidata a partire da tale sottoinsieme;
3. verifica quali corrispondenze sono compatibili con questa trasformazione, classificandole come ***inlier*** o ***outlier***;
4. registra l'omografia che produce il maggior numero di ***inlier***;
5. ristima l'omografia utilizzando solo gli ***inlier*** trovati.

Sebbene l'esempio di questo capitolo utilizzi un'omografia, il RANSAC è un algoritmo di scopo generale e può essere impiegato per stimare diversi modelli geometrici, come rette, circonferenze, piani e altre trasformazioni. In tutti i casi, il principio è lo stesso: generare modelli candidati a partire da piccoli campioni casuali e selezionare quello che presenta il maggiore consenso tra i dati.

Per comprendere questo processo in modo graduale, vengono presentati due simulatori.

Il primo, mostrato nella Figura 8.6, utilizza l'esempio più semplice possibile: l'adattamento di una **retta** a un insieme di punti contenente circa **il 25% di *outlier***. L'obiettivo è comprendere le fasi fondamentali dell'algoritmo — campionare, stimare un modello, identificare gli ***inlier*** e ripetere il processo — senza la complessità della registrazione tra immagini.



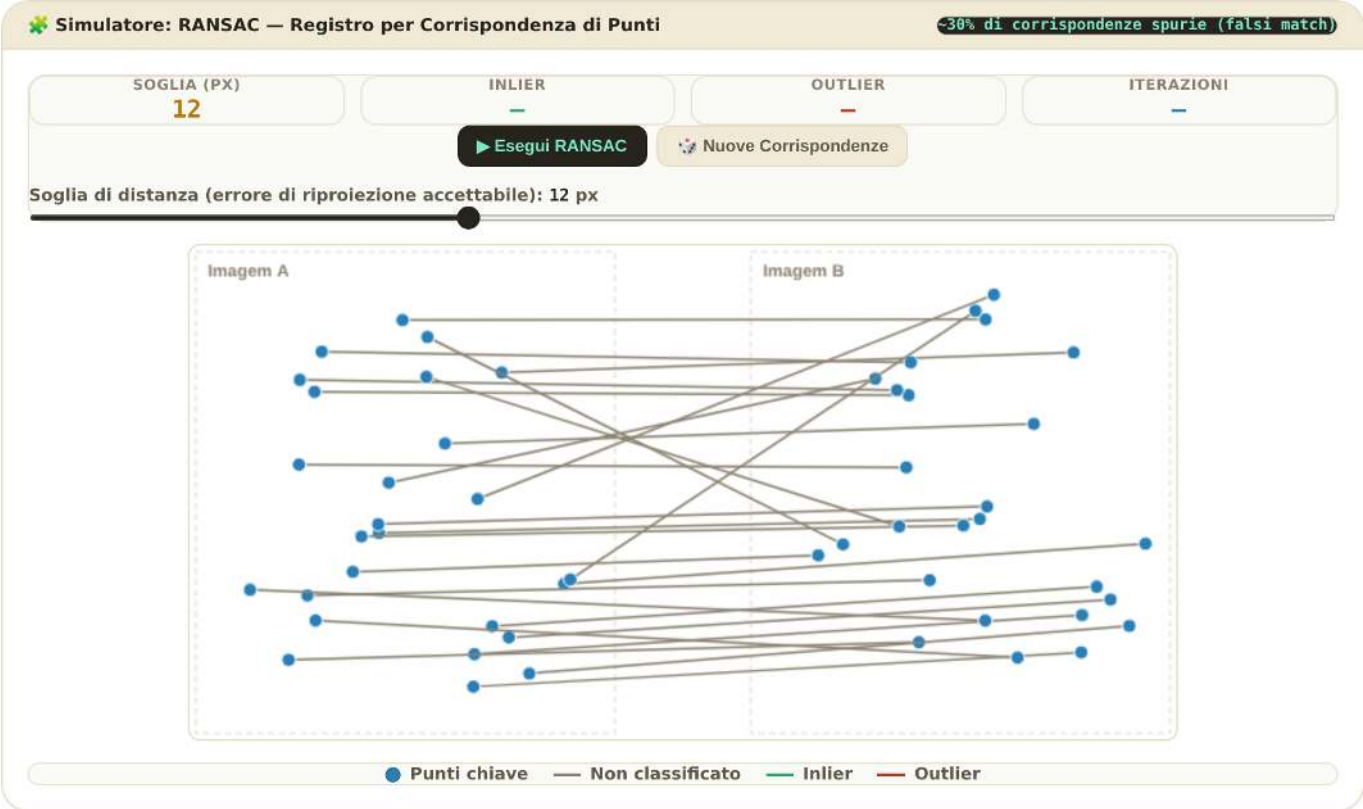
Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-08-ransac.html

Figura 8.6: Simulatore interattivo dell'algoritmo RANSAC: regola la soglia di distanza ed esegui l'algoritmo per osservare la separazione tra *inliers* e *outliers*.

Il secondo simulatore, presentato nella Figura 8.7, si avvicina al problema studiato in questo capitolo. Invece di un unico insieme di punti, vengono considerate **due immagini** contenenti corrispondenze tra punti chiave. Alcune corrispondenze sono corrette (*inliers*), mentre altre sono errate (*outliers*), risultanti da errori nel processo di corrispondenza dei descrittori. In questo simulatore, il modello stimato è una **trasformazione di similarità** (rotazione, scala e traslazione), più semplice rispetto a un'omografia completa, ma sufficiente per illustrare il problema della registrazione tra immagini.

In entrambi i simulatori, l'algoritmo eseguito segue esattamente lo stesso principio utilizzato successivamente per stimare l'omografia. L'unica differenza risiede nel modello geometrico adattato.

Regola la soglia di distanza ed esegui l'algoritmo in ciascun simulatore per osservare come il RANSAC identifichi gli ***inliers***, scarti gli ***outliers*** e stimi un modello coerente utilizzando solo le corrispondenze valide.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-08-ransac-registro.html

Figura 8.7: Simulatore interattivo del RANSAC applicato alla registrazione di immagini: i punti chiave di due immagini vengono accoppiati tramite un descrittore, alcune corrispondenze sono spurie (*outlier*), e il RANSAC stima la trasformazione di similarità che allinea la maggior parte di esse.

8.5.3 Stima dell'Omografia con RANSAC

Dopo aver ottenuto le corrispondenze tra i punti chiave tramite ORB, il passo successivo consiste nello stimare l'**omografia** tra le due immagini. A tal fine, si utilizza la funzione `cv2.findHomography()`, che impiega l'algoritmo **RANSAC** per calcolare questa trasformazione e restituire una maschera che indica quali corrispondenze sono state classificate come ***inliers***.

Il codice seguente esegue quattro operazioni principali:

1. estrae le coordinate dei punti corrispondenti in ciascuna immagine;
2. stima l'omografia tramite `cv2.findHomography(..., cv2.RANSAC)`;
3. riceve la maschera prodotta dal RANSAC, nella quale ciascuna corrispondenza è classificata come ***inlier*** o ***outlier***;
4. utilizza tale maschera per selezionare solo le corrispondenze classificate come ***inliers***.

La Figura 8.8 presenta solo le corrispondenze classificate come ***inliers***. Si osserva che queste coppie di punti sono compatibili con una stessa trasformazione geometrica, mentre le corrispondenze inconsistenti (*outliers*) vengono scartate. Di conseguenza, l'omografia stimata rappresenta in modo più fedele la relazione geometrica tra le due immagini.

```
pts1 = np.float32([kp1[m.queryIdx].pt for m in matches])
pts2 = np.float32([kp2[m.trainIdx].pt for m in matches])

H, mascara_inliers = cv2.findHomography(pts1, pts2, cv2.RANSAC, ransacReprojThreshold=5.0)
n_inliers = int(mascara_inliers.sum())
```

```
print(f"Matrice di omografia stimata:\n{H}\n")
print(f"Inliers: {n_inliers} di {len(matches)} corrispondenze \
      ({100*n_inliers/len(matches):.1f}%)")

matches_inliers = [m for m, ok in zip(matches, mascara_inliers.ravel()) if ok]

img_inliers = draw_matches_destacado(
    img_original, kp1, img_cena, kp2, matches_inliers,
    espessura=2, raio_ponto=4, cor_fixa=(0, 200, 0)  # verde (BGR)
)
mm.show([img_inliers], titles=[f"Corrispondenze Inliers (RANSAC) - \
                               {n_inliers}/{len(matches)}"], cols=1, figsize=(10, 5))
```

```
Matrice di omografia stimata:
[[ 9.39407240e-01  3.41706397e-01 -1.77390658e+02]
 [-3.41929137e-01  9.39717641e-01  5.27534018e+02]
 [-1.12175998e-07 -4.86089683e-08  1.00000000e+00]]

Inliers: 285 di 300 corrispondenze       (95.0%)
```

Figura 8.8: Corrispondenze classificate come *inliers* (verde) dal RANSAC stimando l'omografia tra le due immagini.

8.5.4 Registrazione dell'Immagine

Dopo aver stimato l'omografia, il passo successivo consiste nell'utilizzarla per registrare l'immagine della scena nel sistema di coordinate dell'immagine originale. Questo processo consente di allineare le due immagini, facilitando il confronto tra di esse.

Il codice esegue tre operazioni principali:

1. applica la trasformazione proiettiva tramite `cv2.warpPerspective()`, utilizzando l'opzione `cv2.WARP_INVERSE_MAP`, che applica internamente la trasformazione inversa senza la necessità di calcolare esplicitamente H^{-1};
2. calcola la differenza assoluta pixel per pixel tra l'immagine registrata e l'immagine originale tramite `cv2.absdiff()`;
3. visualizza l'immagine originale, la scena ruotata, l'immagine registrata e una mappa delle differenze tra le due immagini.

La Figura 8.9 presenta il risultato della registrazione. Si osserva che l'immagine registrata diventa visivamente molto simile all'immagine originale, indicando che l'omografia stimata è riuscita ad allineare correttamente le due viste della stessa scena. La mappa delle differenze evidenzia solo le regioni in cui permangono piccole discrepanze derivanti da errori di interpolazione, quantizzazione e dalla stessa stima dell'omografia.

```
h, w = img_original.shape[:2]

# WARP_INVERSE_MAP: applica H "all'indietro", evitando il calcolo manuale di H^-1
img_registrada = cv2.warpPerspective(img_cena, H, (w, h), flags=cv2.WARP_INVERSE_MAP)

erro = cv2.absdiff(img_original, img_registrada)

mm.show( # mostra mm.gray(erro)>10 in toni di grigio
    [img_original, img_cena, img_registrada, mm.gray(erro)>10],
    titles=["Originale", "Scena (ruotata)", "Registrata", "Differenza"],
    cols=2,
    figsize=(14, 8),
)
```

Figura 8.9: Registrazione dell'immagine della scena usando l'omografia inversa stimata da RANSAC.

Perché funziona? — Robustezza per consenso

Molti metodi di stima adattano un modello utilizzando **tutte** le osservazioni disponibili, cercando di minimizzare l'errore totale tra i dati osservati e il modello adattato (approccio noto come **minimi quadrati**). Quando esistono *outlier*, queste osservazioni errate possono spostare significativamente il risultato ottenuto.

Il **RANSAC** segue una strategia diversa. Invece di utilizzare tutti i dati simultaneamente, stima modelli successivi a partire da piccoli campioni casuali. Ogni modello viene poi valutato in base al numero di corrispondenze compatibili con esso. Al termine delle iterazioni, viene selezionato il modello che presenta il maggiore consenso tra i dati, cioè il maggior numero di ***inlier***.

Due parametri svolgono un ruolo fondamentale nell'algoritmo:

- la **soglia di riproiezione**, che definisce la distanza massima affinché una corrispondenza sia classificata come ***inlier***;
- il **numero di iterazioni**, che deve essere sufficientemente grande per aumentare la probabilità di selezionare almeno un campione privo di ***outlier***.

Sebbene piuttosto robusto, il RANSAC presuppone che nei dati esista un modello geometrico predominante. Le sue prestazioni tendono a diminuire quando la proporzione di ***inlier*** è molto piccola o quando diverse strutture geometriche coesistono nella stessa scena, rendendo difficile l'identificazione di un unico modello dominante.

8.6 Rilevamento di Oggetti: *Haar Cascade* (Viola-Jones)

La corrispondenza delle caratteristiche risponde alla domanda: **"dove si trova lo stesso oggetto o pattern osservato in precedenza?"**. Il **rilevamento di oggetti** risolve un problema più generale: localizzare automaticamente istanze di una **categoria** (ad esempio, volti umani), anche se gli oggetti specifici non sono mai stati osservati durante l'addestramento. Mentre ORB richiede due immagini per stabilire corrispondenze tra punti, l'**Haar Cascade** opera su una singola immagine, identificando direttamente le regioni candidate a contenere l'oggetto cercato.

L'algoritmo ***Haar Cascade***, proposto da Viola e Jones [Viola; Jones (2001); Viola (2004)], combina caratteristiche Haar, immagini integrali e una cascata di classificatori per realizzare il rilevamento di oggetti in modo efficiente. Sebbene attualmente esistano metodi più recenti basati su reti neurali convoluzionali, l'Haar Cascade rimane disponibile nella libreria OpenCV e costituisce un esempio classico per lo studio delle tecniche di rilevamento di oggetti.

Il suo funzionamento si basa su tre componenti principali:

1. **Caratteristiche Haar:** filtri rettangolari semplici che misurano differenze di intensità tra regioni vicine dell'immagine, sfruttando **pattern di contrasto caratteristici** dell'oggetto, come la regione degli occhi generalmente più scura della fronte;
2. **Immagine integrale:** struttura dati che consente di calcolare rapidamente la somma dei pixel di qualsiasi regione rettangolare dell'immagine:
$$I_{\text{integrale}}(x, y) = \sum_{x' \leq x,\, y' \leq y} I(x', y'),$$
riducendo significativamente il costo computazionale della valutazione delle caratteristiche Haar. Con questa struttura, la somma dei pixel di qualsiasi rettangolo può essere ottenuta con soli quattro accessi all'immagine integrale;
3. **Cascata di classificatori:** durante l'addestramento, l'algoritmo AdaBoost seleziona e combina classificatori semplici in una sequenza di stadi. Nel rilevamento, le regioni che chiaramente **non corrispondono all'oggetto** vengono scartate già nei primi stadi, mentre solo le candidate più promettenti superano le fasi successive, più precise e computazionalmente più costose. Questa strategia consente di effettuare una ricerca efficiente in diverse posizioni e scale dell'immagine.

Il codice seguente, che genera la Figura 8.10, utilizza classificatori pre-addestrati e messi a disposizione da OpenCV per rilevare **volti** e, successivamente, restringe la ricerca degli **occhi** solo all'interno di ciascun volto rilevato. Questa strategia riduce i falsi positivi e diminuisce il costo computazionale, poiché evita di effettuare la ricerca degli occhi su tutta l'immagine.

Durante il rilevamento, una finestra scorre l'immagine in diverse posizioni e scale. La funzione `detectMultiScale()` esegue automaticamente questa ricerca. Il parametro `scaleFactor` controlla il fattore di riduzione tra scale consecutive della finestra, mentre `minNeighbors` definisce il numero minimo di rilevamenti vicini necessari per confermare un oggetto, riducendo i rilevamenti spuri. Il parametro `minSize` stabilisce la dimensione minima dell'oggetto considerata durante la ricerca.

La funzione restituisce un elenco di rettangoli, ciascuno descritto dalle coordinate dell'angolo superiore sinistro e dalle dimensioni `(x, y, larghezza, altezza)`. Questi rettangoli delimitano le regioni classificate come oggetti dal rilevatore e vengono utilizzati per disegnare le scatole mostrate nella Figura 8.10..

```
def get_cascade(nome):
    """Scarica (se necessario) e carica un classificatore Haar Cascade di OpenCV."""
```

```
    caminho = f"haarcascades/{nome}"
    os.makedirs("haarcascades", exist_ok=True)
    if not os.path.exists(caminho):
      url = f"https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/{nome}"
      urllib.request.urlretrieve(url, caminho)
    return cv2.CascadeClassifier(caminho)

# Più utilizzato
#face_cascade = get_cascade("haarcascade_frontalface_default.xml")

# Più preciso, ma più lento
face_cascade = get_cascade("haarcascade_frontalface_alt2.xml")

# Compromesso tra velocità e precisione
#face_cascade = get_cascade("haarcascade_frontalface_alt.xml")

# Molto veloce, ma meno preciso
#face_cascade = get_cascade("haarcascade_frontalface_alt_tree.xml")

# per gli occhi, più utilizzato
#eye_cascade = get_cascade("haarcascade_eye.xml")
eye_cascade  = get_cascade("haarcascade_eye_tree_eyeglasses.xml")

img_original = mm.read(caminho_local)

# Scala di grigi + equalizzazione dell'istogramma (Capitolo 3), come Haar Cascade si aspetta
img_rgb = mm.rotate(img_original, angle=0)
img_gray = cv2.equalizeHist(cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY))

# Rileva i volti e, all'interno di ciascuno, prova a rilevare gli occhi
faces = face_cascade.detectMultiScale(
    img_gray,
    scaleFactor=1.1,      # Passi più ampi tra le scale: più veloce, ma meno sensibile
    minNeighbors=3,       # N. minimo di rilevazioni sovrapposte per confermare un volto
    minSize=(90, 90)      # Ignora regioni candidate più piccole di 30×30 px
)

img_anotada = img_rgb.copy()
for (x, y, w, h) in faces:
    cv2.rectangle(img_anotada, (x, y), (x + w, y + h), (0, 255, 0), 3)
    olhos = eye_cascade.detectMultiScale(
        img_gray[y:y+h, x:x+w],
        scaleFactor=1.02,
        minNeighbors=4,
        minSize=(15, 15)
    )
    for (ex, ey, ew, eh) in olhos:
        cv2.rectangle(img_anotada, (x+ex, y+ey), (x+ex+ew, y+ey+eh), (255, 0, 0), 4)

print(f"Regioni rilevate come volto: {len(faces)}")
mm.show([img_anotada], titles=
        ["Rilevamento Haar Cascade: volti (verde) e occhi (rosso)"],
        cols=1, figsize=(8, 8))
```

```
Regioni rilevate come volto: 5
```

Rilevamento Haar Cascade: volti (verde) e occhi (rosso)

Figura 8.10: Rilevamento di volti e occhi con *Haar Cascade*.

> 🧠 Perché funziona? — E perché anche fallisce
>
> Nell'immagine utilizzata in questo esempio, il classificatore rileva il viso e gli occhi, ma può anche marcare una seconda regione su parte dello sfondo dell'immagine come se fosse un viso. Questo è un esempio di **falso positivo**: la distribuzione locale delle intensità in quella regione è sufficientemente simile ai modelli appresi durante l'addestramento affinché la cascata la classifichi erroneamente come un volto.
> Questo comportamento evidenzia uno dei principali limiti dell'*Haar Cascade*. Poiché il metodo basa la sua decisione solo su caratteristiche di Haar, cioè differenze di intensità tra regioni rettangolari, non rappresenta esplicitamente la forma o il significato degli oggetti presenti nell'immagine. Pertanto, texture e pattern di contrasto simili a quelli riscontrati nei volti possono produrre rilevamenti errati. Inoltre, le sue prestazioni tendono a diminuire in presenza di grandi variazioni di posa, occlusioni, espressioni facciali e condizioni di illuminazione diverse da quelle presenti nei dati utilizzati per l'addestramento.
> Nonostante questi limiti, l'*Haar Cascade* rimane utile in applicazioni che privilegiano un basso costo computazionale. In situazioni che richiedono una maggiore capacità di generalizzazione rispetto alle variazioni nell'aspetto degli oggetti, i metodi moderni basati su reti neurali profonde tendono a offrire prestazioni migliori.

8.7 Rilevamento di Oggetti: Bounding Box, IoU e NMS

Sebbene i metodi di rilevamento di oggetti utilizzino strategie piuttosto diverse — dal *Haar Cascade* ai rilevatori moderni basati su reti neurali profonde —, i loro risultati sono normalmente rappresentati da **bounding box** (*caixas delimitadoras*), definite dalle coordinate $(x_{min}, y_{min}, x_{max}, y_{max})$.

8.7.1 *Intersection over Union* (IoU)

La metrica ***Intersection over Union*** **(IoU)** quantifica la sovrapposizione tra due bounding box, ad esempio, il rilevamento prodotto da un algoritmo e l'annotazione di riferimento (*ground truth*):

$$\mathrm{IoU}(A, B) = \frac{\mathrm{area}(A \cap B)}{\mathrm{area}(A \cup B)} \in [0, 1].$$

Più il valore è vicino a 1, maggiore è la concordanza tra le box; il valore 0 indica assenza di sovrapposizione. Nelle valutazioni dei rilevatori di oggetti, è comune considerare un rilevamento corretto quando IoU $\geq 0{,}5$, sebbene applicazioni specifiche possano adottare soglie diverse.

La metrica IoU non è utilizzata solo per valutare i rilevatori. Essa costituisce anche il criterio impiegato dall'algoritmo di **Soppressione Non-Massima** per decidere quando due bounding box rappresentano lo stesso oggetto e, pertanto, una di esse deve essere eliminata.

8.7.2 Soppressione Non-Massima (NMS)

Durante il rilevamento, è comune che diverse bounding box siano associate allo stesso oggetto. La **Soppressione Non-Massima** (*Non-Maximum Suppression*, NMS) elimina questa ridondanza in tre fasi:

1. ordina i rilevamenti in base al punteggio di confidenza, dal maggiore al minore;
2. mantiene la box con maggiore confidenza ed elimina quelle il cui IoU con essa supera una determinata soglia;
3. ripete il processo con le box rimanenti fino a quando non esistono sovrapposizioni rilevanti.

L'esempio della Figura 8.11 illustra questa procedura con due oggetti, ciascuno inizialmente rappresentato da diverse box sovrapposte. In questo esempio, ogni bounding box riceve un **punteggio di confidenza**, che rappresenta il grado di fiducia del rilevatore che quella regione contenga l'oggetto cercato.

Invece di implementare manualmente l'algoritmo, si utilizza la funzione `cv2.dnn.NMSBoxes`, che implementa la Soppressione Non-Massima impiegata in diversi rilevatori moderni. La funzione `desenhar_caixas` mostra, affiancate, le box prima e dopo l'NMS, evidenziando la riduzione da cinque rilevamenti a soli due, preservando solo la box con maggiore confidenza per ciascun oggetto.

```
def desenhar_caixas(caixas, pontuacoes, indices, cor, tamanho=(450, 450)):
    """Disegna le scatole (e i loro punteggi) indicate in `indices` su uno sfondo bianco."""
    tela = np.full((tamanho[1], tamanho[0], 3), 255, dtype=np.uint8)
    for i in indices:
        x0, y0, x1, y1 = caixas[i]
        cv2.rectangle(tela, (x0, y0), (x1, y1), cor, 2)
        cv2.putText(tela, f"{pontuacoes[i]:.2f}", (x0, y0 - 8),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, cor, 1, cv2.LINE_AA)
    return tela


caixas = np.array([
    [50, 50, 150, 150], [60, 55, 155, 145], [58, 60, 160, 150],
    [300, 300, 400, 420], [310, 305, 395, 415],
])
pontuacoes = np.array([0.90, 0.75, 0.60, 0.95, 0.70])

# cv2.dnn.NMSBoxes si aspetta scatole nel formato (x, y, larghezza, altezza)
caixas_xywh = np.column_stack([
    caixas[:, 0], caixas[:, 1],
    caixas[:, 2] - caixas[:, 0], caixas[:, 3] - caixas[:, 1]
])

mantidas = cv2.dnn.NMSBoxes(
    bboxes=caixas_xywh.tolist(),
    scores=pontuacoes.tolist(),
    score_threshold=0.0,   # nessun taglio per confidenza minima qui
    nms_threshold=0.5      # soglia IoU per considerare due scatole ridondanti
).flatten()
```

```
img_antes = desenhar_caixas(caixas, pontuacoes, range(len(caixas)), cor=(220, 38, 38))
img_depois = desenhar_caixas(caixas, pontuacoes, mantidas, cor=(22, 163, 74))

mm.show(
    [img_antes, img_depois],
    titles=[f"Prima del NMS ({len(caixas)} scatole)", \
            f"Dopo il NMS ({len(mantidas)} scatola(e))"],
    cols=2, figsize=(9, 4.5)
)
```

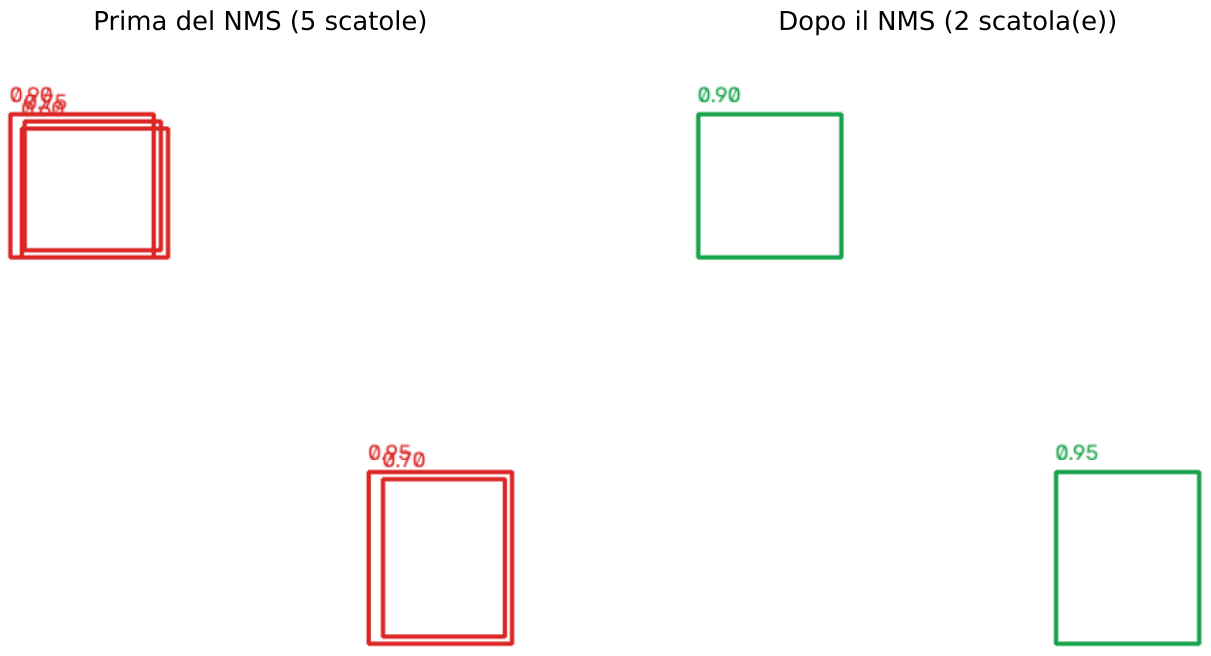


Figura 8.11: Effetto della Soppressione Non-Massimale (NMS): più rilevamenti ridondanti (a sinistra) vengono ridotti a un rilevamento per oggetto (a destra).

> **Perché funziona? — Eliminazione delle ridondanze**
>
> La **NMS** (*Non-Maximum Suppression*) non altera la qualità delle rilevazioni prodotte dal rivelatore. La sua funzione è eliminare **riquadri delimitatori ridondanti** che rappresentano lo stesso oggetto, conservando solo quello con il punteggio di confidenza più elevato.
> La sua efficacia dipende dalla soglia di IoU adottata. Un valore molto basso può eliminare riquadri corrispondenti a oggetti diversi la cui sovrapposizione sia elevata, mentre un valore molto alto può mantenere diversi riquadri sovrapposti per lo stesso oggetto.
> La NMS inoltre **non elimina i falsi positivi isolati**. Se una regione errata viene rilevata da un solo riquadro, essa verrà conservata, poiché non esiste un'altra rilevazione ridondante con cui confrontarla. Per questo motivo, la NMS viene applicata come fase di **post-elaborazione**, utilizzando solo i riquadri delimitatori e i loro punteggi di confidenza prodotti dal rivelatore.

8.8 Segmentazione Visuale: Semantica, per Istanza e Panottica

Una bounding box indica approssimativamente la posizione di un oggetto, ma non identifica quali pixel gli appartengono. La **segmentazione visuale** risolve questo problema assegnando un'etichetta a ogni pixel dell'immagine. A seconda dell'informazione prodotta, si distinguono tre paradigmi principali, riassunti nella Tabella 8.1.

Tabella 8.1: Confronto tra i principali paradigmi di segmentazione visuale.

Paradigma	Domanda a cui risponde	Distingue oggetti della stessa classe?
Semantica	"A quale classe appartiene ogni pixel?"	No. Tutti i pixel di una stessa classe ricevono la stessa etichetta.
Per istanza	"Quali pixel appartengono a ogni singolo oggetto?"	Sì. Ogni oggetto riceve un identificatore proprio.
Panottica	Combina i due precedenti.	Sì. Ogni pixel riceve una classe e, quando applicabile, un identificatore di istanza.

Nella segmentazione **semantica**, ogni pixel riceve solo l'etichetta della propria classe. Nella segmentazione **per istanza**, oltre alla classe, oggetti distinti appartenenti alla stessa categoria vengono differenziati tra loro. La segmentazione **panottica**, invece, combina queste due informazioni, assegnando a ogni pixel una classe e, quando applicabile, un identificatore di istanza.

8.8.1 Segmentazione Semantica per Soglia

Prima della diffusione dei metodi basati su reti neurali profonde — tema del **Capitolo 9** —, molte applicazioni di segmentazione erano risolte con tecniche classiche di elaborazione delle immagini. Uno degli esempi più semplici consiste nel separare oggetto e sfondo mediante **soglia**, producendo una maschera binaria. Tale maschera caratterizza una **segmentazione semantica**, poiché ogni pixel appartiene a una delle due classi: **moneta** o **sfondo**.

La Figura 8.12 utilizza l'immagine di monete della libreria `scikit-image`. Questo esempio riprende i concetti di soglia e morfologia matematica studiati nel **Capitolo 4**, in sostituzione dell'esempio precedente, che impiegava un'altra immagine di monete.

Inizialmente, si applica una **chiusura morfologica** per attenuare piccole imperfezioni sulla superficie delle monete. Successivamente, la **soglia di Otsu** produce la maschera binaria che separa monete e sfondo. Dopo la rimozione degli oggetti connessi al bordo dell'immagine, il riempimento delle regioni interne e l'eliminazione di piccoli rumori tramite operazioni morfologiche, si ottiene una maschera semantica adeguata per le fasi successive.

8.8.2 Segmentazione delle istanze mediante etichettatura

Una maschera semantica fornisce solo la classe di ciascun pixel, ma non distingue oggetti diversi appartenenti alla stessa categoria. Per ottenere una **segmentazione delle istanze**, si applica l'**etichettatura delle componenti connesse**, che assegna un identificatore diverso a ciascuna regione connessa della maschera binaria.

In questo esempio, ogni componente connessa corrisponde a una singola moneta. Pertanto, l'etichettatura produce una segmentazione delle istanze, consentendo di identificare, contare e misurare separatamente ciascuna moneta presente nell'immagine.

```
img_moedas = skdata.coins()

# 1. Chiusura morfologica: smussa i bordi e riempie piccole imperfezioni sulla
#    superficie delle monete prima della sogliatura
fechamento = mm.close(img_moedas, mm.sedisk(4))

# 2. Sogliatura di Otsu (Capitolo 4): separa le monete (chiare) dallo sfondo (scuro)
mascara_bruta = mm.threshold(fechamento)

# 3. Rimuove gli oggetti collegati al bordo dell'immagine - le monete tagliate ai
```

```
#    bordi non formano istanze complete e ostacolerebbero il conteggio
mascara_sem_borda = mm.edgeoff(mascara_bruta)

# 4. Chiusura dei buchi: riempie eventuali regioni interne non rilevate
#    dalla soglia, garantendo che ogni moneta sia un disco solido
mascara_semantica = mm.clohole(mascara_sem_borda)

# 5. Apertura morfologica: rimuove il rumore residuo e scollega le monete che
#    eventualmente si toccano, preparando la maschera per l'Etichettatura
mascara_limpa = mm.open(mascara_semantica, mm.sedisk(6))

# 6. Etichettatura di componenti connessi: ogni moneta isolata riceve un
#    identificatore di istanza distinto
rotulos_instancias = mm.label(mascara_limpa)

print(f"Istanze (monete individuali) identificate: {np.max(rotulos_instancias)}")

mm.show(
    [img_moedas, mascara_bruta, mascara_sem_borda, mascara_semantica,
     mascara_limpa, rotulos_instancias],
    titles=[
        "Originale",
        "1. Sogliatura (Otsu)",
        "2. Rimozione dei bordi",
        "3. Riempimento dei buchi\n(maschera semantica: moneta vs sfondo)",
        "4. Apertura (pulizia)",
        "5. Etichettatura\n(segmentazione di istanze)"
    ],
    cols=3, figsize=(10, 6)
)
```

```
Istanze (monete individuali) identificate: 24
```

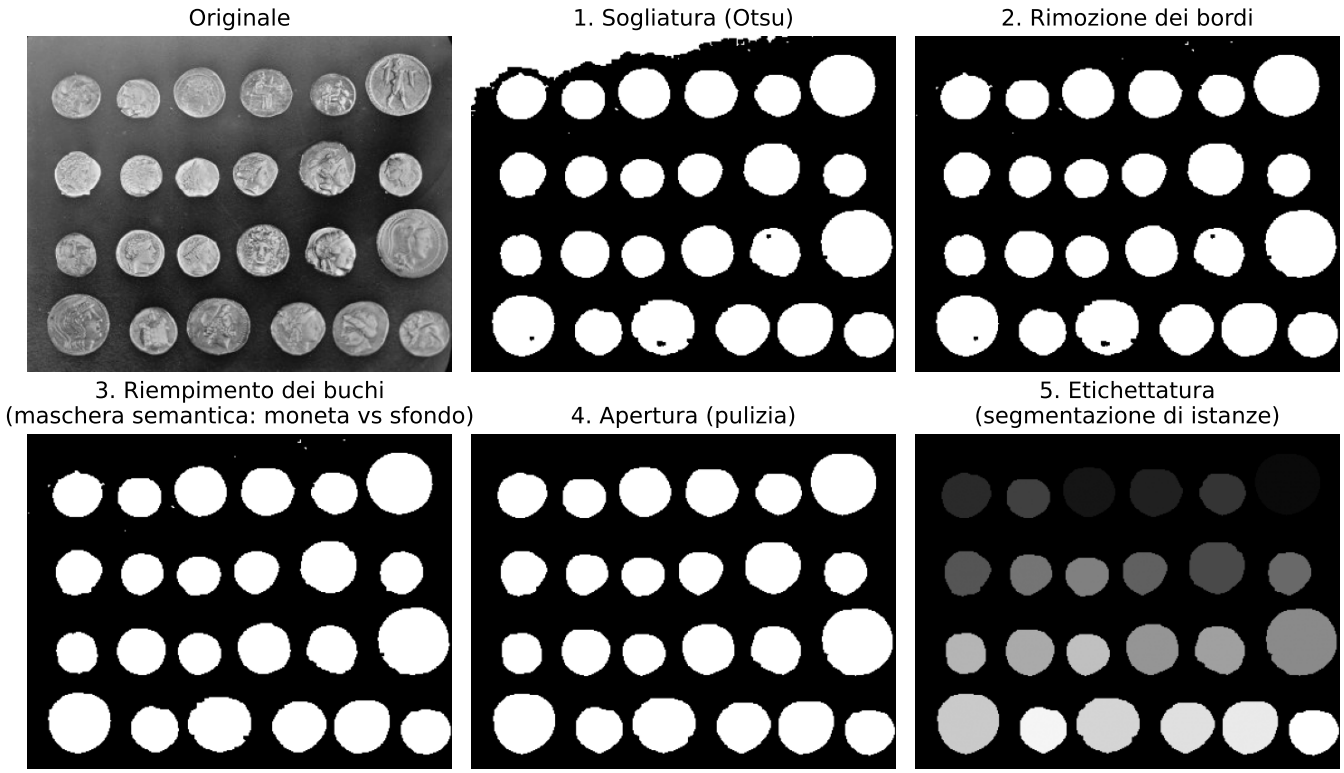


Figura 8.12: Segmentazione classica (non basata su deep learning): maschera semantica (moneta vs sfondo) tramite sogliatura di Otsu, e segmentazione di istanze tramite etichettatura di componenti connessi.

Perché funziona? — E dove l'approccio classico incontra i propri limiti

In questo esempio, la segmentazione è facilitata dal contrasto tra le monete e lo sfondo e dalla relativa omogeneità di intensità all'interno di ciascuna moneta. La sogliatura di Otsu separa efficacemente queste due regioni, mentre le operazioni morfologiche rimuovono piccole imperfezioni dalla maschera binaria. Infine, l'etichettatura delle componenti connesse assegna un identificatore distinto a ciascuna regione connessa, producendo una segmentazione delle istanze.
Questa strategia, tuttavia, dipende direttamente dalla qualità della maschera binaria. Se due oggetti sono uniti, sovrapposti o presentano un contrasto insufficiente rispetto allo sfondo, potrebbero essere rappresentati da un'unica regione o non essere segmentati correttamente. Inoltre, i metodi basati prevalentemente sull'intensità dei pixel hanno una capacità limitata di distinguere oggetti di classi diverse con aspetto simile.
I metodi moderni di segmentazione basati su reti neurali profonde apprendono rappresentazioni visive direttamente dai dati di addestramento, il che generalmente conferisce loro una maggiore capacità di gestire variazioni di illuminazione, texture, forma e occlusione. La segmentazione panottica amplia questo approccio combinando, in un'unica rappresentazione, la classificazione semantica di tutti i pixel e l'identificazione individuale degli oggetti presenti nella scena.

8.8.3 Alternativa: Trasformata della Distanza + *Watershed*

L'apertura morfologica può separare monete che si toccano mediante l'erosione della maschera binaria. Tuttavia, tale operazione modifica il contorno di tutti gli oggetti, inclusi quelli già isolati. Un'alternativa, introdotta nel **Capitolo 4**, consiste nel combinare la **trasformata della distanza** con l'algoritmo ***watershed***, utilizzando marcatori ottenuti dalla stessa maschera binaria.

La procedura si articola in tre fasi:

1. si calcola la **trasformata della distanza** della maschera binaria, assegnando a ciascun pixel dell'oggetto

la sua distanza dallo sfondo più vicino;
2. si applica una soglia alla trasformata della distanza per ottenere **marcatori** situati nelle regioni centrali delle monete;
3. si utilizza l'algoritmo ***watershed*** per espandere questi marcatori fino ai confini tra gli oggetti, separando le monete che si trovano in contatto.

La Figura 8.13 presenta queste fasi. I marcatori sono ottenuti nelle regioni centrali della trasformata della distanza e utilizzati come semi per il *watershed*, che propaga ciascuna etichetta fino a incontrare i confini tra oggetti vicini.

Rispetto all'apertura morfologica, entrambi gli approcci riescono a separare monete in contatto. Tuttavia, poiché il *watershed* utilizza marcatori per dividere regioni connesse, senza applicare erosione direttamente alla maschera, i contorni originali tendono a essere meglio preservati, favorendo l'ottenimento di misure geometriche, come area, perimetro e circolarità.

```
# Riutilizza la maschera semantica PRIMA dell'apertura aggressiva (senza erosione del
contorno)
mascara_base = mascara_semantica

# 1. Trasformata della distanza: ogni pixel della maschera riceve la distanza fino allo sfondo
vicino
distancia = mm.dist(mascara_base)

# Marcatori ottenuti tramite sogliatura della trasformata della distanza.
# Le regioni centrali delle monete rimangono connesse e vengono utilizzate
# come semi per l'algoritmo watershed.
marcadores = mm.label(
    np.uint8(distancia > 0.5 * distancia.max())
)

# 3. Watershed: propaga ogni marcatore all'interno della maschera fino ai punti di contatto
tra monete
rotulos_watershed = mm.watershed(marcadores, mascara_base)

print(f"Apertura morfologica (maschera_pulita): {np.max(rotulos_instancias)} istanze")
print(f"Distanza + watershed: {np.max(rotulos_watershed)} istanze")

mm.show(
    [mascara_base, distancia, mm.dil(marcadores,mm.sedisk(5)), rotulos_watershed],
    titles=[
        "Maschera semantica\n(senza apertura, senza erosione)",
        "Trasformata della distanza",
        "Marcatori\n(massimi regionali)",
        "Watershed\n(istanze separate)"
    ],
    cols=4, figsize=(11, 3.2)
)
```

```
Apertura morfologica (maschera_pulita): 24 istanze
Distanza + watershed: 24 istanze
```

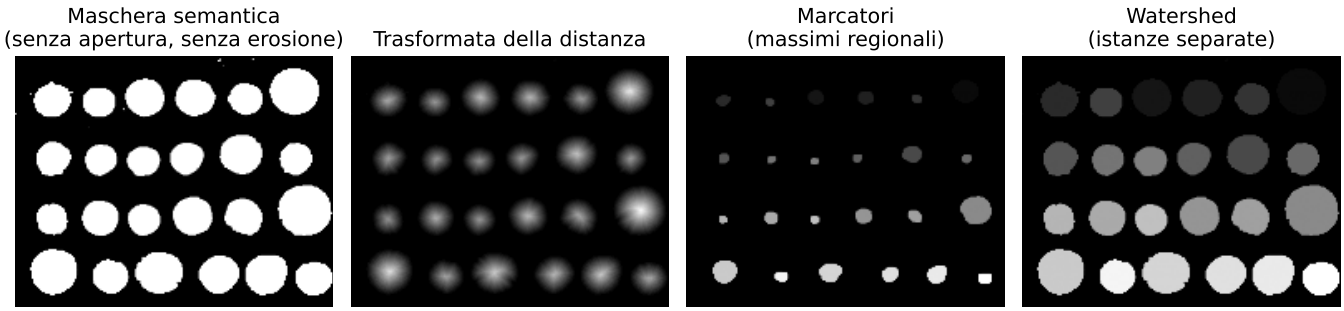


Figura 8.13: Separazione di monete che si toccano tramite trasformata della distanza + *watershed*, senza erosione del contorno delle monete.

> **Perché funziona? — Marcatori e *Watershed***
>
> La trasformata della distanza attribuisce valori più alti ai pixel più lontani dallo sfondo, che tipicamente si trovano nelle regioni più centrali degli oggetti. Applicando una soglia a questa trasformata, si ottengono marcatori localizzati all'interno di ogni moneta, favorendo l'ottenimento di un marcatore per ciascun oggetto.
> L'algoritmo ***watershed*** utilizza questi marcatori come semi e propaga le loro etichette finché due regioni in crescita non si incontrano. I punti di incontro tra queste regioni definiscono i confini tra oggetti adiacenti.
> Le prestazioni di questo metodo dipendono dalla qualità dei marcatori. Oggetti molto allungati, con forme irregolari o contenenti più massimi nella trasformata della distanza possono generare marcatori aggiuntivi, risultando in una segmentazione eccessiva (*oversegmentation*). In applicazioni più complesse, i moderni metodi di segmentazione delle istanze basati su reti neurali profonde apprendono direttamente, a partire dai dati di addestramento, rappresentazioni adeguate per separare gli oggetti, eliminando la necessità di costruire esplicitamente marcatori e altre euristiche geometriche.

8.8.4 Misurazione ed Estrazione di Attributi con `mm.measure`

Dopo la segmentazione e l'etichettatura degli oggetti, il passo successivo consiste nel misurarne le proprietà geometriche. A tal fine, la libreria `morph` mette a disposizione la funzione `mm.measure`, che riceve un'immagine **binaria** e restituisce, per ogni oggetto, un insieme di descrittori geometrici organizzato in una lista di dizionari.

Tra i descrittori calcolati spiccano l'**area**, il **perimetro**, il **centroide**, il **riquadro delimitatore** (*bounding box*), la **circolarità**, la **solidità** e il numero di **vertici** del contorno approssimato. Tali vertici sono ottenuti approssimando il contorno tramite un poligono semplificato, calcolato dalla funzione `approxPolyDP`, che preserva la forma generale dell'oggetto utilizzando un numero ridotto di segmenti.

La circolarità è definita da

$$\text{Circolarità} = \frac{4\pi \cdot \text{Area}}{\text{Perimetro}^2},$$

assumendo valore pari a 1 per un cerchio perfetto e valori inferiori per forme progressivamente meno circolari.

La solidità è data da

$$\text{Solidità} = \frac{\text{Area}}{\text{Area dell'Inviluppo Convesso}},$$

indicando quanto l'oggetto riempie il proprio inviluppo convesso. Valori prossimi a 1 caratterizzano oggetti convessi, mentre valori inferiori indicano la presenza di concavità.

Il numero di vertici fornisce un'indicazione della complessità della forma dell'oggetto. Ad esempio, un triangolo tende a produrre tre vertici e un rettangolo quattro, mentre oggetti con contorno curvo, come le monete, solitamente risultano in poligoni con un numero maggiore di vertici, a seconda della precisione adottata nell'approssimazione.

Poiché le monete di questo esempio presentano contorni approssimativamente circolari e convessi, ci si aspetta che la circolarità assuma valori elevati (tipicamente tra 0,8 e 0,9) e che la solidità rimanga molto vicina a 1. Il numero di vertici dipende dal parametro utilizzato nell'approssimazione poligonale (`precision`), aumentando man mano che l'approssimazione preserva più dettagli del contorno. Nell'insieme, questi descrittori possono essere impiegati nella classificazione degli oggetti, nell'identificazione di componenti spuri residui dalla segmentazione e nell'esecuzione di misurazioni quantitative sull'immagine.

```
# mm.measure riceve un'immagine binaria; ogni componente connesso corrisponde
# a una moneta individuale.
medidas = mm.measure(mascara_limpa, precision=0.01)

if medidas:
    print(f"Totale degli oggetti catalogati: {len(medidas)}")
    print(f"{'ID':<4}{'Área':<10}{'Perímetro':<12}{'Circularidade':<14}",end="")
    print(f" {'Solidez':<10}{'Vértices':<8}")
    for m in medidas:
        print(f"{m['id']:<4}{m['area']:<10.1f}{m['perimeter']:<12.2f}"
              f"{m['circularity']:<15.3f}{m['solidity']:<10.3f}{m['vertices']:<8}")
else:
    print("Nessuna misura restituita.")
```

```
Totale degli oggetti catalogati: 24
ID  Área      Perímetro   Circularidade  Solidez   Vértices
1   1376.5    140.47      0.877          0.979     11
2   1299.5    136.47      0.877          0.978     10
3   1628.0    154.37      0.859          0.980     12
4   1866.0    162.71      0.886          0.985     11
5   2127.0    178.37      0.840          0.973     12
6   2328.0    181.68      0.886          0.983     10
7   1086.5    124.81      0.876          0.979     13
8   1053.5    121.50      0.897          0.983     9
9   1399.5    140.47      0.891          0.985     13
10  1449.0    147.05      0.842          0.972     13
11  1642.5    153.30      0.878          0.982     13
12  2987.5    207.58      0.871          0.981     13
13  1036.5    121.15      0.887          0.981     12
14  1050.5    123.15      0.870          0.979     11
15  1110.0    126.23      0.875          0.975     11
16  1135.0    127.05      0.884          0.986     11
17  1250.5    135.64      0.854          0.975     13
18  1802.0    160.71      0.877          0.982     10
19  1067.5    122.33      0.896          0.986     11
20  1154.5    129.64      0.863          0.977     10
21  1356.0    139.88      0.871          0.978     13
22  1549.5    149.30      0.874          0.982     12
23  1592.0    149.54      0.895          0.984     13
24  2509.5    187.92      0.893          0.987     14
```

Si osserva che sono state identificate **24 monete**. Le misure di solidità rimangono vicine a 1, indicando contorni essenzialmente convessi, mentre la circolarità varia tra circa 0,84 e 0,90 a causa delle irregolarità del contorno discretizzato. Il numero di vertici varia tra 9 e 14, riflettendo l'approssimazione poligonale utilizzata per rappresentare ciascuna moneta.

8.8.5 Esportazione delle Annotazioni: da `mm.measure` al Formato YOLO

Oltre ai descrittori geometrici, `mm.measure` calcola, per ogni oggetto, la sua **bounding box** (*riquadro delimitatore*). Queste informazioni possono essere esportate automaticamente tramite la funzione `mm.saveMeasures`, consentendo di generare file di annotazione per diverse applicazioni senza la necessità di etichettatura manuale.

La funzione supporta tre formati di output:

- `fmt="csv"`: esporta tutti i descrittori geometrici, risultando utile per analisi esplorativa, misurazione e classificazione basata su attributi;
- `fmt="txt"`: registra i descrittori in formato tabellare semplice;
- `fmt="yolo"`: genera annotazioni compatibili con il formato utilizzato dai rilevatori della famiglia **YOLO** (*You Only Look Once*).

Nel formato YOLO, ogni oggetto è rappresentato da una riga contenente

$$\text{classe} \;\; x_c \;\; y_c \;\; w_n \;\; h_n,$$

dove (x_c, y_c) rappresenta il centro del riquadro delimitatore e (w_n, h_n) le sue dimensioni, tutti normalizzati rispetto alle dimensioni dell'immagine:

$$x_c = \frac{x + w/2}{W}, \qquad y_c = \frac{y + h/2}{H}, \qquad w_n = \frac{w}{W}, \qquad h_n = \frac{h}{H},$$

dove (x, y) corrispondono alle coordinate dell'angolo superiore sinistro del riquadro delimitatore, (w, h) alle sue dimensioni in pixel e (W, H) alla larghezza e all'altezza dell'immagine. La normalizzazione rende le annotazioni indipendenti dalla risoluzione dell'immagine, consentendo di utilizzare lo stesso formato per immagini di dimensioni diverse.

Il codice seguente esporta le misure estratte dalle monete nei formati **CSV** e **YOLO**. L'output in CSV preserva tutti i descrittori geometrici calcolati da `mm.measure`, mentre il file in formato YOLO contiene solo la classe e il riquadro delimitatore normalizzato di ciascun oggetto, come richiesto dai rilevatori di questa famiglia.

Questa rappresentazione sarà utilizzata nuovamente nel **Capitolo 9**, dedicato ai metodi di rilevamento degli oggetti basati su *Deep Learning*.

```python
# Esporta le misure estratte dalle monete in due formati di annotazione
altura_img, largura_img = img_moedas.shape[:2]

mm.saveMeasures("moedas.csv", medidas, fmt="csv")
mm.saveMeasures("moedas.txt", medidas, fmt="yolo",
                 img_width=largura_img, img_height=altura_img, class_id=0)

print("--- Estratto di moedas.csv ---")
with open("moedas.csv") as f:
    for linha in f.readlines()[:4]:
        print(linha.rstrip())

print("\n--- Estratto di moedas.txt (formato YOLO) ---")
with open("moedas.txt") as f:
    for linha in f.readlines()[:4]:
        print(linha.rstrip())
```

```
--- Estratto di moedas.csv ---
id,area,perimeter,cx,cy,x,y,w,h,circularity,solidity,vertices
1,1376.5,140.47,359.57,268.85,338,250,45,41,0.877,0.979,11
2,1299.5,136.47,115.16,266.56,95,247,42,42,0.877,0.978,10
```

```
3,1628.0,154.37,245.55,264.02,222,243,49,46,0.859,0.98,12

--- Estratto di moedas.txt (formato YOLO) ---
0 0.938802 0.892739 0.117188 0.135314
0 0.302083 0.884488 0.109375 0.138614
0 0.641927 0.877888 0.127604 0.151815
0 0.789062 0.877888 0.130208 0.158416
```

8.8.6 Validazione delle Annotazioni tramite IoU: `mm.verifyBoundBox`

Prima di utilizzare le annotazioni nell'addestramento o nella valutazione dei modelli, è importante verificare che esse concordino con un insieme di riferimento (*ground truth*).

La funzione `mm.verifyBoundBox` esegue questo confronto utilizzando la metrica **IoU** (*Intersection over Union*), presentata in precedenza in questo capitolo. Per ogni bounding box candidata, la funzione calcola la sua sovrapposizione con le box del riferimento e conta quelle il cui IoU supera una soglia specificata.

Nell'esempio seguente, viene costruito un **riferimento sintetico** a partire dalle prime cinque bounding box ottenute tramite `mm.measure`. Poiché le annotazioni confrontate sono le stesse utilizzate per costruire il riferimento, ci si aspetta che ogni oggetto trovi esattamente una corrispondenza con IoU $\geq 0{,}5$. L'obiettivo è solo illustrare il funzionamento della funzione `mm.verifyBoundBox`; in applicazioni reali, il riferimento deve essere ottenuto in modo indipendente, ad esempio tramite annotazione manuale.

```python
# Matrice di gabariti sintetica nel formato (classe, x1, y1, x2, y2), normalizzato
gabaritos = np.array([
    [0, *[(m["bbox"][0] + d) / largura_img for d in (0,)],
        (m["bbox"][1]) / altura_img,
        (m["bbox"][0] + m["bbox"][2]) / largura_img,
        (m["bbox"][1] + m["bbox"][3]) / altura_img]
    for m in medidas[:5]
])

acertos = 0
for m in medidas[:5]:
    correspondencias = mm.verifyBoundBox(
        object_id=0, bbox=m["bbox"], matrix=gabaritos,
        width=largura_img, height=altura_img, threshold=0.5
    )
    acertos += int(correspondencias > 0)
    print(f"Oggetto {m['id']}: {correspondencias} gabarito/i corrisponde con IoU >= 0.5")

print(f"\nTotale oggetti validati: {acertos}/{len(medidas[:5])}")
```

```
Oggetto 1: 1 gabarito/i corrisponde con IoU >= 0.5
Oggetto 2: 1 gabarito/i corrisponde con IoU >= 0.5
Oggetto 3: 1 gabarito/i corrisponde con IoU >= 0.5
Oggetto 4: 1 gabarito/i corrisponde con IoU >= 0.5
Oggetto 5: 1 gabarito/i corrisponde con IoU >= 0.5

Totale oggetti validati: 5/5
```

Si osserva che i cinque oggetti sono stati correttamente associati ai rispettivi template, ottenendo cinque corrispondenze valide. In applicazioni reali, questa stessa strategia può essere impiegata per confrontare automaticamente le bounding box prodotte da un rilevatore con le annotazioni di riferimento, consentendo di valutare quantitativamente le sue prestazioni tramite la metrica IoU.

8.8.7 Visualizzazione delle Annotazioni Salvate: `mm.showBoundBox`

Dopo aver esportato le annotazioni, è opportuno verificare che le bounding box siano state registrate correttamente. A tal fine, la libreria `morph` mette a disposizione la funzione `mm.showBoundBox`, illustrata nella Figura 8.14.

La funzione legge un file di annotazioni nei formati `yolo`, `csv` o `txt`, ricostruisce le **bounding box** e le sovrappone all'immagine originale, facilitando l'ispezione visiva del risultato. In questo modo, è possibile confermare rapidamente se le annotazioni sono allineate agli oggetti, senza la necessità di esaminarle manualmente.

Questa funzione completa `mm.verifyBoundBox`. Mentre `mm.verifyBoundBox` esegue una **validazione quantitativa**, confrontando le annotazioni con un insieme di riferimento tramite la metrica IoU, `mm.showBoundBox` fornisce una **validazione qualitativa**, consentendo di ispezionare visivamente le bounding box ricostruite.

```
caminho_yolo = "moedas.txt"
img_anotada = mm.showBoundBox(img_moedas, caminho_yolo, fmt="yolo")
```

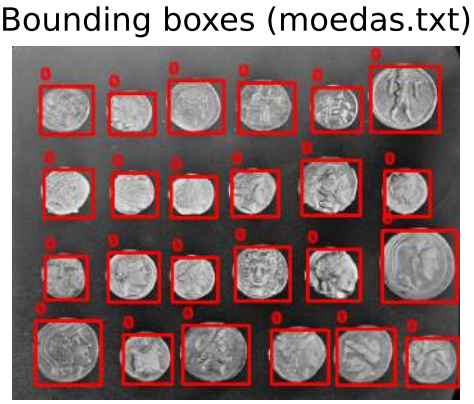


Figura 8.14: *Bounding boxes* ricostruite a partire dal file di annotazioni YOLO esportato da *mm.saveMeasures*, sovrapposte all'immagine originale delle monete.

> **Perché funziona? — Dalla segmentazione alle annotazioni**
>
> Dopo aver segmentato un'immagine, è possibile misurare automaticamente ogni oggetto (`mm.measure`) e convertire queste misurazioni in annotazioni (`mm.saveMeasures`) per diversi formati, incluso quello utilizzato dai rilevatori della famiglia YOLO. Questa procedura consente di generare automaticamente insiemi iniziali di annotazioni in scenari controllati, riducendo significativamente il lavoro di etichettatura manuale.
> Prima di utilizzare queste annotazioni nell'addestramento o nella valutazione dei modelli, si raccomanda di confrontarle con un insieme di riferimento (*ground truth*). La funzione `mm.verifyBoundBox` automatizza questa fase tramite la metrica **IoU**, consentendo di quantificare la concordanza tra le bounding box generate e le annotazioni di riferimento. Complementarmente, `mm.showBoundBox` permette un'ispezione visiva delle annotazioni ricostruite sull'immagine originale, facilitando l'identificazione di errori di normalizzazione, posizionamento o ordinamento delle coordinate che potrebbero non essere evidenti in una validazione puramente numerica.
> Nel loro insieme, `mm.measure`, `mm.saveMeasures`, `mm.verifyBoundBox` e `mm.showBoundBox` implementano un flusso completo per misurare oggetti segmentati, generare annotazioni, valutarle e ispezionarle visivamente. Quando la segmentazione produce risultati affidabili, questo flusso può ridurre significativamente la necessità di etichettatura manuale nella costruzione di insiemi di dati.

8.9 Panoramica dei Modelli Moderni

Le tecniche presentate in questo capitolo — come i rilevatori a cascata, la segmentazione basata su sogliatura e il post-processing tramite IoU e NMS — rimangono importanti per comprendere i fondamenti della Visione Computazionale. Attualmente, tuttavia, molte applicazioni utilizzano modelli di **Deep Learning**, capaci di apprendere automaticamente rappresentazioni discriminative a partire da grandi insiemi di dati, senza richiedere la definizione manuale delle caratteristiche.

La Tabella 8.2 presenta alcune architetture rappresentative per il rilevamento e la segmentazione delle immagini. L'obiettivo è soltanto collocarle nel contesto dei compiti studiati in questo capitolo. I loro principi di funzionamento, addestramento e applicazione saranno discussi in dettaglio nel **Capitolo 9**.

Tabella 8.2: Architetture rappresentative di Deep Learning per il rilevamento e la segmentazione delle immagini.

Modello	Compito	Idea centrale
YOLO (*You Only Look Once*)	Rilevamento di oggetti	Rileva gli oggetti in un'unica passata attraverso la rete, stimando classi e bounding box. Le versioni recenti, come **YOLO26** (2026), eliminano la fase di *Non-Maximum Suppression* (NMS), rendendo l'inferenza completamente *end-to-end*.
Faster R-CNN	Rilevamento di oggetti	Genera regioni candidate e le rifinisce prima della classificazione, privilegiando la precisione.
SSD (*Single Shot Detector*)	Rilevamento di oggetti	Rileva gli oggetti a diverse scale in un'unica passata, cercando un equilibrio tra velocità e precisione.
U-Net	Segmentazione semantica	Produce una maschera che classifica ogni pixel dell'immagine in base alla sua classe.
Mask R-CNN	Segmentazione di istanze	Estende Faster R-CNN aggiungendo una maschera individuale per ciascun oggetto rilevato.
***Segment Anything* (SAM)**	Segmentazione guidata da *prompt*	Segmenta gli oggetti a partire da indicazioni come punti, bounding box o maschere. Si è evoluto in **SAM 2**, con supporto video e tracciamento temporale degli oggetti, e in **SAM 3**, che consente una segmentazione guidata direttamente da descrizioni testuali.

Si osserva un'evoluzione dei compiti affrontati in questo capitolo. Modelli come **YOLO**, **Faster R-CNN** e **SSD** localizzano gli oggetti tramite bounding box. **Mask R-CNN** amplia questa capacità producendo anche una maschera per ciascuna istanza rilevata. Modelli più recenti, come ***Segment Anything* (SAM)**, consentono invece di segmentare gli oggetti a partire da diversi tipi di *prompt*, rendendo il processo più flessibile.

Questa evoluzione segue la sequenza di concetti sviluppata lungo il capitolo: dalla corrispondenza delle

caratteristiche e dalla localizzazione approssimativa tramite bounding box fino alla segmentazione precisa dei pixel appartenenti a ciascun oggetto. Nel **Capitolo 9**, questi compiti saranno ripresi dalla prospettiva del Deep Learning, esplorando come le reti neurali moderne apprendano automaticamente rappresentazioni in grado di superare molti dei limiti dei metodi classici qui presentati.

8.10 Limiti degli Approcci Classici e Motivazione per il *Deep Learning*

Gli esempi sviluppati in questo capitolo illustrano tre problemi fondamentali della Visione Artificiale: **corrispondenza delle feature**, **rilevamento degli oggetti** e **segmentazione delle immagini**. Sebbene le tecniche classiche presentate siano efficienti in diversi scenari, tutte condividono una limitazione importante: si basano su caratteristiche definite manualmente per descrivere o identificare gli oggetti di interesse.

- **ORB** utilizza descrittori binari in grado di stabilire corrispondenze tra punti sotto rotazioni e variazioni moderate di scala, ma non è stato concepito per riconoscere categorie di oggetti.
- **Haar Cascade** impiega un insieme fisso di filtri rettangolari, funzionando bene per oggetti con aspetto relativamente standardizzato, come i volti frontali, ma risultando più suscettibile a falsi positivi in scene complesse.
- La **segmentazione per sogliatura** sfrutta le differenze di intensità tra oggetto e sfondo, essendo adatta per immagini con buon contrasto, ma limitata di fronte a variazioni di illuminazione, trame o scene contenenti molteplici classi di oggetti.

In tutti questi casi, le prestazioni dipendono dalla capacità delle caratteristiche scelte di rappresentare adeguatamente la variabilità presente nelle immagini. Nelle applicazioni reali, fattori come variazioni di posa, illuminazione, scala, occlusioni e diversità delle categorie rendono questo compito sempre più difficile, riducendo la generalizzazione dei metodi classici.

Il **Capitolo 9** presenta un approccio diverso a questi problemi attraverso le **Reti Neurali Convoluzionali (CNN)**. Invece di utilizzare caratteristiche definite manualmente, questi modelli apprendono automaticamente, a partire da grandi insiemi di dati, rappresentazioni adeguate per ciascun compito. Le architetture moderne presentate nella sezione precedente — come **YOLO**, **Faster R-CNN**, **U-Net**, **Mask R-CNN** e ***Segment Anything* (SAM)** — seguono questo principio e rappresentano un'evoluzione delle tecniche classiche studiate in questo capitolo, ampliando la loro capacità di gestire la diversità e la complessità delle immagini del mondo reale.

8.11 Riassunto

In questo capitolo sono stati studiati metodi classici di Visione Artificiale per la **corrispondenza di caratteristiche**, la **rilevazione di oggetti** e la **segmentazione delle immagini**. Al termine, sono state presentate alcune architetture moderne basate sull'Apprendimento Profondo, preparando la transizione al **Capitolo 9**. I principali concetti affrontati sono stati:

- **ORB:** rilevazione di punti chiave, costruzione di descrittori binari e corrispondenza di caratteristiche tramite la distanza di Hamming.
- **Omografia e RANSAC:** stima robusta di trasformazioni proiettive per la registrazione delle immagini e l'eliminazione di corrispondenze incoerenti.
- **Haar Cascade:** rilevazione di oggetti utilizzando caratteristiche di Haar, immagine integrale e classificatori a cascata.
- **Segmentazione classica:** sogliatura, operazioni morfologiche, etichettatura delle componenti connesse, trasformata della distanza e algoritmo *watershed* per la segmentazione delle istanze.
- **Misurazione e annotazione di oggetti:** estrazione di attributi geometrici con `mm.measure`, esportazione automatica di annotazioni (`mm.saveMeasures`) e validazione tramite IoU (`mm.verifyBoundBox`) e ispezione visiva (`mm.showBoundBox`).
- **Valutazione delle rilevazioni:** utilizzo della metrica **Intersection over Union (IoU)** e della **Soppressione dei Non-Massimi (NMS)**.

- **Modelli moderni:** panoramica delle architetture **YOLO**, **Faster R-CNN**, **SSD**, **U-Net**, **Mask R-CNN** e **Segment Anything (SAM)**, che serviranno come base per lo studio dei metodi di Apprendimento Profondo nel **Capitolo 9**.

Prossimi Passi

In questo capitolo sono stati presentati metodi classici per risolvere problemi di **corrispondenza delle caratteristiche**, **rilevamento degli oggetti** e **segmentazione delle immagini**, basati su descrittori, filtri e modelli geometrici definiti manualmente.

Nel **Capitolo 9**, questi stessi problemi saranno rivisitati dalla prospettiva dell'**Apprendimento Profondo**, con particolare enfasi sulle **Reti Neurali Convoluzionali (CNN)**. Invece di utilizzare caratteristiche progettate manualmente, questi modelli apprendono automaticamente rappresentazioni a partire da grandi insiemi di dati, costituendo la base dei principali sistemi moderni di Visione Artificiale per il rilevamento, la segmentazione e il riconoscimento degli oggetti.

8.12 🤖 Uso del Gemini Notebook come Supporto allo Studio

Il **Gemini Notebook** può essere utilizzato come strumento complementare per rivedere i concetti presentati in questo capitolo. A partire dal contenuto reso disponibile come riferimento, esso consente di rispondere a domande, elaborare riassunti, chiarire dubbi ed esplorare gli argomenti in modo interattivo.

Accesso al Notebook del Capitolo

🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 08

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato realizzato esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando dall'edizione in inglese o francese, oppure seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.
Le risposte generate dal Gemini Notebook sono prodotte automaticamente e possono contenere imprecisioni. Utilizzale come materiale complementare di studio, confrontando le informazioni con il contenuto di questo libro e, quando necessario, con altre fonti accademiche.

8.13 Elenco di Esercizi

Gli esercizi seguenti consolidano i concetti presentati in questo capitolo tramite adattamenti, esperimenti ed estensioni degli algoritmi sviluppati nel corso del testo, utilizzando la libreria didattica `morph`.

1. **(10%)** Investigare l'influenza delle rotazioni sulla stabilità del detector e descrittore **ORB**. Utilizzando l'immagine `skimage.data.astronaut()`, generare versioni ruotate ad angoli di $0°, 45°, 90°, 135°, 180°$. Per ogni caso, determinare il numero di corrispondenze ottenute dal `BFMatcher` e la frazione di *inlier* identificati dal **RANSAC**. Presentare i risultati in una tabella e discutere la robustezza del metodo alle rotazioni analizzate.

2. **(15%)** Confrontare il detector di angoli **FAST** con il detector di **Harris** (Capitolo 6). Misurare il tempo medio di esecuzione e il numero di punti rilevati nelle immagini `skimage.data.camera()`, `skimage.data.gravel()` e `skimage.data.brick()`. Discutere i vantaggi e i limiti di ciascun detector in termini di velocità e ripetibilità.

3. **(15%)** Investigare la sensibilità della **Cascata di Haar** alla presenza di rumore e sfocatura. Aggiungere rumore gaussiano ($\sigma \in 5, 10, 20$) e sfocatura da movimento all'immagine `skimage.data.astronaut()`.

Variare i parametri `scaleFactor` e `minNeighbors` della funzione `detectMultiScale()` e discutere gli effetti su falsi positivi, falsi negativi e numero di volti rilevati.

4. **(15%)** Implementare manualmente la metrica **Intersection over Union (IoU)** per bounding box e confrontare i risultati con la funzione `mm.IoU`. Successivamente, creare una box di riferimento e dieci box ottenute tramite traslazioni e variazioni di scala, costruendo un grafico che relazioni la traslazione applicata con il valore di IoU ottenuto.

5. **(15%)** Valutare l'algoritmo di **Soppressione Non-Massimale (NMS)**. Creare una scena sintetica contenente tre oggetti, ciascuno rappresentato da almeno cinque box sovrapposte con diversi punteggi di confidenza. Eseguire l'algoritmo con soglie IoU pari a $0.2, 0.4, 0.6, 0.8$, visualizzare i risultati e discutere l'influenza di questo parametro sulla quantità di box mantenute.

6. **(15%)** Applicare la pipeline di segmentazione presentata in questo capitolo (soglia di Otsu, operazioni morfologiche e `mm.measure`) all'immagine `skimage.data.coins()`. Esportare le misure utilizzando `mm.saveMeasures(fmt="csv")` e produrre istogrammi delle distribuzioni di area, circolarità e solidità. Discutere quali di questi attributi sono più adatti per caratterizzare le monete.

7. **(15%)** Costruire un'immagine sintetica contenente cerchi, rettangoli e triangoli, alcuni parzialmente sovrapposti. Eseguire la segmentazione, estrarre le misure con `mm.measure` e generare automaticamente le annotazioni nel formato YOLO utilizzando `mm.saveMeasures(fmt="yolo")`. Successivamente, modificare deliberatamente alcune bounding box e utilizzare `mm.verifyBoundBox` per valutare, a diverse soglie IoU, quante annotazioni rimangono valide. Discutere l'influenza della sovrapposizione tra oggetti sulla qualità delle annotazioni ottenute.

8. **(Bonus – 10%)** Sviluppare un classificatore semplice, come **k-NN** (Capitolo 7), utilizzando solo gli attributi geometrici prodotti da `mm.measure` (area, circolarità, solidità e numero di vertici). Utilizzare un insieme di forme sintetiche (cerchi, rettangoli e triangoli) con diverse scale e rotazioni, valutare l'accuratezza ottenuta e discutere la capacità di questi descrittori di distinguere diverse classi di oggetti.

Riferimenti del Capitolo

I concetti e gli algoritmi presentati in questo capitolo sono stati fondati su riferimenti classici e contemporanei della letteratura di PDI-VC:

- Gonzalez (2018), per i fondamenti della segmentazione delle immagini, delle operazioni morfologiche, dell'etichettatura delle componenti connesse e dell'estrazione di attributi geometrici.
- Szeliski (2022), per i concetti di corrispondenza delle caratteristiche, trasformazioni geometriche, omografie, rilevamento degli oggetti e segmentazione nella Visione Artificiale.
- Rublee (2011), per la descrizione dell'algoritmo **ORB** (*Oriented FAST and Rotated BRIEF*), utilizzato nel rilevamento, nella descrizione e nella corrispondenza di caratteristiche locali.
- Fischler (1981), per la formulazione originale dell'algoritmo **RANSAC** (*Random Sample Consensus*), impiegato nella stima robusta di modelli geometrici in presenza di dati incoerenti.
- Viola (2001), Viola (2004) e Lienhart (2002), per i fondamenti del rilevatore **Haar Cascade**, incluse le caratteristiche di Haar, l'immagine integrale, i classificatori a cascata e le estensioni utilizzate nelle implementazioni moderne.
- Kirillov (2019), per la definizione della **segmentazione panottica**, che integra la segmentazione semantica e la segmentazione delle istanze in un'unica rappresentazione.
- Redmon (2016), Ren (2015), Ronneberger (2015), He (2016) e Kirillov (2019), per i modelli moderni basati sull'Apprendimento Profondo applicati al rilevamento e alla segmentazione delle immagini, inclusi **YOLO**, **Faster R-CNN**, **U-Net**, **Mask R-CNN** e **Segment Anything (SAM)**.

Executar Colab Abrir GitHub

(Presumo que o texto a seguir seja o conteúdo a ser traduzido; porém, o texto em si não foi fornecido no prompt. Fornecendo-se o texto em Markdown, ele será traduzido conforme as regras acima.)

8.14 💻 Parte Pratica con Esercizi di Programmazione

La presente lista di esercizi di programmazione (EP) consolida le formulazioni teoriche presentate nel Capitolo 8 — Corrispondenza delle Caratteristiche, Rilevamento degli Oggetti e Segmentazione Classica — attraverso un percorso pratico applicato. Come nel capitolo precedente, gli EP isolano le **grandezze intermedie** di ciascuna tecnica — la distanza tra descrittori binari, i termini di un'immagine integrale, il conteggio degli *inlier* di un modello candidato, la sovrapposizione tra bounding box e l'etichetta di ciascuna componente connessa — consentendo di validare manualmente ogni fase del ragionamento senza dipendere da OpenCV né da immagini esterne.

La concatenazione degli esercizi riproduce il flusso concettuale del capitolo: si inizia con la **distanza di Hamming**, cuore della corrispondenza dei descrittori binari come ORB; si prosegue con il conteggio degli ***inlier*** che sostiene **RANSAC** nella stima robusta di un'omografia; si continua con l'**immagine integrale**, l'escamotage computazionale che rende l'***Haar Cascade*** praticabile in tempo reale; si approfondisce con **IoU e Soppressione Non-Massima**, il post-processamento comune a praticamente ogni rilevatore di oggetti; e si conclude con l'**etichettatura delle componenti connesse**, l'approccio classico — e le sue limitazioni — per segmentare istanze individuali in una maschera binaria.

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EPs)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella qui sotto:

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, esegui `TestSuite("EP08_01.estensione").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare il codice Python direttamente, senza salvare un file, usa `run_code(codice)` passando il codice come *stringa* in una variabile `codice`:

```
codice = """
# ... il tuo codice qui ...
"""
TestSuite("EP08_01").run_code(codice)
```

Riepilogo dei Metodi di `morph.py` (Cap. 8)

La libreria `morph.py` fornisce funzioni per l'analisi dei componenti connessi, l'estrazione dei contorni, le metriche geometriche e le annotazioni:

1. **Componenti e Contorni (`connectedComponents`, `findContours`)** Etichettano le regioni connesse ed estraggono i contorni di immagini binarie.
2. **Proprietà del Contorno (`contourArea`, `arcLength`, `convexHull`, `approxPolyDP`, `fitLine`)** Calcolano area, perimetro, inviluppo convesso, approssimazione poligonale e adattamento di una retta per un contorno.
3. **Geometria di Involucro (`boundingRect`, `minAreaRect`, `boxPoints`, `minEnclosingCircle`, `fitEllipse`)** Determinano rettangoli di delimitazione (allineati o orientati), ellissi e il cerchio minimo circoscritto.
4. **Estrazione e Salvataggio delle Misure (`measure`, `saveMeasures`)** Estraggano descrittori geometrici degli oggetti (area, circolarità, solidità, centroide) ed esportano i dati in formato CSV, testo o YOLO.
5. **Valutazione e Visualizzazione (`IoU`, `verifyBoundBox`, `showBoundBox`)** Calcolano l'intersezione su unione (*Intersection over Union*), validano i bounding box con le ground truth e disegnano caselle di delimitazione annotate sull'immagine.

8.14.1 EP08_01 🟢 Distanza di Hamming e Corrispondenza di Descrittori Binari

L'ORB, utilizzato nel Progetto Pratico 1 di questo capitolo, descrive l'intorno di ogni punto di interesse come una sequenza di bit — e, per questo, il confronto tra due descrittori non usa la distanza euclidea del k-NN del Capitolo 7, bensì la **distanza di Hamming**: il numero di posizioni in cui i bit differiscono. Prima di chiamare `cv2.BFMatcher(cv2.NORM_HAMMING)`, ti è stato richiesto di implementare manualmente questa corrispondenza (*matching*) per forza bruta — la stessa fase che, eseguita internamente da OpenCV, precede la stima robusta dell'omografia tramite RANSAC.

8.14.1.1 📋 Linee Guida di Implementazione

1. **Quantità:** Leggere gli interi N e M — numero di descrittori estratti dall'immagine A e dall'immagine B, rispettivamente.
2. **Descrittori di A:** Leggere N righe, ciascuna contenente un descrittore binario (una *stringa* di caratteri `0` e `1`, tutti della stessa lunghezza).
3. **Descrittori di B:** Leggere M righe, nello stesso formato.
4. **Soglia:** Leggere l'intero τ — distanza di Hamming massima accettabile per considerare valida una corrispondenza.
5. **Distanza di Hamming:** Per due descrittori binari a e b della stessa lunghezza,

$$d_H(a, b) = \sum_k \mathbb{1}[a_k \neq b_k],$$

   cioè il conteggio delle posizioni in cui i bit differiscono.
6. **Corrispondenza per vicino più prossimo:** Per ogni descrittore a_i di A (i nell'ordine di lettura, a partire da 0), calcola la sua distanza di Hamming da **tutti** i descrittori di B e trova quello con la distanza minima. In caso di parità tra due o più descrittori di B con la stessa distanza minima, scegli quello con **indice minore**.
7. **Filtraggio tramite soglia:** Se la distanza minima trovata è $\leq \tau$, la corrispondenza è valida; altrimenti, a_i non ha corrispondenza.
8. **Output:** Per ogni i da 0 a $N-1$, nell'ordine di lettura, stampare una riga: `i j d` se esiste una corrispondenza valida (dove j è l'indice del descrittore di B scelto e d la sua distanza), oppure `i -1` in caso contrario. Alla fine, stampare `Total correspondências válidas: X`.

8.14.1.2 📌 Vincoli Computazionali

- **Stessa lunghezza:** tutti i descrittori (di A e di B) hanno esattamente lo stesso numero di bit.

- **Forza bruta:** confronta ogni descrittore di A con **tutti** quelli di B — non è necessaria alcuna indicizzazione o struttura di accelerazione.
- **Pareggio risolto per indice minore in B**, e **mai** per ordine di lettura di A (che è già naturale, poiché ogni a_i è trattato in modo indipendente).

8.14.1.3 🧠 Fondamenti Teorici

Elemento	Ruolo nella corrispondenza ORB
Descrittore binario (BRIEF)	Ogni bit è il risultato di un confronto di intensità tra due pixel dell'intorno
Distanza di Hamming	Metrica di dissimmetria tra *stringhe* binarie; molto più rapida da calcolare rispetto alla distanza euclidea (operazione XOR + conteggio dei bit)
Vicino più prossimo	Criterio di corrispondenza: ogni punto di A è accoppiato al punto di B con descrittore più simile
Soglia τ	Filtra corrispondenze poco affidabili ancor prima del RANSAC — ma, come discusso nel capitolo, alcune corrispondenze errate passano comunque, richiedendo la robustezza del RANSAC

8.14.1.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Interi N e M.
- Prossime N righe: un descrittore binario per riga (*stringa* di `0` e `1`).
- Prossime M righe: un descrittore binario per riga, nello stesso formato.
- Ultima riga: Intero τ.

Output:

- N righe, una per descrittore di A, nel formato `i j d` o `i -1`.
- Ultima riga: `Total correspondências válidas: X`.

8.14.1.5 📌 Esempi

Input	Output	Osservazione
3 3 10101010 11110000 00001111 10101011 00001110 11111111 2	0 0 1 1 -1 2 1 1 Total correspondências válidas: 2	Il descrittore `11110000` non trova corrispondenza: il suo vicino più prossimo è a distanza 4, superiore alla soglia $\tau = 2$.

```
%%writefile EP08_01.py
# Codice Python
```

```
Overwriting EP08_01.py
```

```
TestSuite("EP08_01.py").run()
```

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0801.html

Figura 8.15: Simulatore EP08_01: Distanza di Hamming tra Due Descrittori Binari

```
<IPython.core.display.HTML object>
```

8.14.2 EP08_02 🟢 Omografia e RANSAC: Il Voto per *Inlier*

Il RANSAC, presentato nella sezione "Modellazione Matematica: Omografia e RANSAC", ripete un ciclo di tre passaggi — selezionare un campione minimo, stimare un modello candidato e contare quante corrispondenze sono coerenti con esso (gli ***inlier***) — conservando alla fine il modello più votato. La fase di stima del modello a partire da 4 punti (passo 2) coinvolge algebra lineare che esula dallo scopo di questo EP; qui ricevi direttamente un insieme di omografie **già candidate** — come se ciascuna fosse stata stimata da un campione casuale diverso — e sei incaricato di riprodurre esattamente il passo decisivo dell'algoritmo: **applicare ogni modello a tutte le corrispondenze e contare i suoi *inlier***, scegliendo il vincitore.

8.14.2.1 📋 Linee Guida di Implementazione

1. **Corrispondenze:** Leggere l'intero N e, successivamente, N righe con quattro reali ciascuna, $x\ y\ x'\ y'$ — un punto dell'immagine A e il suo corrispondente (possibilmente errato) nell'immagine B, esattamente come prodotto dalla fase di *matching* dell'EP08_01.
2. **Modelli candidati:** Leggere l'intero K (numero di omografie candidate) e il reale ε (soglia di errore di riproiezione). Successivamente, leggere K righe, ciascuna con nove reali $h_{11}\ h_{12}\ h_{13}\ h_{21}\ h_{22}\ h_{23}\ h_{31}\ h_{32}\ h_{33}$ — gli elementi della matrice H candidata, in ordine di lettura per righe (*row-major*).
3. **Riproiezione:** Per ogni corrispondenza (x, y, x', y') e ogni modello candidato H_k, calcolare il punto proiettato
$$\begin{bmatrix}\hat{x}\\ \hat{y}\\ \hat{w}\end{bmatrix} = H_k \begin{bmatrix}x\\ y\\ 1\end{bmatrix}, \qquad (\hat{x}/\hat{w},\ \hat{y}/\hat{w}) \text{ è il punto proiettato.}$$
4. **Errore di riproiezione:** $e = \sqrt{(\hat{x}/\hat{w} - x')^2 + (\hat{y}/\hat{w} - y')^2}$.
5. **Conteggio degli *inlier*:** Una corrispondenza è un *inlier* del modello H_k se $e \le \varepsilon$.
6. **Selezione del modello migliore:** Il modello vincitore è quello con il maggior numero di *inlier*; in caso di pareggio, scegli quello con **indice minore** k (il primo trovato durante il ciclo iterativo del RANSAC).
7. **Output:** Per ogni modello k da 0 a $K-1$, nell'ordine di lettura, stampare `Modello k: I inliers`. Alla fine, stampare `Modello migliore: k_best con I_best inliers`.

8.14.2.2 📌 Vincoli Computazionali

- **Confronto inclusivo:** un errore di riproiezione **esattamente uguale** a ε conta come *inlier* ($e \le \varepsilon$).
- **Senza stima di H:** le matrici sono già fornite pronte — non è necessario (né previsto) risolvere alcun sistema lineare.

- **Pareggio risolto con l'indice minore**, riflettendo il comportamento naturale di un algoritmo iterativo che esamina i modelli in ordine e sostituisce il migliore trovato finora solo quando un nuovo modello lo **supera strettamente**.

8.14.2.3 🧠 Fondamenti Teorici

Elemento	Ruolo nel RANSAC
Campione minimo (4 coppie)	Sufficiente per determinare gli 8 gradi di libertà di un'omografia
Modello candidato H_k	Stimato da un campione minimo; può essere buono o cattivo, a seconda che il campione contenesse *outlier*
Errore di riproiezione	Misura quanto bene il modello "prevede" ogni corrispondenza osservata
Inlier vs. *outlier*	Corrispondenze coerenti con il modello vincitore (*inlier*) vs. le altre, tipicamente corrispondenze errate del *matching*
Rifinitura finale	In pratica, dopo aver scelto il modello migliore, il RANSAC lo ricalcola usando **solo** i suoi *inlier* — passo non richiesto in questo EP

8.14.2.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Intero N.
- Prossime N righe: quattro reali x y x' y'.
- Riga successiva: Intero K e reale ε.
- Prossime K righe: nove reali (elementi di H_k, *row-major*).

Output:

- K righe nel formato `Modello k: I inliers`.
- Ultima riga: `Modello migliore: k_best con I_best inliers`.

8.14.2.5 📌 Esempi

Input	Output	Osservazione
5 0 0 0 0 1 1 2 2 2 0 4 0 0 2 0 4 5 5 1 1 2 0.5 2 0 0 0 2 0 0 0 1 1 0 0 0 1 0 0 0 1	Modello 0: 4 inliers Modello 1: 1 inliers Modello migliore: 0 con 4 inliers	Il Modello 0 (scala $\times 2$) spiega correttamente 4 delle 5 corrispondenze; la 5ª, $(5,5) \to (1,1)$, è un *outlier* che nessuno dei due modelli spiega bene.

```
%%writefile EP08_02.py
# Codice Python
```

```
Overwriting EP08_02.py
```

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0802.html

Figura 8.16: Simulatore EP08_02: RANSAC — Votazione per Inlier tra Modelli Candidati

```
TestSuite("EP08_02.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.3 EP08_03 🟢 Immagine Integrale: Somme Rettangolari a Tempo Costante

Immagina una telecamera di sorveglianza che elabora 30 fotogrammi al secondo e, per ogni fotogramma, il sistema deve analizzare l'immagine in decine di posizioni e scale diverse, testando in ciascuna un insieme di caratteristiche rettangolari per decidere "c'è un volto qui?". Se calcolare la somma delle intensità di ogni rettangolo richiedesse sommare pixel per pixel, il sistema non avrebbe alcuna possibilità di valutare in tempo reale — il collo di bottiglia sarebbe proprio nella parte più ripetuta dell'algoritmo. È esattamente questo collo di bottiglia che l'immagine integrale elimina.

La Haar Cascade valuta migliaia di caratteristiche rettangolari per finestra, in molteplici posizioni e scale — qualcosa di irrealizzabile in tempo reale se ogni rettangolo richiedesse di sommare i propri pixel uno a uno. L'**immagine integrale**, definita nella sezione sulla Haar Cascade, risolve questo problema: una volta pre-calcolata, la somma delle intensità di **qualsiasi** regione rettangolare si ottiene con solo quattro query e tre operazioni aritmetiche, indipendentemente dalla dimensione del rettangolo.

Sei stato incaricato di implementare questa struttura da zero: in primo luogo, calcolare l'immagine integrale dall'immagine originale; successivamente, rispondere a query rettangolari arbitrarie.

8.14.3.1 📋 Linee Guida di Implementazione

1. **Input:** Leggere le dimensioni $H \times W$ dell'immagine e i suoi $H \times W$ valori interi di intensità.
2. **Immagine integrale:** Calcolare, per ogni posizione (i, j) (indicizzazione a partire da 0, `[riga][colonna]`),
$$II(i,j) = \sum_{i' \le i,\ j' \le j} I(i', j'),$$
ovvero la somma di tutti i pixel sopra e a sinistra di (i, j), inclusa la posizione stessa.
3. **Query:** Leggere l'intero Q e, successivamente, Q righe, ciascuna con quattro interi $x_1\ y_1\ x_2\ y_2$ — gli angoli superiore-sinistro e inferiore-destro di un rettangolo, **entrambi inclusivi**, con $0 \le x_1 \le x_2 < W$ e $0 \le y_1 \le y_2 < H$.
4. **Somma rettangolare in O(1):** Per ogni query, calcolare la somma delle intensità all'interno del rettangolo utilizzando esclusivamente valori già presenti in II (senza scorrere i pixel originali):
$$S(x_1, y_1, x_2, y_2) = II(y_2, x_2) - II(y_2, x_1{-}1) - II(y_1{-}1, x_2) + II(y_1{-}1, x_1{-}1),$$
trattando qualsiasi termine con indice di riga o colonna uguale a -1 come 0.
5. **Output:** In primo luogo, stampare l'immagine integrale completa — H righe con W interi ciascuna. Successivamente, per ogni query, stampare un singolo intero: la somma della regione corrispondente.

8.14.3.2 📌 Vincoli Computazionali

- **Non ricalcolare per forza bruta:** la risposta a ogni query deve utilizzare la formula a quattro termini su II, non una somma diretta dei pixel del rettangolo (sebbene il risultato numerico sia lo stesso, l'obiettivo dell'esercizio è proprio questa tecnica).
- **Rettangoli con coordinate inclusive:** (x_1, y_1) e (x_2, y_2) appartengono alla regione sommata.
- **Gestione dei bordi:** quando si interroga II con indice -1 (quando $x_1 = 0$ o $y_1 = 0$), utilizzare il valore 0.

8.14.3.3 🧠 Fondamentazione Teorica

Elemento	Ruolo nella Haar Cascade
Immagine integrale II	Pre-calcolata una sola volta per immagine, in tempo $O(HW)$
Query in O(1)	Ogni caratteristica Haar (differenza tra somme di regioni rettangolari) viene valutata con poche operazioni, indipendentemente dall'area del rettangolo
Scalabilità	È questa costanza che rende possibile valutare migliaia di caratteristiche, in molteplici posizioni e scale, in tempo reale
Principio di inclusione-esclusione	I quattro termini della formula sommano la regione desiderata e sottraggono esattamente le aree conteggiate in eccesso

8.14.3.4 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Interi H e W.
- Prossime H righe: W interi ciascuna (immagine originale).
- Riga successiva: Intero Q.
- Prossime Q righe: quattro interi x_1 y_1 x_2 y_2.

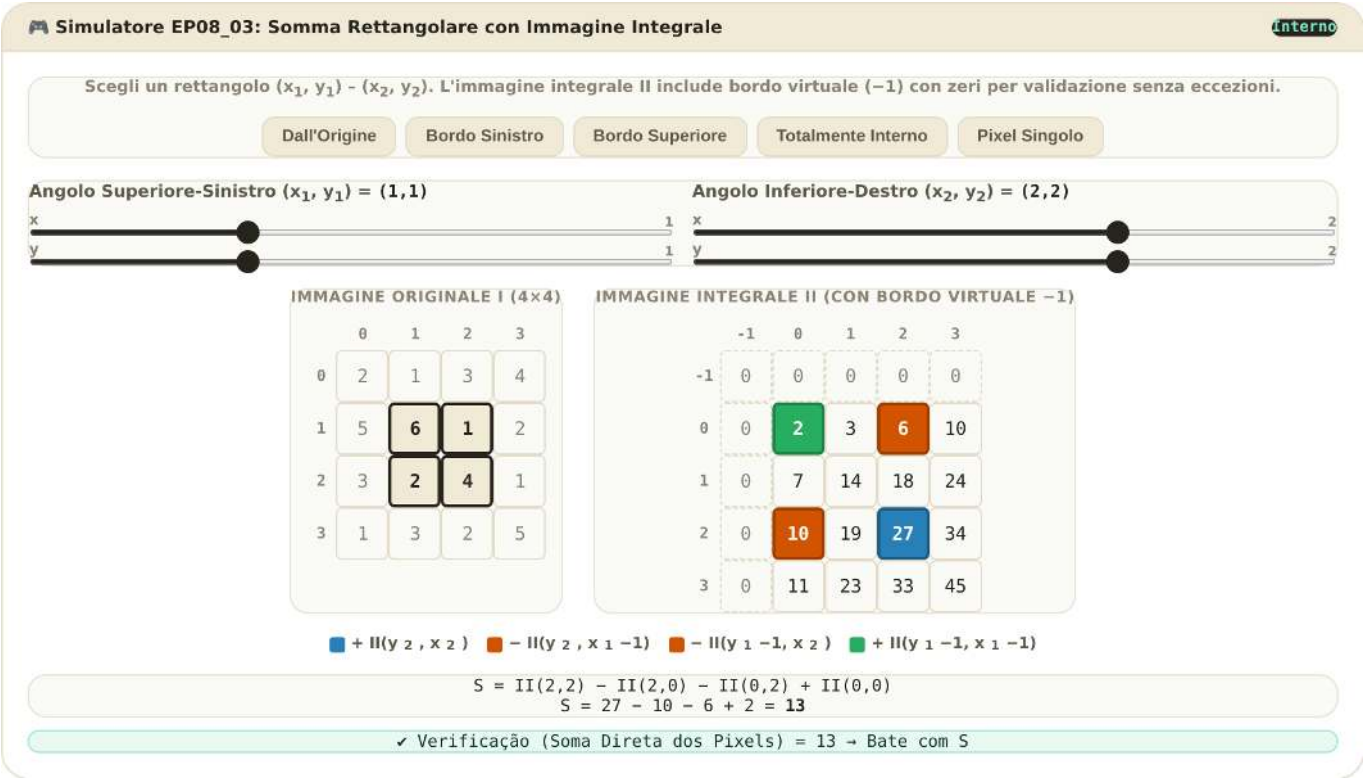
Output:

- H righe con W interi ciascuna (l'immagine integrale).
- Q righe, una per query, con la somma della regione corrispondente.

8.14.3.5 📌 Esempi

Input	Output	Osservazione
3 3 1 2 3 4 5 6 7 8 9 1 0 0 2 2	1 3 6 5 12 21 12 27 45 45	La query copre l'intera immagine; la somma coincide con $II(2, 2)$ e con la somma di tutti i 9 valori.

Input	Output	Osservazione
3 3 1 2 3 4 5 6 7 8 9 2 1 1 2 2 0 0 1 1	1 3 6 5 12 21 12 27 45 28 12	La prima query usa i quattro termini della formula; la seconda coincide direttamente con $II(1,1)$, poiché inizia dall'origine.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0803.html

Figura 8.17: Simulatore EP08_03: Somma Rettangolare in O(1) — Molteplici Situazioni di Bordo

```
%%writefile EP08_03.py
# Codice Python
```

```
Overwriting EP08_03.py
```

```
TestSuite("EP08_03.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.4 EP08_04 🟢 IoU e Soppressione Non-Massima (NMS)

La figura di questa sezione ha mostrato l'effetto della Soppressione Non-Massima su un insieme di riquadri prodotti da un rilevatore di tipo *sliding window*: più rilevazioni ridondanti per oggetto sono state ridotte a un singolo riquadro per oggetto. Ti è stato affidato il compito di reimplementare, byte per byte, le due funzioni che hanno prodotto quel risultato — `calcola_iou` e `soppressione_non_massima` — per confermare, con le tue mani, esattamente i numeri presentati nel capitolo.

8.14.4.1 📋 Linee Guida di Implementazione

1. **Input:** Leggi l'intero N (numero di riquadri) e il reale τ (soglia IoU). Successivamente, leggi N righe, ciascuna con cinque reali x_{min} y_{min} x_{max} y_{max} score.
2. **Intersezione su Unione:** Per due riquadri A e B,

$$\text{IoU}(A, B) = \frac{\text{area}(A \cap B)}{\text{area}(A \cup B)},$$

   con area di intersezione nulla quando i riquadri non si sovrappongono.
3. **Algoritmo NMS** (esattamente come descritto nel capitolo):
   a. Ordina i riquadri per `score` decrescente (i pareggi mantengono l'ordine di lettura originale).
   b. Seleziona il riquadro con il punteggio più alto tra quelli rimanenti; aggiungilo all'output e rimuovilo dalla lista.
   c. Scarta, dalla lista rimanente, **tutti** i riquadri il cui IoU con il riquadro selezionato sia **maggiore o uguale** a τ — solo i riquadri con IoU $< \tau$ rimangono candidati.
   d. Ripeti i passaggi (b)–(c) finché la lista dei rimanenti non è vuota.
4. **Output:** Per ogni riquadro mantenuto, nell'ordine in cui è stato selezionato, stampa il suo indice originale (posizione di lettura, a partire da 0) e il suo `score`, con 2 cifre decimali. Alla fine, stampa `Totale mantenuti: X`.

8.14.4.2 📌 Vincoli Computazionali

- **Attenzione alla direzione della soglia:** contrariamente a quanto si potrebbe supporre, un riquadro viene **soppresso** quando IoU $\geq \tau$ (non solo quando IoU $> \tau$) — segui esattamente questo criterio, lo stesso del codice di riferimento del capitolo.
- **Indici originali:** l'output fa riferimento alla posizione di lettura di ciascun riquadro nell'input, non alla sua posizione dopo l'ordinamento per `score`.
- **Area senza somma di 1 pixel:** usa area $= (x_{max} - x_{min}) \times (y_{max} - y_{min})$, esattamente come nel capitolo (senza l'aggiustamento "+1" talvolta usato in altre convenzioni).

8.14.4.3 🧠 Fondamenti Teorici

Elemento	Ruolo nel post-elaborazione
IoU	Quantifica la sovrapposizione spaziale tra due riquadri delimitatore
Sliding window (Haar Cascade)	Produce tipicamente più rilevazioni sovrapposte per lo stesso oggetto, in posizioni e scale vicine
Soglia τ	Controlla l'aggressività della soppressione: troppo bassa fonde oggetti vicini; troppo alta lascia passare ridondanze
Ordinamento per `score`	Garantisce che, tra riquadri ridondanti, quello con maggiore confidenza sopravviva sempre

8.14.4.4 📦 Specifica di Input e Output (VPL)

Input:

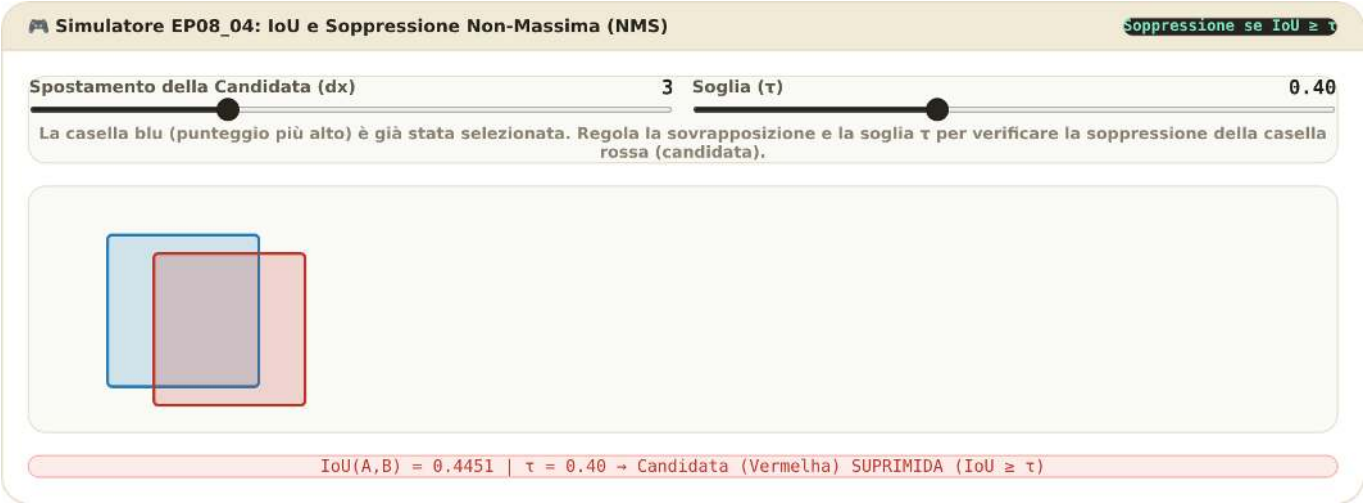
- Riga 1: Intero N e reale τ.
- Prossime N righe: cinque reali x_{min} y_{min} x_{max} y_{max} score.

Output:

- Una riga per riquadro mantenuto, nell'ordine di selezione: `indice score` (score con 2 cifre decimali).
- Ultima riga: `Totale mantenuti: X`.

8.14.4.5 📌 Esempi

Input	Output	Osservazione
5 0.4 50 50 150 150 0.90 60 55 155 145 0.75 58 60 160 150 0.60 300 300 400 420 0.95 310 305 395 415 0.70	3 0.95 0 0.90 Totale mantenuti: 2	Esattamente l'esempio della figura del capitolo: 5 riquadri ridondanti (2 oggetti) diventano 2 rilevazioni finali. L'IoU tra il 1° e il 2° riquadro è $\approx 0{,}775$, ben al di sopra di $\tau = 0{,}4$.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0804.html

Figura 8.18: Simulatore EP08_04: IoU e Soppressione Non-Massimale

```
%%writefile EP08_04.py
# Codice Python
```

```
Overwriting EP08_04.py
```

```
TestSuite("EP08_04.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.5 EP08_05 🟡 Etichettatura dei Componenti Connessi: Segmentazione delle Istanze

L'esempio di segmentazione classica di questo capitolo ha separato le "istanze" delle monete semplicemente tramite la loro disconnessione spaziale nella maschera binaria risultante dalla sogliatura di Otsu. Questa fase finale — etichettare ogni componente connesso con un identificatore di istanza — è esattamente ciò che ti è stato richiesto di implementare qui, da zero, su una maschera binaria già pronta (0 = sfondo, 1 = oggetto), come se fosse una reimplementazione manuale di `cv2.connectedComponents`.

Questo esercizio mette inoltre in luce, in modo molto concreto, la limitazione discussa nel capitolo: il risultato dipende interamente da come si definisce la "vicinanza" tra i pixel — e, come vedrai nel secondo esempio, due pixel in diagonale possono essere considerati la stessa istanza o istanze diverse, dipendendo esclusivamente dalla **connettività** scelta, non da alcuna nozione semantica di oggetto.

8.14.5.1 📋 Linee Guida di Implementazione

1. **Input:** Leggere le dimensioni $H \times W$ della maschera binaria e i suoi $H \times W$ valori (0 o 1).
2. **Connettività:** Leggere l'intero $c \in \{4, 8\}$. Con connettività 4, i vicini di (i, j) sono $(i{-}1, j)$, $(i{+}1, j)$, $(i, j{-}1)$ e $(i, j{+}1)$. Con connettività 8, si aggiungono le quattro diagonali: $(i{-}1, j{-}1)$, $(i{-}1, j{+}1)$, $(i{+}1, j{-}1)$ e $(i{+}1, j{+}1)$.
3. **Scoperta dei componenti:** Scorrendo la maschera in scansione riga per riga, da sinistra a destra e dall'alto verso il basso, ogni volta che si incontra un pixel di valore 1 non ancora etichettato, questo dà inizio a una **nuova componente**: assegnagli la prossima etichetta disponibile (la prima componente scoperta riceve l'etichetta 1, la seconda l'etichetta 2, e così via) e propaga la stessa etichetta a tutti i pixel di valore 1 raggiungibili da esso tramite una catena di vicini (secondo la connettività scelta) — tramite ricerca in ampiezza, in profondità, o *union-find*, a tua scelta.
4. **Pixel di sfondo:** rimangono con etichetta 0 e non appartengono a nessuna istanza.
5. **Output:** Prima, stampare la mappa completa delle etichette — H righe con W interi ciascuna. Successivamente, per ogni etichetta ℓ da 1 a K (nell'ordine di scoperta), stampare `Istanza l: A pixel`, dove A è la quantità di pixel con quella etichetta. Infine, stampare `Totale istanze: K`.

8.14.5.2 📌 Vincoli Computazionali

- **Ordine di scoperta = ordine di scansione:** le etichette sono numerate nell'ordine in cui ogni nuova componente viene trovata dalla scansione riga per riga, non per dimensione o posizione.
- **Connettività esplicita:** due pixel di valore 1 appartengono alla stessa istanza solo se esiste una catena di vicini **secondo** c che li collega — non usare per errore la connettività opposta.
- **Maschera binaria pura:** tutti i valori di input sono esattamente 0 o 1.

8.14.5.3 🧠 Fondamenti Teorici

Elemento	Ruolo nella segmentazione classica delle istanze
Sogliatura (Otsu, Cap. 4)	Fase precedente che produce la maschera binaria a partire dall'immagine di intensità
Componente connessa	Ogni istanza è definita **solo** dalla connettività spaziale dei pixel dell'oggetto, senza alcuna nozione di forma, classe o aspetto
Connettività 4 vs. 8	Parametro che altera il risultato: con connettività 8, due blob uniti solo in diagonale diventano un'unica istanza
Limitazione centrale	La tecnica fonde istanze che si toccano o si sovrappongono (anche se sono oggetti chiaramente distinti), poiché non c'è nozione di "oggetto" — solo di "regione connessa"

8.14.5.4 📦 Specifica di Input e Output (VPL)

Input:

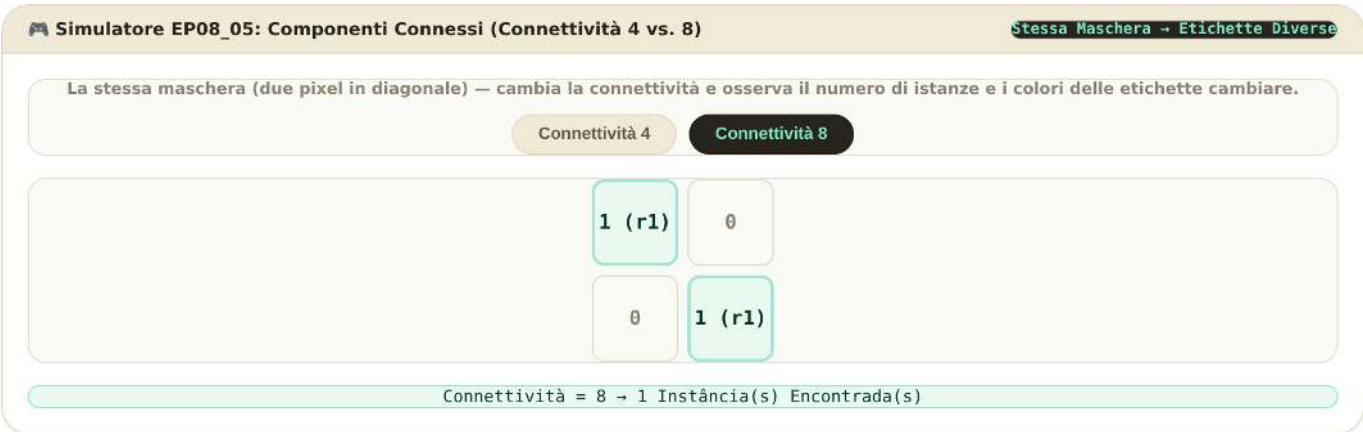
- Riga 1: Interi H e W.
- Prossime H righe: W interi (0 o 1) ciascuna.
- Ultima riga: Intero c (4 o 8).

Output:

- H righe con W interi ciascuna (la mappa delle etichette).
- Una riga per istanza, nell'ordine di scoperta: `Istanza l: A pixel`.
- Ultima riga: `Totale istanze: K`.

8.14.5.5 📌 Esempi

Input	Output	Osservazione
6 6 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 8	0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 0 0 0 2 2 Istanza 1: 4 pixel Istanza 2: 4 pixel Totale istanze: 2	Due blocchi 2×2 chiaramente separati: il risultato è lo stesso con connettività 4 o 8.
2 2 1 0 0 1 8	1 0 0 1 Istanza 1: 2 pixel Totale istanze: 1	Con connettività 8, i due pixel in diagonale appartengono alla **stessa** istanza. Ripeti questo esempio con $c = 4$: il risultato diventa 2 istanze di 1 pixel ciascuna — puramente per il cambiamento di connettività, senza alcuna differenza nella maschera.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0805.html

Figura 8.19: Simulatore EP08_05: Etichettatura delle Componenti Connesse — Connettività 4 vs. 8

```
%%writefile EP08_05.py
# Codice Python
```

```
Overwriting EP08_05.py
```

```
TestSuite("EP08_05.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.6 EP08_06 🟡 *Bounding Boxes*, Centroidi e Proprietà delle Istanze con `mm.measure`

Nell'esercizio precedente (**EP08_05**), si osserva come la segmentazione per componenti connesse etichetti regioni binarie contigue per separare le istanze. Tuttavia, per attività di rilevamento, tracciamento e analisi quantitativa degli oggetti, la semplice mappa delle etichette non è sufficiente. Diventa necessario estrarre **metriche spaziali e geometriche** che caratterizzino ciascuna istanza individualmente.

Questo EP si concentra sul calcolo e sull'estrazione automatica delle proprietà fondamentali della visione artificiale per ogni componente connessa trovata nella maschera binaria, utilizzando il metodo nativo `mm.measure(img)` della libreria `morph`:

1. **Bounding Box:** Il più piccolo rettangolo allineato agli assi che racchiude completamente l'istanza, definito dal suo angolo superiore sinistro (x, y), larghezza w e altezza h.
2. **Centroide Geometrico $(\bar{x}, \bar{y})$:** Il centro di massa dell'istanza sulla griglia discreta, equivalente ai momenti spaziali del primo ordine M_{10}/M_{00} e M_{01}/M_{00}.
3. **Area Geometrica del Contorno (A):** L'area racchiusa dal contorno dell'istanza, calcolata tramite `mm.contourArea(c)`.

8.14.6.1 📋 Linee Guida di Implementazione

1. **Input:** Leggere le dimensioni $H \times W$ della maschera binaria, i valori $H \times W$ (0 o 1) e il parametro di connettività $c \in \{4, 8\}$.
2. **Estrazione Automatica con `mm.measure`:** Passare l'immagine binarizzata alla funzione `mm.measure(img_bin)`, che estrae i contorni OpenCV e restituisce una lista di dizionari contenenti le proprietà geometriche di ciascuna istanza.
3. **Proprietà Restituite:** Per ogni dizionario m della lista restituita da `medidas = mm.measure(img_bin)`:
   - **Area (`area`):** Valore numerico dell'area geometrica del contorno `mm.contourArea(c)`.
   - **Bounding Box (`bbox`):** Tupla (x, y, w, h) che rappresenta l'angolo superiore sinistro, la larghezza e l'altezza.
   - **Centroide (`center`):** Tupla (c_x, c_y) con le coordinate del centro di massa M_{10}/M_{00} e M_{01}/M_{00}. Formattare con **due cifre decimali**.
4. **Output:** Per ogni istanza $1, \ldots, K$ trovata (ordinata per ordine di scoperta/posizione nell'immagine), stampare una riga contenente le sue proprietà. Infine, stampare il numero totale di istanze.
   - Per ordinare, usare `medidas.sort(key=lambda m: (m['bbox'][1], m['bbox'][0]))`.

8.14.6.2 🧠 Fondamenti Teorici

Proprietà in `mm.measure`	Calcolo Matematico / Logica Discreta	Applicazione Pratica nella Visione Artificiale
`bbox` (OpenCV)	$[x, y, w, h] = [\min(c), \min(r), \Delta c + 1, \Delta r + 1]$	Formato classico di OpenCV. *Nota: reti come YOLO convertono questo rettangolo in (c_x, c_y, w, h) normalizzato.*
`center`	$\bar{x} = \frac{M_{10}}{M_{00}}, \quad \bar{y} = \frac{M_{01}}{M_{00}}$	Centro di massa esatto della maschera (usato nel tracciamento e nell'analisi delle traiettorie).
`area`	$A = \text{contourArea}(C)$ (Formula del Poligono)	Metrica continua della superficie dell'oggetto.

8.14.6.3 📦 Specifica di Input e Output (VPL)

Input:

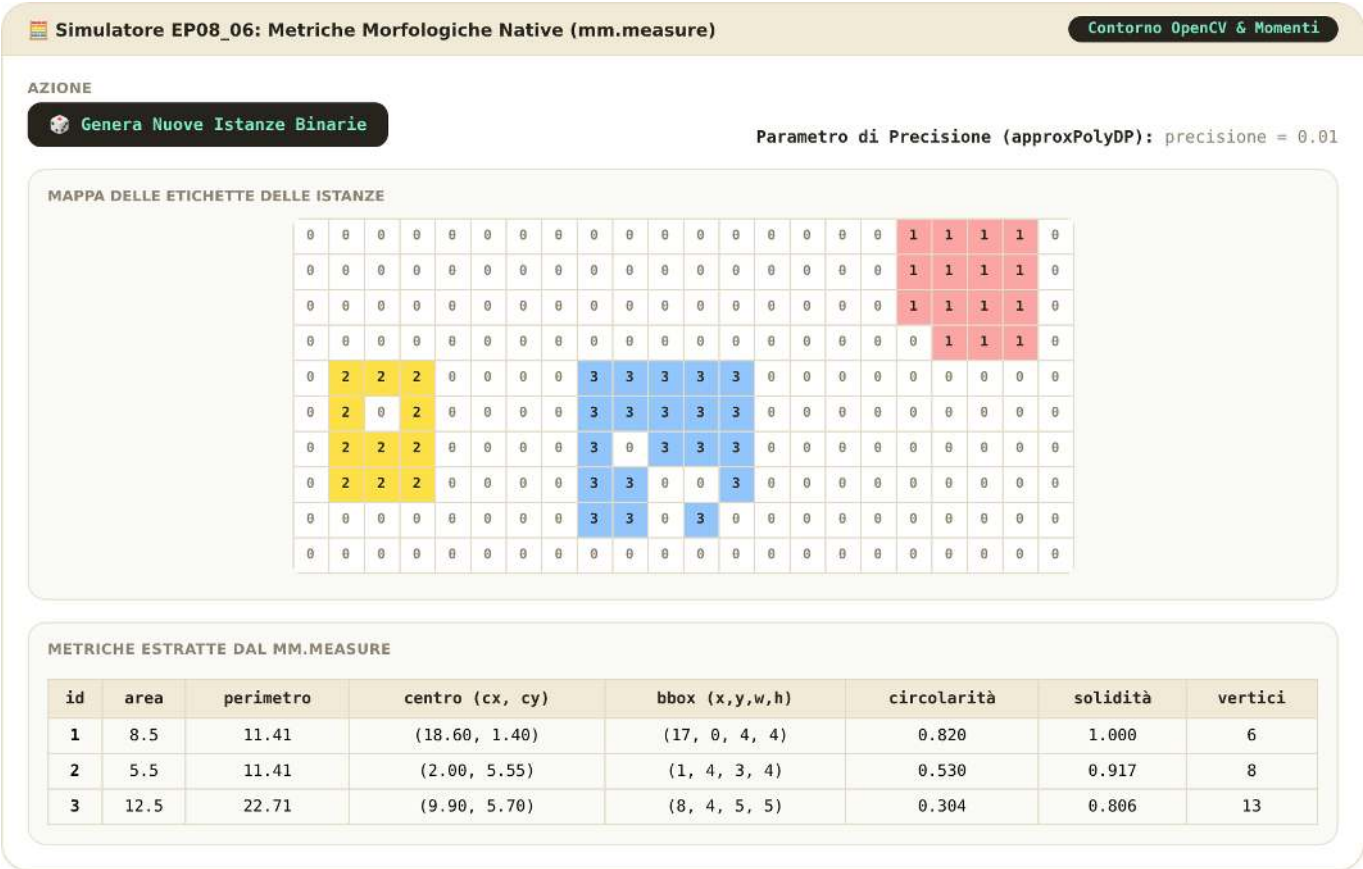
- Riga 1: Interi H e W.
- Prossime H righe: W interi (0 o 1) ciascuna.
- Ultima riga: Intero c (4 o 8).

Output:

- Una riga per istanza nell'ordine di scoperta: `Istanza l: Area=A, BBox=(x,y,w,h), Centroide=(cx,cy)`
- Ultima riga: `Totale istanze: K`.

8.14.6.4 📌 Esempi

Input	Output	Osservazione
6 6 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 1 8	Istanza 1: Area=1.0, BBox=(1,1,2,2), Centroide=(1.50,1.50) Istanza 2: Area=1.0, BBox=(4,4,2,2), Centroide=(4.50,4.50) Totale istanze: 2	Blocchi 2×2 allineati. Il calcolo dell'area geometrica del contorno risulta in 1.0. Il centroide del blocco nelle colonne 1–2 e righe 1–2 è esattamente (1.50, 1.50).
4 6 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 4	Istanza 1: Area=2.0, BBox=(1,1,4,2), Centroide=(2.40,1.20) Totale istanze: 1	Oggetto asimmetrico a forma di "T" invertita. L'area geometrica del contorno è 2.0. Il centroide riflette la distribuzione dei pixel dell'oggetto.



id	area	perimetro	centro (cx, cy)	bbox (x,y,w,h)	circolarità	solidità	vertici
1	8.5	11.41	(18.60, 1.40)	(17, 0, 4, 4)	0.820	1.000	6
2	5.5	11.41	(2.00, 5.55)	(1, 4, 3, 4)	0.530	0.917	8
3	12.5	22.71	(9.90, 5.70)	(8, 4, 5, 5)	0.304	0.806	13

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0806.html

Figura 8.20: Simulatore EP08_06: Estrazione di *Bounding Boxes*, Centroidi e Proprietà con *mm.measure*

```
%%writefile EP08_06.py
# Codice Python
```

```
Overwriting EP08_06.py
```

```
TestSuite("EP08_06.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.7 EP08_07 🟡 Rimozione del Rumore Sale e Pepe e Misurazione degli Oggetti

In questo esercizio, applicherai il filtraggio morfologico per pulire un'immagine binaria corrotta da rumore di tipo **sale e pepe** (pixel isolati di valore `1` nello sfondo e `0` all'interno degli oggetti). Dopo la pulizia, il programma deve estrarre le misurazioni geometriche dei componenti connessi rimanenti, ordinarli e visualizzare la tabella finale delle metriche.

8.14.7.1 📋 Linee Guida di Implementazione

1. **Input:** leggere due interi H e W (altezza e larghezza dell'immagine) nella prima riga e, successivamente, le H righe con la matrice binaria contenente pixel `0` e `1` separati da spazio.
2. **Filtraggio Morfologico:** applicare una sequenza di **Apertura** (per eliminare il rumore "sale" sullo sfondo) seguita da **Chiusura** (per riempire il rumore "pepe" all'interno degli oggetti) con elemento strutturante 3×3.
3. **Stampa dell'Immagine Pulita:** stampare la matrice risultante in valori `0` e `1` separati da spazio.
4. **Misurazioni Geometriche:** per ogni oggetto identificato nella matrice pulita, estrarre:

- `id`: identificatore numerico sequenziale (riassegnato dopo l'ordinamento);
- `area`: area calcolata tramite contorno (`cv2.contourArea`);
- `perimeter`: perimetro del contorno (`cv2.arcLength`);
- `cx`, `cy`: centro di massa (centroide tramite `cv2.moments`);
- `x`, `y`, `w`, `h`: coordinate del rettangolo delimitatore (`cv2.boundingRect`);
- `circularity`: circolarità data da $\frac{4\pi \cdot \text{area}}{\text{perimetro}^2}$;
- `solidity`: solidità data dal rapporto $\frac{\text{area}}{\text{area dell'inviluppo convesso}}$;
- `vertices`: numero di vertici approssimato del poligono (`cv2.approxPolyDP` con $\epsilon = 0.02 \times \text{perimetro}$).

5. **Ordinamento e Uscita:** ordinare gli oggetti in ordine crescente in base alla posizione X del rettangolo delimitatore (`bbox[0]`); in caso di parità, utilizzare la posizione Y (`bbox[1]`). Riassegnare gli `id` da 1 a N e stampare la tabella formattata.
   - Per ordinare, utilizzare `medidas.sort(key=lambda m: (m['bbox'][1], m['bbox'][0]))`, con `medidas = mm.measure(img)`.

8.14.7.2 📌 Vincoli e Regole di Ordinamento

- **Regola di Ordinamento degli Oggetti:**

```
medidas.sort(key=lambda m: (m['bbox'][0], m['bbox'][1]))
```

- **Differenza di Area:** L'area calcolata da OpenCV (`cv2.contourArea`) misura l'area del poligono continuo delimitato dai centri dei pixel di bordo, risultando in valori numerici inferiori rispetto al semplice conteggio discreto dei pixel `1` (`np.sum`).

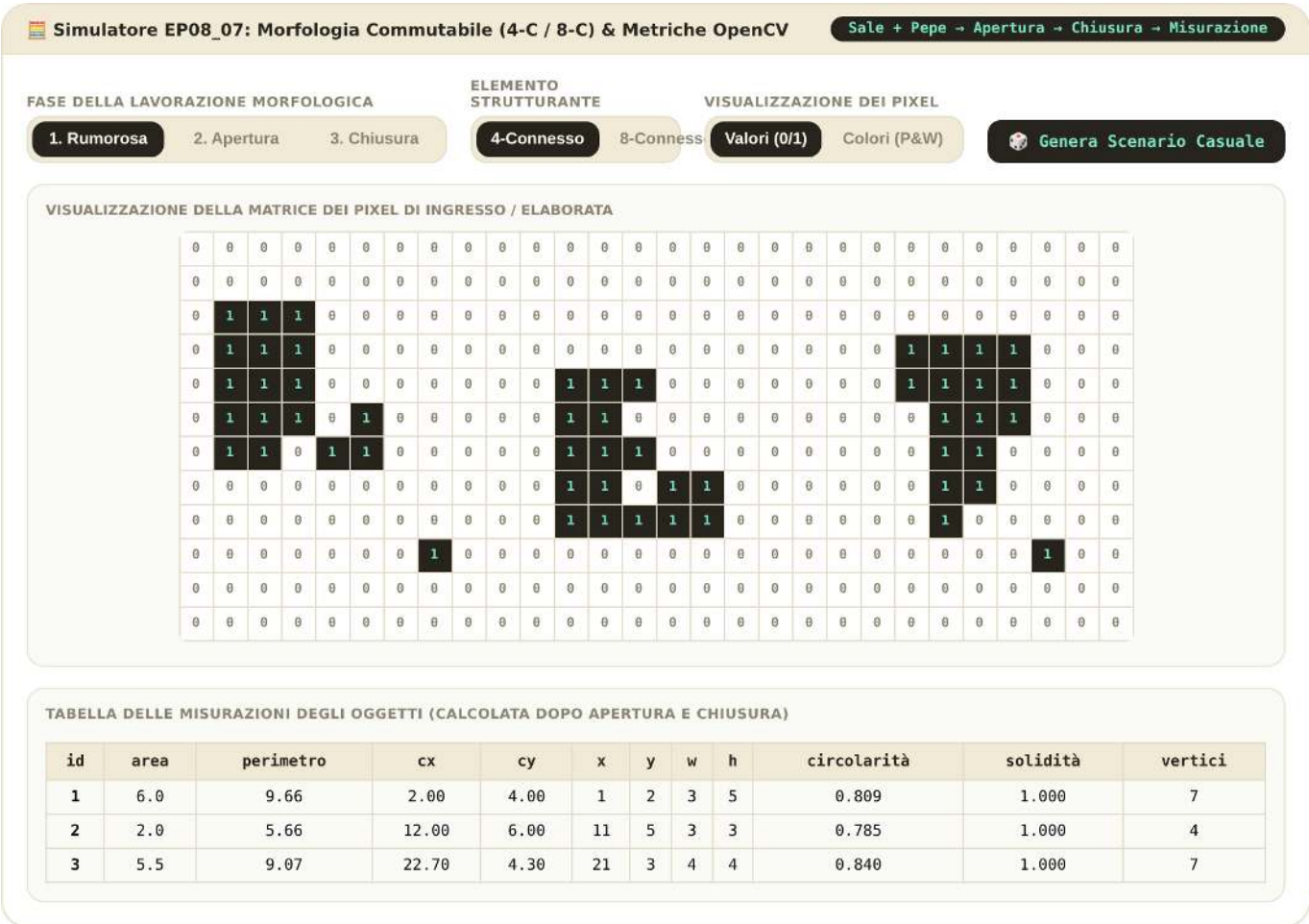
8.14.7.3 🧠 Fondamenti Teorici

Operazione / Metrica	Funzione nel Filtraggio e nella Caratterizzazione
Apertura Morfologica (∘)	Erosione seguita da dilatazione: rimuove rumori brillanti isolati (*sale*).
Chiusura Morfologica (•)	Dilatazione seguita da erosione: riempie piccoli fori scuri all'interno degli oggetti (*pepe*).

Operazione / Metrica	Funzione nel Filtraggio e nella Caratterizzazione
`cv2.boundingRect`	Restituisce (x, y, w, h), il più piccolo rettangolo allineato agli assi che racchiude l'oggetto.
Circolarità e Solidità	Descrivono la compattezza e la convessità geometrica del componente.

8.14.7.4 📌 Esempi

Input	Output
8 9 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0	id area perimeter cx cy x y w h circularity solidity vertices 1 9.0 12.0 3.5 2.0 3 1 4 3 0.79 1.000 4 2 4.0 8.0 7.5 5.5 7 5 2 2 0.79 1.000 4



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0807.html

Figura 8.21: Simulatore EP08_07: Morfologia con Connettività Configurabile e Misurazione

```
%%writefile EP08_07.py
# Codice Python
```

```
Overwriting EP08_07.py
```

```
TestSuite("EP08_07.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.8 EP08_08 🟡 Immagine in Scala di Grigi e Soglia Dinamica

In questo esercizio, l'immagine di ingresso non è più strettamente binaria (`0/1`) ma diventa un'**immagine in scala di grigi** (8 **bit,** $0 \ldots 255$), dove gli oggetti hanno un'intensità media intermedia su uno sfondo scuro (0), oltre a rumore di tipo sale e pepe distribuito su tutta l'immagine.

8.14.8.1 📋 Linee Guida di Implementazione

1. **Ingresso:** leggere H e W nella prima riga, seguiti dalle H righe con valori interi da 0 a 255 in una matrice $H \times W$.
2. **Pre-elaborazione:**

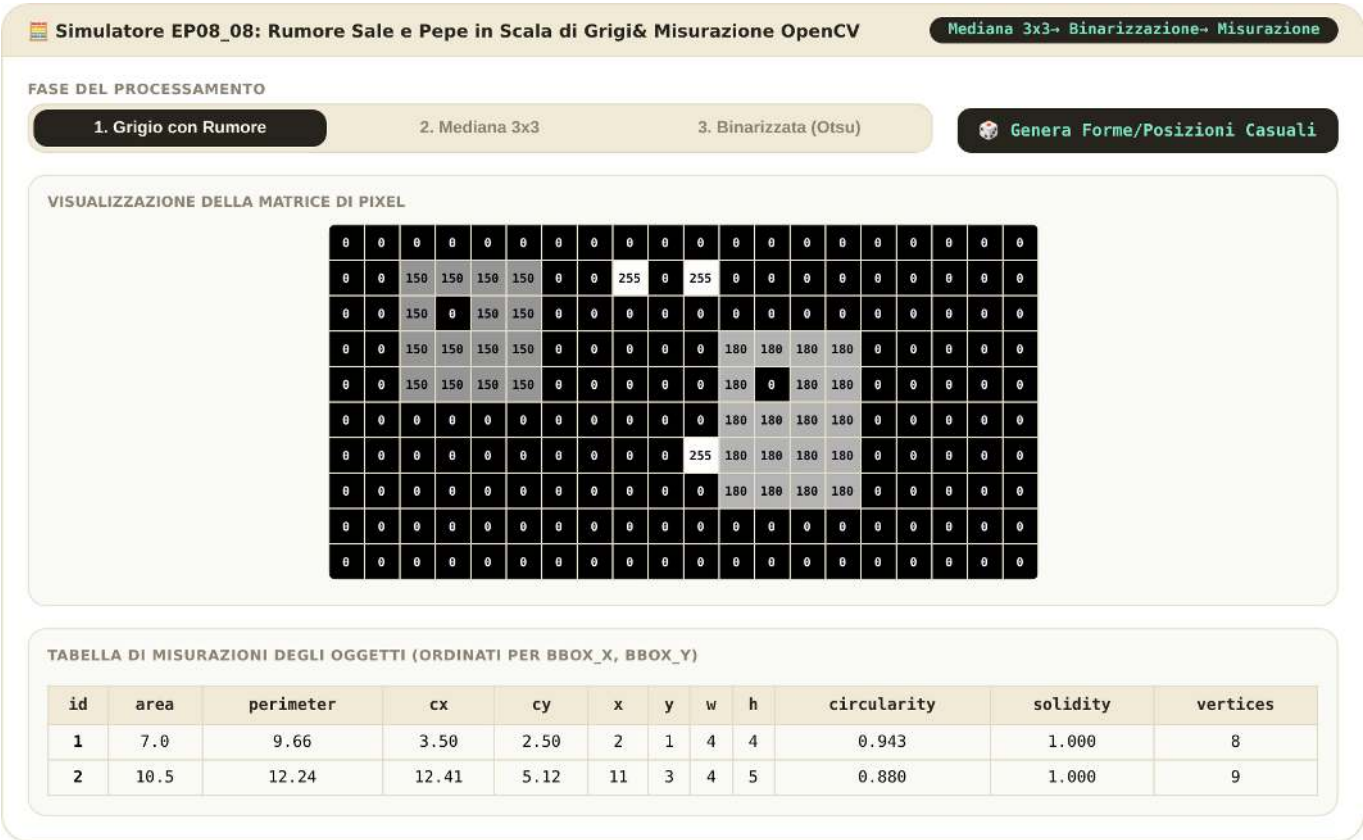
- Applicare un filtro di **Mediana** (3×3) per rimuovere il rumore sale e pepe mantenendo i bordi nitidi.
- Applicare la **Soglia di Otsu** (o una soglia fissa $T = 60$) per binarizzare l'immagine pulita.

3. **Misurazione e Uscita:** estrarre il contorno degli oggetti, calcolare le metriche geometriche (`area`, `perimeter`, `cx`, `cy`, `x`, `y`, `w`, `h`, `circularity`, `solidity`) e ordinare gli oggetti per `bbox[0]` (con `bbox[1]` come criterio di parità). Riassegnare `id` da 1 a N e stampare la tabella.
   - Per ordinare, usare `misure.sort(key=lambda m: (m['bbox'][1], m['bbox'][0]))`, con `misure = mm.measure(img)`.

8.14.8.2 📌 Esempi

Ingresso	Uscita
16 32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 … (immagine binaria contenente un quadrato e un cerchio)	id area perimeter cx cy x y w h circularity solidity vertices 1 16.0 16.0 8.0 8.0 6 6 5 5 0.79 1.000 4 2 28.3 18.8 22.5 8.0 19 5 7 7 1.00 1.000 8

[req 418908] Traduci questo Markdown dal portoghese all'italiano:



id	area	perimeter	cx	cy	x	y	w	h	circularity	solidity	vertices
1	7.0	9.66	3.50	2.50	2	1	4	4	0.943	1.000	8
2	10.5	12.24	12.41	5.12	11	3	4	5	0.880	1.000	9

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0808.html

Figura 8.22: Simulatore EP08_08: Filtraggio Mediano in Scala di Grigi e Misurazione di Oggetti OpenCV

```
%%writefile EP08_08.py
# Codice Python
```

```
Overwriting EP08_08.py
```

```
TestSuite("EP08_08.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.9 EP08_09 🟠 Gradiente di Illuminazione e Soglia Adattativa

In questa variazione, gli oggetti sono immersi in uno sfondo con **illuminazione non uniforme (gradiente morbido di illuminazione)**. La sogliatura semplice a valore singolo fallisce, richiedendo una pre-elaborazione più robusta.

8.14.9.1 📋 Linee guida di implementazione

1. **Input:** immagine in scala di grigi $H \times W$ con variazione di sfondo da 20 a 180.
2. **Pre-elaborazione:**

- Applicare **Soglia Adattativa** (es.: `cv2.adaptiveThreshold` con finestra gaussiana di 15×15 e costante $C = 3$) per isolare gli oggetti indipendentemente dalla variazione dello sfondo.

  `cv2.adaptiveThreshold(img_gray, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, ksize, C) // 255`

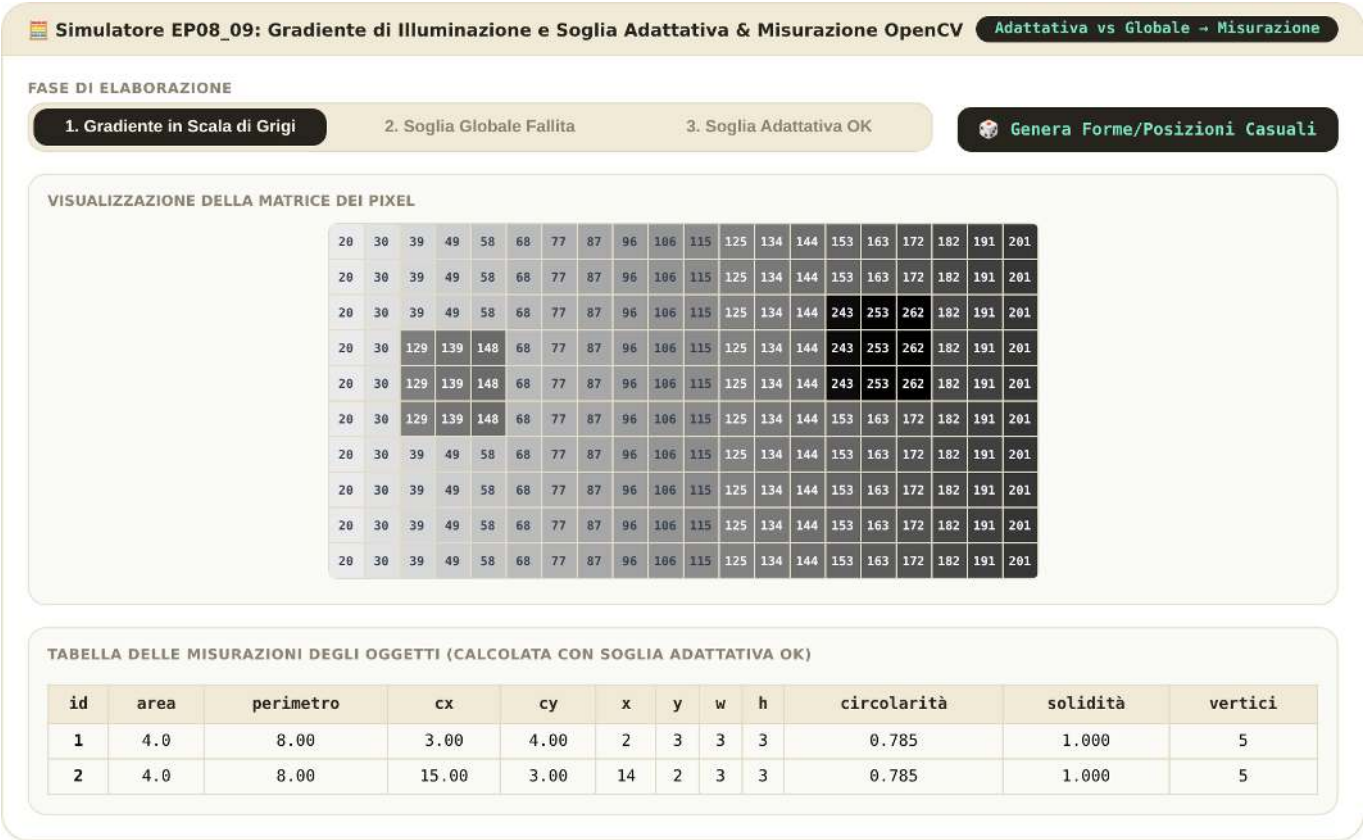
  `ksize` e `C` vengono letti dopo l'immagine.

- Operazione morfologica di **Chiusura** (3×3) per sigillare eventuali difetti nei contorni.

3. **Misurazione e Classificazione:** estrarre le misure.
4. **Ordinamento e Output:** ordinare per `(bbox[0], bbox[1])` e stampare la tabella includendo la colonna `class`.
   - Per ordinare, usare `medidas.sort(key=lambda m: (m['bbox'][1], m['bbox'][0]))`, con `medidas = mm.measure(img, precision=0.02)`.

8.14.9.2 📌 Esempi

Input	Output
16 32 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... k 20	id area perimeter cx cy x y w h circularity solidity vertices 1 9.0 12.0 10.0 5.0 8 3 5 5 0.79 1.000 4 2 28.3 18.8 25.0 12.0 22 9 7 7 1.00 1.000 3



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0809.html

Figura 8.23: Simulatore EP08_09: Gradiente di Illuminazione, Soglia Adattativa e Misurazione OpenCV

```
%%writefile EP08_09.py
# Codice Python
```

```
Overwriting EP08_09.py
```

```
TestSuite("EP08_09.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.10 EP08_10 🔴 Contrasto Basso e Separazione degli Oggetti Tangenti (*Watershed* / Distanza)

In questo esercizio, **alcuni oggetti geometrici sono leggermente a contatto (sovrapposti sui bordi)**. La semplice estrazione dei contorni tratterebbe due oggetti come uno solo.

8.14.10.1 📋 Linee Guida di Implementazione

1. **Input:** matrice $H \times W$ in livelli di grigio con oggetti di intensità $110 \ldots 140$ su sfondo 0, con rumore e coppie di oggetti tangenti.
2. **Pre-elaborazione e Separazione:**

- Applicazione della sogliatura.
- Applicazione della **Trasformata della Distanza** (`mm.dist`).
- Ottenimento dei picchi di distanza per servire come marcatori nella **Trasformata *Watershed*** (`mm.watershed`), separando fisicamente gli oggetti a contatto nella maschera. **Suggerimento:** usare `mm.regmax()` per ottenere i massimi locali e successivamente etichettare con `mm.label0`.
- Dopo il *watershed*, applicare nuovamente la sogliatura con `mm.threshold(water,0)//255`.

3. **Analisi delle Componenti Connesse:** misurare ogni regione isolata post-*Watershed*.
4. **Output:** stampare i componenti ordinati per `(bbox[0], bbox[1])` con le loro metriche individuali di area, centroide e solidità.
   - Per ordinare, usare `medidas.sort(key=lambda m: (m['bbox'][1], m['bbox'][0]))`, con `medidas = mm.measure(img, precision=0.02)`.

8.14.10.2 📌 Esempi

Input	Output
16 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ... (immagine binaria contenente due quadrati)	id area perimeter cx cy x y w h solidity 1 16.0 16.0 5.0 5.0 3 3 5 5 1.000 2 16.0 16.0 11.0 5.0 9 3 5 5 1.000

[req 418908] Traduci questo Markdown dal portoghese all'italiano:

Simulatore EP08_10: Separazione di Dischi Tangenti (Trasformata L2 & Watershed)

`mm.dist L2 → mm.watershed → Misurazione`

FASE DEL PROCESSAMENTO MORFOLOGICO

1. Grigio Rumoroso | 2. Maschera Unita | 3. Distanza L2 | 4. Watershed (Taglio)

Genera Dischi con Raggi Casuali

VISUALIZZAZIONE DELLA MATRICE DI PIXEL

0	0	0	0	142	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	153	152	150	148	154	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	150	158	157	155	140	151	158	0	0	148	0	0	0	0	0	0	0	0	0	0
0	142	157	149	145	153	145	149	0	140	150	154	0	0	0	0	0	0	0	0	0
150	158	154	150	158	145	157	145	156	148	159	153	146	0	0	0	0	0	0	0	0
0	150	140	148	158	151	143	143	0	158	154	154	0	0	0	0	0	0	0	0	0
0	152	147	156	157	156	144	156	0	0	142	0	0	0	0	0	0	0	0	0	0
0	0	152	151	156	146	149	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	152	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TABELLA DELLE MISURAZIONI DEI DISCHI DOPO IL TAGLIO DEL WATERSHED

id	area	perimetro	cx	cy	x	y	w	h	circolarità	solidità	vertici
1	37.0	24.14	3.92	4.00	0	0	8	9	0.798	0.925	8
2	7.0	10.49	10.17	4.00	9	2	4	5	0.800	1.000	6

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0810.html

Figura 8.24: Simulatore EP08_10: Separazione di Dischi Tangenti via Trasformata della Distanza L2 e *Watershed*

```
%%writefile EP08_10.py
# Codice Python
```

```
Overwriting EP08_10.py
```

```
TestSuite("EP08_10.py").run()
```

```
<IPython.core.display.HTML object>
```

8.14.11 EP08_11 🔴 Classificazione e Validazione di Oggetti con Riferimento di *Bounding Box*

In questo esercizio, l'obiettivo è elaborare un'immagine in scala di grigi contenente molteplici oggetti geometrici, estrarne le proprietà con `mm.measure` e validare le scatole delimitatrici (*bounding boxes*) rilevate rispetto a un riferimento reale (*Ground Truth* - GT) fornito in input, utilizzando la metrica IoU (*Intersection over Union*).

8.14.11.1 📋 Linee Guida di Implementazione

1. **Lettura dell'Immagine:** Leggere le dimensioni $H \times W$ e la matrice $H \times W$ di pixel dell'immagine in scala di grigi.
2. ***Pipeline* Morfologico:** Binarizzare l'immagine tramite il metodo di Otsu (`mm.threshold`) e visualizzare la maschera binarizzata risultante utilizzando `mm.drawImg`.
3. **Lettura del Riferimento Reale (*Ground Truth*):**

- Leggere la quantità G di scatole delimitatrici del riferimento.
- Se $G > 0$, leggere G righe contenenti 5 valori ciascuna: `id xmin_norm ymin_norm xmax_norm ymax_norm`.
- **Conversione delle Coordinate:** Le coordinate del riferimento sono normalizzate nell'intervallo $[0.0, 1.0]$. Per convertirle in pixel sulla griglia dell'immagine:

$$x_{\min} = \lfloor \text{xmin_norm} \times W \rfloor, \quad y_{\min} = \lfloor \text{ymin_norm} \times H \rfloor$$

$$w = \lfloor \text{xmax_norm} \times W \rfloor - x_{\min}, \quad h = \lfloor \text{ymax_norm} \times H \rfloor - y_{\min}$$

4. **Estrazione delle Metriche e Calcolo dell'IoU:**

- Estrarre le proprietà delle istanze con `mm.measure(img_bin, precision=0.02)`.
- Per ogni *bounding box* rilevata (x, y, w, h), calcolare la sovrapposizione IoU rispetto alle scatole del riferimento e definire `hits = 1` se esiste una corrispondenza con IoU ≥ 0.50, oppure `hits = 0` in caso contrario.

5. **Output:** Ordinare le istanze per posizione `(bbox[0], bbox[1])` e stampare la tabella CSV con la colonna aggiuntiva `hits`.
   - Per ordinare, usare `medidas.sort(key=lambda m: (m['bbox'][1], m['bbox'][0]))`, con `medidas = mm.measure(img)`.

8.14.11.2 🧠 Fondamenti Teorici e Conversione

Concetto	Formula / Operazione	Descrizione
BBox Rilevata	(x, y, w, h) tramite `mm.measure`	Scatola delimitatrice calcolata sulla griglia discreta in pixel interi.
BBox Riferimento (GT)	$(x_{\min}, y_{\min}, w, h)$ convertiti	Scatola reale fornita in input in coordinate relative $[0.0, 1.0]$.

Concetto	Formula / Operazione	Descrizione
IoU (Intersection over Union)	$\text{IoU} = \frac{\text{Area}(B_{\text{DET}} \cap B_{\text{GT}})}{\text{Area}(B_{\text{DET}} \cup B_{\text{GT}})}$	Valuta il tasso di sovrapposizione delle scatole. È considerata valida se IoU ≥ 0.50.
Stato di Validazione (`hits`)	1 se max(IoU) ≥ 0.50, altrimenti 0	Indicatore binario di correttezza del rilevatore rispetto al riferimento.

8.14.11.3 📦 Specifica di Input e Output (VPL)

Input:

- **Riga 1:** Interi H e W (dimensioni della matrice).
- **Successive H righe:** W interi (0 a 255) che rappresentano l'immagine in scala di grigi.
- **Riga $H+2$:** Intero G (quantità di scatole del riferimento reale).
- **Successive G righe:** 5 valori numerici per riga: `id xmin_norm ymin_norm xmax_norm ymax_norm` (dove le coordinate sono valori fluttuanti tra 0.0 e 1.0).

Output:

1. Matrice binarizzata visualizzata tramite `mm.drawImg(img_bin)`.
2. Intestazione CSV: `id,area,perimeter,cx,cy,x,y,w,h,circularity,solidity,vertices,hits`
3. Una riga CSV per oggetto rilevato contenente le sue proprietà formattate e l'indicatore `hits` (1 o 0).

8.14.11.4 📌 Esempi

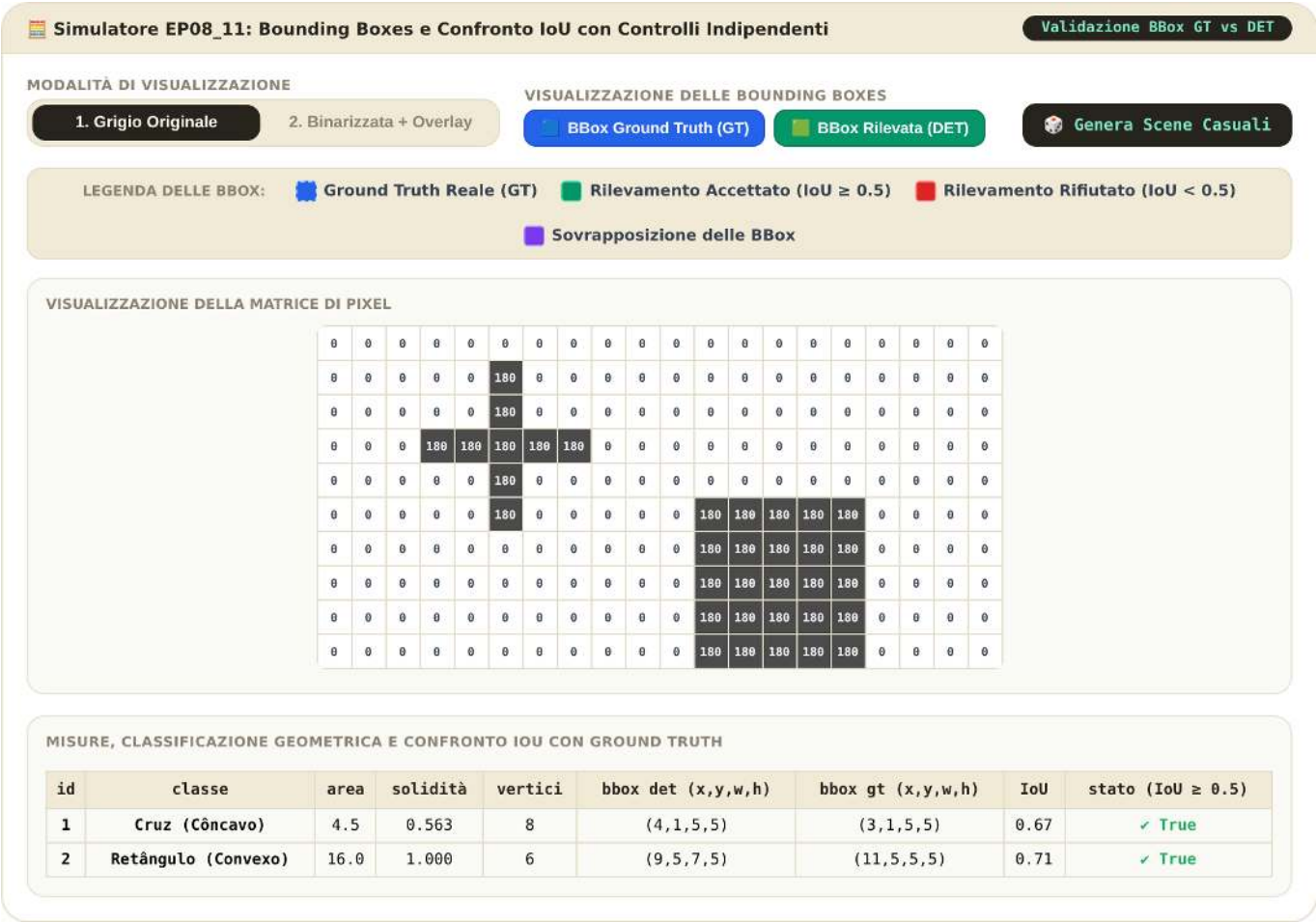
Input	Output
10 20 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 180 0 0 0 0 0 0 0 0 180 180 180 0 0 0 0 0 0 0 180 180 180 0 0 0 0 0 0 0 180 180 180 0 0 0 0 0 0 0 0 180 0 0 0 0 0 0 0 0 180 180 180 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 1 0.10 0.30 0.25 0.60 2 0.60 0.30 0.75 0.60	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 id area perimeter cx cy x y w h circularity solidity vertices hits 1 2.0 5.7 3.0 4.0 2 3 3 3 0.79 1.000 4 1 2 4.0 8.0 13.0 4.0 12 3 3 3 0.79 1.000 4 1

```
%%writefile EP08_11.py
# Codice Python
```

```
Overwriting EP08_11.py
```

```
TestSuite("EP08_11.py").run()
```

```
<IPython.core.display.HTML object>
```



id	classe	area	solidità	vertici	bbox det (x,y,w,h)	bbox gt (x,y,w,h)	IoU	stato (IoU ≥ 0.5)
1	Cruz (Côncavo)	4.5	0.563	8	(4,1,5,5)	(3,1,5,5)	0.67	✔ True
2	Retângulo (Convexo)	16.0	1.000	6	(9,5,7,5)	(11,5,5,5)	0.71	✔ True

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0811.html

Figura 8.25: Simulatore EP08_11: Classificazione Geometrica con Controlli Indipendenti di *Overlay BBox* (GT e DET)

8.14.12 EP08_12 🔴 Segmentazione di Istanza in Immagine Reale: Oggetti Geometrici

L'esempio classico di segmentazione di questo capitolo ha separato le "istanze" delle monete tramite disconnessione spaziale nella maschera binaria risultante dalla sogliatura di Otsu. In questo esercizio applicherai la stessa idea — ma ora su un'immagine reale con oggetti geometrici vari — concatenando pre-elaborazione, binarizzazione, estrazione dei contorni (`cv2.findContours`) e validazione del risultato rispetto a un ground truth di *bounding box*.

A differenza dell'esercizio precedente (etichettatura su maschera già pronta), qui parti dall'**immagine originale**: la qualità della tua segmentazione dipende direttamente dalle scelte di pre-elaborazione (filtraggio, sogliatura, operazioni morfologiche) effettuate prima di etichettare i componenti.

8.14.12.1 📋 Linee Guida di Implementazione

1. **Input:** utilizzare l'immagine `00000.jpg`.
2. **Pre-elaborazione e segmentazione:** applicare le fasi necessarie (filtraggio, binarizzazione e operazioni morfologiche) per separare automaticamente gli oggetti dallo sfondo, senza ritagli manuali.
3. **Etichettatura e misurazione:** per ogni oggetto segmentato, determinare:
   - area;
   - centro di massa (centroide);
   - tipo, secondo il set `obj2`.
4. **Annotazione visiva:** scrivere, all'interno di ciascun oggetto, la sua area e la sigla del tipo (`obj2`).
5. **Validazione (IoU):** calcolare l'*Intersection over Union* (IoU) tra la *bounding box* rilevata (`cv2.boundingRect`) e la *bounding box* di ground truth del tipo corrispondente. Un oggetto è considerato correttamente segmentato solo se esiste esattamente una *bounding box* del tipo corretto con **IoU 0,5**.
6. **Output:** stampare, per ogni oggetto rilevato, il suo identificatore, il tipo e se è stato validato con successo (`acertou=1`) o meno. La stampa deve seguire l'ordine delle classi di `obj2` (0=Tria … 8=Cruz); all'interno della stessa classe, ordinare gli oggetti per coordinata verticale del centroide (`cy`) crescente. Alla fine, stampare l'accuratezza complessiva.

8.14.12.2 📌 Vincoli Computazionali

- **Niente ritaglio manuale:** tutta la segmentazione deve essere eseguita sull'immagine completa.
- **Set di classi fisso:**

```
obj  = ['Triangulo','Quadrado','Pendagono','Hexagono','Heptagono','Circulo',
        'Elipse','Estrela','Cruz']
obj2 = ['Tria','Quad','Pent','Hexa','Hept','Circ','Elip','Estr','Cruz']
```

- **Dimensione dell'immagine:** 608×608 pixel — utilizzata per denormalizzare le coordinate del file TXT.
- **Validazione tramite centro di massa:** un oggetto è considerato correttamente segmentato solo se il suo centroide si trova strettamente all'interno della *bounding box* di ground truth corrispondente allo stesso tipo di oggetto.

8.14.12.3 🧠 Fondamenti Teorici

Elemento	Ruolo nella segmentazione di istanza
Pre-elaborazione (filtraggio, sogliatura)	Fase che produce la maschera binaria a partire dall'immagine di intensità originale
`cv2.findContours`	Estrae i contorni dei componenti connessi nella maschera binaria

Elemento	Ruolo nella segmentazione di istanza
Momenti geometrici (`cv2.moments`)	Consentono di calcolare il centro di massa (centroide) di ogni contorno
`approxPolyDP` / vertici	Aiuta nella classificazione del tipo di oggetto (numero approssimativo di lati)
Validazione tramite *bounding box*	Conferma se l'istanza segmentata corrisponde spazialmente a un oggetto del ground truth, misurando l'accuratezza del metodo

8.14.12.4 📌 Esempio di Output Atteso

```
Objeto 1: tipo=Tria, validado=True
...
Acurácia: 88.89%
```

Parametri fissi per la riproducibilità: affinché l'output corrisponda al ground truth di correzione automatica, utilizza esattamente: filtro di area minima di 300 pixel; `cv2.approxPolyDP` con `epsilon = 0.02 * perimetro`; soglia di solidità 0.92 e conteggio dei vertici 9 (con 11 per distinguere Cruz da Estrela) per forme concave; rapporto d'aspetto 1.15 per distinguere Circulo da Elipse; soglia IoU = 0.5 nella validazione.

8.14.12.5 📌 File di Riferimento (`.jpg` e `.txt`)

Per il debug locale, sono resi disponibili due file di riferimento (inclusi in questa consegna; quando li integri nel repository del capitolo, salvali in `all/cap08/dados/EP08/`):

- 📥 **Immagine (`00000.jpg`)**: immagine di oggetti geometrici utilizzata come input dell'esercizio. L'obiettivo è segmentare automaticamente ogni oggetto, determinarne il tipo e calcolarne le misure.
- 📥 **Ground Truth (`00000.txt`)**: file contenente le *bounding box* normalizzate degli oggetti presenti nell'immagine. Ogni riga contiene l'identificatore della classe e le coordinate normalizzate degli angoli superiore sinistro e inferiore destro, utilizzato per validare automaticamente la segmentazione.

La Figura 8.26 presenta l'immagine di input e la stessa immagine con le *bounding box* disegnate a partire dal file di ground truth.

```python
import os
import urllib.request
from morph import mm

def garantir_e_baixar(nome):
    pasta = "dados/EP12"
    caminho = os.path.join(pasta, nome)

    os.makedirs(pasta, exist_ok=True)

    if not os.path.exists(caminho):
        url = (
            "https://raw.githubusercontent.com/"
            "fzampirolli/pdi-vc/master/all/cap08/dados/EP12/"
            + nome
        )
        print(f"Scaricamento {nome}...")
        urllib.request.urlretrieve(url, caminho)

    return caminho

img_arq = garantir_e_baixar("00000.jpg")
```

```
txt_arq = garantir_e_baixar("00000.txt")

img = mm.read(img_arq)
img_bb = mm.showBoundBox(img, txt_arq, fmt="yolo", show=False)

mm.show(
    [img, img_bb],
    titles=[
        "Immagine originale",
        "Bounding boxes del riferimento"
    ],
    cols=2,
    figsize=(10,5)
)
```

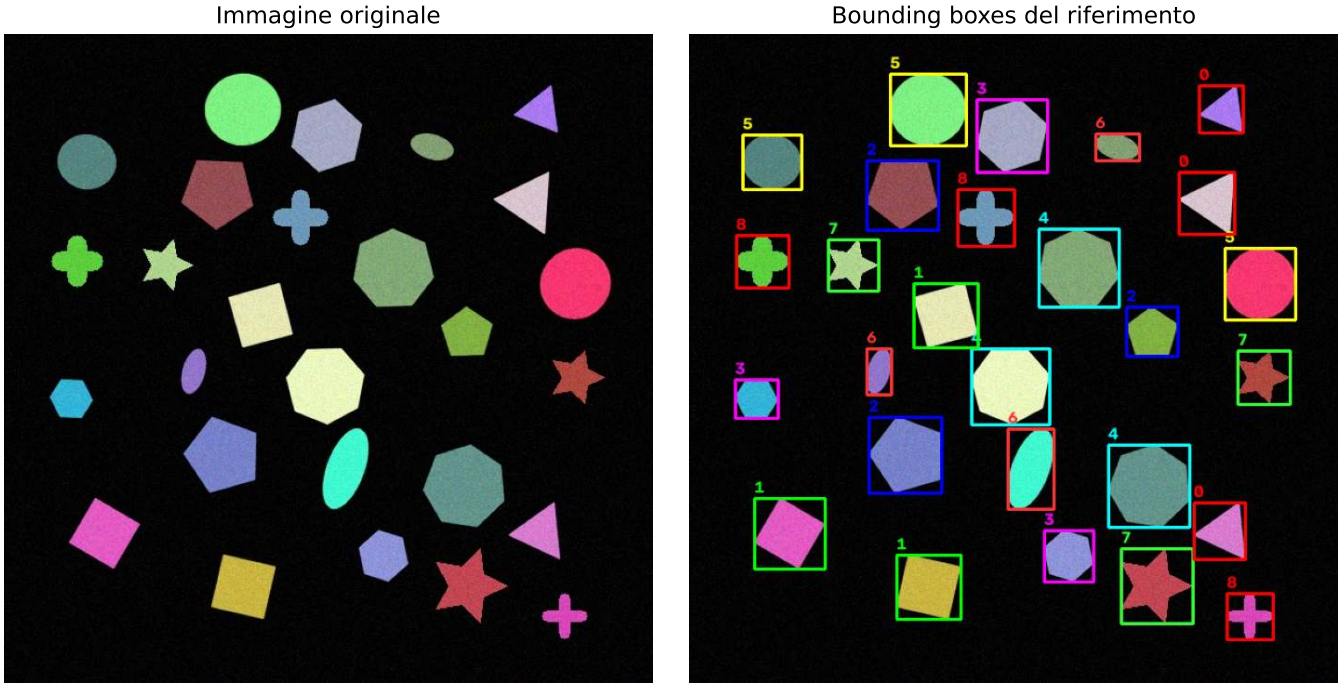


Figura 8.26: Simulatore EP08_12: Immagine utilizzata nell'EP08_12. A sinistra, l'immagine originale. A destra, l'immagine con le *bounding boxes* del file di riferimento.

```
%%writefile EP08_12.py
# Codice Python
```

```
Overwriting EP08_12.py
```

```
TestSuite("EP08_12.py").run()
```

```
<IPython.core.display.HTML object>
```

Simulatore EP08_12: Accuratezza della Segmentazione su Oggetti Multipli

riuscito se IoU ≥ soglia e tipo corretto

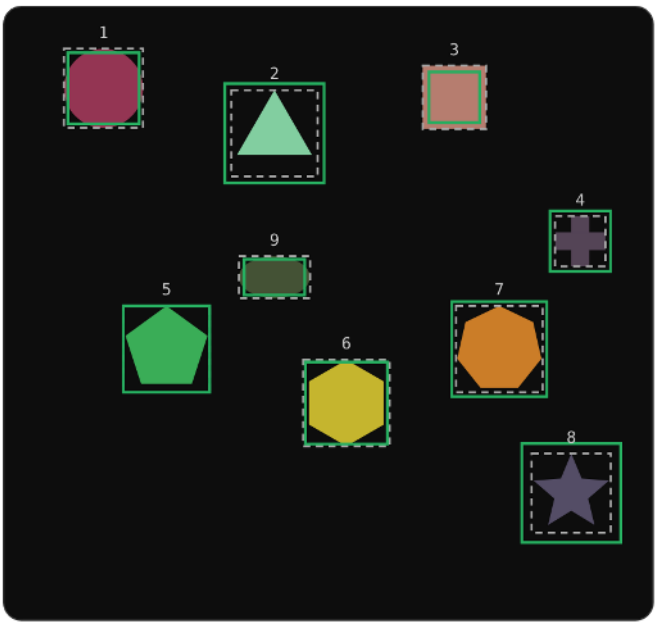
Ogni forma ha un *boundbox* di riferimento (rettangolo tratteggiato, aderente alla forma) e un *boundbox* rilevato (rettangolo pieno, spostato/rumoroso). Regola il rumore, il bias e la soglia IoU per vedere la validazione cambiare.

rumore di segmentazione (px, jitter max per lato) 0

bias sistematico in x (px) 0

soglia IoU 0.50

simula errore di classificazione (2 oggetti con tipo scambiato)



id	tipo reale	tipo rilevato	IoU	≥ soglia	riuscito
1	Circ	Circ	0.81	sim	✔
2	Tria	Tria	0.76	sim	✔
3	Quad	Quad	0.64	sim	✔
4	Cruz	Cruz	0.69	sim	✔
5	Pent	Pent	1.00	sim	✔
6	Hexa	Hexa	0.90	sim	✔
7	Hept	Hept	0.83	sim	✔
8	Estr	Estr	0.64	sim	✔
9	Elip	Elip	0.72	sim	✔

Objetos validados: 9 / 9 → Acurácia = 100.0%

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap08/sim-ep0812.html

Figura 8.27: Simulatore EP08_12: Accuratezza della Segmentazione su Multipli Oggetti (IoU)

Capitolo 9

Apprendimento Profondo per la Visione Artificiale

Executar Colab Abrir GitHub

I capitoli precedenti hanno stabilito le fondamenta della Visione Artificiale (VA) attraverso metodi classici di estrazione e rappresentazione delle caratteristiche. Nel **Capitolo 7**, descrittori come *Local Binary Patterns* (*LBP*) e *Histogram of Oriented Gradients* (*HOG*) hanno mostrato come trame e forme possano essere codificate da descrittori progettati manualmente. Nel **Capitolo 8**, algoritmi come *Oriented FAST and Rotated BRIEF* (*ORB*) e il rivelatore *Haar Cascade* hanno esteso questo principio a compiti di corrispondenza, rilevamento e riconoscimento di oggetti.

Queste tecniche rimangono rilevanti per la loro interpretabilità ed efficienza computazionale, ma dipendono da una fase preliminare di definizione manuale dei descrittori, denominata **ingegneria delle caratteristiche** (*feature engineering*). Questa dipendenza limita l'adattamento del modello a scenari per i quali il descrittore non è stato progettato.

L'**Apprendimento Profondo** (*Deep Learning*) propone un'alternativa: invece di specificare manualmente le caratteristiche rilevanti, il modello apprende automaticamente rappresentazioni dai dati durante l'addestramento — processo noto come **apprendimento di rappresentazioni** (*representation learning*) (GOODFELLOW, 2016). Nella VA, questa strategia è implementata principalmente dalle **Reti Neurali Convoluzionali** (*Convolutional Neural Networks* — CNN), nelle quali i filtri convoluzionali cessano di avere coefficienti fissi e vengono regolati da algoritmi di ottimizzazione.

Sebbene rappresentino un cambiamento nella costruzione di sistemi di riconoscimento di pattern, le CNN preservano concetti già studiati in questo libro: la convoluzione, presentata nel **Capitolo 3**, rimane l'operazione responsabile dell'estrazione locale delle caratteristiche, ora applicata con coefficienti appresi anziché progettati.

Va sottolineato che l'obiettivo di questo capitolo non è esplorare esaustivamente la teoria dell'Apprendimento Profondo, ma piuttosto offrire una panoramica dei suoi fondamenti e dimostrare come queste architetture vengano applicate nel contesto della VA. I lettori interessati a un approfondimento teorico e concettuale nell'area dovrebbero ricorrere a riferimenti specialistici della letteratura, come Goodfellow (2016) e Lecun (2015).

9.1 Obiettivi del Capitolo

Al termine di questo capitolo, lo studente dovrebbe essere in grado di:

- Collegare la convoluzione appresa dalle CNN con la convoluzione di *kernel* fissi presentata nel **Capitolo 3**;
- Descrivere l'architettura di base di una CNN e la funzione dei suoi strati principali;
- Implementare, addestrare e valutare modelli CNN per la classificazione delle immagini;
- Applicare il **transfer learning** (*transfer learning*) per adattare modelli pre-addestrati a nuovi problemi;
- Utilizzare modelli pre-addestrati in compiti di classificazione, rilevamento di oggetti e segmentazione;

- Implementare, addestrare e valutare un'architettura *U-Net* per la segmentazione semantica, confrontandola con approcci classici;
- Preparare set di dati annotati e integrarli in un *pipeline* di addestramento tramite piattaforme come **Roboflow**;
- Integrare geometria computazionale e Deep Learning in applicazioni di realtà aumentata e fotogrammetria.

La Figura 9.1 sintetizza l'organizzazione dei concetti studiati in questo capitolo e le relazioni tra essi.

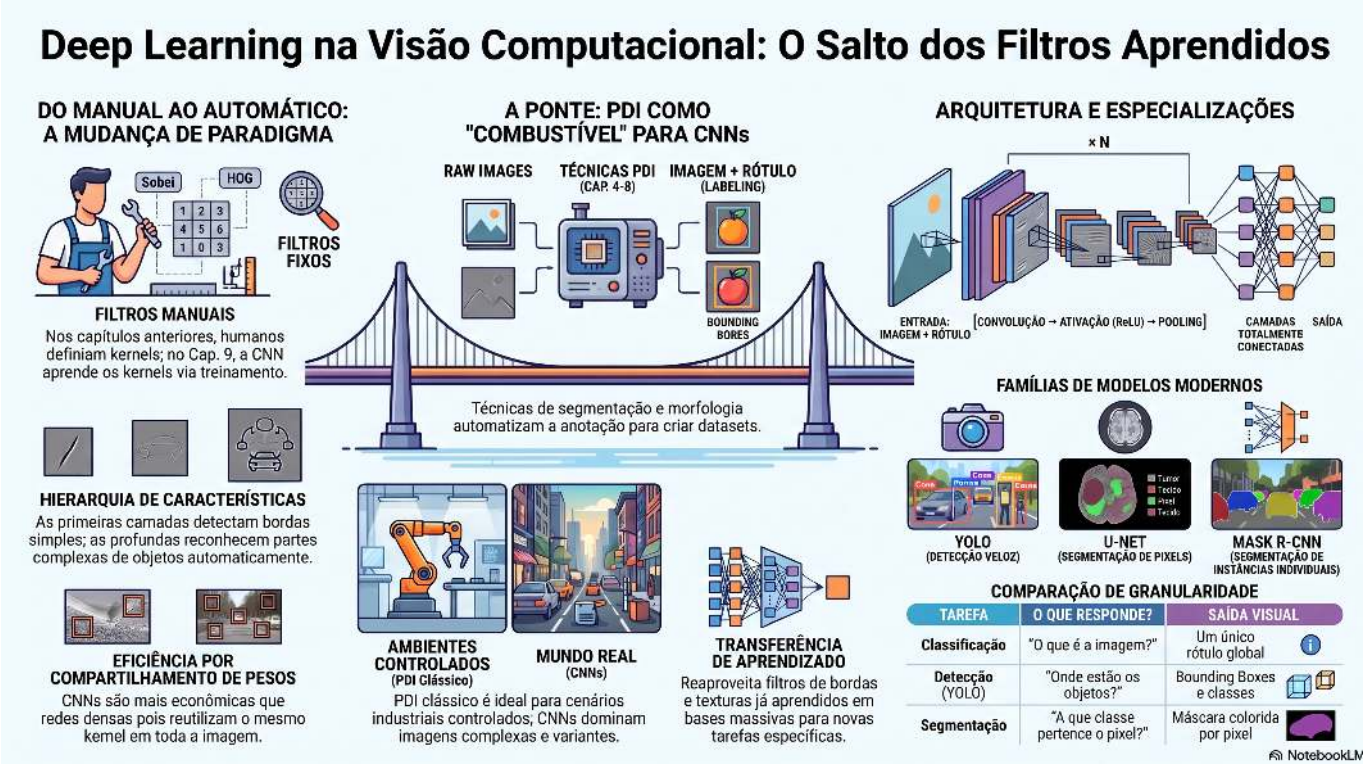


Figura 9.1: Panoramica dei principali concetti trattati in questo capitolo. **Fonte:** elaborato con l'ausilio di *Gemini Notebook* ({GOOGLE}, 2025).

9.2 Panoramica: Classificazione, Rilevamento e Segmentazione

I compiti di VC si differenziano, soprattutto, per l'informazione prodotta come output. La **classificazione** assegna un'unica etichetta all'intera immagine; il **rilevamento di oggetti** localizza e identifica gli oggetti presenti nella scena; la **segmentazione** associa una classe a ogni pixel e, in alcuni approcci, distingue differenti istanze di una stessa categoria.

La Tabella 9.1 riassume i compiti studiati lungo il libro, indicando la domanda a cui ciascuno risponde e la granularità dell'informazione prodotta.

Tabella 9.1: Confronto tra i principali compiti di VC secondo la granularità dell'informazione prodotta.

Compito	Domanda a cui risponde	Granularità dell'output	Capitolo
Classificazione	"Qual è la classe di questa immagine?"	Un'unica etichetta per l'intera immagine	7 e 9
Rilevamento di oggetti	"Quali oggetti esistono e dove si trovano?"	Classe e bounding box per ogni oggetto	8 e 9
Segmentazione semantica	"A quale classe appartiene ogni pixel?"	Un'etichetta di classe per ogni pixel	8 e 9

Compito	Domanda a cui risponde	Granularità dell'output	Capitolo
Segmentazione delle istanze	"Quali pixel appartengono a ogni oggetto?"	Un'etichetta per pixel per ogni istanza	8 e 9
Segmentazione panottica	"Qual è la classe e l'identità di ogni oggetto?"	Classe e identificatore di istanza per ogni pixel	8 e 9

Questi compiti rappresentano livelli crescenti di interpretazione dell'immagine: la classificazione descrive la scena in modo globale, il rilevamento aggiunge la localizzazione degli oggetti e la segmentazione produce una rappresentazione spaziale dettagliata, consentendo di analizzare ciascuna regione individualmente. Questo capitolo si concentra, in primo luogo, sulla classificazione tramite CNN, per poi estendere gli stessi principi al rilevamento e alla segmentazione.

9.3 Configurazione dell'Ambiente

Gli esempi di questo capitolo utilizzano **PyTorch**, un *framework* ampiamente impiegato nello sviluppo e nell'addestramento di modelli di Deep Learning. Il codice seguente verifica la disponibilità delle librerie necessarie e installa automaticamente quelle non ancora presenti nell'ambiente di esecuzione.

Se **PyTorch** non è installato, viene selezionata automaticamente una versione compatibile con l'hardware disponibile: la versione con supporto **CUDA**, se è disponibile una GPU NVIDIA, oppure la versione per l'esecuzione su CPU, in caso contrario.

Successivamente, l'ambiente viene inizializzato con l'importazione delle librerie utilizzate nel corso del capitolo, la definizione di un seed casuale per favorire la riproducibilità degli esperimenti e il recupero del file `morph.py` — la libreria didattica di elaborazione morfologica già utilizzata nei capitoli precedenti —, qualora non sia ancora disponibile nella directory di lavoro.

```
import contextlib, importlib, importlib.metadata, importlib.util
import io, os, random, shutil, subprocess, sys, urllib.request, warnings

# Sopprime gli avvisi di PyTorch e i warning generali
warnings.filterwarnings("ignore", category=UserWarning)

url = ("https://raw.githubusercontent.com/fzampirolli/"
       "pdi-vc/master/morph/config.py")
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config

# Silenzia stdout e stderr sia a livello di Python che di File Descriptors del SO
def setup_silencioso():
    with open(os.devnull, "w") as fnull:
        old_out = os.dup(1)
        old_err = os.dup(2)
        try:
            os.dup2(fnull.fileno(), 1)
            os.dup2(fnull.fileno(), 2)
            with contextlib.redirect_stdout(fnull), contextlib.redirect_stderr(fnull):
                config.setup()
        finally:
            os.dup2(old_out, 1)
            os.dup2(old_err, 2)
            os.close(old_out)
            os.close(old_err)

setup_silencioso()
```

```python
from morph import mm


def setup_cap09():
    """Installa le librerie mancanti per questo capitolo in modo 100% silenzioso."""
    pkgs = {
        "skimage": "scikit-image", "numpy": "numpy",
        "sklearn": "scikit-learn", "matplotlib": "matplotlib",
        "torchviz": "torchviz", "ultralytics": "ultralytics",
        "roboflow": "roboflow",
    }
    for mod, pkg in pkgs.items():
        if importlib.util.find_spec(mod) is None:
            subprocess.run([sys.executable, "-m", "pip", "install", "-q", pkg],
                           stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

    if importlib.util.find_spec("torch") is None:
        args = (["torch", "torchvision"] if shutil.which("nvidia-smi")
                else ["--index-url",
                      "https://download.pytorch.org/whl/cpu",
                      "torch", "torchvision"])
        subprocess.run([sys.executable, "-m", "pip", "install", "-q", *args],
                       stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

    if not shutil.which("dot") and shutil.which("apt-get"):
        subprocess.run(["apt-get", "install", "-y", "-qq", "graphviz"],
                       stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL, check=False)


setup_cap09()

import cv2, numpy as np, torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
import torchvision.models as models
import torchvision.transforms as T
from PIL import Image
from skimage import data as skdata
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from torch.utils.data import DataLoader, TensorDataset
from torchvision.datasets import OxfordIIITPet
from torchvision.models.detection import (
    fasterrcnn_resnet50_fpn, FasterRCNN_ResNet50_FPN_Weights)
from torchvision.models.segmentation import deeplabv3_resnet50
from torchvision.transforms.functional import to_tensor
from torchviz import make_dot
from ultralytics import YOLO

torch.manual_seed(42)
FLAG_LIMPAR_DADOS = False
device = "cuda" if torch.cuda.is_available() else "cpu"
gpu = f" ({torch.cuda.get_device_name(0)})" if device == "cuda" else ""
ver = importlib.metadata.version("ultralytics")
print(f"  Ambiente pronto. OpenCV {cv2.__version__} | "
      f"morph {getattr(mm, '__version__', 'local_file')} | "
      f"PyTorch {torch.__version__} | Ultralytics {ver} | {device}{gpu}")
```

```
  Ambiente pronto. OpenCV 5.0.0 | morph local_file | PyTorch 2.6.0+cu124 | Ultralytics 8.4.113
| cuda (NVIDIA GeForce GTX TITAN X)
```

9.4 Fondamenti di Deep Learning per la VC

Le CNN sono la principale architettura di Deep Learning applicata all'analisi delle immagini. Il loro funzionamento si basa sulla composizione di operazioni convoluzionali organizzate in strati successivi, in cui i filtri appresi durante l'addestramento trasformano l'immagine in rappresentazioni progressivamente più astratte. In questa sezione vengono presentati i concetti fondamentali che collegano la convoluzione spaziale studiata in precedenza ai moderni modelli di VC, inclusa l'estrazione gerarchica delle caratteristiche, il processo di addestramento e l'utilizzo di modelli pre-addestrati.

9.4.1 Dalla Convoluzione Fissa alla Convoluzione Appresa

Il **Capitolo 3** ha presentato la convoluzione spaziale con *kernel* fissi, come gli operatori di Sobel, progettati per evidenziare caratteristiche specifiche di un'immagine. Nei **Capitoli 7** e **8**, lo stesso principio ha sostenuto descrittori come *HOG*, *LBP* e *ORB*, oltre al rilevatore *Haar Cascade*: in tutti questi casi, i filtri sono definiti prima dell'esecuzione dell'algoritmo e rimangono invariati durante l'elaborazione.

La Figura 9.2 riprende il funzionamento della convoluzione spaziale: il simulatore consente di selezionare diversi *kernel* e di seguire lo spostamento della finestra di convoluzione su un'immagine. In ogni posizione, i coefficienti del *kernel* si combinano con l'intorno locale dell'immagine — denominato **campo recettivo** (*receptive field*) — per produrre un valore della **mappa delle caratteristiche** (*feature map*), illustrando anche la condivisione dei pesi (*weight sharing*).

Le CNN preservano questa operazione, ma sostituiscono i *kernel* fissi con **filtri appresi**: invece di coefficienti definiti in precedenza, la rete aggiusta questi valori durante l'addestramento a partire da esempi etichettati, cercando di minimizzare una **funzione di perdita** (*loss function*), che misura la differenza tra le previsioni del modello e le risposte attese.

La differenza essenziale tra i metodi classici e le CNN, quindi, non sta nell'operazione di convoluzione in sé, ma nel modo in cui i filtri vengono ottenuti: mentre i primi utilizzano filtri progettati manualmente, le CNN apprendono, a partire dai dati di addestramento, rappresentazioni adeguate al compito.

Per comprendere come avviene questo apprendimento, è necessario studiare l'unità di base dell'elaborazione delle reti neurali: il **neurone artificiale**.

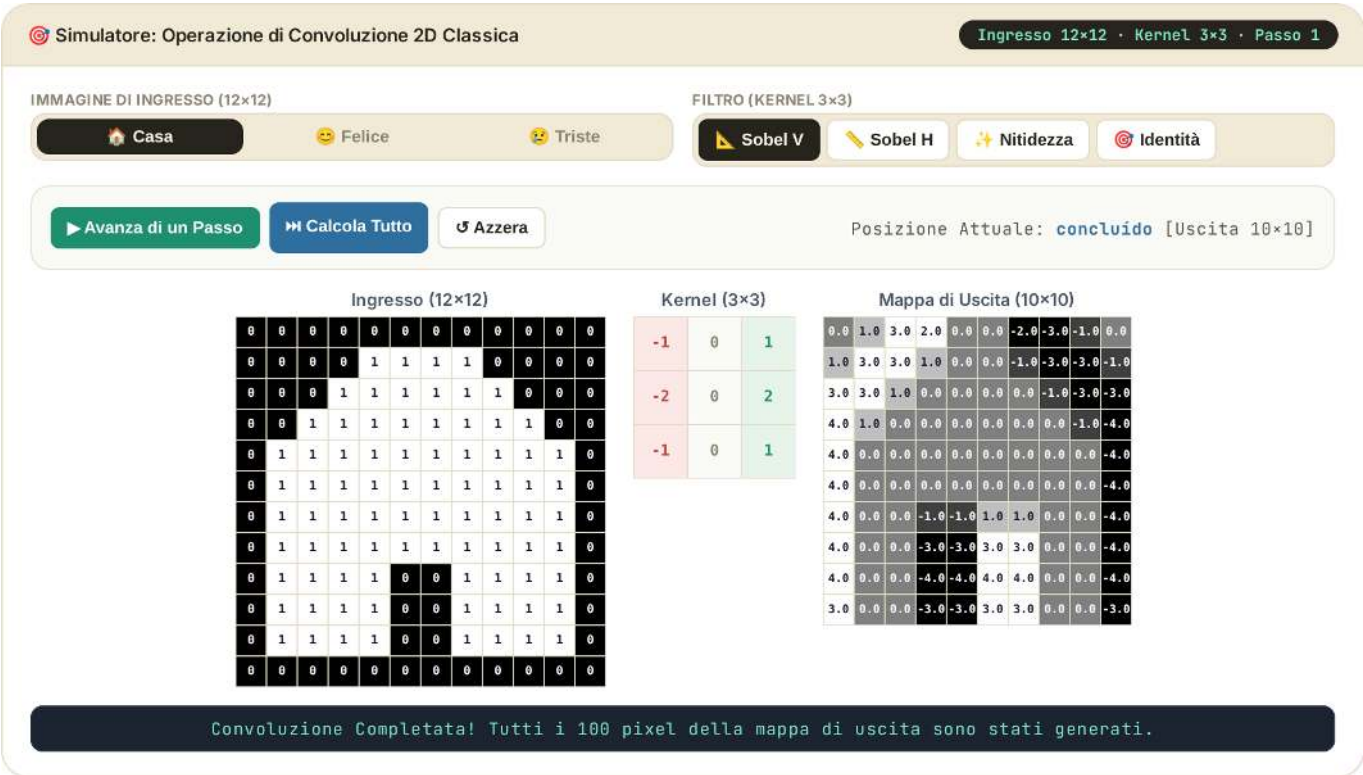
9.4.2 Neurone Artificiale

Il **neurone artificiale** (*artificial neuron*) è l'unità fondamentale di elaborazione di una rete neurale. Il suo primo modello matematico — un insieme di ingressi combinati e confrontati con una soglia — fu proposto da Mcculloch (1943), ancora privo di qualsiasi meccanismo di apprendimento. Il **Percettrone** (ROSENBLATT, 1958) fece avanzare questa formulazione introducendo una regola di aggiustamento dei pesi basata su esempi, diventando il primo modello di neurone artificiale capace di apprendere e la base delle architetture moderne di **Apprendimento Profondo** (*Deep Learning*). Il termine Apprendimento Profondo si riferisce all'uso di reti con più strati di elaborazione, in grado di apprendere rappresentazioni gerarchiche dei dati: i primi strati apprendono caratteristiche semplici, come bordi e trame, e gli strati più profondi combinano progressivamente queste rappresentazioni per identificare strutture e oggetti più complessi.

Ogni neurone riceve un insieme di ingressi, calcola una combinazione lineare di tali valori e applica una **funzione di attivazione** (*activation function*), producendo un unico valore di uscita. Matematicamente, la combinazione lineare è data da

$$z = \sum_{i=1}^{n} w_i x_i + b,$$

dove x_i rappresentano gli ingressi, w_i i pesi associati a ciascun ingresso e b il **bias** (*bias*). L'uscita del neurone si ottiene applicando la funzione di attivazione:



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-convolucao.html

Figura 9.2: Simulatore interattivo di convoluzione 2D classica: scegli tra le tre immagini sintetiche di input 12×12 (casa, faccia felice o triste) e un filtro 3×3 (Sobel V, Sobel H, Nitidezza o Identità) e procedi passo dopo passo per osservare come i prodotti interni locali del campo recettivo costruiscono la mappa delle caratteristiche cella per cella.

$$y = f(z).$$

Nelle CNN, questo principio assume forme diverse a seconda dello strato. Negli **strati convoluzionali** (*convolutional layers*), ogni neurone elabora solo una piccola regione dell'ingresso, denominata **campo recettivo** (*receptive field*), preservando l'organizzazione spaziale dell'immagine. Negli **strati completamente connessi** (*fully connected layers*), ogni neurone riceve tutte le uscite dello strato precedente, combinando le caratteristiche estratte per produrre l'uscita finale della rete, come la classe attribuita all'immagine.

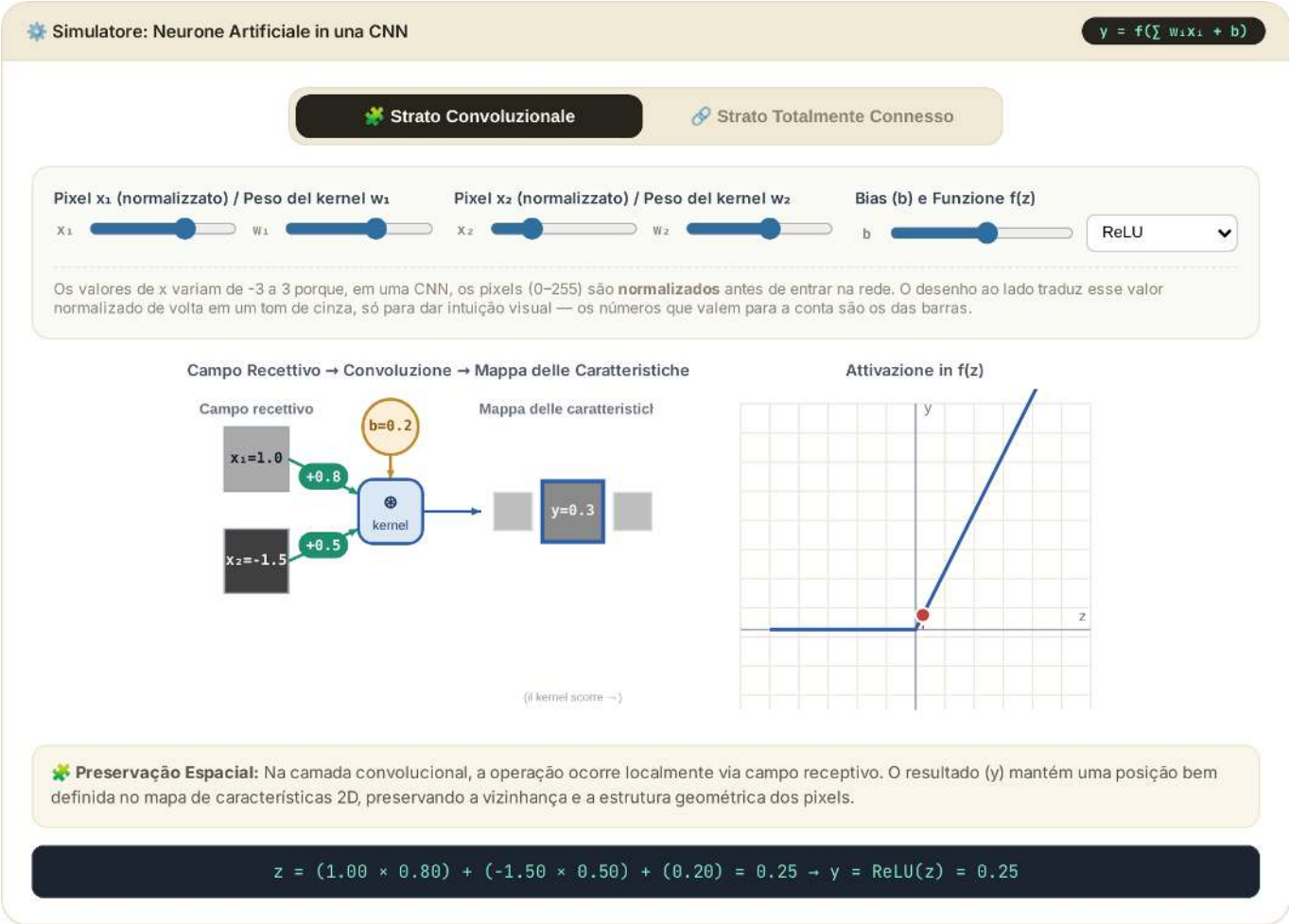
La Figura 9.3 illustra il funzionamento di un neurone artificiale: il simulatore consente di modificare gli ingressi (x_1 e x_2), i pesi (w_1 e w_2), il bias (b) e la funzione di attivazione, osservando in tempo reale il calcolo della combinazione lineare e dell'uscita corrispondente.

In una CNN, migliaia di neuroni si organizzano in strati con funzioni specifiche: i primi sono responsabili dell'estrazione delle caratteristiche tramite la convoluzione, mentre gli ultimi eseguono la classificazione a partire dalle caratteristiche apprese.

9.4.3 Livello Convoluzionale

Il **livello convoluzionale** (*convolutional layer*) è responsabile dell'estrazione delle caratteristiche dell'immagine. Ogni filtro genera una **mappa delle caratteristiche** (*feature map*), la cui intensità in ciascuna posizione indica la risposta del filtro alla regione corrispondente dell'input.

L'operazione eseguita segue lo stesso principio di scorrimento e combinazione locale presentato nel **Capitolo 3** per la convoluzione spaziale. Considerando un *kernel* K di dimensione $k \times k$, il valore prodotto nella posizione (i, j) è dato da



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-neuronio.html

Figura 9.3: Simulatore interattivo del neurone artificiale in un contesto di CNN: passa da un neurone di un livello convoluzionale (dove x_i sono intensità di pixel in un campo ricettivo e w_i sono pesi del *kernel*) a un neurone di un livello completamente connesso, regolando input, pesi, bias e funzione di attivazione per visualizzare il calcolo di z e dell'output y in tempo reale.

$$F(i,j) = \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} K(u,v)\, I(i+u, j+v).$$

Vale la pena registrare una distinzione terminologica: l'espressione sopra corrisponde, formalmente, a una **correlazione incrociata** (*cross-correlation*), e non alla convoluzione matematica rigorosa, che richiede la riflessione del *kernel* prima della combinazione. La maggior parte dei *framework* di Deep Learning, incluso **PyTorch**, implementa questa operazione senza riflessione e la designa, per convenzione, come convoluzione — convenzione adottata anche in questo capitolo. Questa differenza non ha alcun effetto pratico sull'allenamento, poiché i coefficienti del *kernel* vengono appresi e non imposti in anticipo.

La differenza principale rispetto ai metodi classici risiede quindi nell'ottenimento del *kernel* K: nei filtri tradizionali, i suoi coefficienti sono definiti manualmente per evidenziare caratteristiche specifiche dell'immagine; nelle CNN, i coefficienti vengono inizializzati automaticamente e regolati durante l'allenamento tramite la **retropropagazione dell'errore** (*backpropagation*), rendendo ogni filtro specializzato nell'identificare pattern rilevanti per il compito in esame.

Due concetti caratterizzano questo livello:

- **Condivisione dei pesi** (*weight sharing*): lo stesso filtro viene applicato in tutte le posizioni dell'immagine, riducendo significativamente il numero di parametri del modello.
- **Campo recettivo** (*receptive field*): ogni neurone convoluzionale processa solo un piccolo intorno dell'immagine, preservando la struttura spaziale dei dati.

Impilando più livelli convoluzionali, la rete apprende una **gerarchia di caratteristiche**: i primi livelli tendono a rilevare pattern semplici, come bordi e trame, e i livelli più profondi combinano queste informazioni per rappresentare strutture progressivamente più complesse. Dopo la convoluzione, la mappa delle caratteristiche viene sottoposta a una funzione di attivazione, introducendo non linearità nel modello e ampliando la sua capacità di rappresentare relazioni complesse tra le variabili di input.

La Figura 9.4 presenta questo livello in modo interattivo.

9.4.4 Funzione di Attivazione

La convoluzione è un'operazione lineare. Affinché la rete possa modellare relazioni non lineari tra ingressi e uscite, si applica una **funzione di attivazione** (*activation function*) dopo ogni strato convoluzionale.

La funzione più utilizzata nelle CNN è la **ReLU** (*Rectified Linear Unit*), definita da

$$\mathrm{ReLU}(x) = \max(0, x).$$

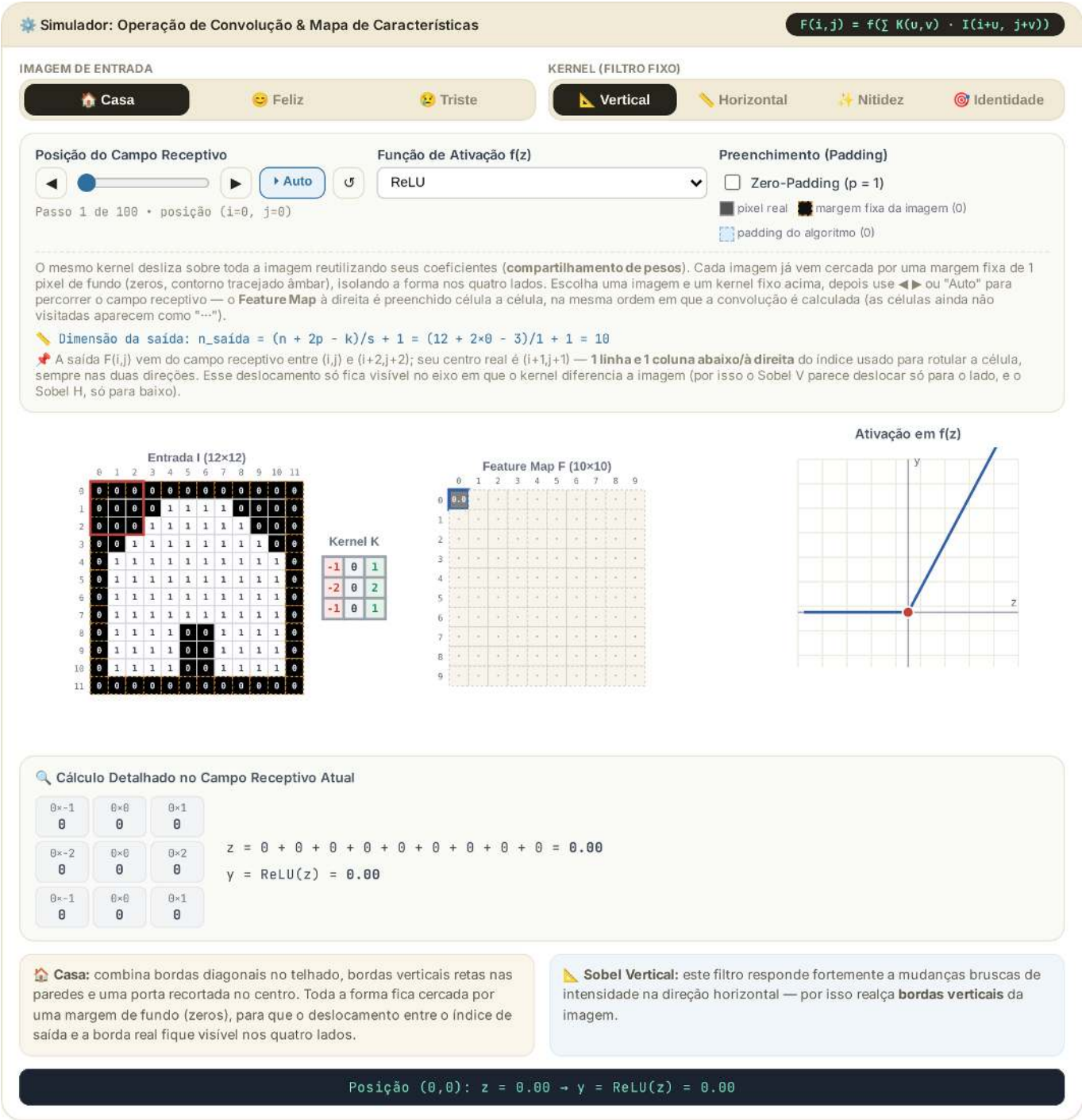
Questa funzione preserva i valori positivi e sostituisce con zero i valori negativi, introducendo non linearità nel modello e favorendo l'addestramento di reti profonde con basso costo computazionale.

La Figura 9.5 illustra il funzionamento della **ReLU** applicata sia a valori individuali sia a una mappa delle caratteristiche, consentendo di confrontare l'uscita prima e dopo l'attivazione.

I mappe di caratteristiche risultanti dalla convoluzione e dall'attivazione preservano la struttura spaziale dell'immagine. In molte architetture, la fase successiva ne riduce la risoluzione tramite un'operazione di *pooling*.

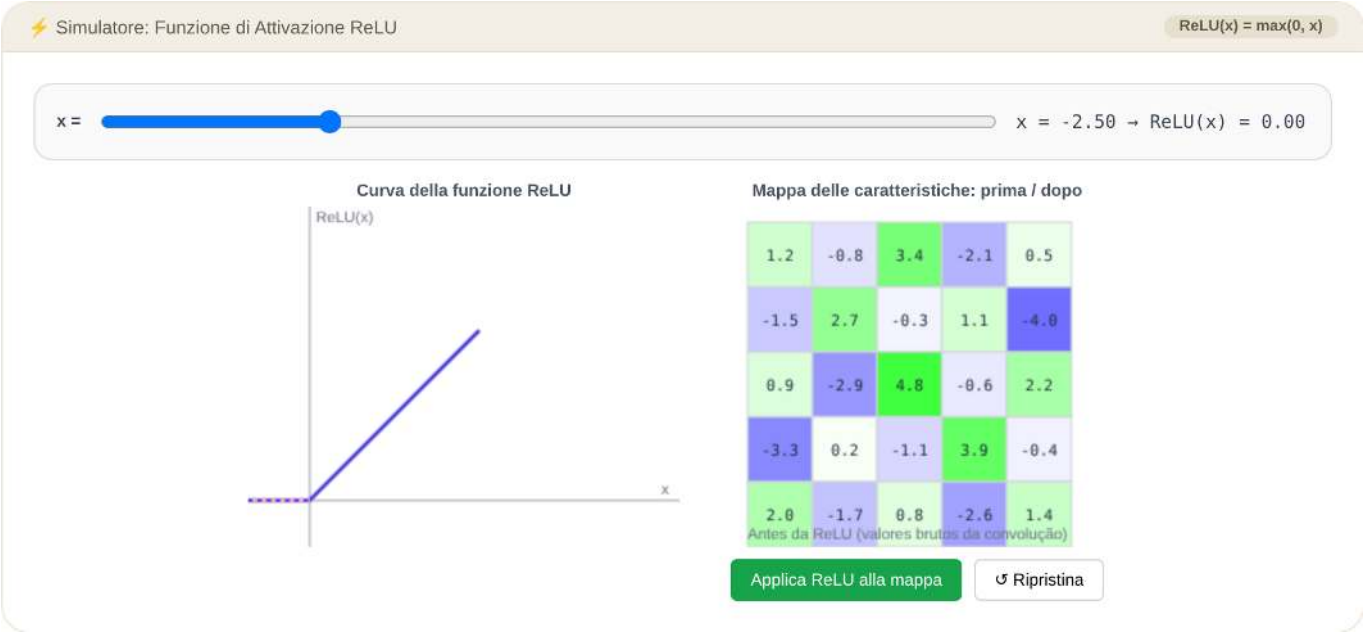
9.4.5 *Pooling*

Il livello di ***pooling*** riduce la risoluzione spaziale delle mappe delle caratteristiche, preservando le informazioni più rilevanti per le fasi successive dell'elaborazione. L'operazione più utilizzata è il ***max-pooling***, che seleziona il valore massimo in ogni finestra dell'immagine:



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-camada-conv.html

Figura 9.4: Simulador interativo da Camada Convolucional: escolha entre três imagens de entrada 12×12 (casa, rosto feliz ou rosto triste) para observar como os mesmos *kernels* fixos reagem a diferentes bordas e formas. Navegue pelo campo receptivo com os botões ou reprodução automática, ajuste a função de ativação e alterne o zero-padding, acompanhando o mapa de características sendo revelado célula a célula, com o cálculo detalhado termo a termo e a fórmula da dimensão de saída em tempo real.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-relu.html

Figura 9.5: Simulatore interattivo della funzione di attivazione ReLU: trascina il controllo per vedere come i valori negativi vengono azzerati e i valori positivi vengono preservati, sia nella curva che in una mappa delle caratteristiche reale.

$$P(i,j) = \max_{(u,v)\in\text{finestra}(i,j)} F(u,v).$$

Questa riduzione diminuisce il costo computazionale dei livelli successivi e rende la rappresentazione più robusta a piccole variazioni nella posizione dei pattern presenti nell'immagine.

La Figura 9.6 presenta questa operazione su una mappa delle caratteristiche di 8×8 pixel, ridotta a 4×4 tramite finestre di 2×2 con passo pari a 2, alternando tra **max-pooling** e **average-pooling** — che calcola, invece del massimo, la media dei valori della finestra corrispondente.

Complessivamente, convoluzione, funzione di attivazione e *pooling* formano il blocco fondamentale utilizzato nella costruzione di una CNN.

9.4.6 Addestramento delle Reti Neurali: Come le CNN Imparano

Una CNN impara regolando automaticamente i propri parametri — i coefficienti dei filtri convoluzionali, i pesi dei livelli completamente connessi e i bias — a partire da esempi etichettati. Questo addestramento è iterativo e coinvolge tre fasi: misurare l'errore prodotto dalla rete tramite una **funzione di perdita** (*loss function*), calcolare come tale errore dipende da ciascun parametro tramite la **retropropagazione** (*backpropagation*) e aggiornare i parametri con un **algoritmo di ottimizzazione** (*optimizer*).

9.4.6.1 Funzione di Perdita (*Loss Function*)

La **funzione di perdita** (*loss function*) quantifica la differenza tra la previsione della rete e la risposta corretta, denominata **verità di riferimento** (*ground truth*). Il risultato è uno scalare L: minore è la perdita, più la previsione è vicina alla risposta attesa.

Nei problemi di classificazione multiclasse, la funzione più utilizzata è l'**Entropia Incrociata** (*Cross-Entropy Loss*), applicata alle probabilità prodotte dal livello **Softmax**:

$$L = -\sum_{c=1}^{C} y_c \log(\hat{y}_c),$$

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-pooling.html

Figura 9.6: Simulatore interattivo di *pooling*: scegli tra *max-pooling* e *average-pooling* e procedi passo dopo passo per osservare la riduzione della risoluzione spaziale della mappa delle caratteristiche.

dove C è il numero di classi, y_c è l'etichetta reale in codifica *one-hot* e $\hat{y}_c$ è la probabilità prevista per la classe c. La perdita si avvicina a zero quando la rete attribuisce alta probabilità alla classe corretta e cresce rapidamente man mano che tale probabilità diminuisce.

La Figura 9.7 illustra questo comportamento: il simulatore consente di selezionare la classe corretta e modificare le probabilità prodotte dalla *Softmax*, mostrando in tempo reale la variazione della funzione di perdita.

9.4.6.2 Retropropagazione (*Backpropagation*)

Dopo il calcolo della perdita, è necessario determinare come ciascun parametro della rete contribuisca a questo risultato. Questa fase è realizzata dalla **retropropagazione** (*backpropagation*), che applica la **Regola della Catena** del calcolo differenziale per ottenere il gradiente della funzione di perdita rispetto a ciascun parametro.

Per un parametro w, questo gradiente è dato da

$$\frac{\partial L}{\partial w}.$$

Il gradiente indica come varia la perdita in relazione a piccole modifiche di w: un gradiente positivo indica che aumentare w aumenta la perdita, mentre un gradiente negativo indica l'effetto opposto.

La Figura 9.8 presenta questo processo in modo visivo, mostrando la propagazione del gradiente dallo strato di output fino ai primi strati convoluzionali.

9.4.6.3 Algoritmi di Ottimizzazione

Dopo il calcolo dei gradienti, un **algoritmo di ottimizzazione** (*optimizer*) aggiorna i parametri della rete per ridurre la funzione di perdita. Nelle reti profonde, questa ricerca avviene in uno spazio ad alta dimensionalità e, in generale, **non convesso**, il che rende l'ottimizzazione un problema complesso.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-loss.html

Figura 9.7: Simulatore interattivo della Funzione di Perdita (*Cross-Entropy*): seleziona la classe reale dell'immagine (Casa, Felice o Triste) e regola le probabilità stimate dalla Softmax per visualizzare il calcolo della penalizzazione scalare e il grafico del logaritmo negativo in tempo reale.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-backprop.html

Figura 9.8: Simulatore interattivo di *Backpropagation*: procedi attraverso i passaggi della Regola della Catena per seguire il flusso del segnale di errore nella direzione inversa della rete, osservando il calcolo delle derivate parziali del gradiente in ogni strato.

Per facilitare la comprensione, la Figura 9.9 utilizza una **superficie di perdita semplificata**, con un minimo globale, un minimo locale e una barriera tra queste regioni. Il **minimo globale** corrisponde al valore più basso della funzione di perdita e rappresenta il miglior insieme di parametri della rete; un **minimo locale** presenta anch'esso una bassa perdita, ma può essere distante dalla soluzione ottimale. Quando l'ottimizzazione rimane intrappolata in un minimo locale, gli aggiustamenti di filtri, pesi e bias diventano molto piccoli, e l'addestramento si ferma prima di raggiungere un modello con errore inferiore.

Discesa del Gradiente Stocastica (*SGD*)

La **Discesa del Gradiente Stocastica** (*Stochastic Gradient Descent — SGD*) aggiorna i parametri nella direzione opposta al gradiente:

$$w_{\text{novo}} = w_{\text{atual}} - \eta \frac{\partial L}{\partial w},$$

dove η è il **tasso di apprendimento** (*learning rate*), responsabile del controllo dell'ampiezza dell'aggiornamento. La *SGD* utilizza solo il gradiente dell'iterazione corrente; quando la ricerca raggiunge un minimo locale, i gradienti diventano molto piccoli e gli aggiornamenti cessano praticamente.

Ottimizzatori Adattivi: *Adam*

L'**Adam** (*Adaptive Moment Estimation*) combina stime adattative dei primi e secondi momenti dei gradienti (KINGMA, 2015), adattando il tasso di apprendimento di ogni parametro individualmente. Questa adattazione favorisce, in molti casi, il superamento dei minimi locali che tratterebbero il *SGD*.

La Figura 9.9 confronta la traiettoria del *SGD* e dell'*Adam* sulla stessa superficie di perdita non convessa.

9.4.7 Architettura di una CNN

Una CNN per la classificazione delle immagini combina i livelli presentati nelle sezioni precedenti. Durante il **passo in avanti** (*forward pass*), l'immagine attraversa successivamente i livelli convoluzionali, le funzioni di attivazione, le operazioni di *pooling*, la fase di **Flatten**, i livelli completamente connessi e, infine, il livello **Softmax**, che produce le probabilità delle classi. Durante l'addestramento, questa previsione viene confrontata con l'etichetta corretta per calcolare la funzione di perdita, eseguire la retropropagazione e aggiornare i parametri tramite un algoritmo di ottimizzazione.

La Figura 9.10 presenta questo flusso di elaborazione e addestramento.

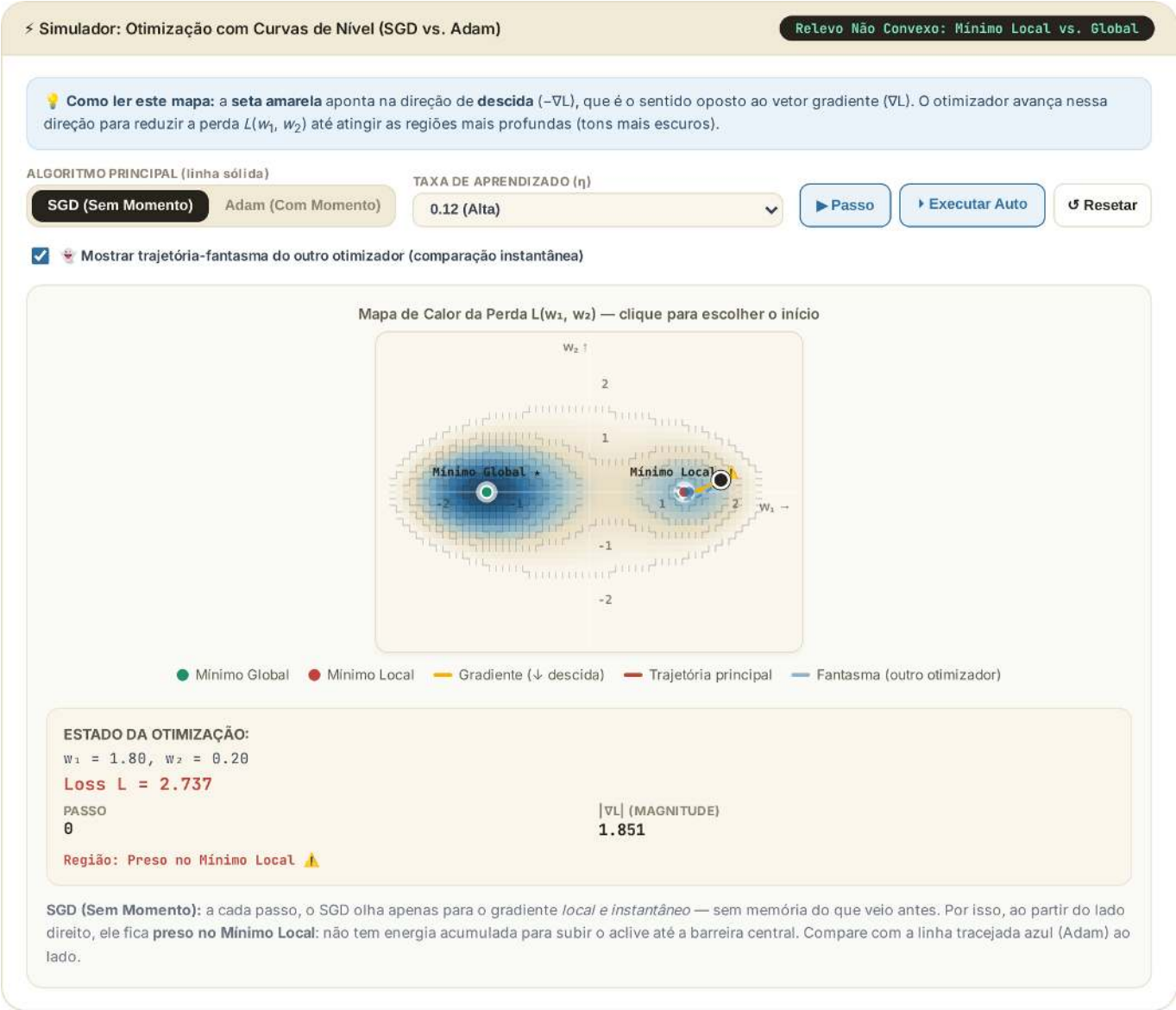
Dopo l'ultimo blocco convoluzionale, l'operazione **Flatten** riorganizza le mappe delle caratteristiche in un vettore unidimensionale, che alimenta i **livelli completamente connessi** (*fully connected layers*), responsabili di combinare le caratteristiche estratte per produrre i punteggi (*logits*) di ciascuna classe. Il livello **Softmax** converte questi punteggi in una distribuzione di probabilità, utilizzata sia per la classificazione sia per il calcolo della funzione di perdita durante l'addestramento.

La Figura 9.11 presenta una versione interattiva di questa architettura, consentendo di eseguire fasi successive di addestramento e di osservare la riduzione della perdita, la retropropagazione dei gradienti e l'aggiornamento dei filtri della rete.

9.4.8 Come il Gradiente Regola i *Kernel* della Convoluzione

La comprensione del processo di apprendimento in una Rete Neurale Convoluzionale (CNN) richiede la chiarificazione di un meccanismo fondamentale: **come i coefficienti casuali di un filtro iniziale si trasformano in rilevatori precisi di bordi, texture e pattern complessi.**

La risposta risiede nel principio della **condivisione dei pesi (*weight sharing*)**. Durante la fase di propagazione in avanti (*forward pass*), lo stesso filtro di dimensione 3×3 scorre su tutta l'estensione



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-opt.html

Figura 9.9: Simulador interativo dos Algoritmos de Otimização: compare a trajetória do SGD e do Adam sobre uma superfície de perda não convexa com mapa de calor e curvas de nível. A linha sólida mostra o otimizador selecionado avançando passo a passo; a linha tracejada mostra, para comparação instantânea, o caminho completo que o outro otimizador percorreria a partir do mesmo ponto inicial. Observe como o SGD fica retido no Mínimo Local à direita, enquanto o Adam pode ou não transpor a barreira central dependendo do impulso acumulado e da taxa de aprendizado. Clique em qualquer ponto do mapa para redefinir o ponto inicial dos pesos.

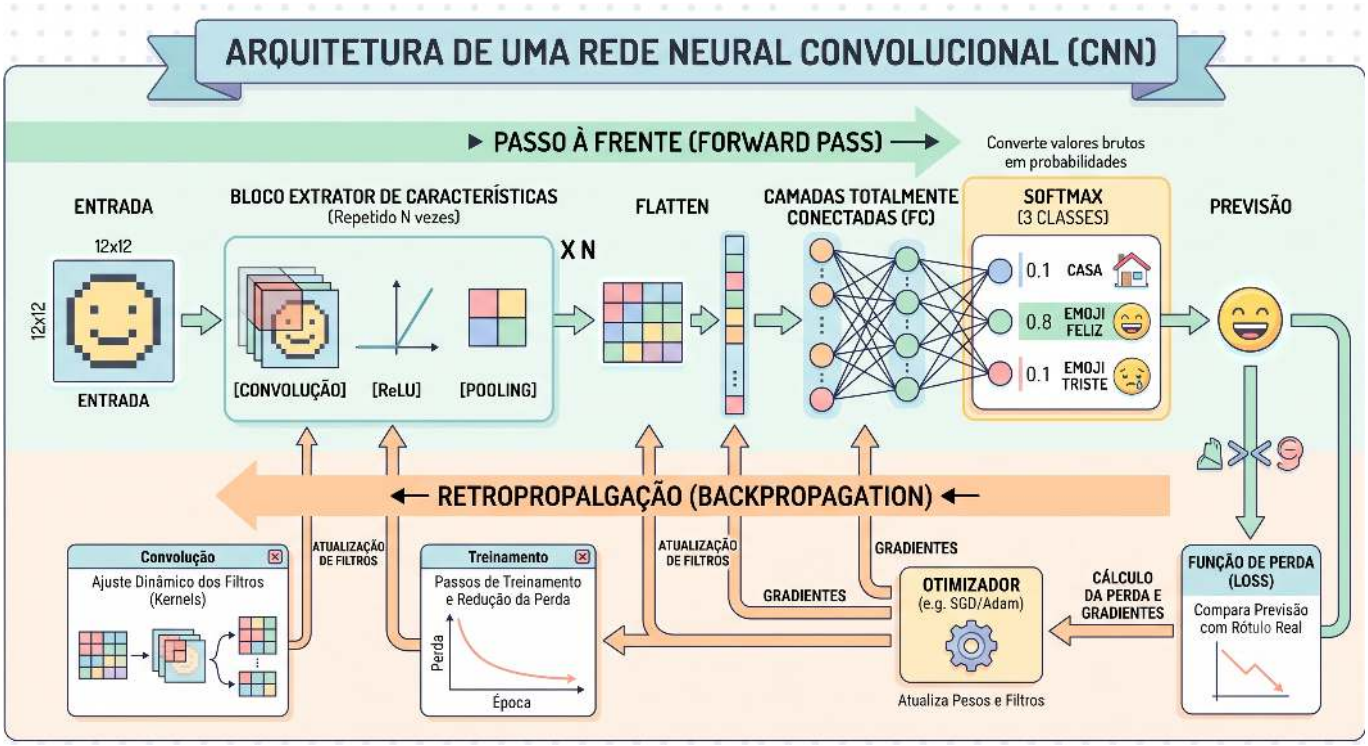


Figura 9.10: Architettura semplificata di una CNN per la classificazione delle immagini, evidenziando il *forward pass* e le fasi di addestramento tramite la funzione di perdita, la retropropagazione e l'algoritmo di ottimizzazione.

dell'immagine di input. Di conseguenza, ogni peso del *kernel* — come l'elemento $K[0][0]$ nell'angolo superiore sinistro — viene riutilizzato più volte lungo le diverse regioni spaziali del dato di input.

Durante la fase di retropropagazione (*backpropagation*), questo riutilizzo stabilisce una dinamica diretta: **ogni posizione spaziale elaborata dal filtro genera un contributo individuale ("voto") per l'aggiornamento del rispettivo peso.**

9.4.8.1 L'Intuizione Dietro il Calcolo

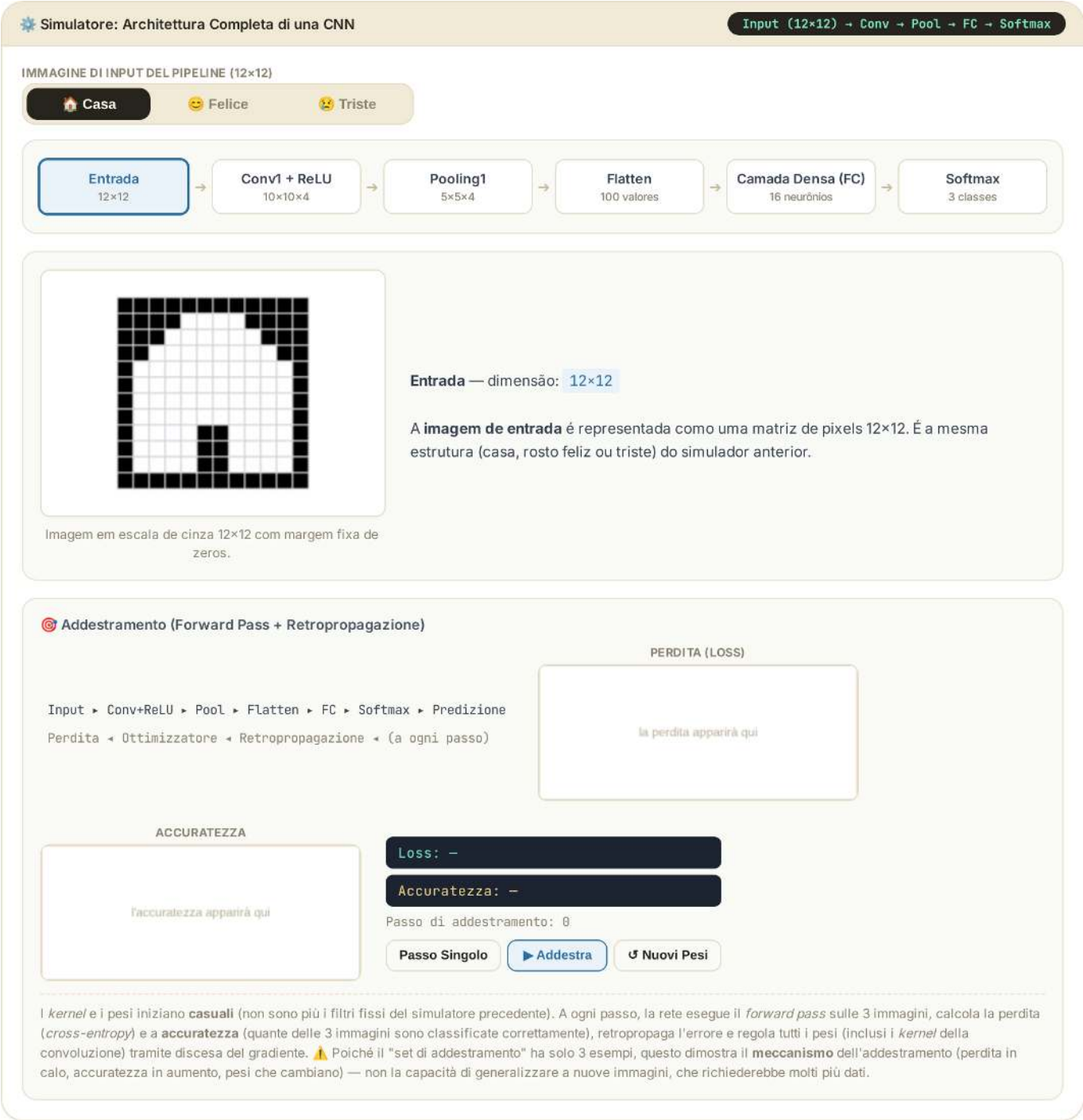
Sia $Z[r][c]$ la mappa delle caratteristiche pre-attivazione nella posizione (r, c) della finestra scorrevole, ottenuta tramite la correlazione incrociata tra il *kernel* K e l'input X:

$$Z[r][c] = \sum_{k_r} \sum_{k_c} K[k_r][k_c] \cdot X[r + k_r][c + k_c]$$

Applicando la regola della catena per determinare il contributo di un peso specifico $K[k_r][k_c]$ nella funzione di perdita L, si ottengono i seguenti passaggi:

1. **Errore Locale (dZ):** In ogni posizione (r, c), si calcola la derivata parziale della funzione di perdita rispetto alla pre-attivazione:
$$dZ[r][c] = \frac{\partial L}{\partial Z[r][c]}$$
che quantifica la responsabilità di quella specifica posizione nell'errore totale della rete (L).

2. **Contributo del Peso:** Poiché $\frac{\partial Z[r][c]}{\partial K[k_r][k_c]} = X[r + k_r][c + k_c]$, l'influenza di un peso specifico $K[k_r][k_c]$ sull'errore della posizione (r, c) si ottiene moltiplicando l'errore locale $dZ[r][c]$ per il valore del pixel di input allineato a quel peso nel momento del calcolo:
$$dZ[r][c] \cdot X[r + k_r][c + k_c]$$



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-arquitetura.html

Figura 9.11: Simulatore interattivo dell'architettura di una CNN: scegli una delle immagini di ingresso 12×12 (casa, viso felice o triste), clicca su ciascun blocco del *pipeline* — Ingresso, *Conv+ReLU*, *Pooling*, *Flatten*, *FC* e *Softmax* — ed esegui passi di addestramento reali (*forward pass* + retropropagazione) per osservare la perdita e l'accuratezza evolversi, i *kernel* essere regolati e il *Softmax* iniziare a puntare alla classe corretta.

3. **Accumulo dei Gradienti:** Il gradiente finale del peso corrisponde alla **somma dei contributi ("voti") di tutte le posizioni** percorse dalla finestra scorrevole:

$$\frac{\partial L}{\partial K[k_r][k_c]} = \sum_{(r,c)} dZ[r][c] \cdot X[r+k_r][c+k_c]$$

Questa formulazione assicura una parità diretta tra la derivazione analitica e i valori calcolati nel simulatore di ispezione del gradiente (Figura 9.12).

9.4.8.2 Il Ruolo della Funzione ReLU come "Filtro di Rilevanza"

L'applicazione della funzione di attivazione **ReLU** ($\max(0, z)$) immediatamente dopo la convoluzione introduce una proprietà di selettività al gradiente:

- **Attivazione Positiva ($Z[r][c] > 0$):** La derivata della ReLU è 1. L'errore locale viene propagato integralmente ($dZ \neq 0$), consentendo alla posizione di contribuire all'aggiornamento dei pesi del *kernel*.
- **Attivazione Inattiva ($Z[r][c] \leq 0$):** La derivata della ReLU è 0. L'errore locale viene annullato ($dZ = 0$), sopprimendo il contributo della posizione al gradiente finale.

  Nota didattica: La ReLU garantisce che solo le regioni spaziali che hanno prodotto risposte attive durante la propagazione in avanti abbiano la capacità di modificare i pesi del *kernel* nel processo di retropropagazione.

9.4.8.3 Aggiornamento dei Pesi tramite Discesa del Gradiente

Dopo il consolidamento dei gradienti accumulati da tutte le posizioni, l'aggiornamento del peso avviene secondo l'algoritmo della Discesa del Gradiente Stocastica (SGD):

$$K[k_r][k_c] \leftarrow K[k_r][k_c] - \eta \cdot \frac{\partial L}{\partial K[k_r][k_c]}$$

dove η denota il **tasso di apprendimento (*learning rate*)**.

- Se la somma dei gradienti è **positiva**, il valore del peso viene ridotto.
- Se la somma è **negativa**, il valore del peso viene incrementato.

9.4.8.4 Esplorando il Simulatore Interattivo

Dall'Architettura Globale all'Ispezione del Gradiente

Nel simulatore di architettura (Figura 9.11), si osserva l'errore dZ derivato dalla completa retropropagazione multistrato, originato dalla perdita di entropia incrociata (*Softmax*) sulle immagini di input 12×12.
Per rendere possibile la verifica analitica del gradiente senza il sovraccarico di 100 posizioni di convoluzione e retropropagazione multistrato, il simulatore di apprendimento del *kernel* (Figura 9.12) adotta un modello di ispezione ridotto (6×6). In questo scenario, si semplifica il problema sostituendo la classificazione complessa con un **meta-dato di calibrazione scalare**: si regola il filtro per produrre una risposta accumulata predefinita (alvo $= 9$) quando identifica un pattern specifico (come un bordo a 45 gradi). Il meccanismo di accumulo dei gradienti ($dZ \cdot X$) rimane rigorosamente identico in entrambe le formulazioni.

Per ispezionare questa dinamica a livello numerico, si utilizza il simulatore nella Figura 9.12:

- **Immagine di Input (X):** Matrice 6×6.
- **Filtro Convoluzionale (K):** Matrice 3×3 (9 pesi).
- **Mappa di Output (Z / A):** Matrice 4×4 (16 posizioni della finestra).

- **Funzione di Perdita (L):** Definita da $L = \frac{1}{2}(S - \text{alvo})^2$, dove $S = \sum A[r][c]$ rappresenta la somma globale delle attivazioni post-ReLU.

  Il ruolo del alvo $= 9$: Il valore scalare alvo $= 9$ rappresenta l' "energia di attivazione" ideale stabilita per l'immagine con bordo diagonale. Poiché la mappa A ha 16 posizioni, questo valore equivale a cercare una risposta media di $\frac{9}{16} \approx 0,56$ per pixel attivato. Quando $S > 9$, la rete identifica che il filtro sta reagendo con intensità eccessiva al pattern, generando un errore $dZ > 0$ che forza la riduzione dei pesi K. Quando $S < 9$, i pesi vengono incrementati per amplificare il segnale.

Percorso Suggerito per la Sperimentazione:

1. **Selezione del Peso:** Nella griglia 3×3, scegli il peso da analizzare (es.: $K[0][0]$).
2. **Scansione della Finestra:** Utilizza il pulsante **"▷ Avanza posizione"** per seguire lo spostamento della finestra attraverso le 16 posizioni spaziali. Nota l'evidenziazione visiva nella cella della mappa di input che allinea il pixel X al peso selezionato.
3. **Analisi del Voto Locale:** Esamina il prodotto dell'errore locale per il pixel di input ($dZ \cdot X$) nel pannello di calcolo della posizione.
4. **Verifica della Cronologia:** Segui il consolidamento dei 16 risultati parziali organizzati nelle quattro colonne della cronologia, osservando l'accumulo del gradiente finale.
5. **Aggiornamento del Kernel:** Clicca su **"▷ Applica passo di discesa del gradiente"** per visualizzare la convergenza della curva di perdita e l'adattamento del *kernel* casuale al pattern di input selezionato.

🧠 Sintesi — Dalla convoluzione all'apprendimento di rappresentazioni

I simulatori di questa sezione dimostrano, in modo sequenziale, come una CNN trasformi un'immagine in ingresso in una stima probabilistica e come i suoi parametri vengano ottimizzati durante l'addestramento:

- **Convoluzione:** applica filtri sull'immagine per estrarre caratteristiche locali, generando mappe di caratteristiche mediante la **condivisione dei pesi**.
- **ReLU:** introduce non linearità nel sistema, consentendo la modellazione di relazioni complesse tra i dati.
- ***Pooling*:** riduce la risoluzione spaziale delle mappe di caratteristiche, diminuendo il costo computazionale e conferendo invarianza a piccole traslazioni locali.
- ***Flatten*:** riorganizza le mappe multidimensionali in un vettore unidimensionale per alimentare i layer successivi.
- **Layer completamente connesso:** combina le caratteristiche estratte per produrre i punteggi grezzi (*logits*) associati a ciascuna classe.
- **Softmax:** converte i *logits* in una distribuzione di probabilità normalizzata.
- **Funzione di perdita:** confronta la distribuzione prevista con la verità di riferimento (*ground truth*), quantificando scalarmente l'errore della rete.
- **Retropropagazione:** applica la regola della catena per calcolare la derivata parziale (gradiente) della funzione di perdita rispetto a ciascun parametro addestrabile.
- **Ottimizzatore:** aggiorna i coefficienti dei filtri, i pesi e i bias nella direzione opposta al gradiente, riducendo la perdita a ogni iterazione.

Nel corso delle iterazioni, i filtri convoluzionali si trasformano da valori stocastici in rilevatori specializzati: i layer iniziali apprendono primitive visive di basso livello (come bordi e trame), mentre i layer più profondi consolidano queste rappresentazioni in strutture astratte e semantiche.

9.5 Applicazioni Pratiche in VC

Dopo la consolidazione teorica dei fondamenti delle CNN e la verifica visiva di ciascuna delle loro operazioni elementari tramite i simulatori interattivi, diventa essenziale osservare l'integrazione di queste fasi in *pipeline* complete di programmazione.

Simulatore: Gradiente di un Peso del Kernel

∂Perdita / ∂K[kr][kc] = Σ dZ · X

MODELLO DI INPUT (IMMAGINE 6×6)

Bordo diagonale | Bordo verticale

TASSO DI APPRENDIMENTO (η)

0.0005 (Lento) | 0.002 (Ideale) | 0.02 (Alto)

Esempio **ridotto**: immagine 6×6 e filtro 3×3 che generano mappe 4×4. Fai clic sulle schede **"Come viene calcolato?"** sotto ogni matrice per capire i calcoli passo dopo passo. **Passa il mouse su qualsiasi cella** di X, Z, A, dZ o K per vedere il calcolo esatto di quel valore, con gli elementi usati nei layer correlati evidenziati con un contorno tratteggiato/blu.

1. PESO DEL KERNEL

K[0][0]	K[0][1]	K[0][2]
K[1][0]	K[1][1]	K[1][2]
K[2][0]	K[2][1]	K[2][2]

KERNEL ATTUALE (K)

-0.34	0.13	-0.20
0.20	0.09	-0.05
0.32	-0.26	0.01

Come viene aggiornato?

INPUT X (6×6)

1	0	0	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0
1	1	1	1	0	0
1	1	1	1	1	0
1	1	1	1	1	1

Come funziona X?

PRE-ATTIVAZIONE Z (4×4)

0.02	0.26	0.32	0.00
0.10	0.02	0.26	0.32
-0.10	0.10	0.02	0.26
-0.10	-0.10	0.10	0.02

Come calcola Z?

ATTIVAZIONE A (4×4)

0.02	0.26	0.32	0.00
0.10	0.02	0.26	0.32
0.00	0.10	0.02	0.26
0.00	0.00	0.10	0.02

Come calcola A?

ERRORE dZ (4×4)

-7.20	-7.20	-7.20	0.00
-7.20	-7.20	-7.20	-7.20
0.00	-7.20	-7.20	-7.20
0.00	0.00	-7.20	-7.20

S = 1.800 (alvo = 9)

L = ½·(1.800 - 9)² = 25.9200

Come calcola dZ, S e Loss?

2. CALCOLO E SOMMA DEI "VOTI" DI OGNI POSIZIONE

$\partial L/\partial K[0][0] = \Sigma_{(r,c)}\, dZ[r][c] \cdot X[r+0][c+0]$

Torna alla posizione | Avanza alla posizione | Somma tutto | Reimposta posizioni

CALCOLO DI QUESTA POSIZIONE

Clique em "Avançar posição" para ver a primeira conta.

SOMMA ACCUMULATA (GRADIENTE)

posições: 0 / 16

total = 0.000

STORICO DELLE 16 POSIZIONI (COLONNE c=0, c=1, c=2, c=3)

p(0,0) -7.20×1=-7.20	p(0,1) -7.20×0=0.00	p(0,2) -7.20×0=0.00	p(0,3) 0.00×0=0.00
p(1,0) -7.20×1=-7.20	p(1,1) -7.20×1=-7.20	p(1,2) -7.20×0=0.00	p(1,3) -7.20×0=0.00
p(2,0) 0.00×1=0.00	p(2,1) -7.20×1=-7.20	p(2,2) -7.20×1=-7.20	p(2,3) -7.20×0=0.00
p(3,0) 0.00×1=0.00	p(3,1) 0.00×1=0.00	p(3,2) -7.20×1=-7.20	p(3,3) -7.20×1=-7.20

3. USA IL GRADIENTE PER AGGIORNARE IL KERNEL

Applica passo di discesa del gradiente | Annulla passo | Nuovo kernel casuale

perda aparecerá aqui

Aggiornamenti: 0

PERDITA NEL CORSO DEGLI AGGIORNAMENTI

Estado Inicial do peso K[0][0] = -0.340:

- **Gradiente visível acumulado (∇K):** -50.400 (soma dos 16 votos dZ × X)
- **Passo de ajuste previsto (η × ∇K):** 0.002 × (-50.400) = **-0.1008**
- **Novo Peso previsto:** K ← -0.340 − (-0.1008) = **-0.239**

Clique em "**▶ Aplicar passo de gradiente descendente**" para atualizar a matriz.

Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-09-gradiente-kernel.html

Figura 9.12: Simulatore interattivo del calcolo del gradiente di un peso del *kernel* convoluzionale.

Nelle sezioni seguenti, la teoria viene tradotta in codice eseguibile in **PyTorch**, esplorando i tre compiti fondamentali della VC: **classificazione**, **rilevamento di oggetti** e **segmentazione semantica**. Questa progressione pratica consente di analizzare dalla costruzione di un'architettura convoluzionale addestrata da zero fino all'applicazione di strategie avanzate di **transfer learning** (*transfer learning*) su modelli pre-addestrati per set di dati sintetici e reali.

9.5.1 Classificazione delle Immagini con CNN

La classificazione delle immagini è una delle applicazioni più tradizionali delle CNN. In questo compito, l'obiettivo è assegnare un'unica etichetta all'immagine in ingresso, come identificare una categoria di oggetto, una specie animale o una classe diagnostica. A tal fine, la CNN trasforma progressivamente i valori dei pixel in rappresentazioni a più alto livello di astrazione, combinando strati convoluzionali, funzioni di attivazione e operazioni di riduzione spaziale fino a produrre una distribuzione di probabilità tra le classi possibili. In questa sezione vengono presentati l'architettura di base di una CNN classificatrice, il flusso di trasformazione dei dati lungo la rete e il processo di addestramento per l'ajustamento dei parametri appresi.

9.5.1.1 Addestramento di una CNN da Zero su Cifre

Per stabilire un confronto diretto con gli approcci presentati nel Capitolo 7, in questa sezione si sviluppa una CNN addestrata sullo stesso insieme di dati di cifre scritte a mano (`load_digits`). La differenza fondamentale risiede nella fase di rappresentazione: mentre i metodi classici dipendono da *pixel* grezzi o da descrittori calcolati manualmente, come l'*Histogram of Oriented Gradients* (*HOG*), la CNN apprende automaticamente i coefficienti dei filtri convoluzionali durante il processo di ottimizzazione.

I codici seguenti (consolidati nella Figura 9.15) eseguono la preparazione dei dati, definiscono un'architettura convoluzionale semplice in **PyTorch**, eseguono il ciclo di addestramento tramite l'algoritmo *Adam* e generano le curve di evoluzione della funzione di perdita e dell'accuratezza.

Blocco 1: Preparazione e Strutturazione dei Dati

La fase iniziale di qualsiasi *pipeline* di Deep Learning consiste nella conversione e nell'adattamento dei dati di input al formato richiesto dal *framework* di calcolo scientifico.

Il Concetto di *Tensor*

Nel Deep Learning, la struttura fondamentale dei dati è il ***tensor***. Dal punto di vista computazionale, un *tensor* consiste in un array multidimensionale di numeri generalizzato a n dimensioni:

- Un *tensor* di ordine 0 è uno scalare (un singolo valore).
- Un *tensor* di ordine 1 è un vettore (lunghezza).
- Un *tensor* di ordine 2 è una matrice (righe e colonne).
- Un *tensor* di ordine 3 o superiore rappresenta un volume o un iper-array di dati.

Nel contesto di **PyTorch**, la classe `torch.Tensor` estende la funzionalità degli array numerici multidimensionali (come quelli di **NumPy**) offrendo supporto a operazioni accelerate su hardware tramite *GPU* (*Graphics Processing Units*) e supporto al calcolo automatico delle derivate (*autograd*), essenziale per l'algoritmo di retropropagazione.

Analisi del Codice di Pre-elaborazione

1. **Caricamento e Normalizzazione delle Intensità:** Il dataset `load_digits` contiene 1.797 campioni di cifre scritte a mano di 8×8 *pixel*, le cui intensità originali variano sulla scala intera da 0 a 16. La divisione per 16.0 esegue la **normalizzazione** dei dati nell'intervallo $[0.0, 1.0]$. Questa trasformazione in scala floating-point (`float32`) è indispensabile nelle reti neurali per evitare la saturazione delle funzioni di attivazione e stabilizzare il calcolo dei gradienti nell'algoritmo di ottimizzazione.

2. **Divisione Stratificata (70% Addestramento / 30% Test):** La funzione `train_test_split` separa il 70% dei campioni per l'adattamento dei parametri della rete e riserva il 30% per la valutazione del modello su dati non visti. Il parametro `stratify=y` garantisce il campionamento stratificato, mantenendo la proporzione esatta di ciascuna delle 10 classi di cifre (0 a 9) in entrambi gli insiemi, prevenendo distorsioni nella distribuzione.

3. **Adattamento Dimensionale per la Convoluzione 2D (`unsqueeze`):** Nelle CNN, i layer convoluzionali bidimensionali (`nn.Conv2d`) richiedono che il *tensor* di input possieda strettamente 4 dimensioni nell'ordinamento (N, C, H, W):

   - N: numero di campioni (*batch size*).
   - C: numero di canali di colore (1 per scala di grigi, 3 per *RGB*).
   - H: altezza dell'immagine in *pixel* (8).
   - W: larghezza dell'immagine in *pixel* (8).

   Poiché l'array originale possiede un formato 3D del tipo $(N, 8, 8)$, la chiamata `.unsqueeze(1)` inserisce una dimensione unitaria specificamente all'**indice 1** (la posizione riservata al canale di colore C), trasformando la struttura in un *tensor* 4D di formato $(N, 1, 8, 8)$, come richiesto da PyTorch.

4. **Conversione delle Etichette (`dtype=torch.long`):** Le etichette delle classi y vengono convertite in *tensor* interi a 64 *bit* (`torch.long`). Questa specifica di tipo è un requisito della funzione di loss di Entropia Incrociata (`nn.CrossEntropyLoss`), che utilizza interi non negativi come indici per associare la classe corretta ai *logits* di output della rete.

> **Attenzione alle Dimensioni:** La struttura finale è rappresentata dal **tensor `(N, 1, 8, 8)`**, dove `N` è il numero di campioni (*batch size*), `1` è il canale di colore (scala di grigi) e `8×8` è la risoluzione spaziale dell'immagine in *pixel*.

La Figura 9.13 illustra una sequenza di campioni dell'insieme di addestramento dopo la pre-elaborazione e l'adattamento dimensionale ai *tensor* di **PyTorch**. Nella fase di visualizzazione, la chiamata `img.squeeze().numpy()` combina due trasformazioni: il metodo `.squeeze()` elimina la dimensione unitaria ridondante del canale di colore, riducendo il *tensor* 3D di formato `(1, 8, 8)` a una matrice 2D di `(8, 8)`; successivamente, il metodo `.numpy()` converte la struttura di **PyTorch** in una matrice nativa di **NumPy**, formato richiesto dagli strumenti di rendering grafico come `mm.show()`.

```
# 1. Caricamento e pre-processamento dei dati
digits = load_digits()
X = digits.images.astype(np.float32) / 16.0  # Normalizzazione all'intervallo [0, 1]
y = digits.target

# Divisione stratificata in set di addestramento (70%) e test (30%)
X_treino, X_teste, y_treino, y_teste = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Adeguamento alla dimensione prevista da PyTorch: (N_campioni, Canali, Altezza, Larghezza)
X_treino_t = torch.tensor(X_treino).unsqueeze(1)   # Dimensione: (N, 1, 8, 8)
y_treino_t = torch.tensor(y_treino, dtype=torch.long)
X_teste_t = torch.tensor(X_teste).unsqueeze(1)
y_teste_t = torch.tensor(y_teste, dtype=torch.long)

# Mostrare un campione
n_amostras = 8
imgs = [img.squeeze().numpy() for img in X_treino_t[:n_amostras]]
imgs_titles = [str(label.item()) for label in y_treino_t[:n_amostras]]
mm.show(imgs, titles=imgs_titles, cols=n_amostras, figsize=(12, 2.5))
```

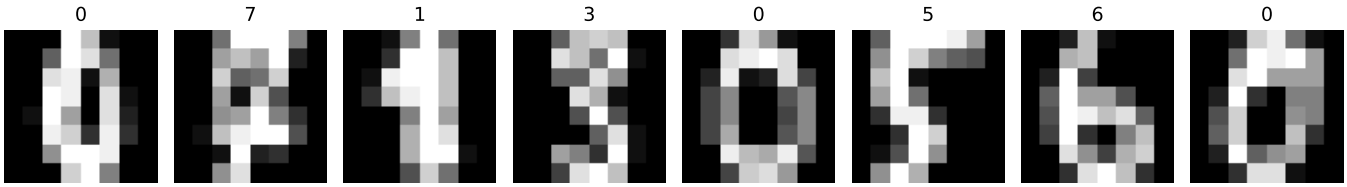


Figura 9.13: Campioni di cifre dal set de addestramento dopo la conversione in *tensori* PyTorch e normalizzazione.

Bloco 2: Definizione dell'Architettura Convoluzionale

La costruzione di modelli in **PyTorch** è strutturata secondo il paradigma dell'orientamento a oggetti, creando una classe specifica per rappresentare la rete neurale (in questo esempio, la classe `CNNDigitos`), che eredita tutte le funzionalità della classe base `nn.Module`. Il costruttore `__init__` è responsabile di istanziare i layer e dichiarare i loro parametri addestrabili, mentre il metodo `forward` stabilisce la sequenza numerica della propagazione in avanti (*forward pass*).

La Figura 9.14 sintetizza le trasformazioni spaziali dei *tensor* e il flusso di dati lungo la classe `CNNDigitos`.

Figura 9.14: Rappresentazione del flusso delle trasformazioni dimensionali dei *tensor* lungo l'architettura *CNNDigitos*.

1. **Costruttore (`__init__`) e Istanziazione dei Componenti:**
   - **Livello Convoluzionale 1 (`self.conv1`):** Applica 8 filtri 3×3 con `padding=1` sull'input in scala di grigi (1 canale), preservando la risoluzione spaziale di 8×8 *pixel*.
   - **Livello Convoluzionale 2 (`self.conv2`):** Processa le 8 mappe di caratteristiche ricevute dal livello precedente applicando 16 filtri 3×3 con `padding=1`.
   - **Sottocampionamento (`self.pool`):** Istanzia l'operazione di *Max-Pooling* con finestra 2×2 e passo (*stride*) 2, riducendo la dimensione spaziale (altezza e larghezza) della metà a ogni applicazione.
   - **Livelli Totalmente Connessi (`self.fc1` e `self.fc2`):** La prima proiezione densa riceve il *tensor* appiattito di dimensione $16 \times 2 \times 2 = 64$ e produce 32 caratteristiche intermedie. La seconda proietta queste 32 caratteristiche nei 10 *logit* finali di output.
2. **Propagazione in Avanti nel Metodo `forward`:**
   - **Primo Blocco Convoluzionale:** Il *tensor* di input di formato $(N, 1, 8, 8)$ passa attraverso `conv1` + ReLU e viene sottocampionato da `pool`, risultando nel formato $(N, 8, 4, 4)$.
   - **Secondo Blocco Convoluzionale:** Il *tensor* $(N, 8, 4, 4)$ viene processato da `conv2` + ReLU e ridotto da `pool` al formato $(N, 16, 2, 2)$.
   - **Appiattimento (*Flatten*):** Il metodo `x.view(x.size(0), -1)` riconfigura la struttura 3D in un vettore 1D di 64 elementi per campione, preservando la dimensione del batch N.
   - **Classificazione:** Il vettore di 64 elementi alimenta `fc1` con attivazione ReLU (32 neuroni) e termina in `fc2`, producendo i 10 *logit* non normalizzati per il calcolo della funzione di perdita.

```python
# 2. Definizione dell'architettura convoluzionale
class CNNDigitos(nn.Module):
    """
    Architettura convoluzionale compatta:
    2 strati convoluzionali con ReLU e Max-Pooling + 2 strati densi.
    """
    def __init__(self, n_classes=10):
        super().__init__()
```

```
        # Conv1: 1 canale di ingresso, 8 filtri 3x3 con padding 1 (uscita: 8x8)
        self.conv1 = nn.Conv2d(1, 8, kernel_size=3, padding=1)
        # Conv2: 8 canali di ingresso, 16 filtri 3x3 con padding 1 (uscita: 4x4)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3, padding=1)

        self.relu = nn.ReLU()

        # Max-Pooling 2x2 con passo (stride) 2
        self.pool = nn.MaxPool2d(2, 2)

        # Strati totalmente connessi (FC)
        self.fc1 = nn.Linear(16 * 2 * 2, 32)
        self.fc2 = nn.Linear(32, n_classes)

    def forward(self, x):
        # Primo blocco: Conv (8x8) -> ReLU -> Pool (4x4)
        x = self.pool(self.relu(self.conv1(x)))

        # Secondo blocco: Conv (4x4) -> ReLU -> Pool (2x2)
        x = self.pool(self.relu(self.conv2(x)))

        # Appiattimento (Flatten): riconfigura la matrice 3D (16, 2, 2) in vettore 1D (64)
        x = x.view(x.size(0), -1)

        # Strato denso intermedio con ReLU
        x = self.relu(self.fc1(x))

        # Strato finale di classificazione (logits)
        return self.fc2(x)
```

Analisi dei Strati e del Flusso della Classe `CNNDigiti`

1. **Costruttore (`__init__`) e Istanziazione dei Componenti:**
   - **Strato Convoluzionale 1 (`self.conv1`):** Applica 8 filtri 3×3 con `padding=1` sull'ingresso in scala di grigi (1 canale), preservando la risoluzione di 8×8 *pixel*.
   - **Strato Convoluzionale 2 (`self.conv2`):** Processa le 8 mappe di caratteristiche ricevute applicando 16 filtri 3×3 con `padding=1`.
   - **Sottocampionamento (`self.pool`):** Istanzia l'operazione di *Max-Pooling* con finestra 2×2 e passo (*stride*) 2, riducendo le dimensioni spaziali (altezza e larghezza) della metà a ogni applicazione.
   - **Strati Totalmente Connessi (`self.fc1` e `self.fc2`):** La prima proiezione densa riceve il *tensore* appiattito di dimensione $16 \times 2 \times 2 = 64$ e produce 32 caratteristiche intermedie. La seconda proietta queste 32 caratteristiche nei 10 *logit* di uscita.
2. **Propagazione in Avanti nel Metodo `forward`:**
   - **Primo Blocco:** Il *tensore* $(N, 1, 8, 8)$ passa attraverso `conv1` + ReLU e viene ridotto da `pool` a $(N, 8, 4, 4)$.
   - **Secondo Blocco:** Il *tensore* $(N, 8, 4, 4)$ passa attraverso `conv2` + ReLU e viene ridotto da `pool` a $(N, 16, 2, 2)$.
   - **Appiattimento (*Flatten*):** Il metodo `x.view(x.size(0), -1)` converte la struttura 3D in un vettore 1D di 64 elementi per campione.
   - **Classificazione:** Il vettore di 64 elementi alimenta `fc1` con attivazione ReLU (32 neuroni) e termina in `fc2`, che produce i 10 *logit* finali per il calcolo della perdita di Entropia Incrociata.

Bloco 3: Istanziazione e Parametri di Ottimizzazione

La fase di configurazione dell'apprendimento richiede l'istanziazione dell'architettura definita e la scelta di due componenti fondamentali: la **funzione di perdita**, che quantifica l'errore del modello, e l'**algoritmo di ottimizzazione**, responsabile dell'aggiustamento dei parametri verso il minimo di tale funzione.

1. **Istanziazione e Conteggio dei Parametri:** Il modello viene creato mediante l'istanziazione dell'oggetto `modello_cnn` della classe `CNNDigiti`. L'espressione `sum(p.numel() for p in modello_cnn.parameters())` esamina tutti i *tensor* dei parametri addestrabili della rete (pesi e bias di ogni livello) e calcola la cardinalità totale del modello, quantificandone la capacità di rappresentazione.

2. **Funzione di Perdita (`nn.CrossEntropyLoss`):** La perdita di Entropia Incrociata (*Cross-Entropy Loss*) rappresenta la scelta standard per i problemi di classificazione multiclasse. In PyTorch, questa implementazione combina internamente l'applicazione della funzione *LogSoftmax* con la Perdita di Log-Verosimiglianza Negativa (*NLLLoss*). Per tale ragione, il livello di output della rete produce *logit* grezzi, eliminando la necessità di applicare esplicitamente la funzione *Softmax* al termine del metodo `forward`.

3. **Ottimizzatore Adattivo (`optim.Adam`):** L'aggiornamento dei parametri utilizza l'algoritmo *Adam* (*Adaptive Moment Estimation*), con un tasso di apprendimento iniziale $\eta = 0,01$ (`lr=1e-2`). L'*Adam* combina i principi del momento con l'adattamento della dimensione del passo basato sulla media mobile delle derivate del primo e secondo ordine, regolando individualmente il tasso di apprendimento di ciascun parametro della rete.

```python
# 3. Inizializzazione del Modello e Parametri di Ottimizzazione
modelo_cnn = CNNDigitos()
num_params = sum(p.numel() for p in modelo_cnn.parameters())
print(f"Parametri addestrabili del modello: {num_params}")

criterio = nn.CrossEntropyLoss()
otimizador = optim.Adam(modelo_cnn.parameters(), lr=1e-2)
```

```
Parametri addestrabili del modello: 3658
```

Blocco 4: Ciclo di Addestramento e Valutazione

L'addestramento di una CNN avviene in modo iterativo tramite l'algoritmo di Discesa del Gradiente Stocastica per *mini-batch* (*Mini-batch SGD*).

Le curve di apprendimento risultanti da questo processo sono presentate nella Figura 9.15, generata al termine dell'esecuzione.

1. **Fase di Addestramento (`modello_cnn.train()`):** Il ciclo principale esegue l'addestramento per 50 epoche. In ogni epoca, si svolgono le seguenti fasi:
   - **Rimescolamento Stocastico:** La funzione `torch.randperm(n)` genera una permutazione casuale degli indici dei campioni, garantendo che l'ordinamento dei *mini-batch* vari a ogni epoca per evitare bias di campionamento.
   - **Divisione in *Mini-batch*:** L'insieme di addestramento viene suddiviso in lotti di 32 campioni (`tam_lote = 32`).
   - **Azzera i Gradient (`otimizzatore.zero_grad()`):** Pulisce i gradienti accumulati nel *tensor* nell'iterazione precedente, evitando la somma indesiderata di derivate tra lotti distinti.
   - **Passo in Avanti e Perdita:** Il *forward pass* calcola le previsioni `saida`, e la chiamata `criterio(saida, y_treino_t[idx])` quantifica l'errore del lotto.
   - **Retropropagazione (`perda.backward()`):** Applica la regola della catena per calcolare le derivate parziali della perdita rispetto a ciascun parametro ($\frac{\partial L}{\partial w}$).
   - **Aggiornamento dei Pesi (`otimizzatore.step()`):** Aggiorna i parametri del modello secondo le equazioni dell'ottimizzatore *Adam*.

2. **Fase di Valutazione (`modello_cnn.eval()`):** Al termine di ogni epoca, il modello viene portato in modalità di valutazione. Il contesto `with torch.no_grad()` disattiva temporaneamente il motore di calcolo automatico delle derivate (*autograd*), riducendo il consumo di memoria e accelerando l'inferenza sull'insieme di test (`X_teste_t`). L'operazione `.argmax(dim=1)` estrae la classe di maggiore probabilità per ciascun campione, consentendo di calcolare l'accuratezza sul test.

3. **Visualizzazione con la Libreria `morph`:** La funzione `mm.showTrainCurves` della libreria didattica `morph` consolida lo storico della perdita di addestramento e l'accuratezza sul test in un unico pannello grafico, permettendo di diagnosticare la convergenza del modello e monitorare la stabilità dell'apprendimento lungo le epoche.

```
# 4. Ciclo di Training (Mini-batch SGD)
n = X_treino_t.size(0)                    # Numero di campioni
tam_lote = 32                             # Dimensione del mini-batch
epocas = 50                               # Totale epoche
historico_perda, historico_acc = [], []   # Storico delle metriche

for epoca in range(epocas):                  # Ripete per epoca
    modelo_cnn.train()                       # Modalità training
    perm = torch.randperm(n)                 # Mescola i campioni
    perda_epoca = 0.0                        # Accumula perdite
    for i in range(0, n, tam_lote):          # Scorre i mini-batch
        idx = perm[i:i + tam_lote]           # Indici del lotto
        otimizador.zero_grad()               # Azzera gradienti
        saida = modelo_cnn(X_treino_t[idx])  # Propagazione diretta
        perda = criterio(saida, y_treino_t[idx]) # Calcola perdita
        perda.backward()                         # Retropropagazione
        otimizador.step()                        # Aggiorna pesi
        perda_epoca += perda.item() * len(idx)   # Somma perdita

    # Valutazione del modello sul set di test alla fine di ogni epoca
    modelo_cnn.eval()                            # Modalità valutazione
    with torch.no_grad():                        # Senza gradienti
        pred_teste = modelo_cnn(X_teste_t).argmax(dim=1)  # Predizioni
        acc_teste = (pred_teste == y_teste_t).float().mean().item()  # Accuratezza
    historico_perda.append(perda_epoca / n)      # Registra perdita
    historico_acc.append(acc_teste)              # Registra accuratezza

acc_final_cnn = historico_acc[-1]                # Ultima accuratezza
print(f"Accuratezza finale della CNN sul set di test: {acc_final_cnn:.4f}")  # Mostra
risultato

final = mm.showTrainCurves(                      # Traccia curve
    historico_perda, historico_acc,
    titulo="Evolução do Treinamento da CNN - Base de Dígitos",
    subtitulo=f"Acurácia final no teste: {acc_final_cnn:.4f}",
)
```

```
Accuratezza finale della CNN sul set di test: 0.9759
```

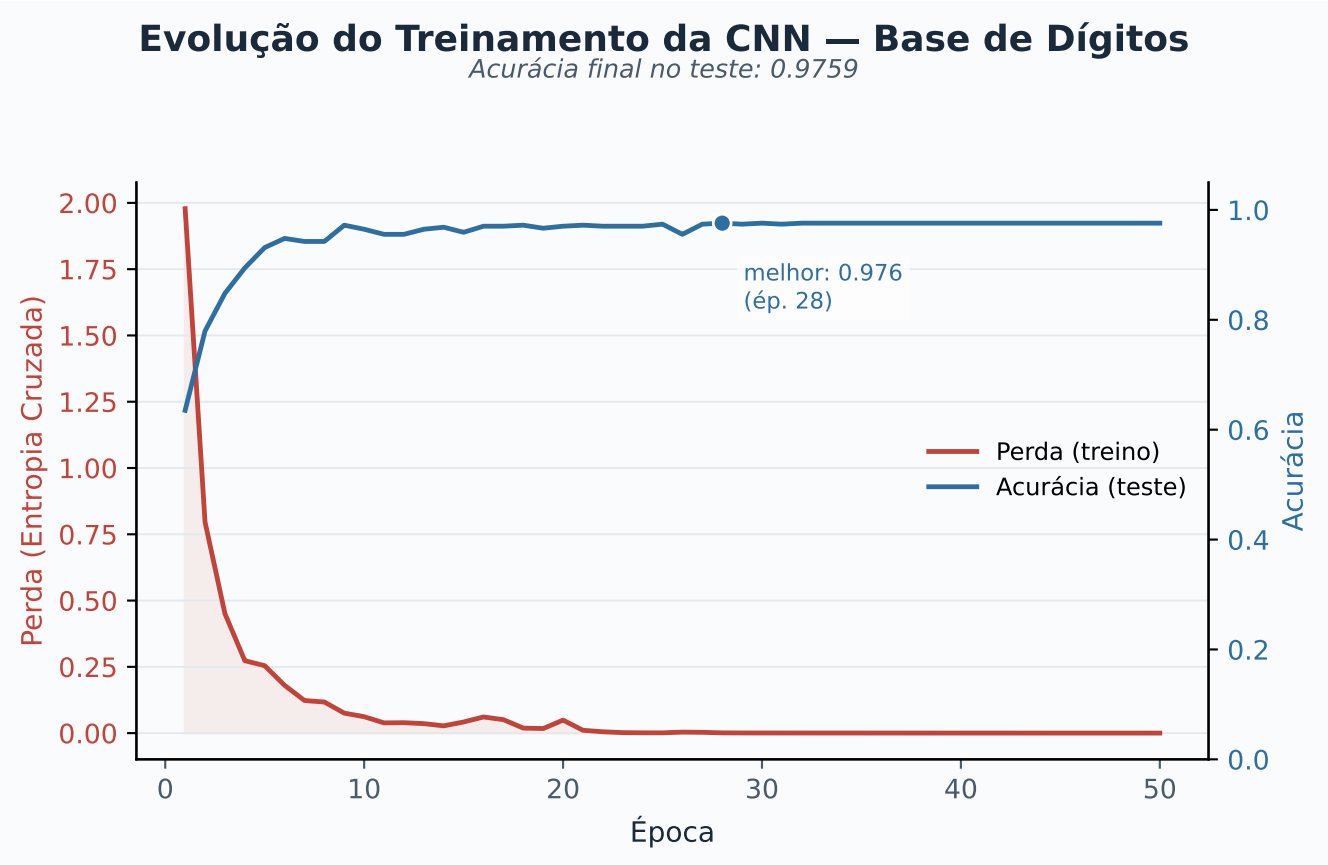


Figura 9.15: Curve di training e valutazione della CNN sul dataset di cifre: evoluzione della perdita di entropia incrociata sul set di training e dell'accuratezza sul set di test nel corso di 50 epoche.

Bloco 5: Visualizzazione del Flusso di Attivazioni

L'ispezione della rete addestrata consente di osservare la trasformazione progressiva del *tensor* di ingresso attraverso i livelli dell'architettura `CNNDigitos`. La Figura 9.16 illustra le dimensioni e le attivazioni intermedie ottenute elaborando un esempio reale della cifra 3.

1. **Selezione e Preparazione del Campione:** Il seme stocastico viene fissato con `torch.manual_seed(7)` per garantire la riproducibilità dei risultati. La prima occorrenza della cifra 3 nel dataset `load_digits` viene isolata, normalizzata nell'intervallo $[0.0, 1.0]$ e riconfigurata come un *tensor* `x` di dimensione $(1, 1, 8, 8)$.

2. **Ispezione Intermedia con `mm.showNet`:** La funzione `mm.showNet` della libreria `morph` esegue la propagazione in avanti (*forward pass*) del *tensor* `x` nell'istanza `modelo_cnn` precedentemente addestrata. Utilizzando *hook* di *forward*, la funzione intercetta lo stato numerico delle attivazioni nei livelli convoluzionali (`nn.Conv2d`), di pooling (`nn.MaxPool2d`) e completamente connessi (`nn.Linear`), restituendoli nel dizionario `acts`. Le funzioni di attivazione non lineare (`nn.ReLU`) non vengono registrate come stadi indipendenti, poiché la loro applicazione avviene direttamente sul *tensor* di uscita del livello corrispondente.

3. **Verifica dei Risultati:** L'istruzione `list(acts.keys())` visualizza la sequenza degli identificatori dei livelli monitorati, consentendo di confermare la riduzione dimensionale progressiva e la generazione del *logit* di valore massimo nell'indice corrispondente alla classe 3, come dimostrato nella Figura 9.16.

```
torch.manual_seed(7)
digits = load_digits()
idx = np.where(digits.target == 3)[0][0]
img = digits.images[idx] / 16.0
```

```
x = torch.tensor(img, dtype=torch.float32).view(1, 1, 8, 8)

# Riutilizzo dell'istanza del modello precedentemente addestrata
acts = mm.showNet(
    modelo_cnn,
    x,
    titulo="Fluxo de transformações dos tensors ao longo da arquitetura CNNDigitos",
    subtitulo=f"Exemplo real do dataset load_digits (classe verdadeira:
{digits.target[idx]})",
)
print("Livelli catturati:", list(acts.keys()))
```

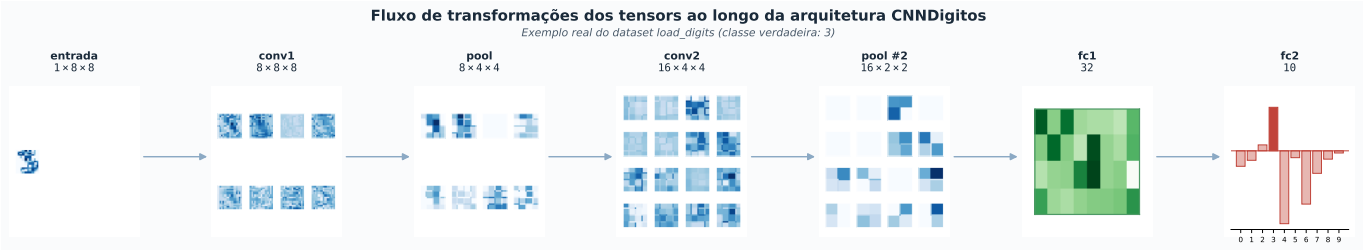


Figura 9.16: Flusso di attivazioni della CNN addestrata durante l'elaborazione di un esempio reale della cifra 3, dal *dataset load_digits*: dimensioni dei tensori strato per strato, dall'input al *logit* di output.

```
Livelli catturati: ['conv1', 'pool', 'conv2', 'pool #2', 'fc1', 'fc2']
```

Ispezionando il Grafo Computazionale con `torchviz`

Mentre `mm.showNet` privilegia la chiarezza didattica — mostrando una colonna per livello con parametri addestrabili —, la libreria `torchviz` proietta il **grafo di autograd** esattamente come PyTorch lo costruisce internamente per il calcolo dei gradienti. La Figura 9.17 illustra questa prospettiva rappresentando l'architettura `CNNDigitos`.

1. **Propagazione in Avanti Tracciata:** Con il modello addestrato in modalità `eval()`, il *forward pass* sul *tensor* `x` della cifra 3 è sufficiente affinché il motore di *autograd* registri tutte le operazioni eseguite, incluse quelle senza parametri addestrabili, come la funzione di attivazione `ReLU` e la riconfigurazione dimensionale `view`.

2. **Generazione del Grafo (`make_dot`):** La funzione `make_dot(saida, params=...)` costruisce il grafo a partire dal *tensor* di uscita, percorrendo retroattivamente la cronologia delle operazioni fino ai nodi foglia (i parametri addestrabili del modello). Ogni nodo del diagramma rappresenta un'operazione del *backward pass* (come `ReluBackward` o `AddmmBackward`), e non solo un blocco concettuale del `nn.Module`.

3. **Esportazione e Renderizzazione (`.render`):** Il metodo `.render(..., format="png", cleanup=True)` richiama l'eseguibile `dot` di **Graphviz** per compilare l'immagine in formato PNG, eliminando automaticamente i file intermedi del codice sorgente.

La Figura 9.17 evidenzia come questo grafo computazionale, anche per un'architettura compatta, presenti una densità maggiore rispetto al pannello di `mm.showNet`, poiché dettaglia ogni operazione atomica responsabile del flusso dei gradienti.

```
# 1. Forward pass con tracciamento del gradiente abilitato
modelo_cnn.eval()
saida = modelo_cnn(x)  # Riutilizzo del tensore x (cifra 3)

# 2. Grafo di base: flusso delle operazioni fino all'uscita
grafo_simples = make_dot(saida, params=dict(modelo_cnn.named_parameters()))
```

```
caminho_simples = grafo_simples.render("cnn_digitos_grafo_simples", format="png",
cleanup=True)

# 3. Visualizzazione diretta nell'ambiente Quarto/Jupyter
# Il .render() restituisce il percorso del file PNG generato; dobbiamo aprirlo come immagine
imagem_simples = np.array(Image.open(caminho_simples).convert("RGB"))
mm.show(imagem_simples, figsize=(5,10))
```

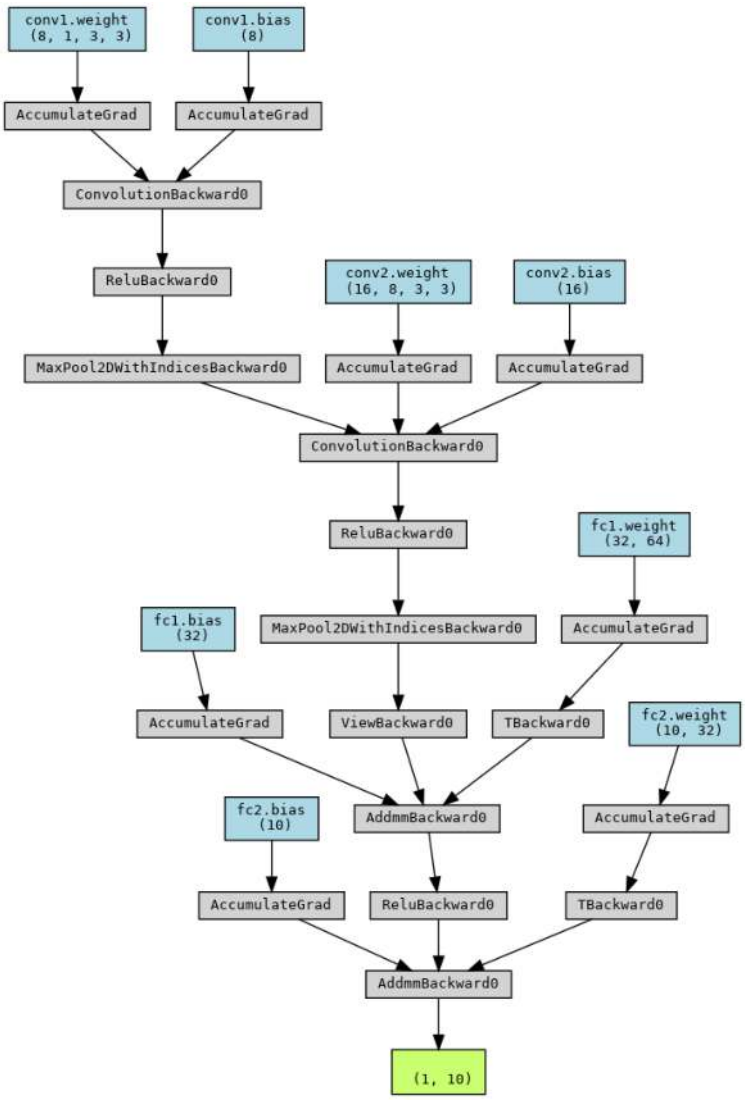


Figura 9.17: Grafo computacional della *CNNDigitos* generato tramite *torchviz*, che mostra le operazioni di *forward* e i nodi di gradiente (*backward*) associati a ciascun parametro addestrabile.

Visão Dettagliata del Grafo Computazionale con `torchviz`

Oltre alla rappresentazione semplificata, la libreria `torchviz` consente di espandere il grafo di *autograd* per ispezionare i dettagli interni dell'esecuzione della rete `CNNDigitos`. La Figura 9.18 presenta questa struttura espansa per lo stesso *tensor* di input `x`.

1. **Tracciamento con Attributi di Operazione (`show_attrs=True`):** L'inclusione degli attributi visualizza le configurazioni iperparametriche associate a ciascun nodo computazionale durante la propagazione in avanti (*forward pass*), come le dimensioni del *kernel* (`kernel_size`), i passi (*stride*) e i riempimenti (*padding*) nelle convoluzioni e nei sottocampionamenti.

2. **Rilevamento dei *Tensor* Salvati in Memoria (`show_saved=True`):** Il parametro forza la

visualizzazione esplicita dei *tensor* intermedi che PyTorch conserva in memoria durante il *forward pass*. Questi dati vengono preservati perché saranno strettamente necessari per il calcolo delle derivate parziali durante la fase di retropropagazione (*backward pass*).

3. **Generazione e Compilazione dei Grafi:** Mentre `grafo_simples` genera una visione diretta del flusso dei gradienti, `grafo_dettagliato` compila il grafo espanso nel file `cnn_digitos_grafo_detalhado.png` tramite l'eseguibile `dot` di **Graphviz**.

Come osservato nella Figura 9.18, questa visualizzazione minuziosa è utile per eseguire il debug del consumo di memoria video (*VRAM*) e per verificare come il motore di PyTorch allochi internamente ciascun nodo della regola della catena.

```
# stesso codice precedente (make_dot + render)...

# 2. Grafo dettagliato: visualizzazione delle dimensioni e dei tensori salvati per la
retropropagazione
grafo_detalhado = make_dot(
    saida,
    params=dict(modelo_cnn.named_parameters()),
    show_attrs=True,   # Mostra gli attributi delle operazioni (es.: kernel_size, stride)
    show_saved=True,   # Mostra i tensori salvati in memoria per la retropropagazione
)

caminho_detalhado = grafo_detalhado.render("cnn_digitos_grafo_detalhado",
                                           format="png", cleanup=True)

# 3. Visualizzazione diretta nell'ambiente Quarto/Jupyter
imagem_detalhado = np.array(Image.open(caminho_detalhado).convert("RGB"))
mm.show(imagem_detalhado, figsize=(8, 16))
```

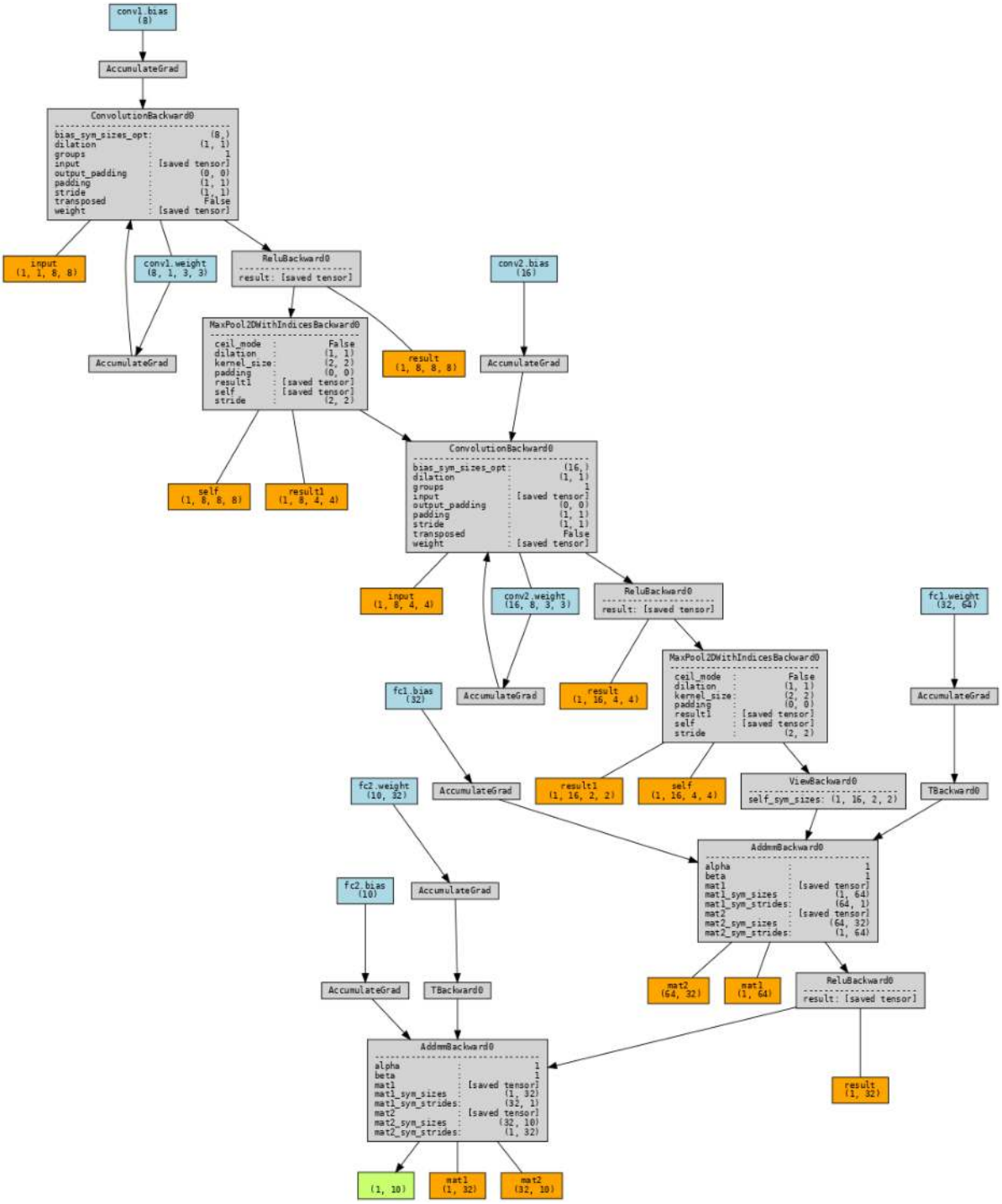


Figura 9.18: Grafo dettagliato della classe CNNDigiti generato tramite *torchviz*.

Confronto con il Capitolo 7

La Figura 9.19 riunisce i risultati ottenuti sullo stesso insieme di dati (`load_digits`), stabilendo un parallelo diretto tra gli approcci classici esplorati in precedenza e la CNN sviluppata in questo capitolo.

1. **Prestazioni dei *Pixel* Grezzi vs. Descrittori Manuali:** Negli esperimenti del Capitolo 7, il classificatore k-NN ($k = 3$) ha raggiunto un'accuratezza del $98,4\%$ quando alimentato direttamente con i *pixel* grezzi delle immagini. Al contrario, l'estrazione preventiva delle caratteristiche tramite *Histogram of Oriented Gradients* (*HOG*) ha prodotto prestazioni significativamente inferiori ($75,8\%$). Questo calo si verifica perché il *HOG* è stato concepito per catturare i gradienti dei bordi in immagini a risoluzione più elevata; in matrici di soli 8×8 *pixel*, la risoluzione spaziale è insufficiente per formare

istogrammi di orientamento informativi.

2. **Equivalenza della CNN e Apprendimento *End-to-End*:** La rete convoluzionale `CNNDigitos` raggiunge prestazioni competitive del $97,6\%$, avvicinandosi all'accuratezza del k-NN con *pixel* grezzi su una base piccola e pre-allineata. Il grande vantaggio concettuale risiede nell'apprendimento della rappresentazione: invece di dipendere da descrittori progettati manualmente (*handcrafted features*) o di mantenere l'intero insieme di dati in memoria per la ricerca dei vicini al momento dell'inferenza, la CNN ottimizza automaticamente i propri filtri convoluzionali durante l'addestramento, generando un modello compatto in grado di eseguire l'estrazione delle caratteristiche e la classificazione in modo integrato (*end-to-end*).

```
import matplotlib.pyplot as plt

# Valori ottenuti nel Capitolo 7 (k-NN, k=3), riprodotti per confronto diretto
ACC_KNN_PIXELS_CAP7 = 0.9844
ACC_KNN_HOG_CAP7 = 0.7578

metodos = ["k-NN\n(pixels brutos)", "k-NN\n(HOG)", "CNN\n(este capítulo)"]
acuracias = [ACC_KNN_PIXELS_CAP7, ACC_KNN_HOG_CAP7, acc_final_cnn]

plt.figure(figsize=(5, 4))
cores = ["#6366f1", "#f97316", "#16a34a"]
plt.bar(metodos, acuracias, color=cores)
plt.ylim(0, max(acuracias) + 0.08)
plt.ylabel("Acurácia (conjunto de teste)")
plt.title("Cap. 7 vs. Cap. 9 - Base de Dígitos")

for i, v in enumerate(acuracias):
    plt.text(i, v + 0.02, f"{v:.3f}", ha="center")

plt.tight_layout()
plt.show()
```

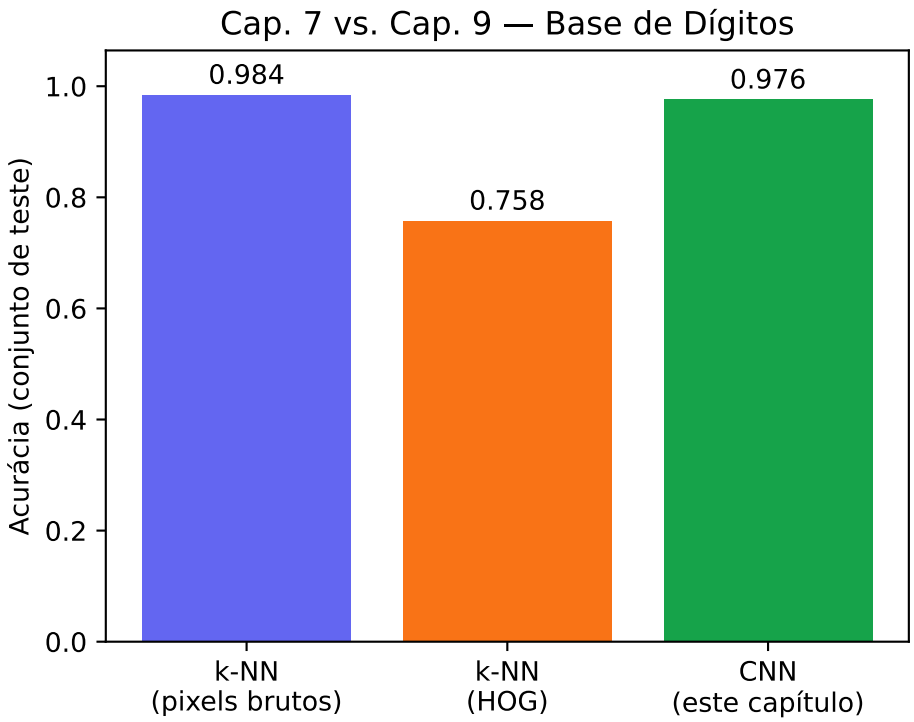


Figura 9.19: Confronto di accuratezza tra i classificatori classici del Capitolo 7 (pixel grezzi e HOG con k-NN) e la CNN addestrata in questo capitolo, sulla stessa base di cifre.

🧠 Perché funziona? — E perché la CNN non "vince" sempre

Il risultato osservato qui **ripete il pattern già visto nel Capitolo 7**: la CNN, pur imparando automaticamente le proprie caratteristiche, non supera necessariamente il k-NN con *pixel* grezzi in questa specifica base. La spiegazione è la stessa: `load_digits` è una base piccola (meno di 1.800 esempi), con immagini già centralizzate, normalizzate e a bassissima risoluzione (8×8) — condizioni in cui il confronto diretto delle intensità è già altamente informativo, e ci sono pochi dati perché la rete impari filtri realmente superiori ai descrittori semplici.
Il vero valore distintivo delle CNN emerge in scenari che descrittori artigianali e classificatori semplici non riescono ad affrontare: immagini più grandi e più realistiche, con migliaia di categorie, variazione sostanziale di posa, illuminazione e sfondo, e insiemi di addestramento massicci — esattamente il regime in cui i modelli presentati nella sezione *"Applicazioni su Larga Scala"*, più avanti, sono stati addestrati.
La lezione pedagogica che attraversa i Capitoli 7, 8 e 9 di questo libro è coerente: **la sofisticazione di un metodo deve essere proporzionale alla complessità del problema** — usare una CNN per un problema che un k-NN risolve ugualmente bene è uno spreco di risorse computazionali, non una virtù.
Questa stessa proporzionalità vale per gli strumenti di ispezione usati lungo il capitolo. Il `mm.showNet` è stato costruito per scopi **didattici** e funziona bene in reti poco profonde come la `CNNDigitos`, ma non scala a architetture profonde: ogni strato tracciato diventa una colonna nella figura, e gli strati convoluzionali con centinaia di canali generano mosaici troppo grandi per un'interpretazione visiva; inoltre, gli *hook* memorizzano tutte le attivazioni in memoria, e il *layout* assume un flusso sequenziale, non rappresentando fedelmente connessioni residue o ramificazioni (come in *ResNet* o moduli *Inception*).
Pertanto, `showNet` deve essere inteso come una lente pedagogica per reti piccole — analoga al ruolo di `mm.showBoundBox` nel debug visivo delle rilevazioni — e non come sostituto di strumenti orientati alla produzione, come *TensorBoard* o *torchviz*.

9.5.1.2 Apprendimento per Trasferimento

Addestrare una CNN da zero richiede generalmente una grande quantità di dati etichettati e notevoli risorse computazionali, poiché il processo di addestramento deve regolare tutti i parametri della rete. In molte applicazioni, tuttavia, è disponibile solo un insieme ridotto di dati per il compito di interesse. In questa situazione, l'**apprendimento per trasferimento** (*transfer learning*) riutilizza le rappresentazioni apprese da un modello precedentemente addestrato su un compito sorgente con un grande volume di dati, riducendo il costo di addestramento e la necessità di nuovi campioni.

Nella visione artificiale, questa strategia sfrutta l'organizzazione gerarchica delle CNN. I livelli iniziali apprendono caratteristiche visive di basso livello, come bordi, texture, gradienti di intensità e pattern di colore, che rimangono utili in diversi domini. I livelli più profondi combinano queste informazioni per formare rappresentazioni progressivamente più astratte e specializzate, legate alle classi presenti nel database di addestramento.

Questa sezione indaga in quali condizioni l'apprendimento per trasferimento produce buoni risultati. Il primo esperimento mostra che un estrattore piccolo, addestrato su un dominio ristretto, può condurre a un **trasferimento negativo** (*negative transfer*). Il secondo dimostra perché modelli profondi pre-addestrati su grandi basi di immagini raggiungono elevate prestazioni in nuovi compiti. Infine, il terzo applica questa strategia a un problema di diagnosi fitosanitaria, illustrando uno scenario vicino alle applicazioni reali.

Esperimento 1 — Limiti di un Estrattore Piccolo e Specializzato

Il primo esperimento mostra che il trasferimento dell'apprendimento non sempre migliora le prestazioni di un modello. A tale scopo, il set di cifre scritte a mano (`load_digits`) viene suddiviso in due domini disgiunti:

- **Dominio A (origine):** cifre da 0 a 4, utilizzate per addestrare una piccola CNN;
- **Dominio B (destinazione):** cifre da 5 a 9, rietichettate da 0 a 4, formando un nuovo compito con solo 20 campioni di addestramento.

L'obiettivo consiste nel valutare l'effetto del riutilizzo dell'estrattore di caratteristiche appreso nel Dominio A senza consentirne l'adattamento al Dominio B.

Bloco 1: Divisione dei Domini, Scarsità e Visualizzazione dei Campioni

Questo blocco prepara il dataset per l'esperimento. Diversamente dal progetto precedente, che utilizzava tutte le cifre in un unico problema di classificazione, la base è suddivisa in due compiti indipendenti: un compito di origine (Dominio A) e un compito di destinazione (Dominio B).

La Figura 9.20 presenta esempi dei due domini dopo la separazione delle classi, la conversione in *tensor* di **PyTorch** e il pre-processing.

1. **Separazione delle classi:** Le maschere booleane `mask_A` e `mask_B` separano gli esempi di ciascun dominio. Successivamente, il codice rietichetta i target del Dominio B (`y[mask_B] - 5`) nell'intervallo $[0,4]‘, consentendo a entrambi i modelli di utilizzare cinque classi di output.

2. **Scarsità dei dati:** Il generatore `np.random.default_rng(0)` seleziona solo 20 campioni per l'addestramento del Dominio B, circa quattro per classe, simulando uno scenario in cui l'addestramento da zero tende a soffrire di *overfitting*.

3. **Conversione in *tensor*:** La funzione `para_tensor` converte le immagini nel formato $(N, 1, 8, 8)$ e i target in `torch.long`, compatibili con i layer `nn.Conv2d` e la funzione di loss.

4. **Visualizzazione dei campioni:** Il codice utilizza `.squeeze().numpy()` per convertire i *tensor* in array **NumPy**. La Figura 9.20 presenta esempi dei due domini ed evidenzia la rietichettatura applicata ai target del Dominio B.

```
# 1. Divisione del dataset in due domini disgiunti
classes_A, classes_B = [0, 1, 2, 3, 4], [5, 6, 7, 8, 9]
mask_A, mask_B = np.isin(y, classes_A), np.isin(y, classes_B)

XA, yA = X[mask_A], y[mask_A]
XB, yB = X[mask_B], y[mask_B] - 5  # Reindicizzazione delle etichette nell'intervallo [0, 4]

# Divisione in addestramento e test per entrambi i domini
XA_tr, XA_te, yA_tr, yA_te = train_test_split(
    XA, yA, test_size=0.25, random_state=42, stratify=yA
)
XB_tr, XB_te, yB_tr, yB_te = train_test_split(
    XB, yB, test_size=0.25, random_state=42, stratify=yB
)

# Simulazione di estrema scarsità nel dominio di destinazione: solo 20 campioni di
addestramento
rng = np.random.default_rng(0)
idx_poucos = rng.choice(len(XB_tr), size=20, replace=False)
XB_tr_poucos, yB_tr_poucos = XB_tr[idx_poucos], yB_tr[idx_poucos]

# Funzione ausiliaria per la conversione in tensori PyTorch
def para_tensor(Ximg, yarr):
    return torch.tensor(Ximg).unsqueeze(1), torch.tensor(yarr, dtype=torch.long)

XA_tr_t, yA_tr_t = para_tensor(XA_tr, yA_tr)
XA_te_t, yA_te_t = para_tensor(XA_te, yA_te)
XB_tr_t, yB_tr_t = para_tensor(XB_tr_poucos, yB_tr_poucos)
XB_te_t, yB_te_t = para_tensor(XB_te, yB_te)

# Visualizzazione di campioni di entrambi i domini
n_amostras = 5
imgs_A = [img.squeeze().numpy() for img in XA_tr_t[:n_amostras]]
```

```
titles_A = [f"A: {label.item()}" for label in yA_tr_t[:n_amostras]]

imgs_B = [img.squeeze().numpy() for img in XB_tr_t[:n_amostras]]
titles_B = [f"B: {label.item()} (orig: {label.item()+5})" for label in yB_tr_t[:n_amostras]]

mm.show(
    imgs_A + imgs_B,
    titles=titles_A + titles_B,
    cols=n_amostras,
    figsize=(12, 4.5)
)
```

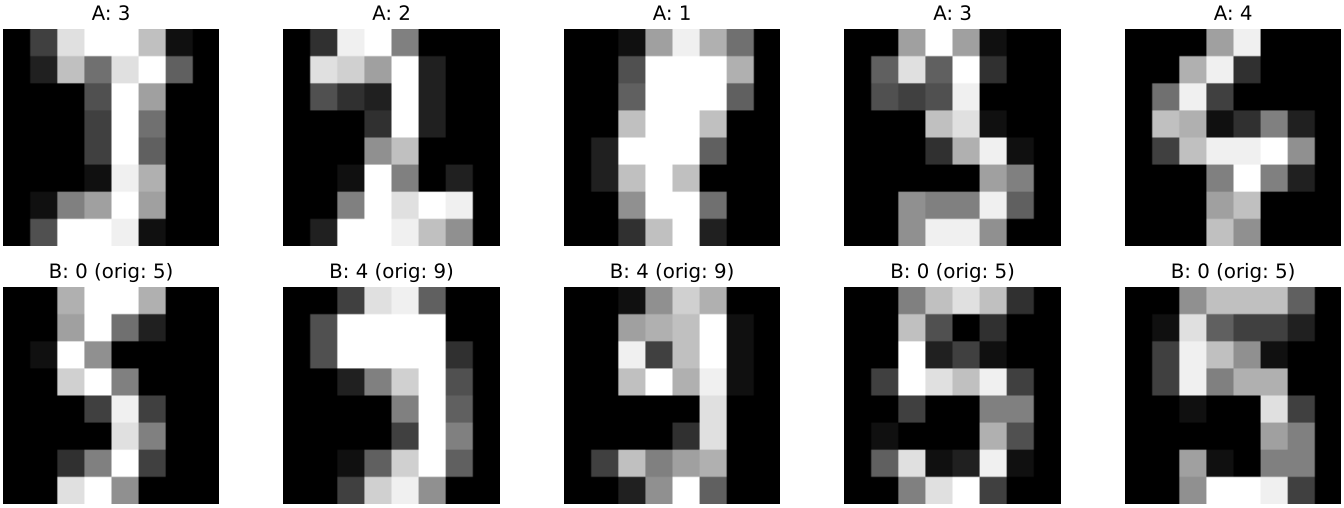


Figura 9.20: Campioni dei set di addestramento dopo il preprocessamento e l'adattamento dimensionale ai tensori PyTorch: Dominio A (cifre da 0 a 4, compito di origine) e Dominio B (cifre da 5 a 9 reindicizzate da 0 a 4, compito di destinazione).

Blocco 2: Architettura Modulare e Routine Generiche

Per consentire il trasferimento dell'apprendimento, l'architettura convoluzionale e il ciclo di addestramento sono stati rifattorizzati rispetto alla classe `CNNDigitos` del progetto precedente.

1. **Modularizzazione dell'Architettura (Differenza rispetto a `CNNDigitos`):**
   - Nel progetto precedente, la classe `CNNDigitos` dichiarava tutti i layer (`conv1`, `conv2`, `pool`, `fc1`, `fc2`) come membri diretti di un'unica classe monolitica.
   - Qui, l'architettura è separata in due componenti: la classe `ExtratorConv` incapsula il blocco spaziale convoluzionale (2 convoluzioni 3×3, 2 *Max-Pooling* 2×2 e l'appiattimento a 64 elementi), mentre la classe `CNNCompleta` istanzia questo estrattore in `self.extrator` e vi aggiunge la "testa" classificatrice (`fc1` e `fc2`).
   - Questa separazione è ciò che consente di copiare lo stato interno dell'estrattore (`state_dict()`) da un modello all'altro in modo isolato.
2. **Adeguamento del Numero di Classi di Uscita:** Mentre `CNNDigitos` nel progetto precedente possedeva 10 *logit* nel layer di uscita (`self.fc2 = nn.Linear(32, 10)`), la classe `CNNCompleta` riceve `n_classes=5` nel costruttore per adattarsi alla suddivisione dei domini A e B.
3. **Flessibilizzazione del Ciclo di Addestramento (`treinar`):**
   - Nel progetto precedente, il ciclo di addestramento iterava direttamente sugli attributi globali del modello (`modelo_cnn.parameters()`) e calcolava metriche specifiche in linea.
   - La funzione `treinar` astrae questo processo e introduce il parametro opzionale `parametros`. Se fornito, l'ottimizzatore *Adam* aggiorna **solo** i parametri di questa lista, ignorando i layer i cui

gradienti sono stati disattivati. Questa flessibilità è cruciale per eseguire l'addestramento con congelamento parziale della rete.

4. **Isolamento della Valutazione (`calcular_acuracia`):** Come nella fase di test del progetto precedente, la funzione pone il modello in `eval()` e utilizza il contesto `torch.no_grad()` per disattivare *autograd*, calcolando l'accuratezza tramite `.argmax(dim=1)`.

```
# Definizione del blocco convoluzionale riutilizzabile (stessa estrazione del progetto
precedente)
class ExtratorConv(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 8, 3, padding=1)
        self.conv2 = nn.Conv2d(8, 16, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        return x.view(x.size(0), -1)

# Architettura modulare che combina l'estrattore e la testa classificatrice
class CNNCompleta(nn.Module):
    def __init__(self, n_classes=5):
        super().__init__()
        self.extrator = ExtratorConv()
        self.fc1 = nn.Linear(64, 32)
        self.fc2 = nn.Linear(32, n_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.extrator(x)
        x = self.relu(self.fc1(x))
        return self.fc2(x)

# Routine generica di addestramento con ottimizzazione selettiva dei parametri
def treinar(modelo, X_t, y_t, epocas, lr, tam_lote=16, parametros=None):
    # Parametri addestrabili
    params = parametros if parametros is not None else modelo.parameters()
    otim = optim.Adam(params, lr=lr)                # Ottimizzatore Adam
    crit = nn.CrossEntropyLoss()                    # Funzione di perdita
    n_amostras = X_t.size(0)                        # Numero di campioni
    for _ in range(epocas):                         # Ripeti per epoca
        perm = torch.randperm(n_amostras)           # Mescola i campioni
        for i in range(0, n_amostras, tam_lote):    # Scorri mini-batch
            idx = perm[i:i + tam_lote]              # Indici del lotto
            otim.zero_grad()                        # Azzera i gradienti
            perda = crit(modelo(X_t[idx]), y_t[idx])  # Calcola la perdita
            perda.backward()                        # Retropropagazione
            otim.step()                             # Aggiorna i pesi

# Routine di valutazione
def calcular_acuracia(modelo, X_t, y_t):
    modelo.eval()                                   # Modalità di valutazione
    with torch.no_grad():                           # Senza gradienti
        pred = modelo(X_t).argmax(dim=1)            # Classi predette
    return (pred == y_t).float().mean().item()      # Restituisce l'accuratezza
```

Bloco 3: Pre-addestramento, Trasferimento e Analisi Comparativa

Questo blocco esegue il confronto tra l'addestramento del modello da zero e l'applicazione del trasferimento con il congelamento statico dell'estrattore. Per garantire piena trasparenza all'esperimento, le dimensioni dei set di addestramento e test di entrambi i domini vengono stampate sul terminale.

1. **Quantificazione dei Campioni per Dominio:**
   - **Dominio A (origine, cifre 0 a 4):** Dispone di 675 campioni di addestramento (75%) e 226 di test (25%), fornendo dati abbondanti affinché il `modello_origine` apprenda l'estrattore convoluzionale fino a raggiungere il 100% di accuratezza.
   - **Dominio B (destinazione, cifre 5 a 9):** Possiede 224 campioni di test in totale, ma il suo set di addestramento è intenzionalmente ridotto da 672 a soli 20 **campioni** (`XB_tr_poucos`), creando uno scenario severo di scarsità di dati.
2. **Fase 1: Pre-addestramento nel Dominio A (Origine):** Il `modello_origine` viene addestrato da zero sui 675 campioni delle cifre 0 a 4. Durante 40 epoche, l'estrattore convoluzionale regola i suoi filtri per identificare le caratteristiche distintive di queste prime cinque cifre, raggiungendo il 100% di accuratezza sul set di test (226 campioni).
3. **Fase 2: Trasferimento dei Pesi e Congelamento:**
   - Viene creato il `modello_trasferimento` per risolvere il compito del Dominio B (cifre 5 a 9).
   - I pesi appresi nel Dominio A vengono copiati tramite:
     - `load_state_dict(modello_origine.estrattore.state_dict())`
   - **Congelamento:** Il ciclo `for p in modello_trasferimento.estrattore.parameters(): p.requires_grad = False` disattiva il calcolo dei gradienti nei layer convoluzionali.
   - **Addestramento Selettivo:** La chiamata `treinar(...)` passa strettamente i parametri dei layer densi (`params_cabeca`), regolando la testa di classificazione con soli 20 campioni di addestramento.
4. **Fase 3: Addestramento da Zero nel Dominio B (Controllo Sperimentale):** Il `modello_da_zero` possiede la stessa architettura, ma viene addestrato da zero sugli stessi 20 campioni del Dominio B, senza alcun riutilizzo di pesi, per le stesse 40 epoche.
5. **Analisi dei Risultati (Figura 9.21):**
   - **Con Trasferimento Congelato ($72,77\%$):** Riutilizzando l'estrattore addestrato nel Dominio A e congelando i suoi parametri, la rete raggiunge il $72,77\%$ di accuratezza sul test (224 campioni) regolando solo i layer densi.
   - **Addestrato da Zero ($76,79\%$):** L'addestramento da zero supera il trasferimento congelato sul set di test del Dominio B.
   - **Causa della Differenza:** Trattandosi di un modello minuscolo (solo 16 filtri convoluzionali in matrici di 8×8), l'estrattore addestrato nel Dominio A è diventato **iper-specializzato** nelle forme geometriche delle cifre 0 a 4. Congelando rigidamente questi pochi filtri, il modello di destinazione è rimasto limitato a rilevatori inadeguati per le cifre 5 a 9. La rete addestrata da zero, anche con soli 20 campioni, è riuscita ad adattare i suoi 16 filtri direttamente ai tratti del Dominio B.

```
# Fixar semente para reprodutibilidade
torch.manual_seed(42)

# Exibição do tamanho dos grupos de treino e teste
print("=== Detalhamento do Tamanho das Bases ===")
print(f"Domínio A (0-4) - Treino: {len(XA_tr_t)} amostras | Teste: {len(XA_te_t)} amostras")
print(f"Domínio B (5-9) - Treino completo: {len(XB_tr)} | Treino reduzido:
{len(XB_tr_poucos)}",
      f"| Teste: {len(XB_te_t)} amostras\n")
```

```
# 1. Pré-treinamento na tarefa de origem (Domínio A: dígitos 0-4)
modelo_origem = CNNCompleta(n_classes=5)                        # Cria CNN

#######
treinar(modelo_origem, XA_tr_t, yA_tr_t, epocas=40, lr=1e-2)   # Treina modelo

acc_A = calcular_acuracia(modelo_origem, XA_te_t, yA_te_t)     # Mede acurácia

print(f"Acurácia no domínio de origem A "                       # Exibe resultado
      f"(dígitos 0-4, {len(XA_te_t)} testes): " f"{acc_A:.4f}")

# 2. Transferência de Aprendizado (Extrator Congelado)
modelo_transferencia = CNNCompleta(n_classes=5)        # Cria CNN
modelo_transferencia.extrator.load_state_dict(         # Copia extrator
    modelo_origem.extrator.state_dict())

for p in modelo_transferencia.extrator.parameters():   # Percorre extrator
    p.requires_grad = False                            # Congela pesos

params_cabeca = list(modelo_transferencia.fc1.parameters())  # FC1
params_cabeca += list(modelo_transferencia.fc2.parameters()) # +FC2

#######
treinar(modelo_transferencia, XB_tr_t, yB_tr_t,              # Treina cabeça
         epocas=40, lr=1e-2, parametros=params_cabeca)

acc_transferencia = calcular_acuracia(modelo_transferencia, XB_te_t, yB_te_t) # Mede acurácia

# 3. Treinamento do Zero no Domínio B
modelo_do_zero = CNNCompleta(n_classes=5)                     # Cria CNN

#######
treinar(modelo_do_zero, XB_tr_t, yB_tr_t, epocas=40, lr=1e-2) # Treina modelo

acc_do_zero = calcular_acuracia(modelo_do_zero,  XB_te_t, yB_te_t) # Mede acurácia


print(f"Domínio de destino B (dígitos 5-9), apenas {len(XB_tr_poucos)} ",
      f"exemplos de treino ({len(XB_te_t)} testes):")
print(f"  Com transferência (extrator congelado): {acc_transferencia:.4f}")
print(f"  Treinando do zero (mesmos dados/épocas): {acc_do_zero:.4f}")

# Visualização comparativa
plt.figure(figsize=(4.5, 4))
plt.bar(["Do zero", "Transferência"], [acc_do_zero, acc_transferencia],
        color=["#dc2626", "#16a34a"])
plt.ylim(0, max([acc_do_zero, acc_transferencia]) + 0.1)
plt.ylabel("Acurácia no domínio B (teste)")
plt.title(f"Efeito da Transferência ({len(XB_tr_poucos)} exemplos de treino)")

for i, v in enumerate([acc_do_zero, acc_transferencia]):
    plt.text(i, v + 0.02, f"{v:.3f}", ha="center")

plt.tight_layout()
plt.show()
```

```
=== Detalhamento do Tamanho das Bases ===
Domínio A (0-4) - Treino: 675 amostras | Teste: 226 amostras
Domínio B (5-9) - Treino completo: 672 | Treino reduzido: 20 | Teste: 224 amostras

Acurácia no domínio de origem A (dígitos 0-4, 226 testes): 1.0000
```

```
Domínio de destino B (dígitos 5-9), apenas 20  exemplos de treino (224 testes):
  Com transferência (extrator congelado): 0.7277
  Treinando do zero (mesmos dados/épocas): 0.7679
```

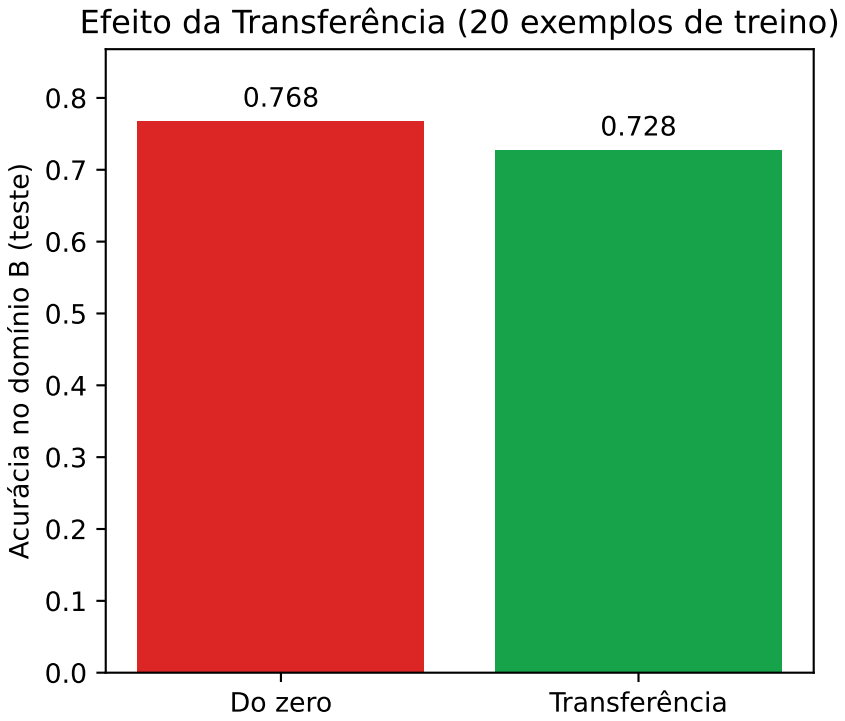


Figura 9.21: Comparação de acurácia no conjunto de teste do Domínio B (dígitos 5 a 9) sob restrição de dados (20 exemplos de treino): demonstração do impacto do congelamento rígido e da transferência negativa em redes de baixa capacidade.

Bloco 4: Visualizzazione del Flusso di Attivazioni con `mm.showNet`

Per confermare che l'estrazione delle caratteristiche riutilizzata preserva le trasformazioni dimensionali studiate nel progetto precedente, si utilizza nuovamente la funzione `mm.showNet` della libreria `morph`. La Figura 9.22 mostra il flusso di attivazioni del `modello_transferimento` durante l'elaborazione di un campione del Dominio B (cifra 7, reindicizzata per la classe 2).

1. **Preservazione del Flusso Convoluzionale:** Poiché l'architettura `EstrazioneConv` replica gli stessi strati di convoluzione e *pooling* della `CNNCifre` del progetto precedente, le dimensioni dei *tensor* intermedi rimangono in $(1, 8, 4, 4)$ nel primo blocco.
2. **Ispezione della Testa Adattata:** La differenza rispetto al progetto precedente emerge nello strato di uscita (`fc2`): mentre il modello del progetto precedente proiettava il vettore intermedio in 10 *logit* (classi da 0 a 9), il modello di trasferimento proietta il vettore in 5 *logit* (classi da 0 a 4), catturando le probabilità relative del Dominio B.

```
# Seleziona il primo campione di test del Dominio B
x_amostra_B = XB_te_t[0:1]  # Tensore di dimensione (1, 1, 8, 8)
classe_verdadeira = yB_te_t[0].item()
classe_original = classe_verdadeira + 5

# Ispezione del flusso di attivazioni nel modello di transfer learning
acts_transfer = mm.showNet(
    modelo_transferencia,
    x_amostra_B,
    titulo="Fluxo de ativações no modelo de transferência (Domínio B)",
```

```
    subtitulo=f"Amostra do dígito {classe_original} (rótulo reindexado: {classe_verdadeira})",
)

print("Livelli catturati nel modello di transfer learning:\n", list(acts_transfer.keys()))
```

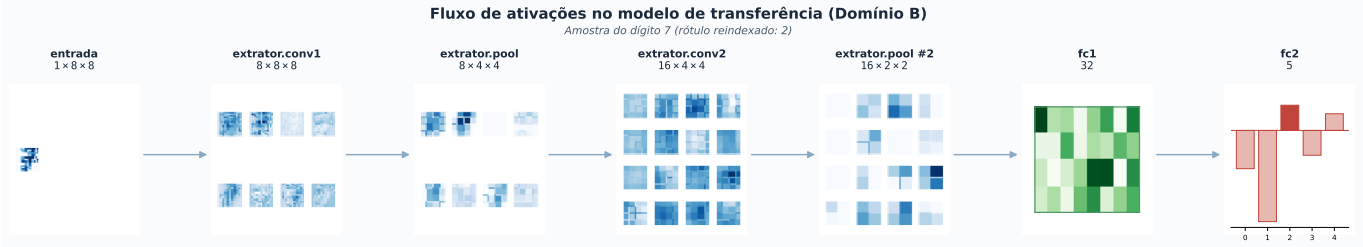


Figura 9.22: Flusso di attivazioni e trasformazioni dimensionali dei tensori nel modello di transfer learning durante l'elaborazione di un campione di test del Dominio B (cifra 7, reindicizzato come classe 2).

```
Livelli catturati nel modello di transfer learning:
 ['extrator.conv1', 'extrator.pool', 'extrator.conv2', 'extrator.pool #2', 'fc1', 'fc2']
```

Analisi dell'Esperimento 1

Il modello addestrato da zero raggiunge un'accuratezza superiore rispetto al modello con transfer learning ed estrattore congelato. Questo risultato caratterizza un caso di **trasferimento negativo** (*negative transfer*) e deriva da tre fattori:

1. **Bassa capacità:** L'estrattore possiede solo 16 filtri 3×3, insufficienti per apprendere rappresentazioni generalizzabili.
2. **Specializzazione nel dominio:** L'addestramento con le cifre da 0 a 4 produce filtri poco discriminativi per le cifre da 5 a 9.
3. **Assenza di adattamento:** Il congelamento impedisce all'estrattore di adattare i propri filtri al nuovo compito.

> Provocazione Pedagogica
>
> Questo esperimento utilizza un estrattore di piccole dimensioni addestrato su un dominio ristretto. Il risultato sarebbe diverso se l'estrattore avesse appreso le proprie rappresentazioni su una base con milioni di immagini e grande diversità di oggetti?

Esperimento 2 — Quando il Transfer Learning Funziona Davvero (*ResNet-18* Pre-addestrata)

Il secondo esperimento replica la stessa struttura del primo — pochi esempi di addestramento, due classi, confronto tra strategie —, ma sostituisce l'estrattore artigianale di 16 filtri con la ***ResNet-18***, un'architettura di 18 strati pre-addestrata su *ImageNet* ($1,4$ milioni di immagini, 1.000 categorie), e il *dataset* sintetico di cifre con fotografie reali del **Oxford-IIIT Pet Dataset** (PARKHI, 2012).

Il compito: distinguere due razze canine — **Carlino** e **Boxer** — a partire da sole 15 fotografie di addestramento per classe.

> Perché questo scenario?
>
> La sfida qui non è la somiglianza visiva tra le razze — Carlino e Boxer hanno corporature e proporzioni ben distinte —, bensì la scarsità di dati: solo 30 fotografie reali in totale, senza alcuna immagine sintetica. È il tipo di problema con budget di dati ridotto che, nella pratica, motiva l'uso di reti pre-

addestrate: non c'è tempo né risorse per fotografare ed etichettare migliaia di cani prima di addestrare un classificatore da zero.

Blocco 1: Caricamento del *Dataset* Reale e Campionamento Sparso

1. **Fonte:** il *Oxford-IIIT Pet Dataset* (PARKHI, 2012) viene caricato tramite `torchvision.datasets.OxfordIIITP` che scarica automaticamente le 7.349 fotografie e le relative etichette di razza alla prima esecuzione.
2. **Filtraggio:** vengono mantenute solo le due razze di interesse (`Pug`, `Boxer`).
3. **Sparsità deliberata:** vengono selezionate solo 15 fotografie di addestramento per classe (30 in totale) — il resto costituisce il set di test, utilizzato esclusivamente per la valutazione.

La Figura 9.23 mostra campioni di addestramento di ciascuna razza.

```
import random

RACAS_ALVO = ["Pug", "Boxer"]
N_TREINO_POR_CLASSE = 15
N_TESTE_POR_CLASSE = 20

# 1. Download del dataset completo (37 razze) - licenza CC BY-SA 4.0
pets_completo = OxfordIIITPet(
    root="dados_pets", split="trainval", target_types="category", download=True
)
nomes_racas = pets_completo.classes
indices_alvo = [nomes_racas.index(r) for r in RACAS_ALVO]

# 2. Filtraggio delle due razze di interesse, separate per classe
por_classe = {idx: [] for idx in indices_alvo}
for img, lbl in pets_completo:
    if lbl in indices_alvo:
        por_classe[lbl].append(img)

# 3. Campionamento: poche immagini di addestramento, più immagini di test
rng = random.Random(42)
imgs_treino, y_treino, imgs_teste, y_teste = [], [], [], []
for classe_idx, idx_original in enumerate(indices_alvo):
    imgs_raca = por_classe[idx_original][:]
    rng.shuffle(imgs_raca)
    imgs_treino += imgs_raca[:N_TREINO_POR_CLASSE]
    y_treino += [classe_idx] * N_TREINO_POR_CLASSE
    imgs_teste += imgs_raca[N_TREINO_POR_CLASSE : N_TREINO_POR_CLASSE + N_TESTE_POR_CLASSE]
    y_teste += [classe_idx] * N_TESTE_POR_CLASSE

print(f"Addestramento: {len(imgs_treino)} immagini | Test: {len(imgs_teste)} immagini")

amostras_pil = imgs_treino[:4] + imgs_treino[N_TREINO_POR_CLASSE:N_TREINO_POR_CLASSE + 4]
amostras_exibicao = [np.array(img.convert("RGB")) for img in amostras_pil]  # PIL -> ndarray
titulos_exibicao = [RACAS_ALVO[0]] * 4 + [RACAS_ALVO[1]] * 4
mm.show(amostras_exibicao, titles=titulos_exibicao, cols=4, figsize=(11, 6))
```

Figura 9.23

Bloco 2: Tre Strategie sulla Stessa Architettura

Per isolare l'effetto del trasferimento di apprendimento, le tre strategie riutilizzano **esattamente la stessa architettura** (*ResNet-18*), variando solo l'origine dei pesi e quali parametri rimangono addestrabili:

1. **`do_zero`:** pesi casuali (`weights=None`) — equivalente ad addestrare l'architettura della *ResNet-18*

interamente da zero, come nel Blocco 2 dell'Esperimento 1.

2. **`congelado`:** pesi pre-addestrati su *ImageNet*, con `requires_grad = False` in tutti i livelli convoluzionali — solo il nuovo livello finale viene addestrato.
3. **`fine_tuning`:** pesi pre-addestrati su *ImageNet* come punto di partenza, ma **senza** congelamento — l'intera rete si adatta al nuovo dominio, con un tasso di apprendimento basso per non distruggere la conoscenza precedente.

In tutti i casi, il livello finale `fc` viene sostituito da `nn.Linear(fc.in_features, 2)`, corrispondente alle due razze di destinazione.

```
transformacao_resnet = T.Compose([
    T.Resize((224, 224)),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

def prepara_tensores(imgs, labels):
    X = torch.stack([transformacao_resnet(img.convert("RGB")) for img in imgs])
    y = torch.tensor(labels, dtype=torch.long)
    return X, y

X_tr, y_tr = prepara_tensores(imgs_treino, y_treino)
X_te, y_te = prepara_tensores(imgs_teste, y_teste)

def cria_modelo_pets(estrategia):
    pesos = None if estrategia == "do_zero" else models.ResNet18_Weights.DEFAULT
    modelo = models.resnet18(weights=pesos)
    if estrategia == "congelado":
        for p in modelo.parameters():
            p.requires_grad = False
    modelo.fc = nn.Linear(modelo.fc.in_features, len(RACAS_ALVO))
    return modelo

modelo_do_zero = cria_modelo_pets("do_zero")
modelo_congelado = cria_modelo_pets("congelado")
modelo_fine_tuning = cria_modelo_pets("fine_tuning")
```

Blocco 3: Addestramento Comparativo e Analisi dell'Accuratezza

Riutilizzando le funzioni generiche `treinar` e `calcular_acuracia`, definite nel Blocco 2 dell'Esperimento 1, i tre modelli vengono addestrati sullo stesso insieme di 30 fotografie e valutati sul set di test (immagini mai viste durante l'addestramento):

- Il modello `do_zero` tende a **sovra-adattarsi** rapidamente alle 30 fotografie di addestramento, senza generalizzare al set di test — 30 esempi sono drasticamente insufficienti per regolare gli 11 milioni di parametri della *ResNet-18* da zero.
- Il modello `congelado` dovrebbe già raggiungere un'accuratezza considerevolmente superiore, poiché riutilizza, senza alcuna regolazione, caratteristiche visive generiche (bordi, trame, contorni) apprese su *ImageNet* — solo il nuovo strato lineare deve essere adattato alle 30 fotografie.
- Il modello `fine_tuning` tende a eguagliare o superare l'estrattore congelato, poiché parte dalla stessa conoscenza preliminare, ma consente inoltre una messa a punto fine dell'intera rete sulle peculiarità visive delle razze.

La Figura 9.24 riassume i tre risultati.

Bloco 4: Ispezione Qualitativa delle Previsioni

Come nell'Esperimento 1 e nella sezione successiva sulla diagnostica fogliare, è istruttivo osservare singolarmente alcune previsioni del modello migliore (tipicamente `fine_tuning` o `congelato`) su fotografie

```python
torch.manual_seed(42)

configuracoes = [
    ("Do zero",              modelo_do_zero,      None, 2e-3),
    ("Extrator congelado",   modelo_congelado,    "fc", 1e-3),
    ("Fine-tuning completo", modelo_fine_tuning,  None, 1e-4),
]

resultados_pets = {}
for nome, modelo, alvo_params, taxa in configuracoes:

    parametros = modelo.fc.parameters() if alvo_params == "fc" else None
    treinar(modelo, X_tr, y_tr, epocas=15, lr=taxa, tam_lote=8, parametros=parametros)
    resultados_pets[nome] = calcular_acuracia(modelo, X_te, y_te)
    print(f"{nome}: {resultados_pets[nome]*100:.1f}%")


plt.figure(figsize=(5.5, 4))
cores = ["#dc2626", "#f59e0b", "#16a34a"]
plt.bar(resultados_pets.keys(), resultados_pets.values(), color=cores)
plt.ylim(0, 1.05)
plt.axhline(0.5, color="gray", linestyle="--", linewidth=1, label="Chute aleatório (50%)")

plt.ylabel("Acurácia no teste")
plt.title("Pug vs. Boxer - 15 fotos de treino/classe")
for i, v in enumerate(resultados_pets.values()):
    plt.text(i, v + 0.02, f"{v*100:.1f}%", ha="center")
plt.xticks(rotation=10)

plt.legend()
plt.tight_layout()
plt.show()
```

Figura 9.24

reali di test, confrontando l'etichetta prevista con la razza effettiva.

```
melhor_modelo.eval()  # oppure modello_congelato, in base al risultato del
blocco 3

idx_amostras = list(range(4)) + list(range(N_TESTE_POR_CLASSE, N_TESTE_POR_CLASSE + 4))

imgs_pred, titulos_pred = [], []
with torch.no_grad():
    for idx in idx_amostras:
        entrada = X_te[idx].unsqueeze(0)
        pred_idx = melhor_modelo(entrada).argmax(dim=1).item()
        real_idx = y_te[idx].item()
        marcador = " " if pred_idx == real_idx else " "
        imgs_pred.append(np.array(imgs_teste[idx].convert("RGB")))  # PIL -> ndarray
        titulos_pred.append(f"{marcador} previsto: {RACAS_ALVO[pred_idx]}\n"
                            f"real: {RACAS_ALVO[real_idx]}")
```

Figura 9.25

```
mm.show(imgs_pred, titles=titulos_pred, cols=4, figsize=(12, 7))
```

```
# Pulizia esplicita dei dati scaricati e liberazione della memoria
if FLAG_LIMPAR_DADOS:
    if os.path.exists('./dados_pets'):
        shutil.rmtree('./dados_pets')
        print('  Directory dati temporanea ./dati_pets rimossa con successo.')

    if torch.cuda.is_available():
        torch.cuda.empty_cache()
```

Esperimento 3 — Diagnostico Fitopatologico con Transferimento di Apprendimento

L'esperimento precedente ha mostrato che una *ResNet-18* preaddestrata su **ImageNet** può adattarsi a un nuovo compito utilizzando poche campioni. Ora, la stessa strategia viene applicata a un problema di diagnostica fitopatologica. La *ResNet-18* deve classificare immagini di foglie in tre categorie: `["foglia_sana", "foglia_malata", "sintomo_sconosciuto"]`.

- **`foglia_sana`:** foglia senza lesioni visibili.
- **`foglia_malata`:** foglia con macchie scure che simulano una malattia fungina.
- **`sintomo_sconosciuto`:** foglia con clorosi giallastra, che rappresenta un pattern diverso dalla malattia nota.

> 🌱 Perché questo scenario?
>
> La diagnostica fitopatologica costituisce un'importante applicazione della visione artificiale nell'agricoltura di precisione. Un modello preaddestrato su **ImageNet** può riutilizzare caratteristiche come bordi, texture e pattern di colore per apprendere questo nuovo compito con poche immagini.

Blocco 1: Generazione del *Dataset* Sintetico

Questo blocco genera un insieme sintetico con **30** immagini per classe per l'addestramento e **8** per la validazione, per un totale rispettivamente di **90** e **24** immagini.

1. **Generazione della foglia:** La funzione `desenha_folha_base` crea il contorno della foglia, variando dimensioni, orientamento e tonalità di verde.
2. **Simulazione della malattia:** La funzione `aplica_manchas_doenca` aggiunge macchie scure irregolari che simulano lesioni fungine.
3. **Simulazione di un altro sintomo:** La funzione `aplica_sintoma_desconhecido` aggiunge regioni giallastre che rappresentano un pattern distinto dalla malattia nota.
4. **Visualizzazione dei campioni:** La Figura 9.26 presenta esempi delle tre categorie dell'insieme sintetico.

```
CLASSES_FOLHA = ["folha_saudavel", "folha_doente", "sintoma_desconhecido"]


def desenha_folha_base(tam_img, rng):
    '''Disegna il contorno ovale di una foglia verde con nervatura centrale,
    con piccole variazioni di tonalità, dimensione e orientamento tra i campioni.'''
    img = np.full((tam_img, tam_img, 3), 245, dtype=np.uint8)  # sfondo chiaro
    cx, cy = tam_img // 2, tam_img // 2
    eixo_a = rng.randint(int(tam_img * 0.30), int(tam_img * 0.38))
    eixo_b = rng.randint(int(tam_img * 0.20), int(tam_img * 0.26))
    angulo = rng.uniform(-15, 15)
    verde = (rng.randint(40, 70), rng.randint(120, 160), rng.randint(40, 70))
    cv2.ellipse(img, (cx, cy), (eixo_a, eixo_b), angulo, 0, 360, verde, -1, cv2.LINE_AA)
    ang_rad = np.deg2rad(angulo)
    dx, dy = np.cos(ang_rad), np.sin(ang_rad)
    p1 = (int(cx - eixo_a * dx), int(cy - eixo_a * dy))
    p2 = (int(cx + eixo_a * dx), int(cy + eixo_a * dy))
    cv2.line(img, p1, p2, (25, 90, 25), 2, cv2.LINE_AA)  # nervatura centrale
    return img, (cx, cy, eixo_a, eixo_b, angulo)


def aplica_manchas_doenca(img, centro_folha, rng, n_manchas=(4, 8)):
    '''Simula lesioni fogliari: macchie scure dai bordi irregolari
    (pattern tipico di malattie fungine).'''
    cx, cy, eixo_a, eixo_b, _ = centro_folha
    for _ in range(rng.randint(*n_manchas)):
        raio = rng.randint(1, 3)
        px = cx + rng.randint(-int(eixo_a * 0.7), int(eixo_a * 0.7))
        py = cy + rng.randint(-int(eixo_b * 0.7), int(eixo_b * 0.7))
        cor_mancha = (rng.randint(50, 90), rng.randint(25, 45), rng.randint(10, 25))
        cv2.circle(img, (px, py), raio, cor_mancha, -1, cv2.LINE_AA)
        cv2.circle(img, (px, py), raio + 1, (120, 85, 30), 1, cv2.LINE_AA)  # alone
    return img


def aplica_sintoma_desconhecido(img, centro_folha, rng):
    '''Simula un pattern distinto (marmorizzato giallastro/clorosi), diverso
    dalle macchie scure della malattia nota.'''
    cx, cy, eixo_a, eixo_b, _ = centro_folha
    for _ in range(rng.randint(3, 5)):
        eixo_m = (rng.randint(1, 3), rng.randint(2, 3))
        px = cx + rng.randint(-int(eixo_a * 0.6), int(eixo_a * 0.6))
        py = cy + rng.randint(-int(eixo_b * 0.6), int(eixo_b * 0.6))
        cor_clorose = (rng.randint(200, 235), rng.randint(195, 225), rng.randint(50, 90))
        ang_m = rng.uniform(0, 180)
        cv2.ellipse(img, (px, py), eixo_m, ang_m, 0, 360, cor_clorose, -1, cv2.LINE_AA)
    return img
```

```python
def aplica_ruido_sal_pimenta(img, prop_ruido=0.02, rng=None):
    '''Applica rumore sale (punti bianchi) e pepe (punti neri) casuali.
    prop_rumore: frazione di pixel modificati (es: 0.02 = 2% dei pixel).'''
    if prop_ruido <= 0:
        return img

    img_ruido = img.copy()
    num_pixels = int(prop_ruido * img.shape[0] * img.shape[1])
    n_sal = num_pixels // 2
    n_pimenta = num_pixels - n_sal

    # Applica Sale (Bianco - [255, 255, 255])
    for _ in range(n_sal):
        y = rng.randint(0, img.shape[0] - 1)
        x = rng.randint(0, img.shape[1] - 1)
        img_ruido[y, x] = [255, 255, 255]

    # Applica Pepe (Nero - [0, 0, 0])
    for _ in range(n_pimenta):
        y = rng.randint(0, img.shape[0] - 1)
        x = rng.randint(0, img.shape[1] - 1)
        img_ruido[y, x] = [0, 0, 0]

    return img_ruido


def gera_folha(classe_idx, tam_img=128, rng=None, prop_ruido=0.02):
    rng = rng or random.Random()
    img, geometria = desenha_folha_base(tam_img, rng)
    nome = CLASSES_FOLHA[classe_idx]

    if nome == "folha_doente":
        img = aplica_manchas_doenca(img, geometria, rng)
    elif nome == "sintoma_desconhecido":
        img = aplica_sintoma_desconhecido(img, geometria, rng)

    ruido_exp = rng.randint(-3, 3)  # leggera variazione di esposizione
    img = np.clip(img.astype(np.int16) + ruido_exp, 0, 255).astype(np.uint8)

    # Applicazione del rumore Sale e Pepe
    img = aplica_ruido_sal_pimenta(img, prop_ruido=prop_ruido, rng=rng)

    return img


def gera_conjunto(n_por_classe, tam_img=128, seed=0, prop_ruido=0.02):
    rng = random.Random(seed)
    imgs, labels = [], []
    for classe_idx in range(len(CLASSES_FOLHA)):
        for _ in range(n_por_classe):
          imgs.append(gera_folha(classe_idx, tam_img=tam_img, rng=rng, prop_ruido=prop_ruido))
          labels.append(classe_idx)
    return imgs, labels


N_POR_CLASSE_TREINO, N_POR_CLASSE_VAL = 30, 8

imgs_treino, labels_treino = gera_conjunto(n_por_classe=N_POR_CLASSE_TREINO, seed=42,
                                           prop_ruido=0.02)
imgs_val, labels_val = gera_conjunto(n_por_classe=N_POR_CLASSE_VAL, seed=123, prop_ruido=0.02)
```

```
print(f"Addestramento: {len(imgs_treino)} immagini ({N_POR_CLASSE_TREINO} per classe) | "
      f"Validazione: {len(imgs_val)} immagini ({N_POR_CLASSE_VAL} per classe)")

# Visualizzazione di 2 campioni per ogni classe (6 immagini in totale)
amostras_exibir, titulos_exibir = [], []
for classe_idx, nome in enumerate(CLASSES_FOLHA):
    for k in range(2):
        idx = classe_idx * N_POR_CLASSE_TREINO + k
        amostras_exibir.append(imgs_treino[idx])
        titulos_exibir.append(nome)

mm.show(amostras_exibir, titles=titulos_exibir, cols=3, figsize=(10, 7))
```

```
Addestramento: 90 immagini (30 per classe) | Validazione: 24 immagini (8 per classe)
```

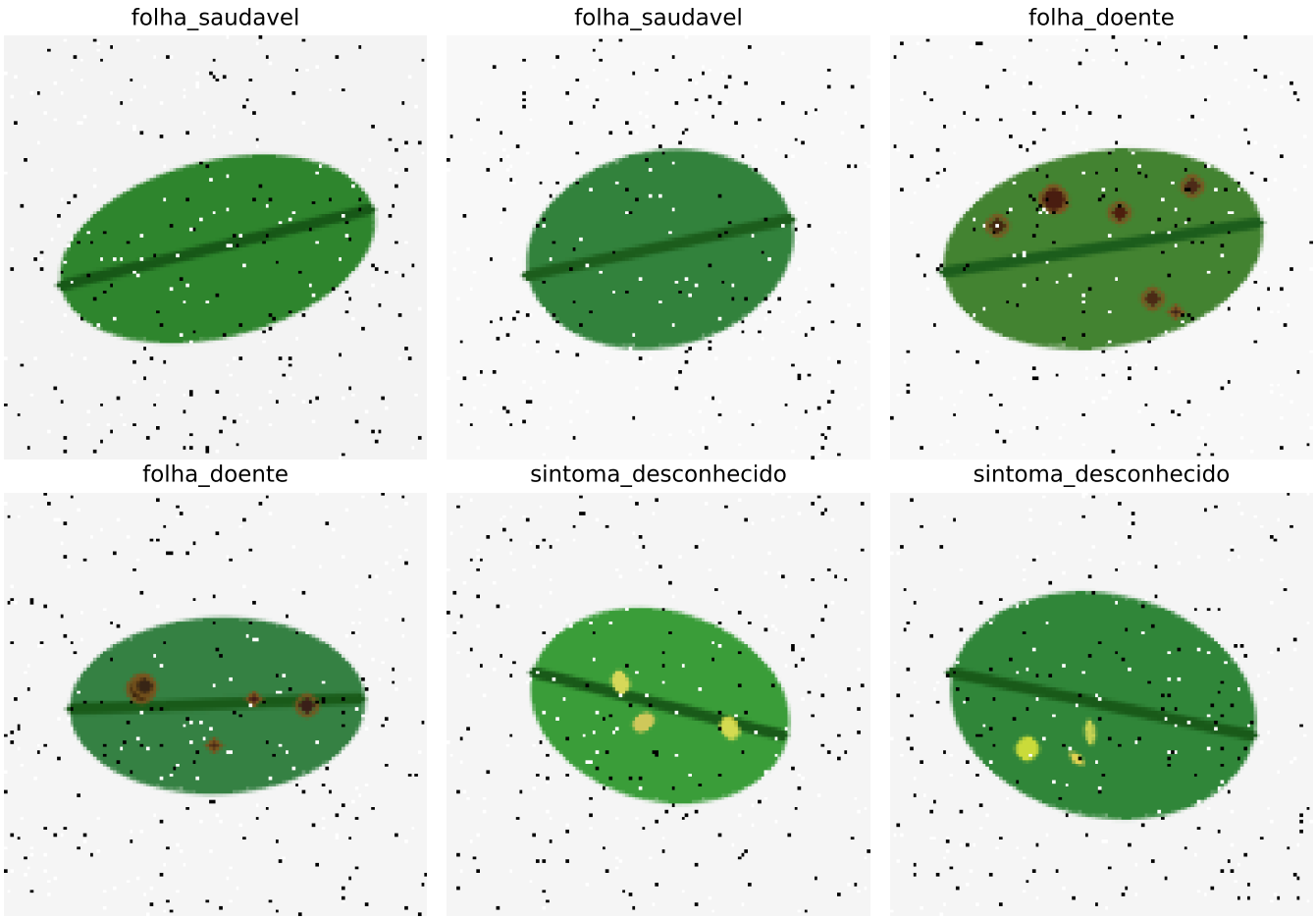


Figura 9.26: Campioni sintetici del *dataset* di diagnosi fogliare: foglia sana, foglia malata (macchie scure) e sintomo sconosciuto (clorosi giallastra) con rumore sale e pepe.

Trappola Comune

Il trasferimento dell'apprendimento richiede che ogni classe presenti pattern visivi distinti. Ripetere la stessa immagine con etichette diverse impedisce al livello classificatore di apprendere una frontiera di decisione, poiché l'estrattore genera praticamente le stesse caratteristiche per tutti i campioni.
In questo esperimento, ogni immagine viene generata in modo indipendente, con pattern visivi compatibili con la propria classe (foglia sana, lesioni fungine o clorosi), fornendo informazioni sufficienti per l'addestramento del livello classificatore.

Bloco 2: Preparazione dei Tensori e Adattamento dell'Architettura

Modelli come la *ResNet-18* richiedono immagini a colori di 224×224 *pixel* normalizzate secondo le statistiche della *ImageNet* ($\mu = [0,485; 0,456; 0,406]$ e $\sigma = [0,229; 0,224; 0,225]$).

1. **Trasformazione di Input (`transforms.Compose`):** si applicano il ridimensionamento e la normalizzazione standard a ciascuna immagine del *dataset* sintetico, producendo i tensori `X_treino`/`X_val` e le etichette `y_treino`/`y_val` — ogni esempio è un'immagine genuinamente distinta, associata all'etichetta corretta della propria classe.
2. **Congelamento dell'Estrattore:** il ciclo `for p in modelo_resnet.parameters(): p.requires_grad = False` disattiva i gradienti nei layer convoluzionali pre-addestrati.
3. **Nuovo Layer Finale:** il layer `modelo_resnet.fc` viene sostituito da una nuova istanza `nn.Linear(modelo_resnet.fc.in_features, n_classes_destino)`, appena inizializzata e con gradienti attivi per impostazione predefinita. Il numero di caratteristiche di input viene ottenuto dinamicamente dal layer originale stesso (`in_features`, pari a 512 nella ResNet-18), invece di essere fissato manualmente nel codice — pratica raccomandata, poiché rende il frammento riutilizzabile per altre varianti dell'architettura senza modifiche.

```
# 1. Pipeline di trasformazioni previste dalla ResNet
transformacao_resnet = T.Compose([
    T.Resize((224, 224)),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])


def prepara_tensores(imgs, labels):
    tensores = [transformacao_resnet(T.functional.to_tensor(img)) for img in imgs]
    X = torch.stack(tensores)
    y = torch.tensor(labels, dtype=torch.long)
    return X, y


# 2. Conversione del dataset sintetico (Blocco 1) in tensori normalizzati
X_treino, y_treino = prepara_tensores(imgs_treino, labels_treino)
X_val, y_val = prepara_tensores(imgs_val, labels_val)

loader_treino = DataLoader(TensorDataset(X_treino, y_treino), batch_size=16, shuffle=True)
loader_val = DataLoader(TensorDataset(X_val, y_val), batch_size=16, shuffle=False)

# 3. Caricamento della ResNet-18 pre-addestrata e congelamento dell'estrattore
n_classes_destino = len(CLASSES_FOLHA)
modelo_resnet = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)

for parametro in modelo_resnet.parameters():
    parametro.requires_grad = False

# 4. Sostituzione del layer finale per le 3 nuove classi di destinazione
modelo_resnet.fc = nn.Linear(modelo_resnet.fc.in_features, n_classes_destino)

print(f"Nuovo layer finale: {modelo_resnet.fc}")
```

Bloco 3: Ciclo di Addestramento e Valutazione

Con l'estrattore congelato e il nuovo strato di uscita correttamente collegato, si esegue il *fine-tuning* della nuova testa di classificazione.

1. **Ottimizzazione Mirata:** l'ottimizzatore *Adam* riceve strettamente `modelo_resnet.fc.parameters()`, aggiornando solo il nuovo strato di uscita — il resto della rete rimane congelato, come definito nel Blocco 2.

2. **Esecuzione del Ciclo:** a ogni epoca, il modello itera sui lotti di addestramento, calcola la perdita tramite Entropia Incrociata e regola i pesi dello strato finale; successivamente, si valuta l'accuratezza sul set di validazione (immagini mai viste durante l'addestramento).
3. **Curve di Addestramento:** la Figura 9.27 monitora l'evoluzione della perdita di addestramento e dell'accuratezza di validazione lungo le epoche — poiché le tre classi sono visivamente distinte tra loro, ci si aspetta una convergenza genuina, ben al di sopra della soglia del 33% corrispondente a una scelta casuale tra 3 classi.

```
dispositivo = torch.device("cuda" if torch.cuda.is_available() else "cpu")
modelo_resnet = modelo_resnet.to(dispositivo)

criterio = nn.CrossEntropyLoss()
otimizador = optim.Adam(modelo_resnet.fc.parameters(), lr=1e-3)

historico_perda, historico_acc = [], []
epocas = 10

for epoca in range(epocas):
    modelo_resnet.train()
    perda_acumulada, n_batches = 0.0, 0

    for X_batch, y_batch in loader_treino:
        X_batch, y_batch = X_batch.to(dispositivo), y_batch.to(dispositivo)

        otimizador.zero_grad()
        saidas = modelo_resnet(X_batch)
        perda = criterio(saidas, y_batch)
        perda.backward()
        otimizador.step()

        perda_acumulada += perda.item()
        n_batches += 1
    historico_perda.append(perda_acumulada / n_batches)

    modelo_resnet.eval()
    acertos, total = 0, 0
    with torch.no_grad():
        for X_batch, y_batch in loader_val:
            X_batch, y_batch = X_batch.to(dispositivo), y_batch.to(dispositivo)
            predicoes = modelo_resnet(X_batch).argmax(dim=1)
            acertos += (predicoes == y_batch).sum().item()
            total += y_batch.size(0)
    acc = acertos / total
    historico_acc.append(acc)

    print(f"Epoca {epoca+1}/{epocas} - perdita: {historico_perda[-1]:.4f} - ",
          f" accuratezza_val: {acc*100:.1f}%")

f=mm.showTrainCurves(
    historico_perda, historico_acc,
    titulo="Fine-Tuning da ResNet-18 - Diagnóstico Foliar",
    subtitulo="Apenas a nova camada linear (fc) é treinada; o extrator permanece congelado",
)
```

Figura 9.27

Blocco 4: Ispezione Qualitativa delle Previsioni

Oltre alla curva di accuratezza aggregata, è istruttivo osservare individualmente alcune previsioni del modello sul set di validazione, confrontando l'etichetta prevista con quella reale. La Figura 9.28 mostra due campioni per ciascuna classe.

```
modelo_resnet.eval()

# Due campioni di ciascuna classe nell'insieme di validazione
idx_amostras = [0, N_POR_CLASSE_VAL, 2 * N_POR_CLASSE_VAL,
                1, N_POR_CLASSE_VAL + 1, 2 * N_POR_CLASSE_VAL + 1]

imgs_pred, titulos_pred = [], []
with torch.no_grad():
    for idx in idx_amostras:
        entrada = X_val[idx].unsqueeze(0).to(dispositivo)
        pred_idx = modelo_resnet(entrada).argmax(dim=1).item()
        real_idx = y_val[idx].item()
        marcador = " " if pred_idx == real_idx else " "
        imgs_pred.append(imgs_val[idx])
        titulos_pred.append(f"{marcador} previsto: {CLASSES_FOLHA[pred_idx]}\n"+
                            f"real: {CLASSES_FOLHA[real_idx]}")

mm.show(imgs_pred, titles=titulos_pred, cols=3, figsize=(10, 7))
```

Figura 9.28

Analisi dell'Esperimento 3

La *ResNet-18* raggiunge un'elevata accuratezza anche utilizzando un insieme ridotto di immagini sintetiche. Questo risultato mostra che le rappresentazioni apprese su **ImageNet** rimangono utili in un dominio completamente diverso, richiedendo solo l'adattamento del livello classificatore.

L'esperimento illustra inoltre una situazione comune nelle applicazioni reali, in cui la disponibilità di dati etichettati è limitata. In questi scenari, il transfer learning riduce i tempi di addestramento e consente di ottenere modelli con buone prestazioni anche senza addestrare l'intera rete.

Sintesi Comparativa — Quando Funziona il Transfer Learning?

I tre esperimenti mostrano che il transfer learning dipende dalla capacità di generalizzazione dell'estrattore di caratteristiche.

- **Esperimento 1:** un estrattore di piccole dimensioni, addestrato su un dominio ristretto, apprende rappresentazioni poco generalizzabili e può produrre **transfer negativo**.
- **Esperimento 2:** una *ResNet-18* pre-addestrata su **ImageNet** trasferisce rappresentazioni generali a un compito di classificazione delle razze di cani e gatti, raggiungendo un'elevata accuratezza con poche immagini.
- **Esperimento 3:** la stessa strategia adatta il modello a un problema di diagnosi fitosanitaria, dimostrando che un unico estrattore può servire come base per diversi domini applicativi.

Confronto tra Approcci di Trasferimento

La Tabella 9.2 riassume i risultati ottenuti nei tre esperimenti.

Tabella 9.2: Confronto tra i tre scenari di trasferimento dell'apprendimento presentati in questa sezione.

Aspetto	Esperimento 1	Esperimento 2	Esperimento 3
Estrattore	Piccola CNN	*ResNet-18*	*ResNet-18*
Addestramento dell'estrattore	Cifre (0–4)	**ImageNet**	**ImageNet**
Capacità di generalizzazione	Bassa	Alta	Alta
Nuovo compito	Cifre (5–9)	Razze di cani e gatti	Diagnosi fitosanitaria
Risultato	Trasferimento negativo	Trasferimento positivo	Trasferimento positivo

9.5.2 Rilevamento di Oggetti

Gli esperimenti precedenti hanno mostrato come il transfer learning adatti modelli pre-addestrati per compiti di classificazione delle immagini. Lo stesso principio è alla base anche delle architetture di rilevamento degli oggetti, in cui un estrattore di caratteristiche pre-addestrato fornisce rappresentazioni visive generali, mentre moduli specializzati localizzano e classificano gli oggetti nell'immagine.

Le sezioni successive presentano la **Faster R-CNN** come esempio di rilevatore pre-addestrato utilizzato direttamente per l'inferenza e, successivamente, un esperimento completo di fine-tuning con l'architettura **YOLO**.

9.5.2.1 Faster R-CNN: Rilevatore Pre-addestrato

La rilevazione degli oggetti estende l'uso di modelli pre-addestrati a un compito più complesso della classificazione. La **Faster R-CNN** utilizza una CNN pre-addestrata, come la *ResNet-50*, come **estrattore di caratteristiche** (*backbone*) e aggiunge moduli specializzati per localizzare e classificare gli oggetti.

Riguardo a questo estrattore, l'architettura incorpora due "teste" principali:

- ***Region Proposal Network* (RPN):** propone regioni dell'immagine con alta probabilità di contenere oggetti.
- **Testa classificatrice:** affina queste regioni, assegna una classe a ciascun oggetto e regola le sue scatole delimitatrici.

Il codice seguente utilizza una Faster R-CNN con pesi pre-addestrati su **COCO** per rilevare oggetti in un'immagine, producendo le loro classi, coordinate e punteggi di confidenza.

> ℹ 🔍 Dove si trova qui il transfer learning?
>
> A differenza degli esperimenti precedenti, questo esempio **non esegue il fine-tuning**. Il modello esegue solo l'inferenza (`eval()`), riutilizzando direttamente i pesi del *backbone*, della RPN e della testa classificatrice addestrati su **COCO**.
> L'adattamento a un nuovo dominio richiederebbe di sostituire il layer `box_predictor` con una nuova testa di classificazione, compatibile con le classi dell'applicazione, e di addestrarla su un insieme di immagini annotate. Questa procedura segue lo stesso principio presentato nella sezione sul transfer learning e costituisce il flusso usuale per applicazioni specifiche, come la rilevazione di parassiti, difetti di fabbricazione o veicoli.

1. **Categorie COCO:** Il codice recupera i nomi delle classi dalle meta-informazioni dei pesi (`FasterRCNN_ResNet50_FPN_Weights.DEFAULT.meta["categories"]`). Sebbene questo elenco contenga 91 voci per ragioni storiche del formato di annotazione di COCO, solo 80 corrispondono a categorie di oggetti.

2. **Inferenza:** L'immagine caricata da `mm.read()` viene convertita in *tensor* e processata dal modello in modalità di valutazione (`eval()`). Il codice mantiene solo le rilevazioni con confidenza superiore

all'80%.

3. **Annotazione dell'immagine:** Per ogni oggetto rilevato, il codice disegna la scatola delimitatrice (`cv2.rectangle`) e scrive la classe predetta e la sua confidenza (`cv2.putText`).
4. **Visualizzazione:** La Figura 9.29 presenta l'immagine annotata con le rilevazioni effettuate dal modello.

9.5.2.2 Rilevamento di Oggetti e Trasferimento dell'Apprendimento con YOLO

Le sezioni precedenti hanno applicato il trasferimento dell'apprendimento a problemi di **classificazione delle immagini**, in cui il modello associa un'unica etichetta all'intera immagine. In questa sezione, lo stesso principio viene esteso al **rilevamento di oggetti**, un compito che richiede di identificare simultaneamente **cosa** è presente nell'immagine e **dove** si trova ciascun oggetto.

La sottosezione precedente ha presentato la **Faster R-CNN** come esempio di rilevatore pre-addestrato utilizzato direttamente per l'inferenza, senza alcun adattamento al nuovo dominio. In questo esperimento, il modello viene sottoposto a una fase di **ottimizzazione fine** (*fine-tuning*): si parte da un'architettura **YOLO** (*You Only Look Once*) pre-addestrata sul dataset **COCO** e si adatta la rete per rilevare e classificare oggetti di un nuovo dominio.

A differenza della Faster R-CNN, che esegue il rilevamento in due fasi, la famiglia **YOLO** adotta un'architettura a stadio singolo (*single-stage detector*), stimando, in un'unica propagazione attraverso la rete, le scatole delimitatrici (*bounding boxes*), la confidenza di ciascun rilevamento e la classe corrispondente. Questa strategia riduce il costo computazionale e rende possibili applicazioni in tempo reale.

Come esempio, l'esperimento utilizza un insieme sintetico di forme geometriche (triangoli, quadrati, stelle, tra le altre), con variazioni di colore, dimensione, rotazione e degrado dovuto a rumore di tipo sale e pepe.

Bloco 1: Generazione del *Dataset* Sintetico

Il blocco seguente genera un insieme sintetico per l'addestramento e la valutazione del rivelatore. Ogni immagine contiene da uno a tre oggetti appartenenti a una delle nove classi:

```
CLASSES = [
    'Triangle', 'Square', 'Pentagon', 'Hexagon',
    'Heptagon', 'Circle', 'Ellipse', 'Star', 'Cross'
]
```

1. **Generazione delle forme:** Le funzioni `poligono_regular`, `poligono_estrela` e `poligono_cruz` costruiscono le coordinate degli oggetti. La funzione `desenha_objeto` disegna ogni forma con posizione, dimensione, orientamento e colore casuali e calcola la sua *bounding box*.
2. **Annotazione nel formato YOLO:** La funzione `gera_imagem_ruidosa` genera da uno a tre oggetti per immagine e converte ogni *bounding box* nel formato YOLO, rappresentato dalla classe e dalle coordinate normalizzate del centro, larghezza e altezza.
3. **Degradazione dell'immagine:** La funzione `adiciona_ruido_sal_pimenta` aggiunge rumore impulsivo, simulando imperfezioni di acquisizione.
4. **Visualizzazione dei campioni:** La Figura 9.30 presenta esempi dell'insieme sintetico con le *bounding boxes* sovrapposte tramite la funzione `mm.showBoundBox()`.

```
CLASSES = [
    'Triangle', 'Square', 'Pentagon', 'Hexagon',
    'Heptagon', 'Circle', 'Ellipse', 'Star', 'Cross'
]
N_LADOS = {'Triangle': 3, 'Square': 4, 'Pentagon': 5, 'Hexagon': 6, 'Heptagon': 7}

def poligono_regular(cx, cy, r, n_lados, rot_graus):
    ang0 = np.deg2rad(rot_graus - 90)
```

```python
# 1. Caricamento dell'immagine e dei nomi delle categorie di COCO
url_imagem = "https://raw.githubusercontent.com/pytorch/hub/master/images/dog.jpg"
url_imagem = "http://images.cocodataset.org/val2017/000000039769.jpg"
img = mm.read(url_imagem)

pesos_coco = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
categorias_coco = pesos_coco.meta["categories"]  # Mappatura indice -> nome della classe

# 2. Caricamento del modello Faster R-CNN pre-addestrato
modelo_detection = fasterrcnn_resnet50_fpn(weights=pesos_coco).eval()

# 3. Esecuzione dell'inferenza senza calcolo dei gradienti
with torch.no_grad():
    predicao = modelo_detection([to_tensor(img)])[0]

# 4. Filtraggio delle rilevazioni con confidenza superiore all'80%
limiar_confianca = 0.8
mascara_confianca = predicao["scores"] >= limiar_confianca

caixas_filtradas = predicao["boxes"][mascara_confianca].numpy()
scores_filtrados = predicao["scores"][mascara_confianca].numpy()
labels_filtrados = predicao["labels"][mascara_confianca].numpy()

img_com_caixas = img.copy()

# 5. Disegno delle bounding box e delle etichette di classe
for box, score, label_idx in zip(caixas_filtradas, scores_filtrados, labels_filtrados):
    x1, y1, x2, y2 = box.astype(int)
    nome_classe = categorias_coco[label_idx]
    texto_rotulo = f"{nome_classe}: {score:.2f}"

    # Disegna il rettangolo rosso (RGB: 255, 0, 0) con spessore di 3 pixel
    cv2.rectangle(img_com_caixas, (x1, y1), (x2, y2), (255, 0, 0), 3)

    # Scrive la classe e la confidenza sopra la bounding box
    cv2.putText(
        img_com_caixas,
        texto_rotulo,
        (x1, max(y1 - 10, 20)),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.8,
        (255, 0, 0),
        2,
        cv2.LINE_AA,
    )

# 6. Visualizzazione grafica dell'immagine risultante
mm.show(img_com_caixas, title="Faster R-CNN (COCO) - Rilevamento con Classe e Confidenza")
```

Figura 9.29

```
    angs = ang0 + 2 * np.pi * np.arange(n_lados) / n_lados
    return np.stack([cx + r * np.cos(angs), cy + r * np.sin(angs)], axis=1)

def poligono_estrela(cx, cy, r_externo, rot_graus, n_pontas=5):
    r_interno = r_externo * 0.45
    ang0 = np.deg2rad(rot_graus - 90)
    angs = ang0 + np.pi * np.arange(2 * n_pontas) / n_pontas
    raios = np.where(np.arange(2 * n_pontas) % 2 == 0, r_externo, r_interno)
    return np.stack([cx + raios * np.cos(angs), cy + raios * np.sin(angs)], axis=1)

def poligono_cruz(cx, cy, r, rot_graus, espessura_rel=0.35):
    w = r * espessura_rel
    base = np.array([
        (-w, -r), (w, -r), (w, -w), (r, -w), (r, w), (w, w),
        (w, r), (-w, r), (-w, w), (-r, w), (-r, -w), (-w, -w),
    ])
    theta = np.deg2rad(rot_graus)
    R = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
    return base @ R.T + np.array([cx, cy])

def desenha_objeto(img, classe_idx, cx, cy, tamanho, rotacao, cor):
    nome = CLASSES[classe_idx]
    if nome in N_LADOS:
        pts = poligono_regular(cx, cy, tamanho, N_LADOS[nome], rotacao)
        cv2.fillPoly(img, [pts.astype(np.int32)], cor)
        xs, ys = pts[:, 0], pts[:, 1]
    elif nome == 'Star':
        pts = poligono_estrela(cx, cy, tamanho, rotacao)
        cv2.fillPoly(img, [pts.astype(np.int32)], cor)
        xs, ys = pts[:, 0], pts[:, 1]
    elif nome == 'Cross':
        pts = poligono_cruz(cx, cy, tamanho, rotacao)
        cv2.fillPoly(img, [pts.astype(np.int32)], cor)
        xs, ys = pts[:, 0], pts[:, 1]
    elif nome == 'Circle':
        cv2.circle(img, (int(cx), int(cy)), int(tamanho), cor, -1)
        xs, ys = np.array([cx - tamanho, cx + tamanho]), np.array([cy - tamanho, cy +
tamanho])
    else:  # Ellisse
        eixo = (int(tamanho), int(tamanho * 0.6))
        cv2.ellipse(img, (int(cx), int(cy)), eixo, rotacao, 0, 360, cor, -1)
        ang = np.deg2rad(rotacao)
        dx = np.hypot(eixo[0] * np.cos(ang), eixo[1] * np.sin(ang))
        dy = np.hypot(eixo[0] * np.sin(ang), eixo[1] * np.cos(ang))
        xs, ys = np.array([cx - dx, cx + dx]), np.array([cy - dy, cy + dy])
    return xs.min(), ys.min(), xs.max(), ys.max()

def adiciona_ruido_sal_pimenta(img, quantidade=0.05):
    img_ruidosa = img.copy()
    h, w, c = img_ruidosa.shape
    num_ruido = int(quantidade * h * w)

    # Sale (255, 255, 255)
    coords_sal = [np.random.randint(0, i - 1, num_ruido) for i in (h, w)]
    img_ruidosa[coords_sal[0], coords_sal[1]] = [255, 255, 255]

    # Pepe (0, 0, 0)
    coords_pimenta = [np.random.randint(0, i - 1, num_ruido) for i in (h, w)]
    img_ruidosa[coords_pimenta[0], coords_pimenta[1]] = [0, 0, 0]

    return img_ruidosa
```

```python
def gera_imagem_ruidosa(tam_img=160, n_objetos=(1, 3), taxa_ruido=0.01, rng=None):
    rng = rng or random.Random()
    img_limpa = np.full((tam_img, tam_img, 3), 255, dtype=np.uint8)
    anotacoes = []

    for _ in range(rng.randint(*n_objetos)):
        classe_idx = rng.randrange(len(CLASSES))
        tamanho = rng.randint(tam_img // 10, tam_img // 5)
        cx = rng.randint(tamanho + 2, tam_img - tamanho - 2)
        cy = rng.randint(tamanho + 2, tam_img - tamanho - 2)
        rotacao = rng.uniform(0, 360)
        cor = tuple(rng.sample(range(30, 226), 3))

        x0, y0, x1, y1 = desenha_objeto(img_limpa, classe_idx, cx, cy, tamanho, rotacao, cor)
        x0, y0 = max(x0, 0), max(y0, 0)
        x1, y1 = min(x1, tam_img), min(y1, tam_img)

        # Formato YOLO: (classe, centro_x, centro_y, larghezza, altezza) normalizzati
        xc, yc = (x0 + x1) / 2 / tam_img, (y0 + y1) / 2 / tam_img
        w, h = (x1 - x0) / tam_img, (y1 - y0) / tam_img
        anotacoes.append((classe_idx, xc, yc, w, h))

    img_ruidosa = adiciona_ruido_sal_pimenta(img_limpa, quantidade=taxa_ruido)
    return img_ruidosa, anotacoes
```

```python
# Generazione di 5 campioni per la visualizzazione iniziale in cima al progetto
n_amostras_iniciais = 5
rng_demo = random.Random(42)

imgs_demo = []
titulos_demo = []

for idx in range(n_amostras_iniciais):
    img_ruid, anotacoes = gera_imagem_ruidosa(rng=rng_demo)

    # Scrittura temporanea dell'annotazione per la lettura nativa tramite mm.showBoundBox
    filename_temp = f"temp_label_{idx}.txt"
    with open(filename_temp, "w") as f:
        for c, xc, yc, w, h in anotacoes:
            f.write(f"{c} {xc:.4f} {yc:.4f} {w:.4f} {h:.4f}\n")

    img_anotada = mm.showBoundBox(img_ruid, filename=filename_temp, fmt="yolo", show=False)
    imgs_demo.append(img_anotada)
    titulos_demo.append(f"Amostra {idx+1}")

# Visualizzazione del pannello di 5 campioni
mm.show(
    imgs_demo,
    titles=titulos_demo,
    cols=n_amostras_iniciais,
    figsize=(14, 3)
)
```

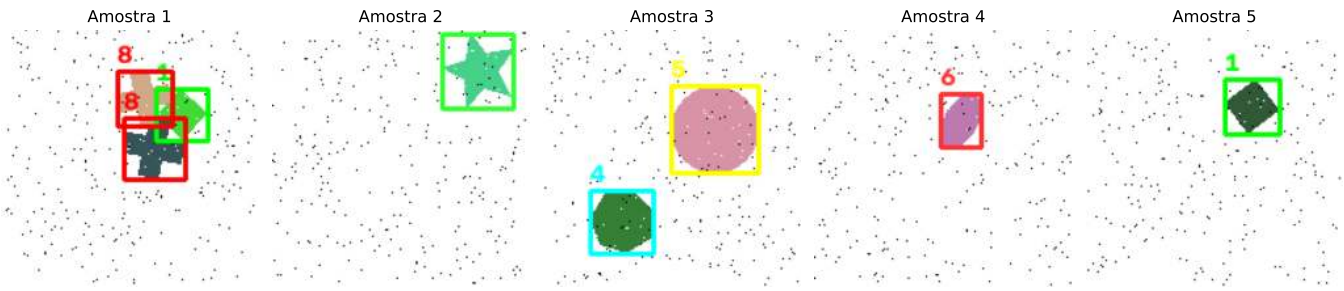


Figura 9.30: Campioni iniziali del *dataset* sintetico rumoroso di oggetti geometrici con le bounding box in formato YOLO sovrapposte.

Bloco 2: Organizzazione del *Dataset* e Creazione del File `data.yaml`

Questo blocco organizza il set di dati nel formato previsto dalla libreria **Ultralytics YOLO**. Le immagini e le annotazioni sono distribuite in directory separate per l'addestramento e la validazione, mentre il file `data.yaml` raccoglie le informazioni necessarie per l'addestramento del rilevatore.

```
shapes_dataset/
   data.yaml
   images/
       train/
       val/
   labels/
        train/
        val/
```

1. **Generazione del set di dati:** Il codice crea **90** immagini per l'addestramento e **20** per la validazione. Per ciascuna immagine, scrive un file `.txt` contenente una riga per oggetto, nel formato YOLO (`classe`, `x_c`, `y_c`, `larghezza`, `altezza`), con tutte le coordinate normalizzate.
2. **Organizzazione dei file:** Le immagini sono archiviate in `images/train` e `images/val`, mentre le annotazioni corrispondenti vengono salvate in `labels/train` e `labels/val`, preservando lo stesso nome del file.
3. **Creazione del file `data.yaml`:** Il codice genera automaticamente il file di configurazione contenente il percorso del *dataset*, le directory di addestramento e validazione e la mappatura tra gli indici numerici e i nomi delle nove classi.

```
base_dir = "shapes_dataset"
rng_global = random.Random(42)

for split, n_imgs in [("train", 90), ("val", 20)]:
    os.makedirs(f"{base_dir}/images/{split}", exist_ok=True)
    os.makedirs(f"{base_dir}/labels/{split}", exist_ok=True)

    for i in range(n_imgs):
        img_ruid, anotacoes = gera_imagem_ruidosa(rng=rng_global)
        cv2.imwrite(f"{base_dir}/images/{split}/{i:04d}.jpg", img_ruid)

        with open(f"{base_dir}/labels/{split}/{i:04d}.txt", "w") as f:
            for c, xc, yc, w, h in anotacoes:
                f.write(f"{c} {xc:.4f} {yc:.4f} {w:.4f} {h:.4f}\n")

with open(f"{base_dir}/data.yaml", "w") as f:
    f.write(
        f"path: {os.path.abspath(base_dir)}\n"
        "train: images/train\nval: images/val\nnames:\n"
```

```
    )
    for i, nome in enumerate(CLASSES):
        f.write(f"  {i}: {nome}\n")

print("Set di dati rumoroso generato con successo: 90 immagini di addestramento e 20 di
validazione.")
```

```
Set di dati rumoroso generato con successo: 90 immagini di addestramento e 20 di validazione.
```

Bloco 3: Pre-elaborazione con Filtro Mediano

Questo blocco applica una pre-elaborazione per ridurre l'effetto del rumore di tipo sale e pepe introdotto nella generazione del *dataset*. Il **Filtro Mediano** (`cv2.medianBlur`) rimuove questo tipo di degradazione preservando meglio i bordi degli oggetti rispetto ai filtri di smoothing convenzionali.

1. **Filtraggio dell'immagine:** Il codice applica un filtro mediano con finestra 3×3 all'immagine rumorosa, riducendo i *pixel* impulsivi senza alterare le annotazioni del dataset.

2. **Visualizzazione comparativa:** La Figura 9.31 confronta l'immagine originale e l'immagine filtrata, mantenendo le *bounding boxes* sovrapposte tramite la funzione `mm.showBoundBox()`.

```
# 1. Caricamento del primo campione rumoroso del dataset
caminho_img = f"{base_dir}/images/train/0000.jpg"
caminho_label = f"{base_dir}/labels/train/0000.txt"

img_ruidosa = mm.read(caminho_img)

# 2. Pre-elaborazione con Filtro Mediano (finestra 3x3)
img_filtrada = cv2.medianBlur(img_ruidosa, ksize=3)

# 3. Sovrapposizione delle bounding box con mm.showBoundBox
img_ruid_anotada = mm.showBoundBox(img_ruidosa, filename=caminho_label, fmt="yolo",
show=False)
img_filt_anotada = mm.showBoundBox(img_filtrada, filename=caminho_label, fmt="yolo",
show=False)

# 4. Visualizzazione comparativa con mm.show
mm.show(
    [img_ruid_anotada, img_filt_anotada],
    titles=[
        "1. Immagine Rumorosa Originale (Sale e Pepe)",
        "2. Pre-elaborata (Filtro Mediano 3x3)"
    ],
    cols=2,
    figsize=(9, 4)
)
```

Figura 9.31: Confronto tra l'immagine originale con rumore sale e pepe e l'immagine dopo l'applicazione del Filtro Mediano (3x3). Le *bounding box* in formato YOLO rimangono invariate.

Bloco 4: Pre-elaborazione del *Dataset* e Fine-Tuning di YOLOv8

Questo blocco applica il Filtro Mediano all'insieme di immagini ed esegue il *fine-tuning* del rilevatore **YOLOv8n** pre-addestrato sull'insieme **COCO**. La filtrazione riduce l'effetto del rumore impulsivo introdotto nella generazione delle immagini, mentre l'addestramento adatta i parametri della rete al nuovo dominio delle forme geometriche.

1. **Filtraggio in batch:** Il codice scorre le directory `train` e `val` e applica `cv2.medianBlur` con finestra 3×3 su tutte le immagini, mantenendo le annotazioni YOLO originali.

2. **Fine-tuning del rilevatore:** La rete `YOLO("yolov8n.pt")`, inizialmente addestrata su COCO, viene adattata all'insieme geometrico tramite la funzione `.train()`. L'addestramento utilizza immagini con risoluzione 320×320 *pixel* per 30 epoche.

3. **Valutazione del modello:** La funzione `.val()` calcola le metriche di rilevamento sull'insieme di validazione, includendo **precisione** (*precision*), **revocazione** (*recall*) e **mAP50** (*mean Average Precision* con soglia IoU pari a $0,5$).

```
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

base_dir = "shapes_dataset"
base_dir_filt = "shapes_dataset_filtrado"

# 1. Crea una CÓPIA filtrata del dataset in una cartella separata
#    (il dataset originale in base_dir rimane rumoroso, intatto)
for split in ["train", "val"]:
    pasta_imgs_orig = f"{base_dir}/images/{split}"
    pasta_labels_orig = f"{base_dir}/labels/{split}"
    pasta_imgs_filt = f"{base_dir_filt}/images/{split}"
    pasta_labels_filt = f"{base_dir_filt}/labels/{split}"

    os.makedirs(pasta_imgs_filt, exist_ok=True)
```

```
    os.makedirs(pasta_labels_filt, exist_ok=True)

    for nome_arq in os.listdir(pasta_imgs_orig):
        if nome_arq.endswith(".jpg"):
            img_ruidosa = cv2.imread(f"{pasta_imgs_orig}/{nome_arq}")
            img_filtrada = cv2.medianBlur(img_ruidosa, ksize=3)
            # salva la versione filtrata nella cartella NUOVA, non sovrascrive l'originale
            cv2.imwrite(f"{pasta_imgs_filt}/{nome_arq}", img_filtrada)

    # copia le etichette (non cambiano con il filtro)
    for nome_arq in os.listdir(pasta_labels_orig):
        shutil.copy(f"{pasta_labels_orig}/{nome_arq}", f"{pasta_labels_filt}/{nome_arq}")

# 2. data.yaml che punta al dataset FILTRATO
with open(f"{base_dir_filt}/data.yaml", "w") as f:
    f.write(
        f"path: {os.path.abspath(base_dir_filt)}\n"
        "train: images/train\nval: images/val\nnames:\n"
    )
    for i, nome in enumerate(CLASSES):
        f.write(f"  {i}: {nome}\n")

print("Pre-elaborazione completata: dataset filtrato salvato in una cartella separata.\n")

# Scaricare il modello YOLOv8 pre-addestrato (yolov8n.pt) se non è già presente
with contextlib.redirect_stdout(io.StringIO()), \
    contextlib.redirect_stderr(io.StringIO()):
    modelo_yolo = YOLO("yolov8n.pt")
print("Modello YOLOv8 caricato.")
```

```
Pre-elaborazione completata: dataset filtrato salvato in una cartella separata.

Modello YOLOv8 caricato.
```

```
# 3. Callback personalizzato per stampare solo l'epoca in esecuzione
def on_train_epoch_start(trainer):
    epoch_atual = trainer.epoch + 1
    total_epochs = trainer.epochs
    # Scrive direttamente sullo stdout originale (bypassando il silenziatore)
    sys.__stdout__.write(f"  Processando Época {epoch_atual}/{total_epochs}...\n")
    sys.__stdout__.flush()

# Aggiunge il callback al modello
modelo_yolo.add_callback("on_train_epoch_start", on_train_epoch_start)

# 4. Gestore di contesto per silenziare la spazzatura di Ultralytics (C/C++ e Python)
@contextlib.contextmanager
def silenciar_logs():
    logger = logging.getLogger("ultralytics")
    disabled_state = logger.disabled
    logger.disabled = True

    with open(os.devnull, "w") as fnull:
        old_stdout_fd = os.dup(1)
        old_stderr_fd = os.dup(2)
        try:
            os.dup2(fnull.fileno(), 1)
            os.dup2(fnull.fileno(), 2)
            with contextlib.redirect_stdout(fnull), contextlib.redirect_stderr(fnull):
                yield
```

```
        finally:
            os.dup2(old_stdout_fd, 1)
            os.dup2(old_stderr_fd, 2)
            os.close(old_stdout_fd)
            os.close(old_stderr_fd)
            logger.disabled = disabled_state

# Esecuzione dell'Addestramento
print("--- Avvio Addestramento YOLOv8 ---")
with silenciar_logs():
    resultados_treino = modelo_yolo.train(
        data=f"{base_dir_filt}/data.yaml",   # <-- addestra sul dataset filtrato
        epochs=30,
        imgsz=320,
        batch=16,
        device=device,
        verbose=False,
        plots=False
    )
    metricas = modelo_yolo.val(verbose=False)

# 5. Metriche Finali
precision = metricas.results_dict["metrics/precision(B)"]
recall = metricas.results_dict["metrics/recall(B)"]
map50 = metricas.results_dict["metrics/mAP50(B)"]

print("\n--- Prestazioni Ottimizzate del Modello YOLOv8 ---")
print(f"Precisione: {precision*100:.2f}%")
print(f"Revocazione (Recall): {recall*100:.2f}%")
print(f"mAP al 50% (IoU 0.50): {map50*100:.2f}%")
```

```
--- Avvio Addestramento YOLOv8 ---

--- Prestazioni Ottimizzate del Modello YOLOv8 ---
Precisione: 67.29%
Revocazione (Recall): 90.17%
mAP al 50% (IoU 0.50): 84.45%
```

Blocco 5: Confronto di Inferenza: Immagine Rumorosa e Immagine Ripristinata

Dopo la messa a punto della YOLOv8 sull'insieme ripristinato, si esegue un confronto visivo tra il rilevamento applicato direttamente a un'immagine degradata dal rumore sale e pepe e la stessa immagine dopo il Filtro Mediano.

L'obiettivo è osservare come una semplice fase di preelaborazione possa influenzare la qualità delle previsioni di un rilevatore già adattato al nuovo dominio.

1. **Inferenza con la YOLO:** La funzione `modello_yolo.predict()` esegue il rilevamento nelle due versioni dell'immagine, utilizzando una soglia di confidenza del 25% (`conf=0.25`).
2. **Visualizzazione delle Previsioni:** La funzione `plot()` genera le immagini annotate con i riquadri delimitatori e le etichette previste dal modello. La Figura 9.32 presenta il confronto tra i due scenari.

```
# Pulizia esplicita dei dati scaricati e liberazione della memoria
if FLAG_LIMPAR_DADOS:
    if os.path.exists(base_dir):
        shutil.rmtree(base_dir)
        print('  Diretório de dados temporários {} removido com sucesso.'.format(base_dir))

    if os.path.exists(base_dir_filt):
```

```
# 1. Caricamento di un campione di test originale (senza il filtro salvato in batch)
caminho_teste = f"{base_dir}/images/val/0002.jpg"
img_ruidosa_teste = mm.read(caminho_teste)

# 2. Applicazione puntuale del Filtro Mediano (3x3) per il confronto
img_filtrada_teste = cv2.medianBlur(img_ruidosa_teste, ksize=3)

# 3. Inferenza con il modello YOLOv8 addestrato

pred_ruidosa = modelo_yolo.predict(img_ruidosa_teste, conf=0.25, verbose=False)[0]
pred_filtrada = modelo_yolo.predict(img_filtrada_teste, conf=0.25, verbose=False)[0]

# 4. Estrazione delle matrici annotate dal generatore di YOLO (conversione BGR -> RGB)
img_pred_ruid = cv2.cvtColor(pred_ruidosa.plot(), cv2.COLOR_BGR2RGB)
img_pred_filt = cv2.cvtColor(pred_filtrada.plot(), cv2.COLOR_BGR2RGB)

# 5. Visualizzazione comparativa standardizzata tramite mm.show

mm.show(
    [img_pred_ruid, img_pred_filt],
    titles=[
        f"Inferenza sull'Immagine Rumorosa ({len(pred_ruidosa.boxes)} oggetti)",
        f"Inferenza sull'Immagine Filtrata ({len(pred_filtrada.boxes)} oggetti)"
    ],
    cols=2,
    figsize=(10, 4)

)
```

Figura 9.32

```
        shutil.rmtree(base_dir_filt)
        print('  Diretório de dados temporários {} removido com
        sucesso.'.format(base_dir_filt))

    if torch.cuda.is_available():
        torch.cuda.empty_cache()
```

Un dominio molto più distante di quello delle cifre

Nell'esperimento di transfer learning tra cifre scritte a mano (domini A e B), il compito di origine e quello di destinazione condividevano statistiche visive molto simili: entrambi erano tratti in scala di grigi su sfondo uniforme. Qui, la distanza tra i domini è molto maggiore — YOLO è stato pre-addestrato su **fotografie naturali a colori** di COCO (persone, animali, veicoli, oggetti quotidiani), e il compito di destinazione consiste in **forme geometriche sintetiche, di colore pieno e contorno ben definito**, senza texture, illuminazione o sfondo complesso.
Ciononostante, il transfer learning risulta vantaggioso: i primi strati di un rilevatore addestrato su COCO apprendono filtri generici — rilevatori di bordi, angoli e regioni di contrasto — che restano utili per delimitare il contorno di un triangolo o di una stella, anche se il contenuto visivo finale è piuttosto diverso. È per questo che consentire il *fine-tuning* di tutti gli strati, combinato con un preprocessamento coerente tra addestramento e inferenza per attenuare il rumore sale e pepe, rende possibili tassi di accuratezza elevati nel rilevamento di oggetti del nuovo dominio.

Utilizzando un Set di Dati Reale

Il *pipeline* sopra descritto è stato costruito interamente attorno al **formato di annotazione YOLO** (classe, x_{centro}, y_{centro}, larghezza, altezza, normalizzati per larghezza e altezza dell'immagine) esattamente affinché

possa essere riutilizzato senza modifiche qualora il lettore abbia accesso a un set di immagini reali annotate nello stesso modo — ad esempio, un set di immagini di oggetti geometrici fotografati o renderizzati, ciascuno con un file `.txt` corrispondente nello stesso formato usato qui. A tal fine, basterebbe:

1. Organizzare le immagini reali in `shapes_dataset/images/train` e `shapes_dataset/images/val`, e i file `.txt` di annotazione corrispondenti nelle cartelle `labels/train` e `labels/val` (un file di annotazione per immagine, stesso nome di base, estensione `.txt`);
2. Modificare il file `data.yaml` qualora il numero o i nomi delle classi fossero diversi;
3. Eseguire le stesse celle di addestramento, ottimizzazione e visualizzazione già presentate, senza alcun'altra modifica al codice.

Questa separazione tra **generazione/organizzazione dei dati** e **addestramento del modello** è, in pratica, il motivo per cui formati di annotazione standardizzati (come quello di YOLO) sono così ampiamente adottati: essi consentono di sostituire il set di dati di input — da sintetico a reale, da un dominio a un altro — mantenendo invariato tutto il resto del *pipeline* di transfer learning.

9.5.3 Segmentazione degli Oggetti

Lo stesso principio di trasferimento dell'apprendimento è alla base anche delle architetture di segmentazione delle immagini, in cui un estrattore di caratteristiche pre-addestrato fornisce rappresentazioni visive generali, mentre una testa specializzata esegue la classificazione densa, pixel per pixel.

Le sezioni successive presentano **DeepLabV3** come esempio di segmentatore pre-addestrato utilizzato direttamente per l'inferenza e, successivamente, l'architettura **U-Net**, addestrata da zero e confrontata con una baseline morfologica classica.

9.5.3.1 DeepLabV3: Segmentatore Pre-addestrato

Nella segmentazione semantica, l'obiettivo non si limita alla localizzazione degli oggetti tramite bounding box. La rete assegna una classe a ogni pixel dell'immagine, producendo una mappa di etichette con la stessa risoluzione dell'input. Architetture come la **DeepLabV3**, con backbone **ResNet-50**, utilizzano un estrattore di caratteristiche pre-addestrato e una testa specializzata per eseguire questa classificazione densa.

1. **Caricamento e inferenza:** Il modello `deeplabv3_resnet50(weights="DEFAULT")` carica pesi pre-addestrati sul dataset **Pascal VOC**, che definisce 21 classi di segmentazione. Il codice converte l'immagine in tensore, aggiunge la dimensione del batch (`unsqueeze(0)`) ed esegue l'inferenza.

2. **Mappa delle classi:** L'output del modello ha dimensioni $(1, 21, H, W)$, contenente un valore per ogni classe in ogni pixel. L'operazione `.argmax(dim=1)` seleziona la classe con la risposta più alta in ciascuna posizione, generando una matrice bidimensionale di etichette con dimensioni (H, W).

3. **Visualizzazione:** Il codice converte la mappa delle etichette nel formato atteso da `mm.show()`, che mostra il risultato della segmentazione nella Figura 9.33.

9.5.3.2 Segmentazione Semantica con Architettura U-Net

La sottosezione precedente ha presentato un modello pre-addestrato (*DeepLabV3*) che produce direttamente un'etichetta di classe per *pixel*. Questa sezione completa la sequenza di progetti pratici — classificazione e rilevamento — affrontando la segmentazione semantica implementata e addestrata da zero con la **U-Net**, l'architettura di riferimento introdotta da Ronneberger (2015).

Nella classificazione, la mappa delle caratteristiche finale veniva appiattita (*flatten*) in un vettore, scartando l'informazione spaziale a favore di un'unica etichetta per immagine. La U-Net, invece, produce un'uscita con la **stessa risoluzione spaziale dell'ingresso**: una mappa bidimensionale in cui ogni *pixel* riceve la propria classificazione. Questo requisito — preservare il dettaglio spaziale ad alta risoluzione mentre si costruisce il contesto semantico negli strati profondi — motiva l'architettura encoder-decoder con connessioni di salto (*skip connections*).

```
# 1. Caricamento dell'immagine (restituisce numpy.ndarray)
url_imagem = "https://raw.githubusercontent.com/pytorch/hub/master/images/dog.jpg"
url_imagem = "http://images.cocodataset.org/val2017/000000039769.jpg"
img = mm.read(url_imagem)

# 2. Caricamento del modello DeepLabV3 pre-addestrato in modalità di valutazione
modelo_segmentacao = deeplabv3_resnet50(weights="DEFAULT").eval()

# 3. Esecuzione dell'inferenza senza calcolo dei gradienti
with torch.no_grad():
    tensor_entrada = to_tensor(img).unsqueeze(0)  # Formato (1, C, H, W)
    saida = modelo_segmentacao(tensor_entrada)["out"]

    # Selezione della classe con probabilità maggiore per pixel (argmax sull'asse dei canali)
    mapa_classes = saida.argmax(dim=1).squeeze(0).byte().cpu().numpy()
```

```
# 4. Visualizzazione della mappa di segmentazione semantica
mm.show(
    [img,mapa_classes],
    title=["Immagine originale","Segmentazione Semantica (DeepLabV3)"]
)
```

Figura 9.33

Lo scenario utilizzato simula la segmentazione di noduli in esami medici sintetici: immagini in scala di grigi contengono una regione circolare ("nodulo") sovrapposta a uno sfondo, entrambi contaminati da rumore gaussiano e con medie di intensità molto vicine — una sfida intenzionale a basso contrasto, ideale per dimostrare il guadagno dell'apprendimento spaziale rispetto alla sogliatura puntuale.

Bloco 1: Generatore dell'Insieme Sintetico di Noduli e Visualizzazione Iniziale

Il generatore seguente produce coppie (immagine, maschera): l'immagine contiene una regione circolare di intensità leggermente superiore a quella dello sfondo, entrambe affette dalla stessa deviazione standard del rumore gaussiano. La maschera binaria delimita esattamente la regione del nodulo e funge da verità di riferimento (*ground truth*).

1. **Costruzione del Nodulo:** La funzione `gera_imagem_com_nodulo` sovrappone il nodulo allo sfondo su una matrice 64×64 e applica rumore gaussiano.
2. **Visualizzazione con `mm.show`:** La Figura 9.34 illustra i primi due campioni e le rispettive maschere.

```
TAM_IMG = 64


def gera_imagem_com_nodulo(
    tam=TAM_IMG,
    raio_min=7,
    raio_max=15,
    media_fundo=95,
    media_nodulo=118,
    sigma_ruido=26,
    rng=None,
):
    rng = rng or np.random.default_rng()
    fundo = rng.normal(media_fundo, sigma_ruido, (tam, tam))
    nodulo = rng.normal(media_nodulo, sigma_ruido, (tam, tam))
    mascara = np.zeros((tam, tam), dtype=np.uint8)
```

```
    raio = int(rng.integers(raio_min, raio_max))
    cx = int(rng.integers(raio + 4, tam - raio - 4))
    cy = int(rng.integers(raio + 4, tam - raio - 4))

    cv2.circle(mascara, (cx, cy), raio, 255, -1)
    imagem = np.clip(np.where(mascara > 0, nodulo, fundo), 0, 255).astype(
        np.uint8
    )
    return imagem, mascara


# Generazione dei set di training e validazione
rng_dados = np.random.default_rng(42)
N_TREINO, N_VAL = 160, 40

imgs_treino, masks_treino = zip(
    *[gera_imagem_com_nodulo(rng=rng_dados) for _ in range(N_TREINO)]
)
imgs_val, masks_val = zip(
    *[gera_imagem_com_nodulo(rng=rng_dados) for _ in range(N_VAL)]
)

print(
    f"Set di training: {N_TREINO} immagini | Set di validazione: {N_VAL} immagini\n"
)

# Visualizzazione dei campioni iniziali
mm.show(
    [imgs_treino[0], masks_treino[0], imgs_treino[1], masks_treino[1]],
    titles=["Immagine 1", "Maschera 1", "Immagine 2", "Maschera 2"],
    cols=4,
    figsize=(11, 3),
)
```

```
Set di training: 160 immagini | Set di validazione: 40 immagini
```

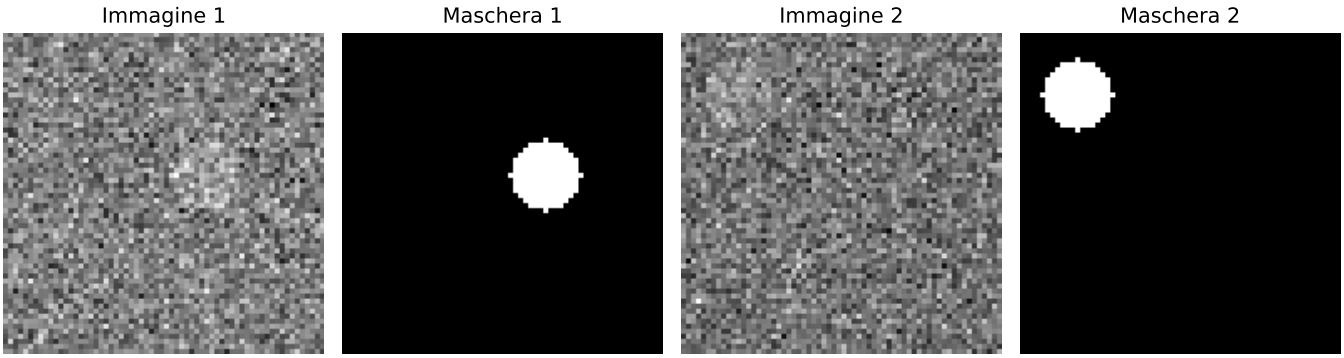


Figura 9.34: Campioni del dataset sintetico di noduli: immagine in scala di grigi sotto rumore e rispettiva maschera binaria di riferimento mostrate.

Bloco 2: Baseline Classica (Filtraggio, Otsu e Morfologia)

Prima di impiegare la U-Net, si valuta le prestazioni di un *pipeline* morfologico classico costruito con la libreria `morph`: uno **smorzatore gaussiano** (`mm.blur`), una **soglia di Otsu** (`mm.threshold`) e un'**apertura morfologica** (`mm.open`) per l'eliminazione di rumori isolati.

1. **Metrica di Intersezione su Unione (IoU):** La funzione `iou_mascaras` calcola il grado di sovrapposizione *pixel per pixel* tra la predizione e la maschera reale.

2. **Esecuzione e Confronto:** La Figura 9.35 mostra il risultato della segmentazione classica su un'immagine di test, evidenziando le limitazioni della soglia globale in condizioni di basso contrasto.

```
def iou_mascaras(predita, referencia):
    p, r = predita > 0, referencia > 0
    intersecao = np.logical_and(p, r).sum()
    uniao = np.logical_or(p, r).sum()
    return intersecao / uniao if uniao else 1.0


def segmenta_classico(imagem, elemento_estrutural):
    suavizada = mm.blur(imagem, 7)
    binaria = mm.threshold(suavizada)
    return mm.open(binaria, elemento_estrutural)


elemento_estrutural = mm.sedisk(5)
ious_classico = [
    iou_mascaras(segmenta_classico(img, elemento_estrutural), mask)
    for img, mask in zip(imgs_val, masks_val)
]
iou_classico_medio = float(np.mean(ious_classico))
print(
    f"IoU medio (linea di base classica) sulla validazione: {iou_classico_medio:.4f}\n"
)

predicao_classica_exemplo = segmenta_classico(imgs_val[0], elemento_estrutural)

mm.show(
    [imgs_val[0], masks_val[0], predicao_classica_exemplo],
    titles=["Immagine", "Maschera di Riferimento", "Predizione Classica"],
    cols=3,
    figsize=(9, 3.2),
)
```

```
IoU medio (linea di base classica) sulla validazione: 0.7516
```

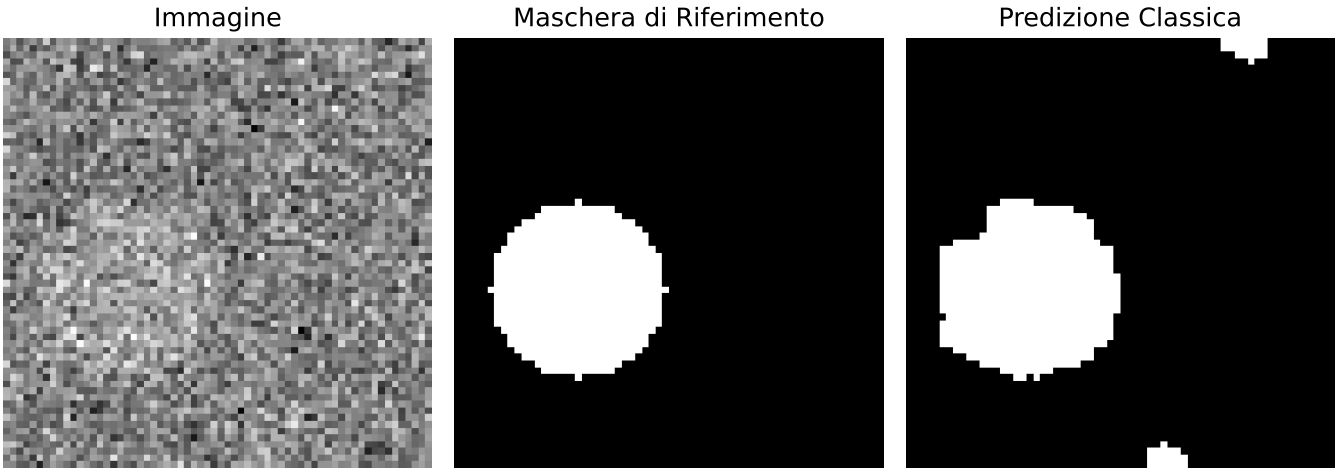


Figura 9.35: Linea di base classica di segmentazione: smoothing, sogliatura di Otsu e apertura morfologica mostrate.

Bloco 3: Costruzione dell'architettura U-Net e funzioni di perdita

La U-Net è un'architettura a forma di "U" (da cui il nome), pensata specificamente per la segmentazione delle immagini. È composta da due percorsi che lavorano insieme:

- ↓ **Codificatore (encoder):** scende attraverso l'immagine, riducendo la risoluzione spaziale a ogni passo mentre estrae caratteristiche sempre più astratte (bordi → trame → forme → contesto).
- **Decodificatore (decoder):** risale, ricostruendo la risoluzione originale attraverso convoluzioni trasposte (*upsampling*), fino a generare una maschera delle stesse dimensioni dell'immagine di input.

L'elemento che rende la U-Net speciale sono le **connessioni di salto** (*skip connections*): esse trasportano le mappe delle caratteristiche dal codificatore direttamente alla fase corrispondente del decodificatore, alla stessa risoluzione. Questo evita che i dettagli fini — come contorni e bordi — si perdano durante la compressione spaziale.

Flusso generale dell'architettura:

```
Input


Codificatore (Conv → Conv → Pool) × 3

      le connessioni di salto reintroducono mappe ad alta risoluzione

Base (collo di bottiglia)


Decodificatore (Upsample → Concat → Conv → Conv) × 3


Output 1×1 (logits)
```

Componenti principali:

1. **Blocco convoluzionale base (`BloccoConv`)** L'unità fondamentale ripetuta in tutta la rete. Applica due convoluzioni 3×3 in sequenza, ciascuna seguita da attivazione ReLU, con *padding* che preserva le dimensioni spaziali dell'input. È questo blocco che appare sia nel codificatore che nel decodificatore.

2. **Convoluzione trasposta (`nn.ConvTranspose2d`)** È l'operazione responsabile dell'*upsampling* nel decodificatore: invece di ridurre la risoluzione spaziale (come fa il `MaxPool2d` nel codificatore), la aumenta, apprendendo i pesi necessari per "annullare" la compressione e recuperare gradualmente la dimensione originale dell'immagine.

3. **Perdita combinata (BCE + Dice)** La funzione `perdita_segmentazione` somma due metriche complementari:

   - **Entropia incrociata binaria (BCE):** valuta l'accuratezza *pixel per pixel*.
   - **Coefficiente di Dice:** valuta la *sovrapposizione globale* tra la maschera prevista e quella reale.

   Insieme, esse bilanciano la precisione locale con la fedeltà della forma segmentata nel suo complesso.

```python
class BlocoConv(nn.Module):

    def __init__(self, canais_entrada, canais_saida):
        super().__init__()
        self.rede = nn.Sequential(
            nn.Conv2d(canais_entrada, canais_saida, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(canais_saida, canais_saida, kernel_size=3, padding=1),
            nn.ReLU(inplace=True),
        )
```

```
    def forward(self, x):
        return self.rede(x)


class UNetCompacta(nn.Module):

    def __init__(self, canais_entrada=1, base=8):
        super().__init__()
        self.enc1 = BlocoConv(canais_entrada, base)
        self.enc2 = BlocoConv(base, base * 2)
        self.enc3 = BlocoConv(base * 2, base * 4)
        self.pool = nn.MaxPool2d(2)

        self.fundo = BlocoConv(base * 4, base * 8)

        self.up3 = nn.ConvTranspose2d(
            base * 8, base * 4, kernel_size=2, stride=2
        )
        self.dec3 = BlocoConv(base * 8, base * 4)
        self.up2 = nn.ConvTranspose2d(
            base * 4, base * 2, kernel_size=2, stride=2
        )
        self.dec2 = BlocoConv(base * 4, base * 2)
        self.up1 = nn.ConvTranspose2d(
            base * 2, base, kernel_size=2, stride=2
        )
        self.dec1 = BlocoConv(base * 2, base)

        self.saida = nn.Conv2d(base, 1, kernel_size=1)

    def forward(self, x):
        e1 = self.enc1(x)
        e2 = self.enc2(self.pool(e1))
        e3 = self.enc3(self.pool(e2))
        f = self.fundo(self.pool(e3))

        d3 = self.dec3(torch.cat([self.up3(f), e3], dim=1))
        d2 = self.dec2(torch.cat([self.up2(d3), e2], dim=1))
        d1 = self.dec1(torch.cat([self.up1(d2), e1], dim=1))
        return self.saida(d1)


def para_tensores(imagens, mascaras):
    X = (
        torch.tensor(np.stack(imagens), dtype=torch.float32).unsqueeze(1)
        / 255.0
    )
    Y = (
        torch.tensor(np.stack(mascaras), dtype=torch.float32).unsqueeze(1)
        / 255.0
    )
    return X, Y


def perda_dice(logits, alvo, eps=1e-6):
    probs = torch.sigmoid(logits)
    intersecao = (probs * alvo).sum(dim=(1, 2, 3))
    uniao = probs.sum(dim=(1, 2, 3)) + alvo.sum(dim=(1, 2, 3))
    dice = (2 * intersecao + eps) / (uniao + eps)
    return 1 - dice.mean()
```

```
def perda_segmentacao(logits, alvo):
    return nn.functional.binary_cross_entropy_with_logits(
        logits, alvo
    ) + perda_dice(logits, alvo)
```

Blocco 4: Addestramento della U-Net e Valutazione delle Prestazioni

L'addestramento viene eseguito per 35 epoche utilizzando l'ottimizzatore *Adam*. Ad ogni epoca, si monitorano la Perdita di Addestramento e l'indice IoU medio sul set di validazione.

1. **Ciclo di Addestramento:** I parametri vengono aggiornati con *mini-batches* di 16 campioni.
2. **Evoluzione Grafica:** La Figura 9.36 mostra il grafico con il progresso della perdita e dell'IoU.

```
modelo_unet = UNetCompacta()
print(
    f"Parametri addestrabili della U-Net: {sum(p.numel() for p in modelo_unet.parameters())}"
)

X_treino_unet, Y_treino_unet = para_tensores(imgs_treino, masks_treino)
X_val_unet, Y_val_unet = para_tensores(imgs_val, masks_val)

otimizador_unet = optim.Adam(modelo_unet.parameters(), lr=1e-3)
n = X_treino_unet.size(0)
tam_lote = 16
epocas_unet = 35
historico_perda_unet, historico_iou_unet = [], []

for epoca in range(epocas_unet):
    if epoca % 5 == 0:  # Stampa ogni 5 epoche
        print(epoca + 1, "/", epocas_unet)
    modelo_unet.train()
    perm = torch.randperm(n)
    perda_epoca = 0.0

    for i in range(0, n, tam_lote):
        idx = perm[i : i + tam_lote]
        otimizador_unet.zero_grad()
        logits = modelo_unet(X_treino_unet[idx])
        perda = perda_segmentacao(logits, Y_treino_unet[idx])
        perda.backward()
        otimizador_unet.step()
        perda_epoca += perda.item() * len(idx)

    modelo_unet.eval()
    with torch.no_grad():
        predicao_val = (torch.sigmoid(modelo_unet(X_val_unet)) > 0.5).float()
        intersecao = (predicao_val * Y_val_unet).sum(dim=(1, 2, 3))
        uniao = ((predicao_val + Y_val_unet) > 0).float().sum(dim=(1, 2, 3))
        iou_epoca = (intersecao / uniao.clamp(min=1e-6)).mean().item()

    historico_perda_unet.append(perda_epoca / n)
    historico_iou_unet.append(iou_epoca)

iou_unet_final = historico_iou_unet[-1]
print(f"IoU medio finale della U-Net nella validazione: {iou_unet_final:.4f}")

# Grafico dell'evoluzione dell'allenamento
r = mm.showTrainCurves(historico_perda_unet, historico_iou_unet,
```

```
        titulo="Treinamento da U-Net - Segmentação de Nódulos Sintéticos",
        subtitulo="Perda no conjunto de treino e IoU médio no conjunto de validação \
            ao longo de 35 épocas")
```

```
Parametri addestrabili della U-Net: 120681
1 / 35
6 / 35
11 / 35
16 / 35
21 / 35
26 / 35
31 / 35
IoU medio finale della U-Net nella validazione: 0.8876
```

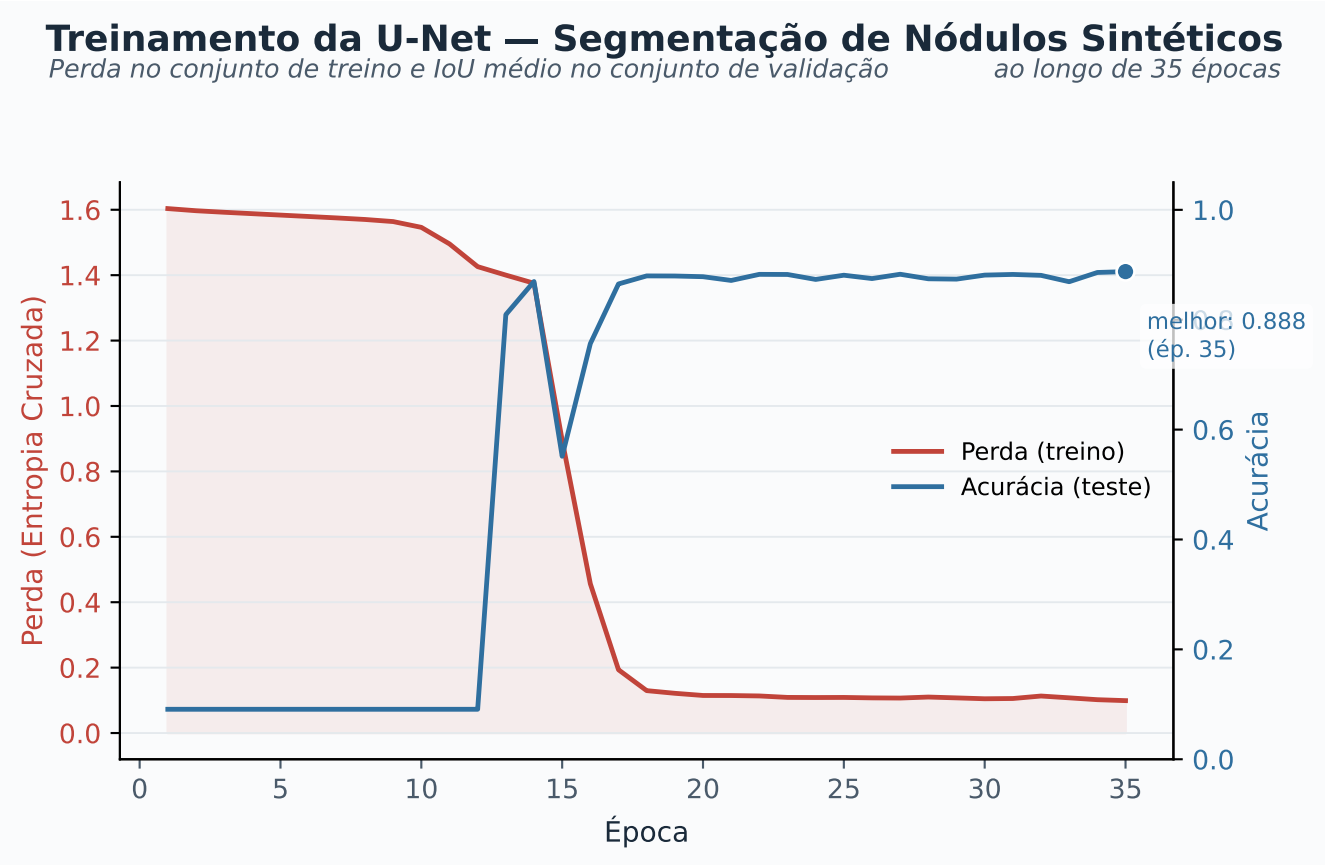


Figura 9.36: Evoluzione dell'allenamento della U-Net compatta: riduzione della perdita e aumento dell'indice IoU medio sul set di validazione lungo 35 epoche.

Blocco 5: Confronto Quantitativo e Qualitativo (Classico vs. U-Net)

Il confronto tra l'approccio classico e la U-Net rende esplicita la superiorità dell'apprendimento di rappresentazioni in scenari di basso contrasto.

1. **Pannello delle metriche:** Il grafico a barre in Figura 9.37 contrappone lo IoU medio di entrambi i metodi in validazione.
2. **Visualizzazione qualitativa:** Il confronto visivo su tre campioni dimostra come le connessioni di shortcut recuperino il contorno del nodulo anche in presenza di rumore accentuato.

```
# 1.  confronto
print(f"IoU medio (classico Smoothing + Otsu): {iou_classico_medio:.4f}")
print(f"IoU medio (U-Net): {iou_unet_final:.4f}")
```

```python
# 2. Visualizzazione qualitativa affiancata su 3 campioni tramite mm.show
imgs_comparativas = []
titulos_comparativos = []

for i in range(3):
    with torch.no_grad():
        pred_unet = (
            (torch.sigmoid(modelo_unet(X_val_unet[i : i + 1])) > 0.5)
            .float()
            .squeeze()
            .numpy()
            * 255
        )

    pred_classico = segmenta_classico(imgs_val[i], elemento_estrutural)

    imgs_comparativas.extend(
        [imgs_val[i], masks_val[i], pred_classico, pred_unet.astype(np.uint8)]
    )

    t_prefix = f"Amostra {i+1}"
    titulos_comparativos.extend(
        [
            f"{t_prefix}: Imagem",
            f"{t_prefix}: Referência",
            f"{t_prefix}: Clássico",
            f"{t_prefix}: U-Net",
        ]
    )

mm.show(
    imgs_comparativas,
    titles=titulos_comparativos,
    cols=4,
    figsize=(11, 7.5),
)
```

```
IoU medio (classico Smoothing + Otsu): 0.7516
IoU medio (U-Net): 0.8876
```

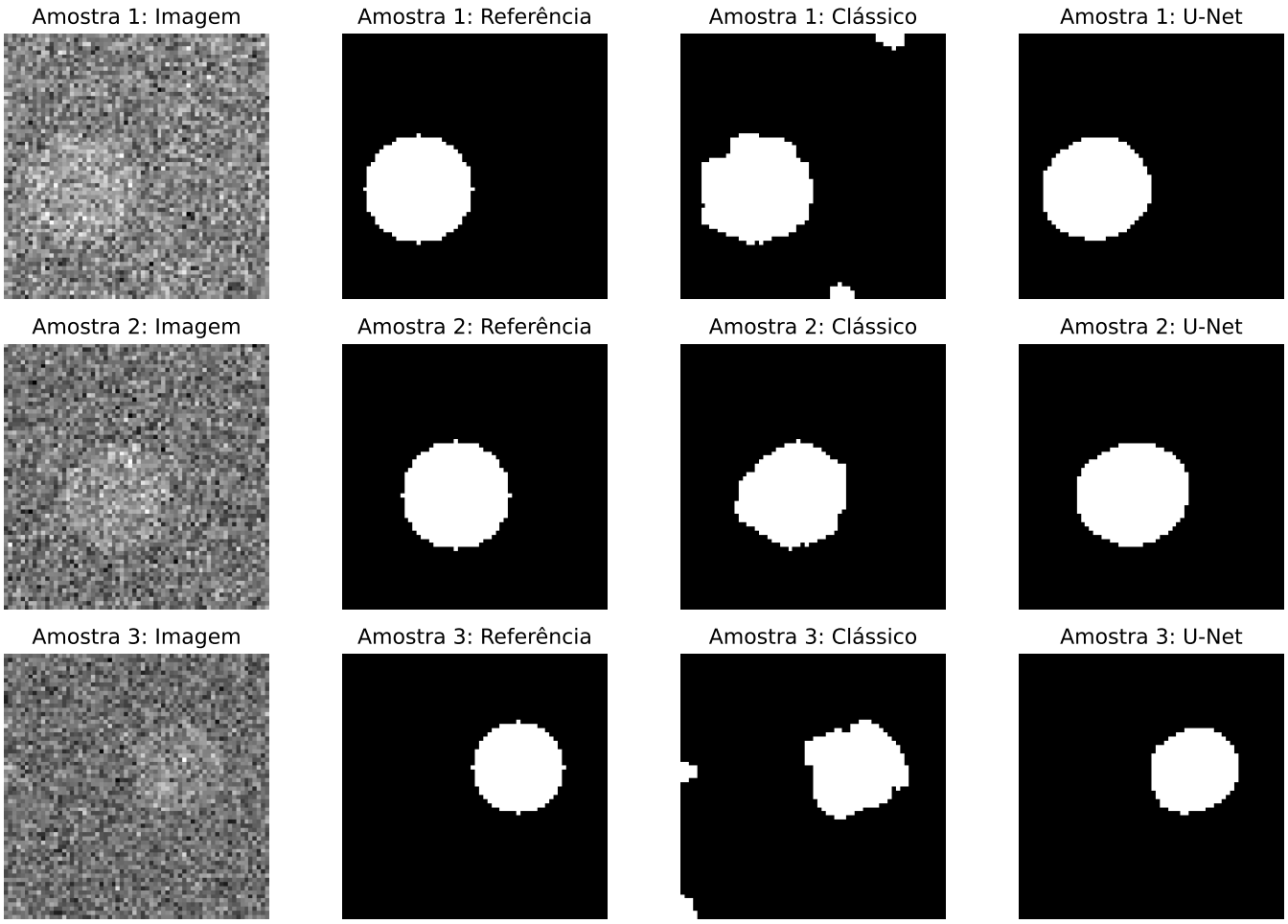


Figura 9.37: Confronto quantitativo (IoU medio) e qualitativo tra l'approccio classico e la U-Net addestrata su tre campioni di validazione.

Perché la U-Net supera la sogliatura fissa?

La sogliatura di Otsu applica un valore di taglio globale sull'intensità locale. Quando la differenza di media tra il nodulo e lo sfondo è piccola rispetto al rumore gaussiano, questa regola commette errori sistematici ai bordi.
La U-Net supera questa limitazione combinando l'ampio contesto semantico estratto dall'encoder con i dettagli spaziali fini preservati dalle connessioni di salto. Ciò consente di identificare la presenza del nodulo e di delinearne i contorni con precisione, anche in presenza di rumore intenso.

9.5.4 Ingegneria dei Dati per la Visione Artificiale (*Roboflow*)

Una U-Net può essere addestrata a partire da immagini annotate su piattaforme di ingegneria dei dati per la Visione Artificiale (VA), come *Roboflow*. Queste piattaforme consentono di organizzare set di dati, eseguire annotazioni, applicare fasi di pre-elaborazione e *data augmentation*, addestrare modelli ed esportare i dati in diversi formati. In questa sezione, tuttavia, l'esempio riprende la **rilevazione di oggetti geometrici**, utilizzando set di dati già presentati in questo libro.

Dipendenza da connessione e chiave API

Le celle di questa sezione richiedono una connessione a Internet e una chiave API gratuita di *Roboflow* (`app.roboflow.com`). Nel *Workspace* del progetto, accedi a **§ → Roboflow API** e copia la **Private API Key**.

Crea, in questa cartella, il file `chave_roboflow.txt` contenente **solo la chiave**, senza virgolette. Un modello di questo file è disponibile in `chave_roboflow.txt.exemplo`.
Aggiungi `chave_roboflow.txt` al file `.gitignore`, poiché contiene una credenziale di accesso che non deve essere versionata né condivisa.

9.5.4.1 Rilevamento di oggetti utilizzando *Roboflow*

Roboflow consente di eseguire inferenze su immagini locali, permettendo di valutare le prestazioni del modello ospitato nell'identificazione degli oggetti di interesse.

Oltre all'inferenza, il *dataset* può essere esportato nel formato `png-mask-semantic`, in cui ogni immagine è accompagnata da una maschera di segmentazione semantica. In questa maschera, ogni pixel rappresenta la classe a cui appartiene l'oggetto corrispondente. Le coppie immagine-maschera vengono utilizzate come dati di addestramento per la `UNetCompacta`.

9.5.4.2 Connessione, *download* e verifica del *dataset*

Il codice stabilisce una connessione con il *Workspace* `mctest` e il progetto `geometric-test00`, versione 6, ed esegue il *download* del *dataset* in `dados/datasetRoboFlow`. Successivamente, verifica la *shape* delle immagini in ciascuno *split*. La funzione di verifica viene riutilizzata successivamente nella sezione.

```
from roboflow import Roboflow
from pathlib import Path
from PIL import Image
from collections import Counter

def contar_shapes(raiz, splits=("train", "valid", "test")):
    """Conta la shape (altezza, larghezza, canali) delle immagini per split."""
    for split in splits:
        pasta = raiz / split / "images"
        if not pasta.exists():
            print(f"{split}: cartella non trovata")
            continue

        shapes = Counter()
        for arquivo in pasta.iterdir():
            if arquivo.is_file():
                with Image.open(arquivo) as img:
                    shapes[(img.height, img.width, len(img.getbands()))] += 1

        txt = ", ".join(f"{s}: {n}" for s, n in shapes.items())
        print(f"{split}: {txt}")


chave = Path("chave_roboflow.txt")

if not chave.exists():
    print("Chiave Roboflow non trovata: chiave_roboflow.txt")
else:
    with open(chave) as f:
        api_key = f.read().strip()

    # Progetto:
    # https://app.roboflow.com/mctest/geometric-test00/models
    # geometric-test00/6

    rf = Roboflow(api_key=api_key)
    projeto = rf.workspace("mctest").project("geometric-test00")
    versao = projeto.version(6)
```

```
    print(
        f"ID: {versao.version} | Nome: {versao.name} | "
        f"Immagini: {versao.images}"
    )

    raiz = Path("dados/datasetRoboFlow")
    versao.download("yolov8", location=str(raiz))

    print("Dataset:", raiz)
    contar_shapes(raiz)
```

9.5.4.3 Scaricare il *dataset* in modo riproducibile (alternativa)

In alternativa al *dataset* ottenuto tramite *Roboflow*, è possibile utilizzare un *dataset* reso disponibile in un repository GitHub, anch'esso organizzato negli stessi *splits* e contenente le stesse classi di oggetti. I set di dati, tuttavia, non sono identici: le immagini su GitHub hanno una risoluzione di `608×608` pixel, mentre le immagini esportate da *Roboflow* hanno `640×640` pixel.

Il codice seguente esegue il *download* del *dataset* da GitHub, nel caso in cui non sia già disponibile localmente, e riutilizza `contar_shapes` per verificare le dimensioni delle immagini in ciascuno *split*.

```
import os

if not os.path.exists("dados/dataset"):
    cmd = (
        "git clone --no-checkout --depth 1 --filter=blob:none "
        "https://github.com/fzampirolli/pdi-vc.git tmp_repo && "
        "cd tmp_repo && git sparse-checkout set all/cap09/dados/dataset "
        "&& git checkout && cd .. && mkdir -p dados && "
        "cp -r tmp_repo/all/cap09/dados/dataset dados/dataset && "
        "rm -rf tmp_repo"
    )
    !{cmd}

if not chave.exists():
    print("Chave do Roboflow não encontrada: chave_roboflow.txt")
else:
  versao_recente = projeto.versions()[-1]
  print(f"Versão mais recente: {versao_recente.version.split('/')[-1]}")

  contar_shapes(Path("dados/dataset"))
```

9.5.4.4 Confronto di un'immagine da ciascun *dataset*

I due *dataset* contengono immagini con risoluzioni diverse: `608×608` su GitHub e `640×640` su *Roboflow*. Per un confronto visivo diretto, le immagini vengono ridimensionate alla stessa dimensione prima di essere mostrate affiancate (Figura 9.38).

9.5.4.5 Inferenza con il modello addestrato

Il modello addestrato nella versione 6 viene utilizzato per eseguire l'inferenza su un'immagine di test locale. La previsione considera le soglie di confidenza e di sovrapposizione impiegate dalla soppressione non massima (*Non-Maximum Suppression*, NMS), e il risultato viene presentato nell'immagine annotata della Figura 9.39.

9.5.4.6 Valutando le predizioni con IoU e classe

Roboflow restituisce ogni riquadro con le coordinate del centro (`x`, `y`) in *pixel* e la classe prevista, mentre le etichette locali (`dati/dataset/test/labels/00001.txt`) seguono il formato YOLO, con centro e dimensioni

```python
import cv2

if not chave.exists():
    print("Chiave Roboflow non trovata: chave_roboflow.txt")
else:
    caminho1 = next((raiz / "train/images").iterdir())
    caminho2 = next((Path("dados/dataset") / "train/images").iterdir())

    img1 = mm.read(str(caminho1))
    img2 = mm.read(str(caminho2))


    # Ridimensiona entrambe alla stessa dimensione (la più piccola tra le due)
    largura = min(img1.shape[1], img2.shape[1])
    altura = min(img1.shape[0], img2.shape[0])
    img1_r = cv2.resize(img1, (largura, altura))
    img2_r = cv2.resize(img2, (largura, altura))

    mm.show(
        [img1_r, img2_r],
        title=[f"Roboflow {img1.shape}", f"GitHub {img2.shape}"],
    )
```

Figura 9.38

```python
# version.model è deprecato; usare version.models()

if not chave.exists():
    print("Chiave Roboflow non trovata: chave_roboflow.txt")
else:

    modelo = versao.models()[0]

    img_path = "dados/dataset/test/images/00001.jpg"
    pred = modelo.predict(img_path, confidence=40, overlap=30)
    resp = pred.json()  # include le caselle rilevate in resp["predictions"]


    altura, largura, _ = mm.read(img_path).shape
    print("Dimensioni dell'immagine di test:", (altura, largura))

    pred.save("resultado.jpg")  # immagine annotata


    mm.show(
        mm.read("resultado.jpg"),
        title="Risultato dell'inferenza con il modello di Roboflow",
        figsize=(6, 6)

    )
```

Figura 9.39

normalizzati nell'intervallo `[0, 1]`. Prima del confronto tramite `mm.IoU`, i riquadri devono essere convertiti nello stesso formato, con le coordinate dell'angolo superiore sinistro e le dimensioni espresse in *pixel*.

Una predizione è considerata corretta solo quando la classe prevista coincide con la classe del riquadro reale e la sua *Intersection over Union* (IoU) è maggiore o uguale alla soglia definita, adottando, in questo esempio, `0,5` (50%) come valore predefinito.

! Il `class_id` di *Roboflow* non corrisponde all'indice delle *label* locali

Nell'esportazione, *Roboflow* riordina le classi in **ordine alfabetico** nel `data.yaml`, indipendentemente dall'ordine utilizzato nel progetto originale, mantenuto nel `data.yaml` del GitHub. Così, `class_id = 0` corrisponde a `Circulo` nella risposta dell'API, mentre lo stesso indice corrisponde a `Triangulo` nelle *label* locali.
Pertanto, il confronto deve essere effettuato tramite il **nome della classe** (`p["class"]`), convertendolo successivamente nell'indice corrispondente nella lista locale. I `class_id` non devono essere confrontati direttamente.

```
# resp, largura e altura sono stati definiti nella cella precedente

# Ordine delle classi usato nelle etichette locali (dati/dataset/*/labels/*.txt)
CLASSES_LOCAIS = ["Triangulo", "Quadrado", "Pentagono", "Hexagono",
                  "Heptagono", "Circulo", "Elipse"]

def predicao_correta(pred: tuple, real: tuple, limiar: float = 0.5) -> bool:
    """Vero se stessa classe e mm.IoU(scatola_pred, scatola_reale) >= soglia."""
    classe_pred, caixa_pred = pred
    classe_real, caixa_real = real
    return classe_pred == classe_real and mm.IoU(caixa_pred, caixa_real) >= limiar

def caixa_roboflow(p: dict) -> tuple:
    """Converte la predizione di Roboflow in (classe, (x, y, w, h))."""
    caixa = (p["x"] - p["width"] / 2, p["y"] - p["height"] / 2,
             p["width"], p["height"])
    # usa il nome della classe (non il class_id!) per corrispondere all'ordine locale
    classe = CLASSES_LOCAIS.index(p["class"])
    return (classe, caixa)

def carrega_labels_yolo(caminho_txt: str, largura: int, altura: int) -> list:
    """Legge etichette YOLO (normalizzate) e le converte in (classe, (x, y, w, h))."""
    caixas = []
    with open(caminho_txt) as f:
        for linha in f:
            classe, xc, yc, w, h = map(float, linha.split())
            w_px, h_px = w * largura, h * altura
            x_px = xc * largura - w_px / 2
            y_px = yc * altura - h_px / 2
            caixas.append((int(classe), (x_px, y_px, w_px, h_px)))
    return caixas

if not chave.exists():
    print("Chiave Roboflow non trovata: chiave_roboflow.txt")
else:
    caixas_pred = [caixa_roboflow(p) for p in resp["predictions"]]
    caixas_real = carrega_labels_yolo(
        "dados/dataset/test/labels/00001.txt", largura, altura
    )

    acertos = sum(
        any(predicao_correta(cp, cr, limiar=0.5) for cr in caixas_real)
```

```
        for cp in caixas_pred
    )
    print(f"{acertos}/{len(caixas_pred)} predizioni corrette (classe + IoU >= 50%)")
```

Per visualizzare il risultato, ogni previsione viene disegnata sull'immagine: in verde quando è un successo (classe + IoU soglia) e in rosso quando è un errore, come mostra la Figura 9.40.

```
import cv2

VERDE, VERMELHO = (0, 255, 0), (255, 0, 0)

if not chave.exists():
    print("Chave do Roboflow não encontrada: chave_roboflow.txt")
else:
    img_acertos = mm.read(img_path).copy()

    if not chave.exists():
        print("Chave do Roboflow não encontrada: chave_roboflow.txt")
    else:
        for cp in caixas_pred:
            classe_pred, (x, y, w, h) = cp
            correta = any(predicao_correta(cp, cr, limiar=0.5) for cr in caixas_real)
            cor = VERDE if correta else VERMELHO

            p1, p2 = (int(x), int(y)), (int(x + w), int(y + h))
            cv2.rectangle(img_acertos, p1, p2, cor, 2)
            cv2.putText(img_acertos, str(classe_pred), (p1[0], p1[1] - 5),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, cor, 2)

    mm.show(
        img_acertos,
        title=f"{acertos}/{len(caixas_pred)} predições corretas "
            f"(classe + IoU >= 50%)",
        figsize=(6, 6)
    )
```

Figura 9.40

9.5.5 Applicazioni Geometriche e Integrate

Il capitolo si conclude integrando due pilastri della VC: la **geometria proiettiva** (studiata nei Capitoli 6 e 8) e l'**apprendimento profondo**. La combinazione di questi approcci supporta applicazioni pratiche nel mondo reale, come illustrato di seguito.

9.5.5.1 Realtà Aumentata con Marcatori e Omografia

Immagina una telecamera puntata su un tavolo dove qualcuno ha incollato un piccolo marker ArUco. A seconda dell'angolazione della telecamera, questo marker appare **ruotato, inclinato, in prospettiva** — mai perfettamente quadrato. È proprio questa distorsione che l'omografia sa "leggere" e annullare (o, nel nostro caso, replicare per una nuova immagine).

Il flusso completo di un'applicazione di RA basata su marcatori segue tre passaggi:

1. **Ambientazione:** il marker viene inserito in una scena reale, subendo una trasformazione di prospettiva (simulando l'angolazione della telecamera).
2. **Rilevamento:** l'algoritmo localizza il marker nella scena e recupera le coordinate esatte dei suoi 4 angoli con `cv2.aruco.ArucoDetector`.

3. **Sostituzione:** con l'omografia tra il marker "ideale" e il marker "rilevato", si proietta una **nuova immagine virtuale** esattamente sull'area del marker — come se si fosse trasformato in una finestra verso un altro contenuto, come mostrato in Figura 9.41.

> **Nota**
>
> **Dettaglio tecnico importante:** l'ArUco necessita di un margine bianco attorno al pattern (la "zona di silenzio") affinché il rilevatore possa distinguere il marker dallo sfondo. Per questo motivo, il codice seguente aggiunge un bordo con `cv2.copyMakeBorder` e utilizza l'interpolazione `cv2.INTER_NEAREST` quando deforma l'immagine, evitando che la rotazione sfumi i piccoli quadrati bianchi e neri e impedisca il rilevamento.

```
# 1. Generazione del marker ArUco sintetico, già con margine bianco (quiet zone)
# → questo margine è essenziale affinché il rilevatore riesca a "vedere" il marker
dic = cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_4X4_50)
aruco_bruto = cv2.aruco.generateImageMarker(dic, 7, 200)
aruco_gray = cv2.copyMakeBorder(
    aruco_bruto, 40, 40, 40, 40, cv2.BORDER_CONSTANT, value=255
)  # 200 -> 280px, con 40px di cornice bianca su ogni lato
aruco = cv2.cvtColor(aruco_gray, cv2.COLOR_GRAY2BGR)
lado = aruco.shape[0]  # 280

# 2. Scena reale di sfondo, usando un'immagine di esempio da skimage.data
fundo = cv2.cvtColor(skdata.coffee(), cv2.COLOR_RGB2BGR)
cena = cv2.resize(fundo, (640, 480))

# Angoli del marker "frontale" (src) e la sua posizione ruotata/inclinata nella scena (dst)
src = np.float32([[0, 0], [lado, 0], [lado, lado], [0, lado]])
dst = np.float32([[190, 160], [420, 70], [470, 330], [150, 370]])  # rotazione + prospettiva

# 3. Proietta il marker (con bordi nitidi) sulla scena reale
H_cena, _ = cv2.findHomography(src, dst)
mask_cena = cv2.warpPerspective(
    np.full((lado, lado), 255, np.uint8), H_cena, (640, 480),
    flags=cv2.INTER_NEAREST,
)
cena[mask_cena > 0] = cv2.warpPerspective(
    aruco, H_cena, (640, 480), flags=cv2.INTER_NEAREST
)[mask_cena > 0]

# 4. Rilevamento del marker all'interno della scena (come farebbe una telecamera)
det = cv2.aruco.ArucoDetector(dic, cv2.aruco.DetectorParameters())
corners, ids, _ = det.detectMarkers(cena)

assert ids is not None and len(corners) > 0, \
    "Marcador não detectado - confira iluminação/contraste da cena."

# 5. Immagine virtuale (un'altra immagine da skimage.data) che "sostituirà" il marker
# Nota: usiamo il quadrato INTERNO del marker (senza margine) come area di proiezione,
# quindi l'omografia dell'immagine virtuale usa 'src' originale (200x200), non il 'lato' con
bordo
src_interno = np.float32([[0, 0], [200, 0], [200, 200], [0, 200]])
virtual = cv2.resize(
    cv2.cvtColor(skdata.camera(), cv2.COLOR_RGB2BGR), (200, 200)
)

# Gli angoli rilevati corrispondono al marker CON il margine (280x280),
# quindi ricalcoliamo H_ra usando 'src' con margine, per mantenere la proiezione coerente
H_ra, _ = cv2.findHomography(src, corners[0][0])
```

```
# L'omografia rilevata viene applicata all'immagine virtuale (ridimensionata a 'lato')
virtual_grande = cv2.resize(virtual, (lado, lado))
ra = cena.copy()
mask_ra = cv2.warpPerspective(
    np.full((lado, lado), 255, np.uint8), H_ra, (640, 480), flags=cv2.INTER_NEAREST
)
ra[mask_ra > 0] = cv2.warpPerspective(
    virtual_grande, H_ra, (640, 480), flags=cv2.INTER_NEAREST
)[mask_ra > 0]

# Visualizzazione: scena con marker vs. scena con Realtà Aumentata applicata
mm.show(
    [cv2.cvtColor(cena, cv2.COLOR_BGR2RGB), cv2.cvtColor(ra, cv2.COLOR_BGR2RGB)],
    titles=["Marker nella Scena Reale (ruotato)", "Sovrapposizione Virtuale tramite
Omografia"],
    cols=2,
    figsize=(9, 4),
)
```

Figura 9.41: Realtà aumentata basata su marker ArUco: marker inserito in scena reale e ruotato, rilevato e sostituito con immagine virtuale tramite omografia.

Riepilogo del *pipeline*:

Fase	Funzione	Funzione chiave
1. Generazione	Marker ArUco con margine bianco	`generateImageMarker` + `copyMakeBorder`
2. Ambientazione	Inserisce marker distorto nella scena	`findHomography` + `warpPerspective`
3. Rilevamento	Localizza marker e restituisce angoli	`ArucoDetector.detectMarkers`
4. Sostituzione	Proietta immagine virtuale sul marker	`findHomography` + `warpPerspective`

💡 **Lezione:** è comune nella visione artificiale che il rilevatore "non trovi nulla". Chiediti sempre: **"ho fornito abbastanza contrasto e spazio?"** — vale per ArUco, codici QR e riconoscimento facciale.

9.5.5.2 Fotogrammetria e Riferimento di Scala

La **fotogrammetria** consente di stimare le dimensioni fisiche degli oggetti a partire da immagini digitali. A tal fine, si utilizza un oggetto di riferimento con dimensioni note, posizionato nella stessa scena dell'oggetto di interesse. Questa procedura stabilisce una relazione tra le distanze misurate in *pixel* e le corrispondenti dimensioni nel mondo reale.

Si consideri, ad esempio, una carta di credito, le cui dimensioni seguono lo standard internazionale ISO/IEC 7810. Poiché la sua larghezza è esattamente **8,56 cm**, è sufficiente determinare quanti *pixel* occupa tale larghezza nell'immagine per calcolare il fattore di conversione tra *pixel* e centimetri. Se la carta corrisponde a 140 *pixel*, allora ogni *pixel* rappresenterà approssimativamente **0,061 cm**. Questo stesso fattore di scala può essere applicato per stimare le dimensioni di qualsiasi altro oggetto situato nello stesso piano della scena, come illustrato in Figura 9.42..

La procedura può essere suddivisa in due fasi principali:

1. **Segmentazione e *bounding box*:** individuare nell'immagine sia l'oggetto di riferimento sia l'oggetto di interesse, utilizzando tecniche come la segmentazione per colore, la sogliatura, il rilevamento dei contorni o metodi di rilevamento degli oggetti.
2. **Conversione in dimensioni fisiche:** calcolare il rapporto cm/pixel a partire dalla larghezza nota dell'oggetto di riferimento e utilizzarlo per convertire le misure dell'oggetto di interesse da *pixel* a centimetri.

> **Nota**
>
> **Condizione per misurazioni affidabili**
> La conversione tra *pixel* e centimetri presuppone che l'oggetto di riferimento e l'oggetto di interesse si trovino approssimativamente sullo stesso piano e alla stessa distanza dalla fotocamera. In queste condizioni, la scala rimane pressoché costante in tutta l'immagine. Differenze di profondità, inclinazione della fotocamera o distorsioni dell'obiettivo possono introdurre errori nelle misure stimate.

```
COR_REFERENCIA = (200, 200, 200)  # Carta di riferimento (grigia)
COR_OBJETO = (60, 60, 220)  # Oggetto bersaglio (rosso)

# Disegno della scena sintetica
cena_medicao = np.full((300, 500, 3), 255, dtype=np.uint8)
cv2.rectangle(
    cena_medicao, (30, 200), (30 + 140, 200 + 88), COR_REFERENCIA, -1
)
cv2.rectangle(cena_medicao, (250, 100), (250 + 220, 100 + 150), COR_OBJETO, -1)


def caixa_delimitadora_por_cor(imagem_bgr, cor_bgr, tolerancia=40):
    diferenca = np.abs(imagem_bgr.astype(int) - np.array(cor_bgr)).sum(axis=2)
    mascara = (diferenca < tolerancia).astype(np.uint8) * 255
    contornos, _ = cv2.findContours(
        mascara, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )
    maior_contorno = max(contornos, key=cv2.contourArea)
    return cv2.boundingRect(maior_contorno)


x_ref, y_ref, w_ref_px, h_ref_px = caixa_delimitadora_por_cor(
    cena_medicao, COR_REFERENCIA
)
x_obj, y_obj, w_obj_px, h_obj_px = caixa_delimitadora_por_cor(
    cena_medicao, COR_OBJETO
)
```

```
# Calcolo della scala fisica
LARGURA_REFERENCIA_CM = 8.56
razao_cm_por_px = LARGURA_REFERENCIA_CM / w_ref_px
largura_obj_cm = w_obj_px * razao_cm_por_px
altura_obj_cm = h_obj_px * razao_cm_por_px

# Disegno delle scatole e delle misurazioni stimate
resultado = cena_medicao.copy()
cv2.rectangle(
    resultado, (x_ref, y_ref), (x_ref + w_ref_px, y_ref + h_ref_px), (0, 180, 0), 2
)
cv2.rectangle(
    resultado, (x_obj, y_obj), (x_obj + w_obj_px, y_obj + h_obj_px), (0, 180, 0), 2
)

cv2.putText(
    resultado,
    f"{LARGURA_REFERENCIA_CM:.2f} cm",
    (x_ref, y_ref - 8),
    cv2.FONT_HERSHEY_SIMPLEX,
    0.55,
    (0, 120, 0),
    2,
)

cv2.putText(
    resultado,
    f"{largura_obj_cm:.1f} x {altura_obj_cm:.1f} cm",
    (x_obj, y_obj - 8),
    cv2.FONT_HERSHEY_SIMPLEX,
    0.6,
    (0, 120, 0),
    2,
)

mm.show(
    [
        cv2.cvtColor(cena_medicao, cv2.COLOR_BGR2RGB),
        cv2.cvtColor(resultado, cv2.COLOR_BGR2RGB),
    ],
    titles=[
        "Scena Originale",
        "Misurazione tramite Riferimento di Scala (Centimetri)",
    ],
    cols=2,
    figsize=(9, 4),
)
```

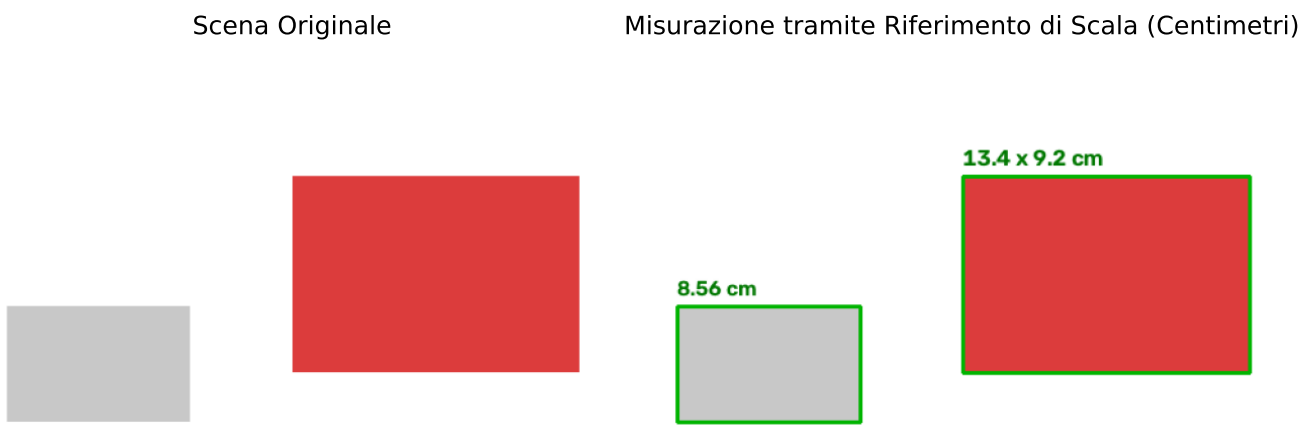


Figura 9.42: Misurazione di dimensioni fisiche reali utilizzando una carta di riferimento di scala nota (8,56 cm) renderizzata. Il rettangolo grigio rappresenta la carta di riferimento e il rettangolo blu rappresenta l'oggetto bersaglio. Le dimensioni stimate dell'oggetto sono visualizzate in centimetri.

Sintesi del *pipeline*:

Fase	Funzione	Funzione chiave
1. Scena sintetica	Disegna la carta di riferimento e l'oggetto target con colori distinti	`cv2.rectangle`
2. Segmentazione	Isola ciascun oggetto per colore e ne estrae il contorno	`cv2.findContours`
3. Bounding Box	Ottiene il riquadro delimitatore (posizione e dimensioni in pixel) di ciascun oggetto	`cv2.boundingRect`
4. Ridimensionamento	Converte i pixel in centimetri usando la larghezza nota della carta	Regola del tre: $\text{cm/pixel} = \frac{8{,}56}{w_{ref_px}}$
5. Annotazione	Disegna i riquadri e mostra le misure stimate sull'immagine	`cv2.rectangle` + `cv2.putText`

💡 **Applicazione nel mondo reale:** questa è esattamente la tecnica usata dalle *app* di *e-commerce* che stimano la dimensione di un prodotto a partire da una foto scattata accanto a una carta, da sistemi agricoli che misurano frutta su nastri trasportatori, e persino da perizie forensi che calcolano le dimensioni di tracce su scene del crimine — tutto con la stessa idea: **un righello noto all'interno della stessa foto.**

9.6 Riassunto

Questo capitolo, che chiude la Parte II del libro, ha presentato:

- **Convoluzione e *pooling* appresi:** la stessa operazione matematica di convoluzione del Capitolo 3, ma con *kernel* trattati come parametri regolati mediante addestramento, anziché definiti manualmente;
- **Condivisione dei pesi e gerarchia delle caratteristiche** come proprietà che rendono le CNN efficienti e capaci di apprendere rappresentazioni sempre più astratte in livelli successivi;
- **Addestramento di una CNN da zero**, con prestazioni paragonabili — e non necessariamente superiori — ai classificatori classici del Capitolo 7 su una base piccola e semplice, a sottolineare che la scelta del metodo deve essere proporzionale alla complessità reale del problema;

- **Transfer learning**, dimostrato sperimentalmente come una strategia efficace per compiti con pochi dati etichettati, riutilizzando un estrattore di caratteristiche già addestrato su un compito correlato, e le sue limitazioni, evidenziate dal transfer negativo tra domini molto diversi;
- **Applicazioni su larga scala** con modelli pre-addestrati di classificazione, rilevamento (*Faster R-CNN*) e segmentazione (*DeepLabV3*), seguendo lo stesso principio di transfer learning su scala industriale;
- **Transfer learning applicato al rilevamento di oggetti**, adattando un YOLO pre-addestrato su COCO per localizzare e classificare oggetti geometrici sintetici, illustrando lo stesso principio di congelamento parziale in un dominio di origine e destinazione ancora più distanti tra loro;
- **Segmentazione semantica con U-Net**, implementata e addestrata da zero su un insieme sintetico a basso contrasto, superando una linea di base classica di sogliatura grazie alle connessioni di skip tra codificatore e decodificatore;
- **Ingegneria dei dati per la Visione Artificiale con *Roboflow***, utilizzando un *dataset* e un modello pre-addestrato per eseguire inferenza nel rilevamento di oggetti geometrici, oltre a confrontare insiemi di dati con diverse risoluzioni, ma con le stesse classi di oggetti;
- L'integrazione di **geometria computazionale** (omografia e calibrazione) e **deep learning** in due applicazioni reali che chiudono il libro: **realtà aumentata** e **fotogrammetria**.

9.7 🤖 Uso del Gemini Notebook come Tutor Complementare

In questa edizione, l'uso del **Gemini Notebook** è incoraggiato come strumento complementare di apprendimento. Basato sull'intelligenza artificiale, il sistema utilizza esclusivamente i documenti forniti dall'autore come fonte di conoscenza, producendo risposte allineate ai contenuti e all'approccio adottato nel corso di questo capitolo.

🎓 Studia con il Tutor Intelligente

🚀 ACCEDI AL GEMINI NOTEBOOK: CAPITOLO 09

🌐 **Lingua e Linguaggio di Programmazione**

Il progetto di questo capitolo nel Gemini Notebook è stato realizzato esclusivamente con il testo in **portoghese** e gli esempi di codice in **Python**. Se stai studiando utilizzando l'edizione in inglese o francese, o seguendo il percorso in C++, le risposte del tutor potrebbero non corrispondere esattamente alla versione che stai leggendo.

⚠️ **Avvertenza sul Contenuto Generato dall'IA**

Sebbene sia uno strumento prezioso di supporto allo studio, il Gemini Notebook può talvolta produrre risposte incomplete, imprecise o errate. Si consiglia di validare le informazioni consultando il materiale del capitolo, libri, articoli scientifici e altre fonti accademiche affidabili. Quando possibile, esegui e sperimenta gli esempi pratici presentati nel corso del testo per consolidare la comprensione dei concetti.

9.8 Elenco di Esercizi

Gli esercizi seguenti consolidano i concetti presentati in questo capitolo tramite adattamenti, esperimenti ed estensioni degli algoritmi sviluppati nel testo, utilizzando le librerie `PyTorch`, `ultralytics` e la libreria didattica `morph`.

1. **(10%)** Indagare l'impatto della profondità in un'architettura convoluzionale. Partendo dalla rete a due strati del Progetto Pratico 1, aggiungere un terzo strato convoluzionale con 32 filtri prima degli strati completamente connessi. Addestrare la nuova architettura mantenendo lo stesso numero di epoche e la

stessa suddivisione dei dati. Confrontare l'accuratezza sul set di test e il numero totale di parametri addestrabili rispetto alla rete originale, discutendo se l'aumento di profondità abbia apportato un beneficio misurabile per immagini di dimensione 8×8.

2. **(15%)** Valutare la soglia di dati necessari nel dominio di destinazione affinché l'addestramento di una CNN da zero diventi competitivo con il **transfer learning**. Variando il numero di campioni di addestramento disponibili nel dominio B tra $\{5, 10, 20, 40, 80\}$, misurare l'accuratezza di test per entrambe le strategie. Presentare i risultati in un grafico a linee e determinare a partire da quale volume di dati l'addestramento da zero raggiunge prestazioni equivalenti all'estrattore pre-addestrato.

3. **(15%)** Indagare la strategia di **fine-tuning parziale** rispetto al congelamento totale dei pesi. Nello scenario di transfer learning tra domini di cifre, scongelare il secondo strato convoluzionale (`conv2`) dell'estrattore affinché venga aggiornato insieme alla testa di classificazione durante l'addestramento nel dominio B. Confrontare l'accuratezza ottenuta con il congelamento totale e con l'addestramento da zero, discutendo il compromesso tra capacità di adattamento e rischio di overfitting.

4. **(20%)** Valutare l'influenza della profondità del congelamento (*freeze*) sulle prestazioni dei rilevatori **YOLO** sottoposti a transfer learning. Utilizzando il set di dati sintetico di forme geometriche, eseguire il fine-tuning variando il parametro di congelamento del *backbone* per $\{0, 5, 10, 15\}$. Registrare la metrica mAP_{50} sul set di validazione per ogni configurazione, presentare i dati in una tabella e discutere se il congelamento parziale sia vantaggioso quando i domini di origine (COCO) e di destinazione (forme geometriche) sono significativamente distinti.

5. **(20%)** Studiare l'importanza delle **connessioni di salto** (*skip connections*) nell'architettura **U-Net** per la segmentazione semantica. Implementare una variazione `UNetSenzaSalti` che funzioni come un *autoencoder* convoluzionale tradizionale, rimuovendo le concatenazioni tra gli stadi dell'encoder e del decoder. Addestrare entrambi i modelli sulla stessa base di noduli sintetici, confrontare l'IoU medio sul set di validazione e presentare visivamente la differenza nella precisione dei bordi segmentati da ciascun metodo.

6. **(20%) — Sfida: segmentazione semantica con *Roboflow*.** Utilizzare un progetto *Roboflow* di tipo *instance segmentation*, contenente le sette classi di forme geometriche utilizzate in questo capitolo. Esportare il *dataset* nel formato `coco-segmentation` e sviluppare una procedura per convertire i poligoni memorizzati nei file `_annotations.coco.json` in maschere semantiche multiclasse, in cui ogni pixel riceve l'indice della classe corrispondente e il valore `0` rappresenta lo sfondo. Utilizzare le immagini e le maschere risultanti per addestrare la `UNetCompatta`. Valutare l'IoU medio sul set di test e confrontare visivamente le maschere previste con le annotazioni originali. Discutere le principali difficoltà incontrate nella conversione delle annotazioni COCO in maschere e gli effetti di oggetti sovrapposti o appartenenti a classi diverse.

7. **(Bonus – 10%)** Sviluppare un sistema interattivo che combini **rilevamento di oggetti (YOLO)** con **misurazione tramite riferimento di scala (fotogrammetria)**. Addestrare il rilevatore per identificare due classi in una scena: una "Carta di Riferimento" (dimensione nota di 8,56 cm $\times$ 5,39 cm) e un "Oggetto Target". Durante l'inferenza su una nuova immagine, utilizzare la dimensione in pixel della *bounding box* della carta rilevata per convertire le dimensioni della scatola dell'oggetto target in centimetri. Visualizzare l'immagine processata con le etichette di classe, la probabilità di confidenza e le dimensioni fisiche stimate sovrapposte.

8. **(Bonus – 10%)** Sviluppare un sistema di **stima della profondità tramite visione stereo** a partire da due immagini della stessa scena acquisite da posizioni diverse, simulando una coppia di telecamere stereo. Considerare nota la distanza tra le due posizioni di acquisizione (*baseline*).

   Utilizzare uno dei metodi di **rilevamento di oggetti** presentati nel capitolo, come YOLO, per localizzare gli oggetti di interesse nelle due immagini. Per ogni rilevamento, stabilire la corrispondenza tra lo stesso oggetto nelle due posizioni e determinare la sua **disparità**. A partire dalla disparità, dalla *baseline* e dai parametri della telecamera, utilizzare la geometria stereo per stimare la distanza di ciascun oggetto rispetto alle telecamere.

Come estensione della libreria didattica `morph`, modificare il metodo `showBoundBox` affinché, oltre alla **classe** e alla **confidenza del rilevamento**, presenti su ogni *bounding box* la **distanza stimata dell'oggetto**. Il risultato deve consentire di visualizzare, direttamente nelle immagini, la classe, l'accuratezza (confidenza) e la profondità di ciascun oggetto rilevato.

Presentare le due immagini con i rilevamenti, le corrispondenze tra gli oggetti, l'immagine di disparità e una rappresentazione della profondità stimata. Discutere come la distanza tra le telecamere, la precisione del rilevamento e della corrispondenza, la risoluzione delle immagini e la posizione dell'oggetto nella scena influenzino la qualità della stima.

Per la validazione, utilizzare almeno un oggetto la cui distanza dalla telecamera sia nota. Confrontare la profondità stimata con il valore reale e riportare l'**errore assoluto** e l'**errore relativo**. Discutere inoltre le limitazioni del metodo quando un oggetto non viene rilevato correttamente in entrambe le immagini o quando la corrispondenza tra le regioni osservate è ambigua.

9.9 Chiusura della Parte II

Questo capitolo conclude la Parte II del libro e chiude la sequenza di contenuti iniziata nel Capitolo 6, dedicata alla rappresentazione, rilevazione, descrizione e corrispondenza delle caratteristiche nelle immagini. Nel corso di questi capitoli, sono stati presentati metodi classici di CV basati su **caratteristiche progettate manualmente**, come Sobel, LBP, HOG, ORB e Haar Cascade, nonché metodi fondati su **caratteristiche apprese automaticamente**, rappresentati dalle CNN.

Gli esempi e gli esperimenti sviluppati evidenziano che nessuno di questi approcci è universalmente superiore. La scelta della tecnica più adeguata dipende dalle caratteristiche del problema, dalla disponibilità di dati per l'addestramento, dai requisiti di precisione e dai vincoli computazionali dell'applicazione. In problemi ben strutturati e con pochi dati, i descrittori classici offrono spesso soluzioni semplici ed efficienti. Al contrario, compiti più complessi tendono a trarre beneficio dalla capacità di apprendimento offerta dalle CNN.

Diverse direzioni di studio possono approfondire i concetti presentati in questa parte del libro, tra le quali si distinguono:

- **Architetture moderne di CNN**, come ResNet, EfficientNet e *Vision Transformers*, che ampliano la capacità di rappresentazione e le prestazioni in compiti di classificazione e riconoscimento visivo;
- **Rilevazione e segmentazione di oggetti**, con particolare attenzione alla famiglia YOLO e ai modelli di segmentazione basati su *prompt*, come *Segment Anything*;
- **Ricostruzione tridimensionale e SLAM** (*Simultaneous Localization and Mapping*), che utilizzano più immagini per stimare la geometria della scena e la traiettoria di telecamere in movimento;
- **Modelli generativi di immagini**, come le reti generative avversarie (GAN) e i modelli di diffusione, capaci di sintetizzare immagini realistiche a partire da esempi o descrizioni testuali.

Le basi sviluppate nel corso della Parte II costituiscono il fondamento per queste e altre aree avanzate della CV, nelle quali la rappresentazione adeguata delle informazioni visive rimane l'elemento centrale per l'analisi e la comprensione delle immagini.

Riferimenti del Capitolo

I concetti e gli algoritmi presentati in questo capitolo sono stati fondati su riferimenti classici e contemporanei della letteratura sul Deep Learning applicato alla VC:

- Mcculloch (1943), per la proposta della prima astrazione matematica e logica del neurone artificiale, che getta le basi concettuali dell'elaborazione neurale computazionale.
- Rosenblatt (1958), per la formulazione originale del **Perceptron**, modello precursore del neurone artificiale utilizzato nelle architetture moderne di deep learning.
- Goodfellow (2016) e Lecun (2015), per i fondamenti delle reti neurali, della convoluzione, delle funzioni di attivazione e dell'addestramento di modelli profondi.

- Bishop (2006), per i concetti rigorosi di riconoscimento di pattern, probabilità, stima di massima verosimiglianza e metodi statistici applicati al machine learning.
- Ronneberger (2015) per l'architettura **U-Net**, utilizzata nella segmentazione semantica con connessioni di skip tra codificatore e decodificatore.
- Redmon (2016), per l'architettura **YOLO** (*You Only Look Once*), utilizzata negli esperimenti di rilevamento degli oggetti con transfer learning.
- Ren (2015), per l'architettura **Faster R-CNN**, impiegata come modello pre-addestrato per il rilevamento degli oggetti.
- He (2016), per l'architettura **ResNet**, base di diversi estrattori di caratteristiche pre-addestrati utilizzati in questo capitolo.
- Chen (2018), per l'architettura **DeepLabV3**, utilizzata come modello pre-addestrato per la segmentazione semantica.
- Kirillov (2023), per il modello **Segment Anything (SAM)**, menzionato come direzione di studio per la segmentazione promptable.
- {google} (2025), relativo allo strumento **Gemini Notebook**, utilizzato nell'elaborazione dell'infografica di sintesi del capitolo e reso disponibile come supporto complementare allo studio.

Executar Colab Abrir GitHub

9.10 💻 Parte Pratica con Esercizi di Programmazione

La presente lista di **Esercizi di Programmazione (EP)** consolida le formulazioni teoriche presentate nel Capitolo 9 — Deep Learning per la Visione Artificiale — tramite un percorso pratico applicato. Diversamente dall'addestramento di reti neurali complete con PyTorch, che richiede tempi di esecuzione e, talvolta, GPU, gli EP di questo capitolo isolano le **grandezze intermedie** di una *pipeline* reale di deep learning — l'output di un singolo strato convoluzionale, il risultato di un'operazione di *pooling*, il conteggio dei parametri addestrabili di un'architettura, la sovrapposizione tra bounding box candidate, la qualità di una maschera di segmentazione e il filtro di soppressione non-massima — consentendo di validare manualmente ogni fase del ragionamento senza dipendere da librerie di machine learning né da un addestramento reale.

L'incatenamento degli esercizi riproduce il flusso concettuale del capitolo e cresce in difficoltà a ogni passo: si inizia con il calcolo manuale dell'output di uno **strato convoluzionale addestrato** (🟢), a partire da un *kernel* e un bias già addestrati; si prosegue con l'operazione di ***pooling*** (🟢, massimo e media), che riduce la risoluzione spaziale tra blocchi convoluzionali; si continua con il **conteggio dei parametri addestrabili** (🟡) di un'architettura CNN completa, evidenziando perché la condivisione dei pesi rende queste reti così più economiche rispetto a uno strato completamente connesso equivalente; si approfondisce il calcolo dell'**Intersezione su Unione (IoU)** e della **Soppressione Non-Massima (NMS)** (🟡), fase di post-elaborazione comune a rilevatori come Faster R-CNN e YOLO; si passa alla **valutazione delle maschere di segmentazione** (🟠) con le stesse metriche IoU e Dice utilizzate per confrontare U-Net con la baseline morfologica classica; e si conclude con una ***pipeline* integrata** (🔴), unendo l'output di un rilevatore di oggetti (dopo NMS) a una misurazione del mondo reale tramite riferimento di scala — lo stesso principio della fotogrammetria studiato nell'integrazione finale del capitolo.

Ogni volta che ha senso, ciascun esercizio indica i metodi della libreria didattica `morph.py` (la stessa utilizzata nel capitolo, importata come `mm`) che risolvono una fase del problema o che servono da riferimento per verificare i propri calcoli — senza, tuttavia, sostituire il ragionamento che devi implementare.

! Linee Guida per la Risoluzione degli Esercizi di Programmazione

In tutti gli esercizi di questo capitolo, le fasi di discretizzazione o arrotondamento numerico devono impiegare l'arrotondamento standard all'intero più vicino (*round half away from zero*), mitigando le ambiguità in valori con frazione esattamente uguale a 0,5. Salvo indicazione esplicita contraria:

(i) l'operazione di "convoluzione" segue la convenzione adottata dai *framework* di deep learning — **correlazione incrociata**, senza inversione spaziale del *kernel*, esattamente come presentato nella Sezione "Strato Convoluzionale"; (ii) il riempimento (*padding*) è effettuato con zeri; (iii) le bounding box sono specificate nel formato angolo-a-angolo (x_1, y_1, x_2, y_2), con $x_1 < x_2$ e $y_1 < y_2$; e (iv) vettori/matrici seguono l'indicizzazione a partire da 0, con la convenzione `[riga][colonna]` per strutture bidimensionali.

🎯 Obiettivo di questo Quaderno

Il quaderno consente di sviluppare, validare, organizzare e testare soluzioni di **Esercizi di Programmazione (EPs)** in ambienti interattivi, come Colab, con gli stessi casi di test di Moodle, copiandoli lì solo al momento di registrare il voto ufficiale.

Download

Scarica `morph.py` e `testsuite.py` eseguendo la cella sottostante:

```
import os, urllib.request

url = "https://raw.githubusercontent.com/fzampirolli/pdi-vc/master/morph/config.py"
if not os.path.exists("config.py"):
    urllib.request.urlretrieve(url, "config.py")

import config
config.setup(testsuite=True)
from morph import mm
from testsuite import TestSuite
```

```
Ambiente pronto. Morph: 1.1.9 | OpenCV: 5.0.0 | TestSuite: 1.1.2
```

Esecuzione dei Test

Per valutare i test, esegui `TestSuite("EP09_01.extensão").run()` in una nuova cella, sostituendo l'estensione con quella del linguaggio utilizzato (`.py`, `.java`, `.c`, `.cpp`, `.js` o `.r`). Il sistema scarica i casi di test da GitHub, esegue il programma e calcola automaticamente il voto.

Per testare direttamente il codice Python, senza salvare il file, usa `run_code(codigo)` passando il codice come *stringa* in una variabile `codigo`:

```
codigo = """
# ... il tuo codice qui ...
"""
TestSuite("EP09_01").run_code(codigo)
```

9.10.1 EP09_01 🟢 Convoluzione 2D Manuale (*Forward* di un Livello Appreso)

Il PyTorch, presentato in questo capitolo, esegue `nn.Conv2d(x)` in un'unica chiamata — ma dietro di essa c'è solo la correlazione incrociata tra un *kernel* (già addestrato) e un intorno dell'input, seguita dalla somma di un bias e di un'attivazione, esattamente come formalizzato nella Sezione "Livello Convoluzionale". La differenza essenziale rispetto alla convoluzione con *kernel* fissi del Capitolo 3 è che, qui, i valori del *kernel* e del bias **sono già pronti** (come se fossero stati appresi per gradiente), e spetta a te riprodurre manualmente il passaggio diretto (*forward pass*) che il *framework* esegue internamente.

Prima di addestrare una vera CNN, ti è stato affidato il compito di implementare questo passaggio diretto da zero, per un singolo livello convoluzionale con un singolo canale di input e un singolo filtro di output, incluso il supporto a *padding* e *stride* arbitrari.

9.10.1.1 📋 Linee Guida di Implementazione

1. **Input:** Leggere le dimensioni $H \times W$ della mappa delle caratteristiche di input e, successivamente, i suoi $H \times W$ valori reali.
2. ***Kernel* e bias:** Leggere le dimensioni $k_h \times k_w$ del *kernel* (già addestrato), i suoi valori reali, e il bias b (reale, scalare).
3. **Iperparametri:** Leggere il *padding* p (intero, numero di zeri aggiunti su ciascun bordo) e lo *stride* s (intero, passo dello scorrimento).
4. **Riempimento:** Aggiungere p zeri su ciascuno dei quattro bordi della mappa di input prima della correlazione.
5. **Correlazione incrociata:** Per ogni posizione di output (i, j), calcolare

$$z(i,j) = b + \sum_{u=0}^{k_h-1} \sum_{v=0}^{k_w-1} K(u,v) \cdot X_{pad}(i \cdot s + u,\ j \cdot s + v),$$

   scorrendo l'input **senza** invertire il *kernel* (convenzione dei *framework* di deep learning, diversa dalla convoluzione matematica classica).
6. **Attivazione:** Applicare ReLU a ogni valore: $a(i,j) = \max(0, z(i,j))$.
7. **Dimensioni di output:** $O_h = \lfloor (H + 2p - k_h)/s \rfloor + 1$ e $O_w = \lfloor (W + 2p - k_w)/s \rfloor + 1$.
8. **Output:** Stampare O_h e O_w nella prima riga, seguiti da O_h righe con O_w valori reali ciascuna (la mappa delle caratteristiche di output, già con ReLU applicata), formattati con 4 cifre decimali.

9.10.1.2 📌 Vincoli Computazionali

- **Un canale di input, un filtro di output:** non è necessario gestire più canali o più filtri in questa versione semplificata.
- **Senza inversione del *kernel*:** implementare la correlazione incrociata, non la convoluzione matematica classica con *kernel* invertito — è questa l'operazione che PyTorch (e la maggior parte dei *framework*) chiama "convoluzione".
- **Riempimento con zeri:** i p pixel aggiunti su ciascun bordo valgono sempre 0.
- **Formattazione:** tutti i valori di output devono avere esattamente 4 cifre decimali, anche quando il valore è un intero (es.: `2.0000`).

9.10.1.3 🧠 Fondamenti Teorici

Elemento	Ruolo nel livello convoluzionale
Kernel K	Parametri appresi per gradiente, analoghi ai coefficienti di un filtro fisso del Capitolo 3, ma regolati tramite backpropagation
Bias b	Offset appreso, sommato dopo la correlazione — consente al neurone di "attivarsi" anche con input nullo
Padding	Controlla la dimensione spaziale dell'output e previene la perdita di informazioni ai bordi a ogni livello
Stride	Controlla il passo dello scorrimento; valori > 1 riducono la risoluzione spaziale, come una forma di sottocampionamento integrato nella convoluzione stessa

Elemento	Ruolo nel livello convoluzionale
ReLU	Introduce non linearità dopo la combinazione lineare, esattamente come nella Sezione "Funzione di Attivazione"

9.10.1.4 🧩 Metodi di `morph.py` che possono aiutare

- `mm.readImg(h, w, dtype='float')` — legge direttamente una matrice $h \times w$ di valori reali dall'input standard, evitando il *parsing* manuale della mappa delle caratteristiche e del *kernel*.
- `mm.correlacao0(f, kernel, bias)` — implementa la stessa somma di correlazione incrociata + bias che dovrai calcolare a mano, ma **senza** supporto per *padding* o *stride*, e converte il risultato in `uint8` (tronca valori negativi e decimali). Può servire come riferimento concettuale o per verificare il caso più semplice ($p = 0$, $s = 1$), ma non sostituisce la tua implementazione completa — che deve preservare segno, cifre decimali, *padding*, *stride* e ReLU.

9.10.1.5 📦 Specifica di Input e Output (VPL)

Input:

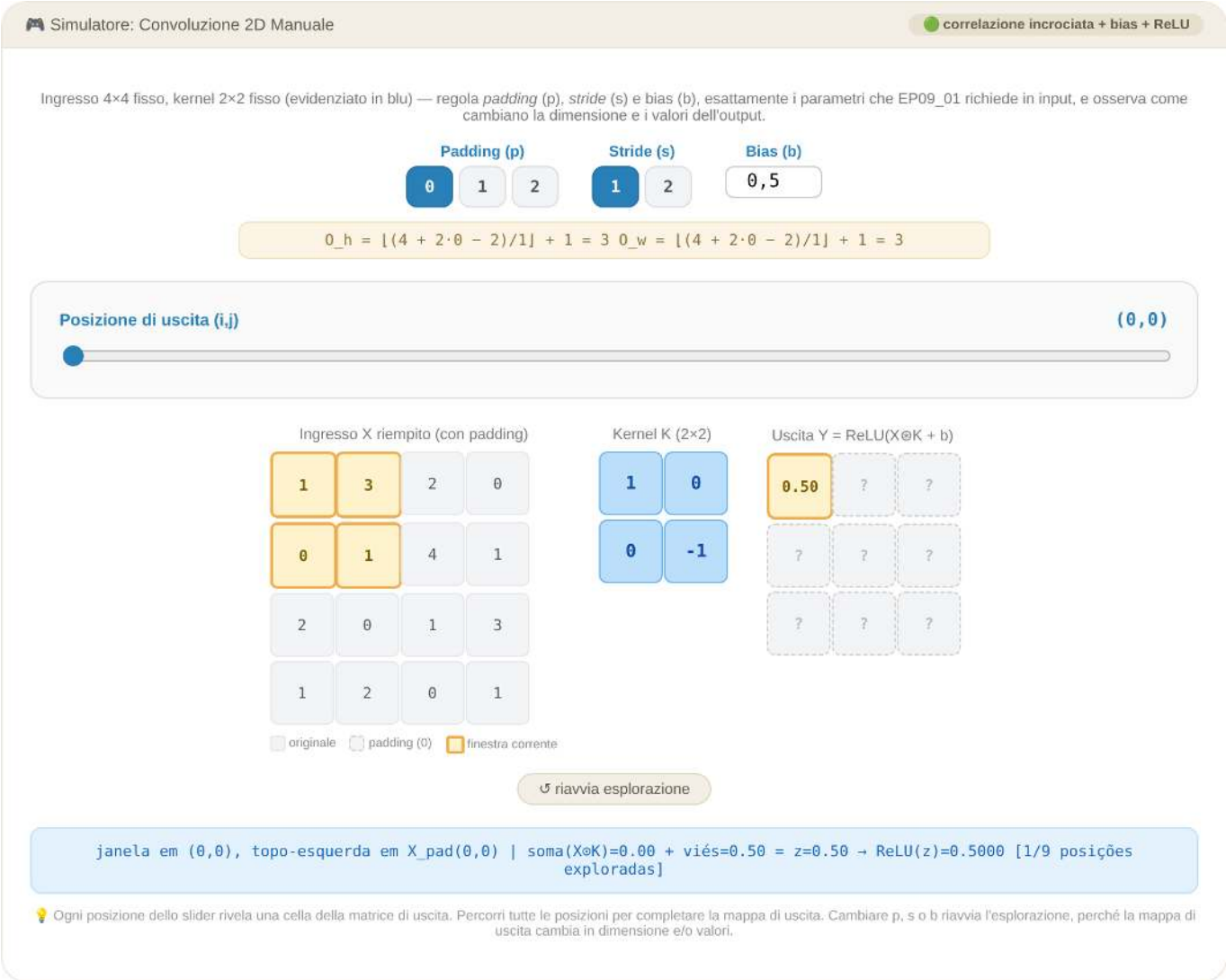
- Riga 1: Interi H e W.
- Prossime H righe: W valori reali ciascuna (mappa di input).
- Riga successiva: Interi k_h e k_w.
- Prossime k_h righe: k_w valori reali ciascuna (*kernel*).
- Riga successiva: Reale b (bias).
- Riga successiva: Interi p e s.

Output:

- Riga 1: Interi O_h e O_w.
- Prossime O_h righe: O_w valori reali ciascuna, con 4 cifre decimali.

9.10.1.6 📌 Esempi

Input	Output	Osservazione
3 3 1 2 0 0 1 2 1 0 1 2 2 1 1 1 1 -2 0 1	2 2 2.0000 3.0000 0.0000 2.0000	*Padding* 0, *stride* 1: output 2×2 senza riempimento.
3 3 1 2 0 0 1 2 1 0 1 2 2 1 0 0 1 0 1 2	2 2 1.0000 0.0000 1.0000 2.0000	*Padding* 1, *stride* 2: input riempito con zeri prima della correlazione.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-ep0901.html

Figura 9.43: Simulatore EP09_01: Convoluzione 2D Manuale (correlazione incrociata + bias + ReLU, con *padding* e *stride* regolabili)

```
%%writefile EP09_01.py
# Codice Python
```

```
Overwriting EP09_01.py
```

```
TestSuite("EP09_01.py").run()
```

```
<IPython.core.display.HTML object>
```

9.10.2 EP09_02 🟢 *Pooling* Manuale (Massimo e Media)

Tra i blocchi convoluzionali, l'architettura tipica di una CNN intercala livelli di ***pooling***, che riducono la risoluzione spaziale della mappa delle caratteristiche senza introdurre nuovi parametri addestrabili — a differenza della convoluzione, il *pooling* non ha pesi: si limita a riassumere ogni finestra dell'ingresso in un singolo valore, tramite un massimo o una media, esattamente come formalizzato nella Sezione "*Pooling*".

Sei stato incaricato di implementare questa operazione a partire da una finestra scorrevole quadrata, senza sovrapposizione parziale sui bordi (solo finestre complete), supportando i due tipi più comuni: `max` (preserva il valore più saliente, tipicamente usato per mantenere bordi e texture forti) e `avg` (smussa la regione, preservando l'informazione di intensità media).

9.10.2.1 📋 Linee Guida di Implementazione

1. **Ingresso:** Leggere le dimensioni $H \times W$ della mappa delle caratteristiche di ingresso e i suoi $H \times W$ valori reali.
2. **Finestra:** Leggere gli interi k (dimensione della finestra quadrata $k \times k$) e s (*stride*).
3. **Tipo:** Leggere una *stringa*, `max` o `avg`, che indica il tipo di *pooling*.
4. **Senza riempimento:** Questa operazione **non** utilizza *padding*; le finestre che supererebbero il bordo dell'ingresso vengono scartate.
5. **Calcolo:** Per ogni posizione di uscita (i, j), calcolare il massimo o la media dei $k \times k$ valori della finestra corrispondente, iniziando da $(i \cdot s,\ j \cdot s)$.
6. **Dimensioni di uscita:** $O_h = \lfloor (H-k)/s \rfloor + 1$ e $O_w = \lfloor (W-k)/s \rfloor + 1$.
7. **Uscita:** Stampare O_h e O_w nella prima riga, seguiti da O_h righe con O_w valori reali ciascuna, formattati con 4 cifre decimali.

9.10.2.2 📌 Vincoli Computazionali

- **Finestra quadrata:** $k \times k$, senza supporto per finestre rettangolari in questa versione.
- **Senza *padding*:** solo le finestre interamente contenute nell'ingresso sono considerate — le dimensioni che "avanzeranno" sono semplicemente scartate.
- **`avg` usa divisione reale:** la media è sempre somma$/k^2$, anche quando il risultato ha molte cifre decimali — arrotondare solo nella formattazione finale, secondo la linea guida generale del capitolo.
- **Formattazione:** tutti i valori di uscita con esattamente 4 cifre decimali.

9.10.2.3 🧠 Fondamento Teorico

Elemento	Ruolo nell'architettura
Pooling massimo	Preserva l'attivazione più forte della finestra; comune dopo livelli convoluzionali per mantenere bordi e texture salienti
Pooling medio	Smussa la regione, preservando l'intensità media; comune nei livelli finali (*global average pooling*)

Elemento	Ruolo nell'architettura
Assenza di parametri	Differenzia il *pooling* dalla convoluzione: riduce la risoluzione spaziale senza costi aggiuntivi di addestramento
Riduzione della risoluzione	Contribuisce all'invarianza rispetto a piccole traslazioni e alla riduzione del costo computazionale dei livelli successivi

9.10.2.4 🧩 Metodi di `morph.py` che possono aiutare

Il `morph.py` non implementa il *pooling* con sottocampionamento direttamente, ma due famiglie di operazioni mostrano la stessa idea sotto un'altra ottica, utile per verificare la tua intuizione:

- `mm.dil(f, Bc)` / `mm.dil0(f, B)` — dilatazione morfologica: sostituisce ogni pixel con il **massimo** del suo intorno definito dall'elemento strutturante B (es.: `mm.sebox(n)` per una finestra $(2n+1)\times(2n+1)$). È concettualmente un "*max-pooling* senza sottocampionamento" (produce un'immagine della stessa dimensione, invece che ridotta).
- `mm.blur(f, N)` — smussatura per media in una finestra $N \times N$, analoga all'*avg-pooling*, anch'essa senza riduzione della risoluzione.
- `mm.readImg(h, w, dtype='float')` — utile per leggere la mappa di ingresso in virgola mobile.

9.10.2.5 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

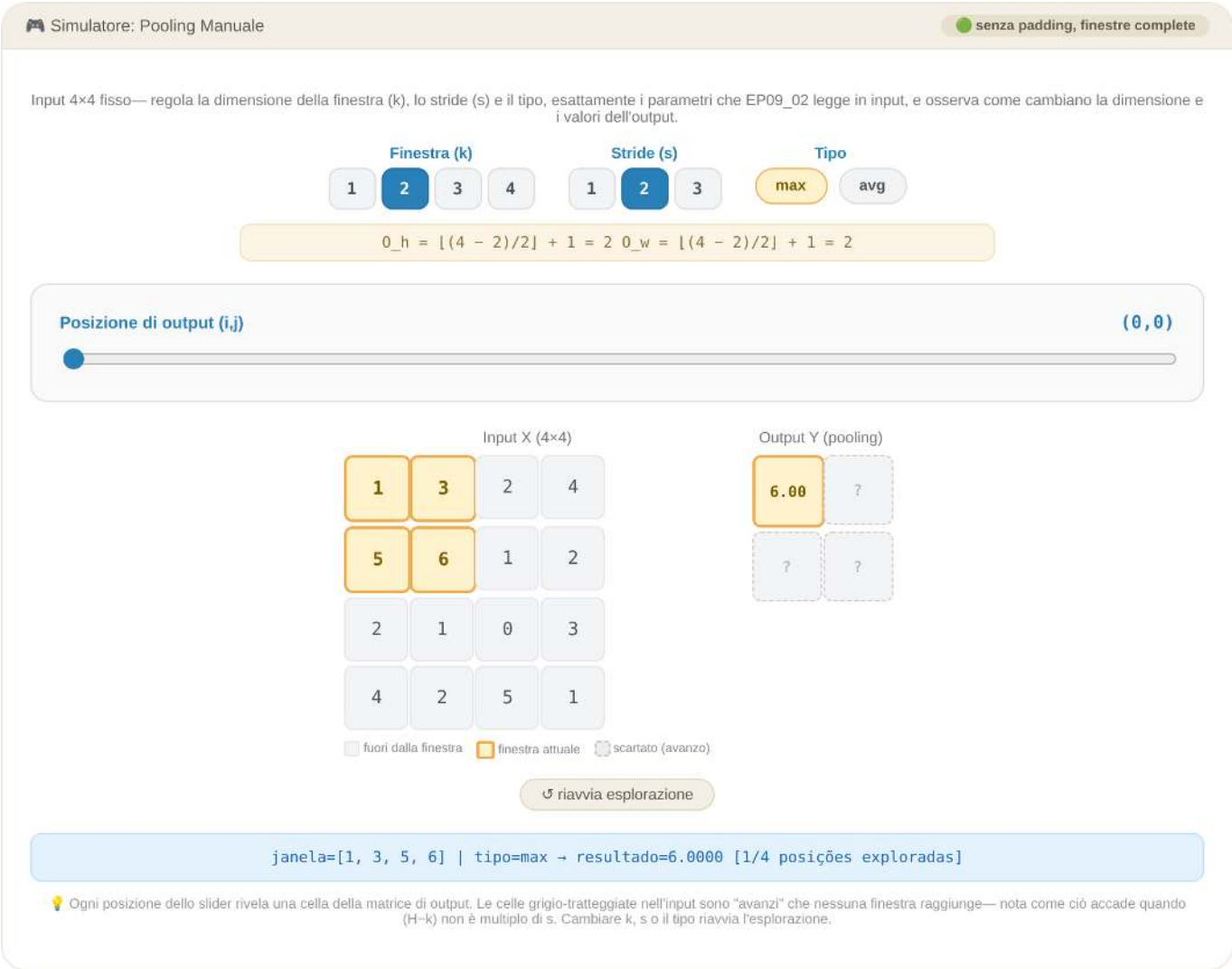
- Riga 1: Interi H e W.
- Prossime H righe: W valori reali ciascuna.
- Riga successiva: Interi k e s.
- Riga successiva: `max` o `avg`.

Uscita:

- Riga 1: Interi O_h e O_w.
- Prossime O_h righe: O_w valori reali ciascuna, con 4 cifre decimali.

9.10.2.6 📌 Esempi

Ingresso	Uscita	Osservazione
4 4 1 3 2 4 5 6 1 2 2 1 0 3 4 2 5 1 2 2 max	2 2 6.0000 4.0000 4.0000 5.0000	*Pooling* massimo, finestra 2×2, *stride* 2.
4 4 1 3 2 4 5 6 1 2 2 1 0 3 4 2 5 1 2 2 avg	2 2 3.7500 2.2500 2.2500 2.2500	*Pooling* medio sulle stesse finestre.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-ep0902.html

Figura 9.44: Simulatore EP09_02: Pooling Manuale (massimo vs. media, con finestra k e stride s regolabili)

```
%%writefile EP09_02.py
# Codice Python
```

```
Overwriting EP09_02.py
```

```
TestSuite("EP09_02.py").run()
```

```
<IPython.core.display.HTML object>
```

9.10.3 EP09_03 🟡 Conteggio dei Parametri Addestrabili di una CNN

Questo EP formalizza il conteggio dei parametri addestrabili di una *CNN*. Data la descrizione testuale di una piccola architettura, composta da layer convoluzionali, di *pooling* e completamente connessi, determinare, per ogni layer, il numero di parametri addestrabili e il totale della rete.

L'architettura deve essere interpretata **sequenzialmente**: l'uscita di un layer convoluzionale diventa l'ingresso del layer successivo compatibile. Pertanto, il numero di canali prodotti da un layer `CONV` determina il numero di canali di ingresso (`cin`) del layer convoluzionale successivo.

In un layer convoluzionale, è importante distinguere **canali di ingresso** e **canali di uscita**:

- c_{in} (*channels in*) è il numero di **canali che entrano nel layer**. Un'immagine in scala di grigi ha $c_{in} = 1$, mentre un'immagine RGB ha $c_{in} = 3$. In un layer convoluzionale intermedio, `cin` è normalmente uguale al numero di canali prodotti dal layer `CONV` precedente.
- c_{out} (*channels out*) è il numero di **canali prodotti dal layer**. È uguale al numero di filtri utilizzati. Pertanto, se un layer ha 16 filtri, produce $c_{out} = 16$ canali.

Ad esempio, si consideri la sequenza:

```
CONV 3 3 1 8 1
POOL
CONV 3 3 8 16 1
POOL
FC 784 10 1
```

La prima convoluzione riceve un'immagine con un canale e produce 8 canali. Dopo il *pooling*, la seconda convoluzione riceve questi 8 canali e ne produce 16. Il layer `POOL` non altera il numero di canali, può solo ridurre le dimensioni spaziali. Il layer `FC` riceve la quantità di ingressi indicata nella propria descrizione.

Ogni filtro convoluzionale ha dimensioni

$$k_h \times k_w \times c_{in}.$$

Pertanto, un layer con c_{out} filtri ha

$$k_h \cdot k_w \cdot c_{in} \cdot c_{out}$$

pesi. Se c'è un bias, si aggiunge un parametro per ogni filtro, totalizzando altri c_{out} parametri.

Il punto centrale di questo esercizio è osservare che la quantità di parametri di un layer convoluzionale **non dipende dalle dimensioni spaziali** ($H \times W$) della mappa delle caratteristiche. Ciò è dovuto alla **condivisione dei pesi**: lo stesso filtro viene riutilizzato in diverse posizioni dell'ingresso.

9.10.3.1 📋 Linee Guida di Implementazione

1. **Ingresso:** Leggere l'intero L (numero di layer dell'architettura, nell'ordine in cui vengono applicati).

2. **Layer:** Leggere L righe, ciascuna descrive un layer in uno dei tre formati:
   - `CONV kh kw cin cout bias` — layer convoluzionale con *kernel* $k_h \times k_w$, c_{in} canali di ingresso, c_{out} canali di uscita e `bias` (0 o 1), che indica se c'è un bias per filtro;
   - `POOL` — layer di *pooling* (massimo o medio), che non ha parametri addestrabili e preserva il numero di canali;
   - `FC in out bias` — layer completamente connesso con `in` ingressi, `out` uscite e `bias` (0 o 1), che indica se c'è un bias per neurone.
3. **Coerenza tra layer `CONV`:** in una sequenza di layer convoluzionali, il `cin` di un layer deve corrispondere al `cout` del layer convoluzionale precedente. Un layer `POOL` non altera questo numero di canali.

   Ad esempio:

   ```
   CONV 3 3 1 8 1
   POOL
   CONV 3 3 8 16 1
   ```

   La prima `CONV` produce 8 canali, che vengono ricevuti dalla seconda `CONV`. Pertanto, nel secondo layer, `cin=8` e `cout=16`.
4. **Parametri di un layer `CONV`:**

   Ciascuno dei c_{out} filtri ha $k_h \cdot k_w \cdot c_{in}$ pesi. Pertanto,

   $$P_{\text{CONV}} = k_h \cdot k_w \cdot c_{in} \cdot c_{out} + c_{out} \cdot \text{bias}.$$

5. **Parametri di un layer `FC`:**

   $$P_{\text{FC}} = \text{in} \cdot \text{out} + \text{out} \cdot \text{bias}.$$

6. **Parametri di un layer `POOL`:** sempre 0.
7. **Totale della rete:** sommare i parametri addestrabili di tutti i layer.
8. **Uscita:** Per ogni layer, nell'ordine di lettura, stampare `Camada i: P`, dove i inizia da 1 e P è il numero di parametri di quel layer. Alla fine, stampare `Total: T`.

9.10.3.2 📐 Esempio per capire `cin` e `cout`

Si consideri la sequenza:

```
CONV 3 3 1 8 1
POOL
CONV 3 3 8 16 1
```

Nel primo layer:

- `cin=1`: entra un canale;
- `cout=8`: ci sono 8 filtri e, quindi, escono 8 canali.

Ogni filtro ha

$$3 \cdot 3 \cdot 1 = 9$$

pesi. Poiché ci sono 8 filtri:

$$9 \cdot 8 = 72$$

pesi. Con un bias per filtro:

$$72 + 8 = 80.$$

Nel secondo layer:

- `cin=8`: entrano gli 8 canali prodotti dalla prima `CONV`;
- `cout=16`: ci sono 16 filtri e, quindi, escono 16 canali.

Ogni filtro ha

$$3 \cdot 3 \cdot 8 = 72$$

pesi. Poiché ci sono 16 filtri:

$$72 \cdot 16 = 1152$$

pesi. Con 16 bias:

$$1152 + 16 = 1168.$$

Pertanto, i due layer hanno, rispettivamente, **80** e **1168 parametri addestrabili**.

Si noti che `cout` **non è** *cin* moltiplicato per il numero di filtri. Il numero di filtri è esattamente `cout`: ogni filtro combina tutti i canali di ingresso e produce **un singolo canale di uscita**.

9.10.3.3 📌 Vincoli Computazionali

- **Indipendenza dalla dimensione spaziale:** l'ingresso non fornisce $H \times W$. Il conteggio di un layer `CONV` dipende solo da `kh`, `kw`, `cin` e `cout`.
- **Coerenza dei canali:** per due layer `CONV` consecutivi, il `cin` del secondo deve essere uguale al `cout` del primo. Un layer `POOL` preserva il numero di canali.
- **`bias` sempre 0 o 1:** moltiplicare direttamente il termine di bias per questo valore.
- **Layer `POOL` senza argomenti aggiuntivi:** la riga contiene solo la parola `POOL`.
- **Layer `FC`:** il numero di ingressi `in` è fornito esplicitamente. Non è necessario calcolare le dimensioni spaziali prodotte dai layer precedenti.
- Tutti i valori numerici di ingresso sono interi non negativi.

9.10.3.4 🧠 Fondamenti Teorici

Elemento	Ruolo nel conteggio dei parametri
c_{in}	Numero di canali ricevuti dal layer
c_{out}	Numero di filtri e, quindi, di canali prodotti dal layer
Filtro convoluzionale	Ogni filtro ha $k_h \cdot k_w \cdot c_{in}$ pesi e produce un canale di uscita
Condivisione dei pesi	Lo stesso filtro viene riutilizzato in diverse posizioni dell'ingresso, rendendo il conteggio indipendente da $H \times W$
Bias	Un singolo parametro aggiuntivo per filtro (`CONV`) o per neurone (`FC`)
Pooling	Può alterare $H \times W$, ma non ha parametri addestrabili e preserva il numero di canali
Layer `FC`	Ha un peso per ogni combinazione tra ingresso e neurone di uscita

9.10.3.5 🧩 Metodi di `morph.py` che possono aiutare

Questo esercizio è puramente aritmetico e non utilizza direttamente le funzioni di `morph.py`. Il conteggio può, tuttavia, essere verificato in un'architettura reale implementata in PyTorch tramite:

```
sum(p.numel() for p in modelo.parameters())
```

Questa espressione calcola i parametri del modello, inclusi pesi e bias.

9.10.3.6 📦 Specifica di Ingresso e Uscita (VPL)

Ingresso:

- Riga 1: Intero L.
- Prossime L righe: descrizione di ogni layer, nel formato `CONV kh kw cin cout bias`, `POOL` o `FC in out bias`.

Uscita:

- L righe nel formato `Camada i: P`.
- Ultima riga: `Total: T`.

9.10.3.7 📌 Esempi

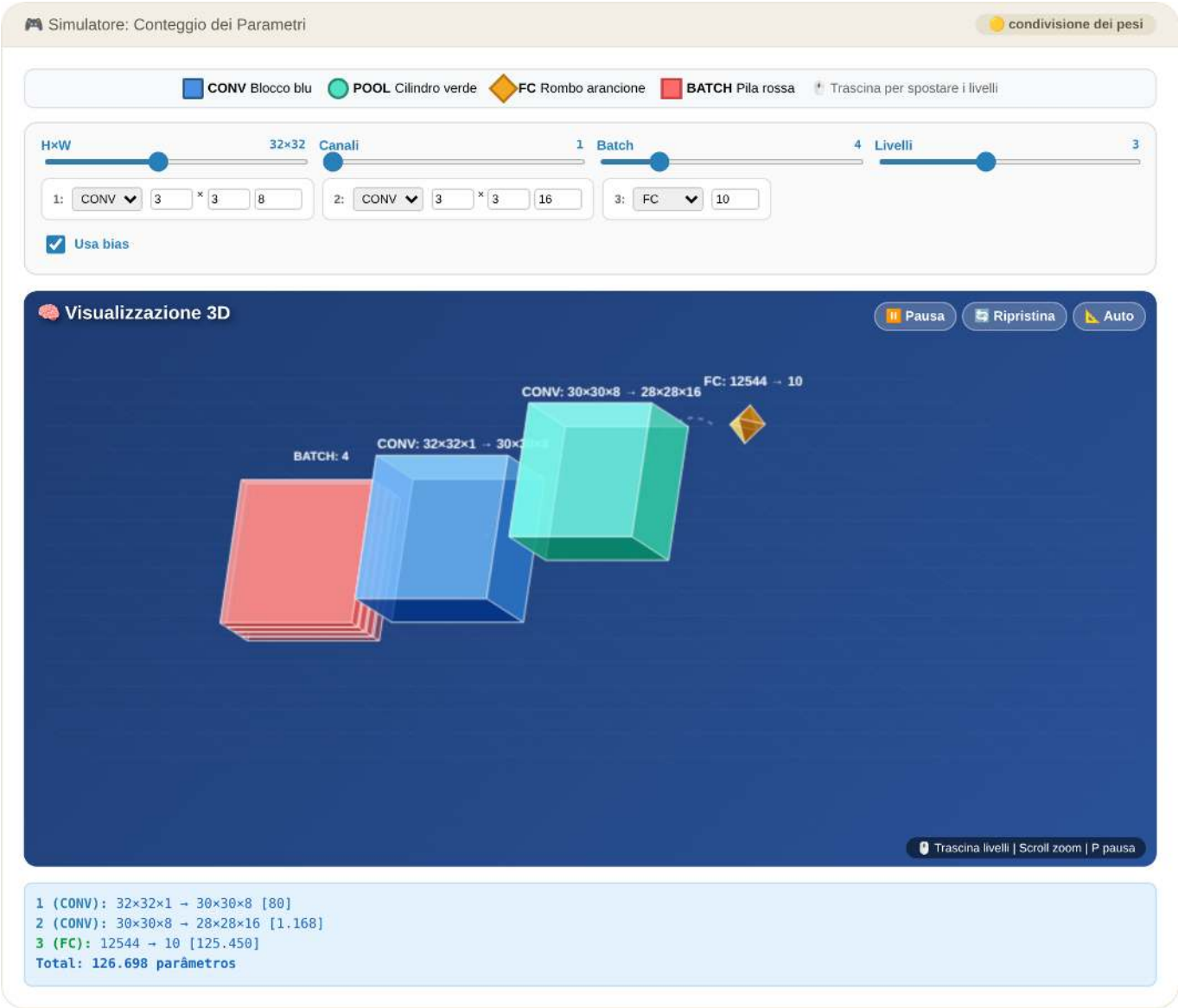
Ingresso	Uscita	Osservazione
3 CONV 3 3 1 8 1 POOL FC 1352 10 1	Camada 1: 80 Camada 2: 0 Camada 3: 13530 Total: 13610	Rete semplice con una convoluzione, *pooling* e layer di classificazione.
5 CONV 3 3 1 8 1 POOL CONV 3 3 8 16 1 POOL FC 400 10 1	Camada 1: 80 Camada 2: 0 Camada 3: 1168 Camada 4: 0 Camada 5: 4010 Total: 5258	Piccola CNN con due convoluzioni, due *pooling* e un layer completamente connesso.
6 CONV 3 3 1 8 1 POOL CONV 3 3 8 16 1 POOL FC 256 32 1 FC 32 10 1	Camada 1: 80 Camada 2: 0 Camada 3: 1168 Camada 4: 0 Camada 5: 8224 Camada 6: 330 Total: 9802	Piccola CNN con due convoluzioni, *pooling* intermedio e due layer completamente connessi per la classificazione.

```
%%writefile EP09_03.py
# Codice Python
```

```
Overwriting EP09_03.py
```

```
TestSuite("EP09_03.py").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-ep0903.html

Figura 9.45: Simulatore EP09_03: Conteggio dei Parametri — Convoluzione vs. Strato Totalmente Connesso

9.10.4 EP09_04 🟡 Intersezione su Unione (IoU) e Soppressione dei Non-Massimi (NMS)

I modelli di rilevamento degli oggetti possono produrre **diverse bounding box candidate** per lo stesso oggetto, con posizioni e punteggi di confidenza differenti. La fase di post-elaborazione responsabile dell'eliminazione di queste rilevazioni ridondanti è la **Soppressione dei Non-Massimi (NMS)**, la cui operazione fondamentale utilizza la metrica di **Intersezione su Unione (IoU)**.

La NMS utilizza questa misura per decidere quali box devono essere mantenute. In generale, la box con la confidenza più alta viene selezionata per prima; successivamente, le box che presentano un'IoU superiore a una certa soglia con la box selezionata sono considerate ridondanti e vengono rimosse. Il processo viene ripetuto finché non rimangono box candidate.

In questo esercizio, dovrai implementare l'algoritmo NMS da zero, calcolando l'IoU tra le box e applicando successivamente il criterio di selezione e soppressione per produrre l'insieme finale di rilevazioni.

9.10.4.1 📋 Linee Guida di Implementazione

1. **Input:** Leggere l'intero N (numero di box candidate) e la soglia reale τ (soglia IoU per la soppressione), sulla stessa riga.
2. **Box:** Leggere N righe, ciascuna con cinque valori reali:

   ```
   x1 y1 x2 y2 score
   ```

   dove (x_1, y_1) rappresenta l'angolo superiore sinistro, (x_2, y_2) l'angolo inferiore destro e `score` il punteggio di confidenza.
3. **Intersezione su Unione:** Per due box A e B,

   $$IoU(A, B) = \frac{\text{Area}(A \cap B)}{\text{Area}(A \cup B)}.$$

   L'area di intersezione deve essere calcolata dalla sovrapposizione degli intervalli in x e y. Se non c'è sovrapposizione, l'area di intersezione è zero.
4. **Algoritmo greedy di NMS:**
   a. Ordina le box per `score` decrescente. In caso di parità, mantieni l'ordine originale di lettura.
   b. Seleziona la box con il punteggio più alto tra le box rimanenti e aggiungila all'insieme di output.
   c. Calcola l'IoU tra la box selezionata e **tutte le box ancora rimanenti**. Sopprimi le box per cui

   $$\text{IoU} > \tau.$$

   d. Ripeti i passaggi (b) e (c) finché non rimangono box.
5. **Output:** Per ogni box mantenuta, nell'ordine in cui è stata selezionata, stampa il suo indice originale (posizione di lettura, a partire da 0) e il suo `score`, formattato con 4 cifre decimali. Alla fine, stampa:

   ```
   Totale mantenute: X
   ```

9.10.4.2 📌 Vincoli Computazionali

- **Soppressione stretta:** solo le box con IoU $> \tau$ vengono soppresse. Le box con IoU $= \tau$ vengono mantenute.
- **Indici originali:** l'output fa riferimento alla posizione in cui ogni box è stata letta nell'input (a partire da 0), non alla sua posizione dopo l'ordinamento.
- **Ordinamento stabile:** in caso di `score` uguali, deve essere preservato l'ordine originale di lettura.
- **Rettangoli allineati agli assi:** tutte le box sono specificate da due angoli, con $x_1 < x_2$ e $y_1 < y_2$ garantiti nell'input.

- **Coordinate e punteggi:** i valori reali possono essere positivi o negativi, secondo i limiti definiti dall'input, ma le dimensioni delle box sono sempre positive.

9.10.4.3 🧠 Fondamento Teorico

Elemento	Ruolo nella post-elaborazione della rilevazione
IoU	Quantifica la sovrapposizione spaziale tra due box; IoU = 1 per box identiche e IoU = 0 per box senza sovrapposizione
Ordinamento per confidenza	Fa sì che la box con `score` più alto venga analizzata per prima
Soglia τ	Definisce la quantità di sovrapposizione necessaria affinché una box sia considerata ridondante
Soppressione	Rimuove le box che presentano una grande sovrapposizione con una box già selezionata
Box distanti	Hanno IoU vicina a zero e, in generale, non vengono soppresse da questa regola

9.10.4.4 🧩 Metodi di `morph.py` che possono aiutare

- `mm.IoU(boxA, boxB)` — calcola la metrica IoU, ma si aspetta le box nel formato (x, y, w, h), cioè angolo superiore sinistro, larghezza e altezza. L'input di questo esercizio utilizza il formato (x_1, y_1, x_2, y_2). La conversione è diretta:

$$w = x_2 - x_1, \qquad h = y_2 - y_1.$$

  L'uso di questa funzione è facoltativo. L'obiettivo principale dell'esercizio è implementare correttamente il processo di selezione e soppressione della NMS.

9.10.4.5 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: intero N e reale τ.
- Prossime N righe: x_1 y_1 x_2 y_2 score.

Output:

- Una riga per box mantenuta, nell'ordine di selezione: `indice score`.
- Ultima riga: `Totale mantenute: X`.

```
%%writefile EP09_04.py
# Codice Python
```

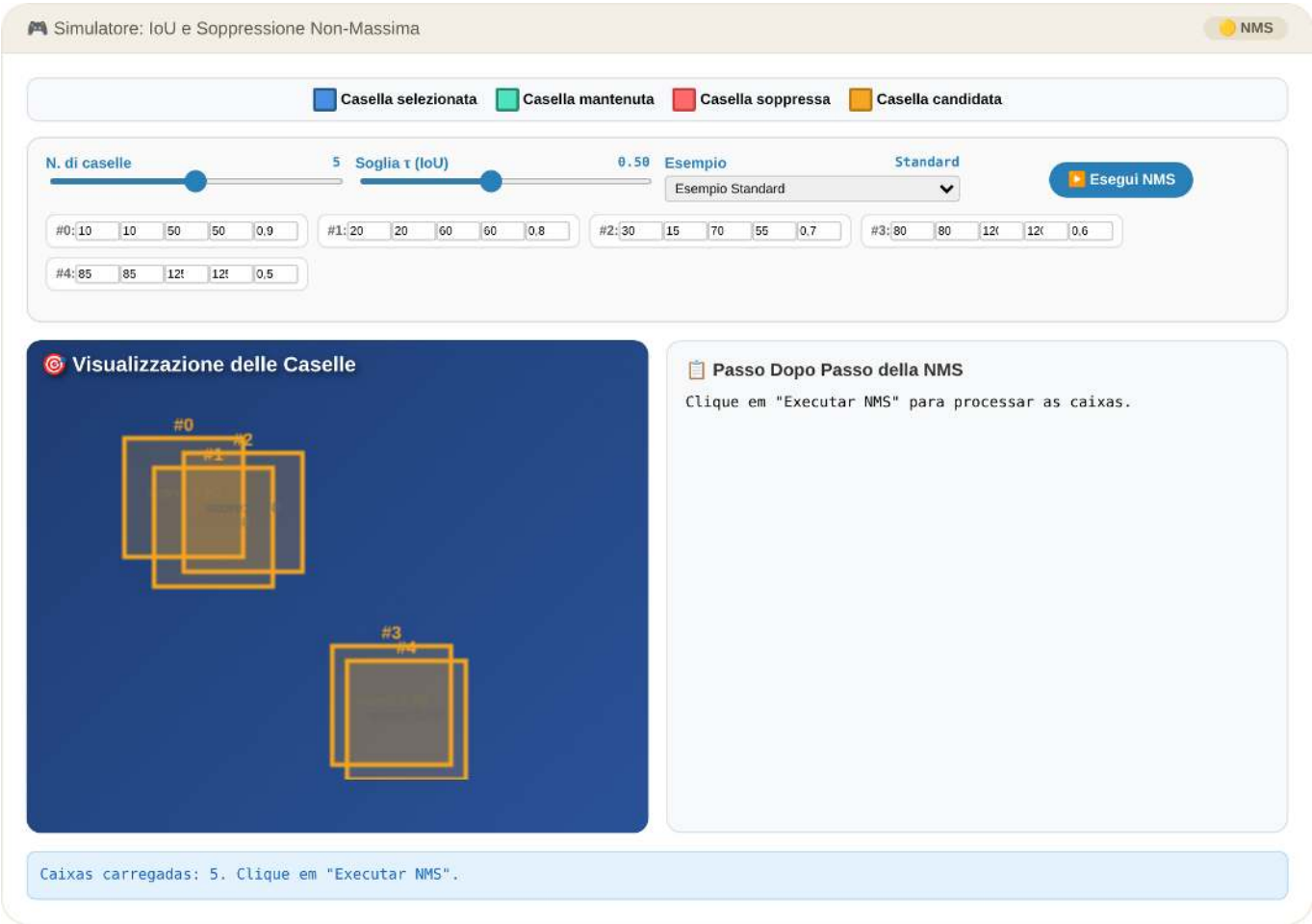
```
Overwriting EP09_04.py
```

```
TestSuite("EP09_04.py").run()
```

```
<IPython.core.display.HTML object>
```

9.10.5 EP09_05 🟠 Valutazione della Segmentazione: IoU e Dice Pixel per Pixel

Il Blocco 2 della sezione "Segmentazione Semantica con Architettura U-Net" definisce, in poche righe, la funzione `iou_mascaras`, utilizzata per misurare la qualità della baseline morfologica classica (smoothing + Otsu + apertura) e, più avanti, della stessa U-Net addestrata. Diversamente dall'IoU dell'EP09_04 — calcolato su **bounding box** (regioni rettangolari descritte da quattro numeri) —, l'IoU di segmentazione è



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-ep0904.html

Figura 9.46: Simulatore EP09_04: IoU e Soppressione Non-Massima (NMS)

calcolato **pixel per pixel**: ogni posizione dell'immagine viene confrontata individualmente tra la maschera predetta e la maschera di riferimento.

Ti è stato affidato il compito di generalizzare questa valutazione, implementando non solo l'IoU pixel per pixel, ma anche il **coefficiente di Dice**, un'altra metrica di sovrapposizione ampiamente utilizzata in segmentazione medica (inclusa nella funzione `perda_dice`, menzionata nello stesso blocco del capitolo come base della funzione di perdita utilizzata per addestrare la U-Net).

9.10.5.1 📋 Linee Guida di Implementazione

1. **Input:** Leggere le dimensioni $H \times W$ delle maschere.
2. **Maschera predetta:** Leggere H righe con W valori interi (0 o 1) ciascuna — ad esempio, l'output di una U-Net dopo la sogliatura a 0,5 sulla sigmoide, come nel Blocco 4 del capitolo.
3. **Maschera di riferimento:** Leggere altre H righe con W valori interi (0 o 1) ciascuna — il *ground truth*.
4. **Intersezione e unione:** Considerando ogni pixel come appartenente all'oggetto quando il suo valore è diverso da zero,
$$\text{intersezione} = \sum_{i,j} \mathbb{1}[P_{ij} = 1 \wedge R_{ij} = 1], \qquad \text{unione} = \sum_{i,j} \mathbb{1}[P_{ij} = 1 \vee R_{ij} = 1].$$
5. **IoU pixel per pixel:**
$$\text{IoU} = \frac{\text{intersezione}}{\text{unione}}.$$
6. **Coefficiente di Dice:**
$$\text{Dice} = \frac{2 \cdot \text{intersezione}}{|P| + |R|},$$
dove $|P|$ e $|R|$ sono il numero totale di pixel dell'oggetto in ciascuna maschera.
7. **Convenzione per maschere vuote:** se **entrambe** le maschere non hanno alcun pixel dell'oggetto (unione $= 0$ e $|P| + |R| = 0$), considera la corrispondenza banalmente perfetta: IoU = Dice = 1,0.
8. **Output:** Due righe, `IoU: X.XXXX` e `Dice: X.XXXX`, ciascun valore con 4 cifre decimali.

9.10.5.2 📌 Vincoli Computazionali

- **Qualsiasi valore non nullo conta come oggetto:** tratta i valori diversi da 0 (non solo 1) come appartenenti alla maschera, replicando il controllo `predita > 0` usato in `iou_mascaras` nel capitolo.
- **Stesse dimensioni:** le due maschere hanno sempre esattamente $H \times W$ elementi.
- **Convenzione del vuoto:** applica la regola del punto 7 **solo** quando entrambe le maschere sono completamente vuote; se solo una è vuota, l'intersezione è 0 e l'IoU/Dice risultante sarà anch'esso 0.

9.10.5.3 🧠 Fondamento Teorico

Elemento	Ruolo nella valutazione della segmentazione
IoU pixel per pixel	Generalizza la metrica dell'EP09_04 a regioni di forma arbitraria — non solo rettangoli — confrontando la maschera predetta e il riferimento posizione per posizione
Coefficiente di Dice	Metrica correlata all'IoU (sempre Dice $\geq$ IoU), più sensibile a piccole intersezioni e ampiamente utilizzata come funzione di perdita in segmentazione (funzione `perda_dice` del capitolo)

Elemento	Ruolo nella valutazione della segmentazione
Convenzione per maschere vuote	Evita la divisione per zero e riconosce che "nessun oggetto previsto, nessun oggetto reale" è, per definizione, un successo
Confronto classico vs. U-Net	Il capitolo usa esattamente questo tipo di metrica per giustificare, numericamente, perché la U-Net supera la baseline morfologica in scenari a basso contrasto

9.10.5.4 🧩 Metodi di `morph.py` che possono aiutare

- `mm.readImg(h, w, dtype='uint8')` — legge direttamente ogni maschera binaria $h \times w$ dall'input standard (i valori 0/1 rientrano perfettamente nel tipo intero standard).
- La stessa funzione `iou_mascaras`, definita nel Blocco 2 della sezione U-Net del capitolo (non fa parte di `morph.py`, ma del codice del capitolo), è l'ispirazione diretta di questo esercizio — vale la pena rileggere quelle poche righe prima di programmare.
- Per un'estensione opzionale (non richiesta da questo EP), `mm.connectedComponents` o `mm.label0` (visti nel contesto dell'analisi delle componenti connesse) permetterebbero di etichettare ogni nodulo individualmente e calcolare l'IoU **per componente**, invece che sull'intera maschera.

9.10.5.5 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Interi H e W.
- Prossime H righe: W valori interi (0 o 1) — maschera predetta.
- Prossime H righe: W valori interi (0 o 1) — maschera di riferimento.

Output:

- Riga 1: `IoU: X.XXXX`.
- Riga 2: `Dice: X.XXXX`.

💡 Esempio Illustrativo

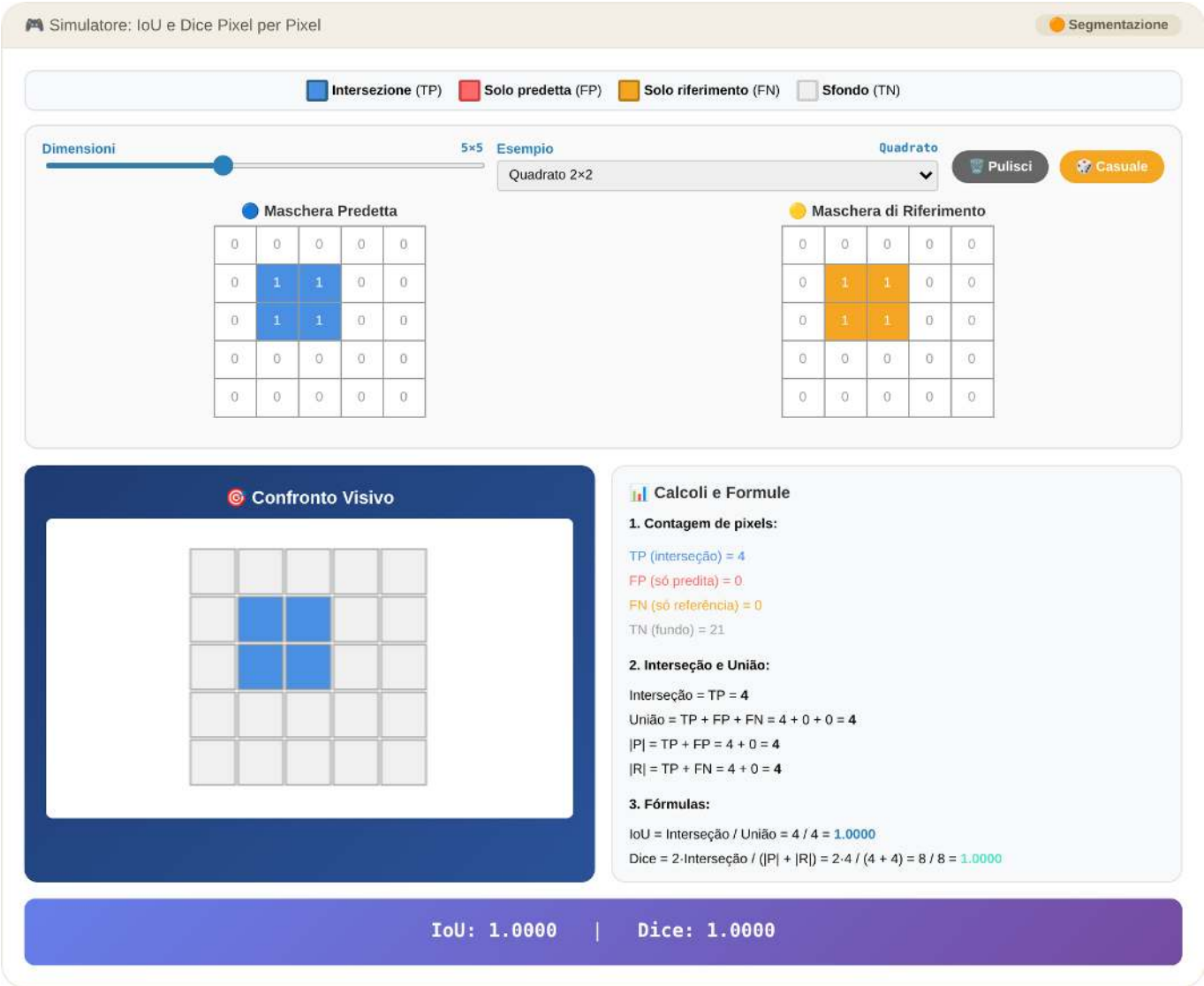
Considera una maschera predetta con un quadrato 2×2 di pixel attivi e un riferimento spostato di una colonna, sovrapponendosi solo per metà dell'area:

```
Predetta        Riferimento
0 0 0 0         0 0 0 0
0 1 1 0         0 0 1 1
0 1 1 0         0 0 1 1
0 0 0 0         0 0 0 0
```

Intersezione $= 2$ pixel, unione $= 6$ pixel $(4+4-2)$, quindi IoU $= 2/6 \approx 0{,}3333$ e Dice $= 2 \cdot 2/(4+4) = 0{,}5000$ — nota che il Dice è sempre uguale o maggiore dell'IoU per la stessa sovrapposizione.

9.10.5.6 📌 Esempi

Input	Output	Osservazione
4 4 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 1 0 0 0 0	IoU: 0.3333 Dice: 0.5000	Maschere 4×4 con sovrapposizione parziale di 2 pixel.



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-ep0905.html

Figura 9.47: Simulatore EP09_05: Valutazione della Segmentazione — IoU e Dice Pixel per Pixel

```
%%writefile EP09_05.py
# Codice Python
```

```
Overwriting EP09_05.py
```

```
TestSuite("EP09_05.py").run()
```

```
<IPython.core.display.HTML object>
```

9.10.6 EP09_06 🔴 *Pipeline* Integrato: Dalla Rilevazione alla Misurazione nel Mondo Reale

Questo esercizio finale integra i due esercizi di rilevazione e il principio di **fotogrammetria** presentato nella sezione "Fotogrammetria e Riferimento di Scala" — esattamente lo stesso calcolo implementato nella figura di misurazione per riferimento di scala di questo capitolo. Lo scenario riproduce una situazione realistica: un rilevatore (Faster R-CNN o YOLO) genera **diverse scatole candidate sovrapposte** per lo stesso oggetto di interesse; dopo averle filtrate tramite NMS, la scatola sopravvissuta con maggiore confidenza viene utilizzata, insieme a una scatola di riferimento di larghezza reale nota (come la carta da 8,56 cm), per stimare le dimensioni reali dell'oggetto rilevato.

9.10.6.1 📋 Linee Guida di Implementazione

1. **Riferimento noto:** Leggere il valore reale L_{ref} (larghezza reale dell'oggetto di riferimento, in cm) e successivamente i quattro reali x_1 y_1 x_2 y_2 della sua scatola delimitante in pixel (già nota, senza necessità di rilevazione).
2. **Candidati dell'oggetto da misurare:** Leggere l'intero N (numero di scatole candidate prodotte dal rilevatore per l'oggetto di interesse) e la soglia reale τ; successivamente, leggere le N righe di scatole candidate, ciascuna con x_1 y_1 x_2 y_2 score.
3. **Fase 1 — NMS:** Applicare esattamente l'algoritmo di Soppressione Non Massima dell'EP09_04 alle N scatole candidate, utilizzando la soglia τ, per eliminare rilevazioni ridondanti dello stesso oggetto.
4. **Fase 2 — Selezione della scatola finale:** Dopo il NMS, la scatola con il `score` più alto tra quelle mantenute è la rilevazione finale dell'oggetto (l'input garantisce che tutte le scatole candidate corrispondano a un singolo oggetto fisico, quindi la prima scatola selezionata dal NMS è già il risultato finale).
5. **Fase 3 — Misurazione per riferimento di scala:** Calcolare il rapporto cm/pixel $= L_{ref}$/larghezza del riferimento in pixel e applicarlo sia alla larghezza che all'altezza (in pixel) della scatola finale dell'oggetto, ottenendo le sue dimensioni reali stimate in centimetri.
6. **Output:** Prima, una riga per ogni scatola mantenuta dopo il NMS (stesso formato dell'EP09_04): `índice score`. Successivamente, la riga `Total mantidas: X`. Infine, la riga `Objeto: L x A cm`, dove L e A sono la larghezza e l'altezza stimate dell'oggetto, ciascuna con 2 cifre decimali.

9.10.6.2 📌 Vincoli Computazionali

- **Riutilizzare integralmente il NMS dell'EP09_04** — stessa regola di parità, stesso criterio di soppressione (IoU $> \tau$).
- **Il riferimento non passa attraverso il NMS:** la sua scatola è data direttamente, senza candidati concorrenti.
- **Rapporto unico per larghezza e altezza:** così come nella figura di fotogrammetria del capitolo, lo stesso rapporto cm/pixel (derivato dalla larghezza del riferimento) viene applicato sia alla larghezza che all'altezza dell'oggetto — non vi è calibrazione verticale separata.

9.10.6.3 🧠 Fondamenti Teorici

Fase	Concetto del capitolo
Multiple scatole candidate	Output grezzo di un rilevatore come Faster R-CNN o YOLO, prima della post-elaborazione
NMS (EP09_04)	Filtra le rilevazioni ridondanti, preservando solo la più affidabile per l'oggetto

Fase	Concetto del capitolo
Riferimento di scala noto	Stesso principio della carta da 8,56 cm utilizzata nella sezione "Fotogrammetria e Riferimento di Scala"
Conversione pixel → centimetro	Regola del tre semplice: cm/pixel $= L_{ref}/w_{ref_px}$, applicata alla scatola finale dell'oggetto

9.10.6.4 🧩 Metodi di `morph.py` che possono aiutare

- `mm.IoU(boxA, boxB)` — la stessa funzione suggerita nell'EP09_04, qui riutilizzata all'interno della fase di NMS di questo *pipeline* integrato (ricordarsi della conversione di formato: $w = x_2 - x_1$, $h = y_2 - y_1$).
- Se hai già risolto l'EP09_04 incapsulando il NMS in una funzione propria, questo è il momento ideale per **riutilizzare quel codice** — l'integrazione di moduli già testati singolarmente è esattamente la pratica ingegneristica che questo esercizio vuole rafforzare.

9.10.6.5 📦 Specifica di Input e Output (VPL)

Input:

- Riga 1: Reale L_{ref}.
- Riga 2: x_1 y_1 x_2 y_2 della scatola di riferimento.
- Riga 3: Intero N e reale τ.
- Prossime N righe: x_1 y_1 x_2 y_2 score delle scatole candidate dell'oggetto.

Output:

- Una riga per ogni scatola mantenuta dopo il NMS: `índice score`.
- Riga successiva: `Total mantidas: X`.
- Ultima riga: `Objeto: L x A cm`.

9.10.6.6 📌 Esempi

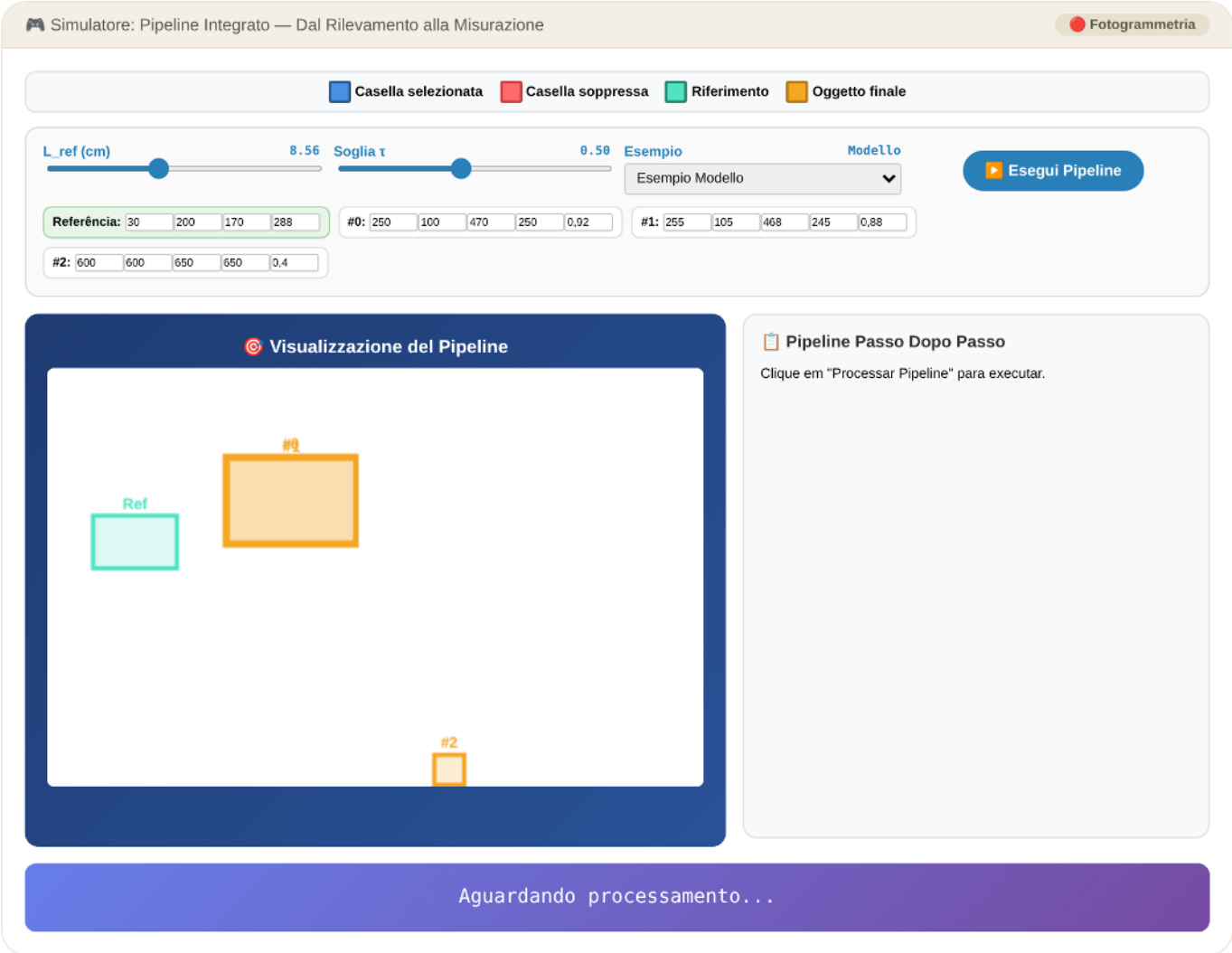
Input	Output	Osservazione
8.56 30 200 170 288 3 0.5 250 100 470 250 0.92 255 105 468 245 0.88 600 600 650 650 0.40	0 0.9200 2 0.4000 Total mantidas: 2 Objeto: 13.45 x 9.17 cm	La scatola 1 viene soppressa perché si sovrappone fortemente alla scatola 0; la rilevazione finale dell'oggetto è la scatola 0.

```
%%writefile EP09_06.py
# Codice Python
```

```
Overwriting EP09_06.py
```

```
TestSuite("EP09_06.py").run()
```

```
<IPython.core.display.HTML object>
```



Versão interativa: https://fzampirolli.github.io/pdi-vc/simuladores/py.it/cap09/sim-ep0906.html

Figura 9.48: Simulatore EP09_06: Pipeline Integrato — Rilevamento alla Misurazione del Mondo Reale

Riferimenti

BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. *In*: 2015. Disponível em: <https://arxiv.org/abs/1409.0473>

BARRERA, Junior *et al.* MMach: a mathematical morphology toolbox for the KHOROS system. **Journal of Electronic Imaging**, v. 7, n. 1, p. 174–210, 1998.

DOUGHERTY, Edward R.; LOTUFO, Roberto A. **Hands-on morphological image processing**. *[S.l.]*: SPIE press, 2003. v. 59

KNUTH, Donald E. Literate Programming. **The Computer Journal**, v. 27, n. 2, p. 97–111, 1984.

LOTUFO, Roberto A. *et al.* MMachLib functions and MMach operators. *In*: 1997.

MATHERON, Georges. **Random Sets and Integral Geometry**. New York: John Wiley & Sons, 1975.

VIOLA, Paul; JONES, Michael. Rapid Object Detection Using a Boosted Cascade of Simple Features. *In*: IEEE Computer Society, 2001.

ZAMPIROLLI, Francisco A. **MCTest: Como Criar e Corrigir Exames Parametrizados**. *[S.l.]*: Independente, 2023.

ZAMPIROLLI, Francisco de Assis *et al.* A Practical Digital Image Processing Course with morph.py. *In*: Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação (SBC), 2024. Disponível em: <https://sol.sbc.org.br/index.php/educomp/article/view/28199>

ZAMPIROLLI, Francisco de Assis *et al.* Teaching Hands-On Digital Image Processing with morph.py: Methods and Comprehensive Results. **Revista Brasileira de Informática na Educação**, v. 33, p. 990–1026, 2025.

ZAMPIROLLI, Francisco de Assis *et al.* Intelligent Feedback for Individualized Introductory Programming Exercises. **Computer Applications in Engineering Education**, v. 34, n. 1, p. e70132, 2026.

Appendice A

Sobre o MCTest

O **MCTest** é um sistema de código aberto para criação e correção automática de exames parametrizados (Zampirolli, 2023). Ele oferece suporte a questões de múltipla escolha, dissertativas e de programação, estas últimas integradas ao VPL do Moodle, descrito no Apêndice B.

A versão do MCTest utilizada neste livro é a **5.4**, disponível em https://github.com/fzampirolli/mctest. A subpasta `book` do repositório contém as versões **5.3**, descrita em (Zampirolli, 2023), e **5.4**, utilizada para gerar as provas apresentadas neste livro e atualmente em produção na UFABC (https://mctest.ufabc.edu.br).

A.1 Instalação do MCTest

A instalação recomendada utiliza o VirtualBox com Ubuntu 22.04. No terminal do Ubuntu, como *root*, execute:

```
wget https://raw.githubusercontent.com/fzampirolli/mctest/master/_setup-all.sh
```

Em seguida, substitua `yourLogin` pelo nome do usuário local e execute:

```
sed -i 's/\/home\/fz\//\/home\/yourLogin\//g' _setup-all.sh
source _setup-all.sh
pip install mysqlclient
```

Após alguns minutos, o MCTest estará configurado. Para executá-lo:

```
source /home/yourLogin/PycharmProjects/runDjango.sh
```

O sistema fica então acessível em `http://127.0.0.1:8000`.

A.2 Criando um exame no MCTest

No MCTest, todo exame é configurado em uma tela de **Exame**, que reúne Turmas, Tópicos (associados a uma disciplina, como PDI-VC) e Questões — cada questão pertence a um único tópico. Além dos atributos de banco de dados, o exame possui parâmetros próprios, como o número de questões sorteadas e o número de variações geradas.

No caso das duas provas descritas neste apêndice, foi definido um conjunto de questões parametrizadas por prova, das quais um subconjunto é sorteado aleatoriamente para cada variação, totalizando **110 variações** distintas — uma por estudante matriculado nas turmas.

O botão **Create-Variations** gera um novo conjunto de variações a cada acionamento. Quando as questões estão associadas a atividades VPL, o professor recebe por e-mail um arquivo `*linker.json` contendo todos os casos de teste das variações geradas. O botão **Create-PDF** gera, para cada turma, um PDF com as provas sorteadas por estudante, além de um arquivo `*students_variations.csv` com nome, e-mail e variação sorteada de cada estudante.

! Atenção

Cada acionamento do botão **Create-Variations** gera um novo conjunto de variações de exames. Após imprimir o PDF, é fundamental **não alterar os atributos do exame**, sob risco de invalidar a correção automática das provas já impressas ou aplicadas.

A.3 Criando uma questão paramétrica

Uma questão paramétrica no MCTest combina três elementos:

1. **Descrição** — o enunciado da questão, escrito em LaTeX, contendo variáveis entre `[[code:variavel]]`, que são substituídas pelo valor sorteado para cada estudante;
2. **Bloco de definição** (`[[def: ... ]]`) — um trecho de código Python, embutido na própria questão, responsável por sortear os parâmetros, calcular a solução de referência e montar os casos de teste;
3. **Casos de teste para o Moodle** (bloco `moodle_cases`) — dicionário serializado em JSON contendo as entradas e saídas esperadas, consumido pela atividade VPL.

Um exemplo simplificado de definição de questão (adaptado da geração de um tabuleiro de xadrez `h × w`) é:

```
[[code:texto]]

\vspace{2mm}\noindent\textbf{Exemplo de Entrada:}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_inp]]
\end{verbatim}
\vspace{-2mm}\noindent\textbf{Exemplo de Saída:}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_out]]
\end{verbatim}

\begin{comment}
[[code:moodle_cases]]
\end{comment}

[[def:
import json
import numpy as np
from topic.morph import mm # TEM QUE INCLUIR "topic." no MCTest

def chess(h, w):
    m = np.zeros((h, w), dtype='int')
    for i in range(h):
        for j in range(w):
            if (i + j) % 2:
                m[i][j] = 1
    return m

# PARÂMETROS USADOS NA DESCRIÇÃO DA QUESTÃO
height = int(np.random.randint(5, 15))

# ENUNCIADO COMPLETO, COM O VALOR DE height JÁ INTERPOLADO
texto = (
    r"\vspace{2mm}\noindent\textbf{Descrição:}\vspace{-2mm} "
    r"Escreva um programa que leia um número inteiro \texttt{W}, representando a "
    r"largura (número de colunas) de um tabuleiro, e imprima esse tabuleiro no "
    r"formato de um tabuleiro de xadrez, usando os dígitos \texttt{0} e \texttt{1}. "
    r"O tabuleiro impresso terá sempre altura (número de linhas) igual a "
    f"\\textbf{{{height}}} "
    r"e largura igual ao valor \texttt{W} lido da entrada. "
```

```
    r"Considerando a posição $(i, j)$ do tabuleiro (indexada a partir de 0, com "
    r"$i$ representando a linha e $j$ a coluna), o valor impresso nessa posição "
    r"deve ser \texttt{1} se $i + j$ for ímpar, e \texttt{0} caso contrário. Cada "
    r"linha do tabuleiro deve ser impressa em uma linha separada, com os valores "
    r"de cada coluna separados por um espaço."
)

inp_list, out_list, test_cases = [], [], 4
for i in range(test_cases):
    width = int(np.random.randint(500, 1500) / 100)
    inp = str(width) + '\n'
    out = mm.drawImg(chess(height, width)) + '\n'
    inp_list.append(inp)
    out_list.append(out)

cases = {}
cases['skills']      = ["matrizes", "laços de repetição", "formatação de saída"]
cases['description'] = [{"text": latex_to_text(texto)}]
cases['input']       = inp_list
cases['output']      = out_list

moodle_cases = json.dumps(cases)

caso0_inp = cases['input'][0]
caso0_out = cases['output'][0]
]]
```

O valor de `height` é sorteado uma única vez por variação (para aparecer no enunciado), enquanto `width` é sorteado a cada caso de teste, aumentando a robustez da correção automática.

Vale destacar o papel duplo da variável `texto` nesse mecanismo. Ela é ao mesmo tempo:

- **conteúdo do enunciado impresso**, inserida na descrição LaTeX da questão através da tag `[[code:texto]]`, aparecendo no PDF gerado pelo botão **Create-PDF**; e
- **entrada do dicionário exportado para o Moodle**, através da chave `cases['description']`, que passa a compor o JSON `moodle_cases`. É justamente esse campo que a atividade VPL exibe ao estudante quando ele abre a questão no Moodle para avaliá-la ou submeter uma solução.

Há, porém, uma diferença importante entre esses dois usos: o PDF é composto em LaTeX e, portanto, interpreta normalmente comandos como `\textbf{}`, `\texttt{}` ou `$...$`. Já o VPL do Moodle **não** processa LaTeX — ele espera texto puro. Por isso, ao montar `cases['description']`, `texto` não é inserido diretamente, mas sim passado pela função utilitária `latex_to_text()`, própria do MCTest, que remove (ou converte) os comandos e símbolos LaTeX do enunciado, produzindo uma versão em texto simples equivalente. Assim, `[[code:texto]]` no PDF continua recebendo o `texto` original, com toda a formatação LaTeX, enquanto `cases['description']` recebe `latex_to_text(texto)`, garantindo que o enunciado exibido no Moodle seja legível mesmo sem suporte a LaTeX.

Como `texto` é montada dentro do próprio bloco `[[def: ...]]` — no mesmo escopo em que `height`, `width` e os demais parâmetros são sorteados —, ela é recalculada a cada variação. Isso garante que o enunciado visto pelo estudante no Moodle seja **sempre idêntico** ao que foi impresso na prova em papel, mesmo quando o texto muda de estudante para estudante junto com os valores sorteados. A Seção A.4 retoma esse ponto em um caso real, no qual o enunciado é significativamente mais longo e descreve, além dos parâmetros numéricos, o próprio cenário visual gerado para cada variação.

A Figura A.1 mostra a questão do tabuleiro de xadrez efetivamente gerada pelo MCTest para uma das variações, com o enunciado já contendo o valor sorteado de `height` e o exemplo de entrada/saída correspondente ao primeiro caso de teste:

Ao acionar **Create-PDF** na tela da questão, uma nova variação é gerada a cada clique, permitindo conferir o enunciado antes de publicá-lo.

1. **Descrição:** Escreva um programa que leia um número inteiro `W`, representando a largura (número de colunas) de um tabuleiro, e imprima esse tabuleiro no formato de um tabuleiro de xadrez, usando os dígitos `0` e `1`. O tabuleiro impresso terá sempre altura (número de linhas) igual a **6** e largura igual ao valor `W` lido da entrada. Considerando a posição (i, j) do tabuleiro (indexada a partir de 0, com i representando a linha e j a coluna), o valor impresso nessa posição deve ser `1` se $i + j$ for ímpar, e `0` caso contrário. Cada linha do tabuleiro deve ser impressa em uma linha separada, com os valores de cada coluna separados por um espaço.

Exemplo de Entrada:

```
9
```

Exemplo de Saída:

```
0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
0 1 0 1 0 1 0 1 0
1 0 1 0 1 0 1 0 1
```

Figura A.1: Questão do tabuleiro de xadrez gerada pelo MCTest, ilustrando o enunciado parametrizado com o valor de `height` sorteado para a variação e o exemplo de entrada/saída do primeiro caso de teste.

Para as duas provas da disciplina, esse mecanismo foi utilizado de forma mais ampla: cada questão paramétrica define não apenas os valores numéricos, mas também, em alguns casos, pequenas variações estruturais no enunciado (por exemplo, qual direção cardinal — Norte, Sul, Leste, Oeste — deve ser avaliada), dificultando a busca por soluções prontas na internet ou em ferramentas de IA generativa.

A.4 Exemplo real: uma questão do Simulado 4

Para ilustrar o mecanismo descrito na seção anterior com um caso concreto, apresenta-se a seguir uma questão efetivamente aplicada no **Simulado 4** (tópico *im4-Operadores Morfológicos*, dificuldade 1, taxonomia de Bloom "remember"), do tipo questão de programação parametrizada com integração Moodle+VPL.

A questão pede ao estudante que escreva um programa capaz de:

- ler uma imagem binária, corrompida por ruído sal e pimenta, contendo ao menos dois objetos geométricos isolados;
- aplicar filtragem morfológica (abertura seguida de fechamento) para eliminar o ruído;
- extrair, a partir da imagem filtrada, as medidas geométricas de cada objeto (área, perímetro, centro, bounding box, circularidade, solidez e número de vértices), usando a função auxiliar `mm.measure`;
- ordenar os objetos por posição (coordenada X do bounding box, com Y como critério de desempate) e reatribuir os identificadores sequencialmente;
- imprimir a matriz limpa e a tabela de medidas no formato esperado pelo corretor automático.

No bloco `[[def: ... ]]` correspondente, um gerador de cena (`gerarCena`) sorteia aleatoriamente a altura e a largura da imagem, o tipo, o tamanho e a posição de cada objeto (quadrado, retângulo, triângulo ou bloco), garantindo que eles não se sobreponham, e em seguida acrescenta ruído sal e pimenta com 3% de probabilidade por pixel. Dez casos de teste distintos são gerados dessa forma para cada variação da prova, e o par entrada/saída do primeiro caso é reaproveitado no próprio enunciado como exemplo para o estudante. A biblioteca de morfologia (`mm`) é importada diretamente de `morph.py`, também disponível como módulo utilitário do próprio MCTest.

Em resumo, o enunciado pede ao estudante um programa que:

1. leia, na primeira linha, dois inteiros H e W (altura e largura da imagem);
2. leia as H linhas seguintes da matriz binária (0s e 1s separados por espaço), usando `mm.readImg(H, W)`;
3. aplique filtragem morfológica para remover o ruído sal e pimenta;
4. imprima a matriz já filtrada, em 0s e 1s separados por espaço;
5. extraia as medidas geométricas dos objetos com `mm.measure(imgLimpa)`;
6. ordene os objetos e imprima a tabela de medidas no formato esperado.

O enunciado ainda traz duas observações importantes para a correção automática: (i) a área calculada

pelo OpenCV (`cv2.contourArea`) corresponde ao polígono contínuo delimitado pelos centros dos pixels de borda, sendo por isso menor do que a simples contagem de pixels 1 (`np.sum`); e (ii) a lista de objetos deve ser ordenada em ordem crescente pela coordenada X do bounding box (`bbox[0]`), usando a coordenada Y (`bbox[1]`) como critério de desempate, com os IDs reatribuídos sequencialmente de 1 a N após a ordenação.

Assim como no exemplo da Seção A.3, o texto do enunciado aqui também não é fixo: ele é montado dentro do próprio bloco `[[def: ... ]]`, na variável Python `texto`, e desempenha o mesmo papel duplo já descrito — alimenta o PDF via `[[code:texto]]` e é exportado, já convertido por `latex_to_text()`, em `cases['description']` dentro de `moodle_cases`, sendo essa a versão exibida ao estudante no VPL do Moodle. A diferença é que, neste caso real, é o **cenário visual** (imagem ruidosa, número e posição dos objetos) que muda de estudante para estudante a cada variação, enquanto a redação de `texto` permanece fixa entre variações, já que apenas os dados numéricos (imagem e casos de teste) são sorteados. O ideal seria que a redação também variasse a cada geração, como no exemplo anterior, em que o valor de `height` era interpolado diretamente no texto da descrição. A estrutura real utilizada (com trechos de sorteio de cenário omitidos por brevidade) é:

```
[[code:texto]]

\vspace{2mm}\noindent\textbf{Exemplo de Entrada:}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_inp]]
\end{verbatim}
\vspace{-2mm}\noindent\textbf{Exemplo de Saída:}\vspace{-2mm}\
\begin{verbatim}
[[code:caso0_out]]
\end{verbatim}

\begin{comment}
[[code:moodle_cases]]
\end{comment}

[[def:
import json
import numpy as n
import cv2
from topic.morph import mm # TEM QUE INCLUIR "topic." no MCTest

# ... geração da cena, do ruído e dos 10 casos de teste (inplist/outlist) ...

# ENUNCIADO COMPLETO, COM EXPLICAÇÃO DE ÁREA E DE ORDENAÇÃO
texto = (
    "Uma imagem binária corrompida por ruído sal e pimenta contém no mínimo dois objetos
    geométricos isolados.\n\n"
    "Escreva um programa que:\n"
    "1. Leia dois inteiros ..."
)

cases = {}
cases['skills']      = ["morfologia matematica", "remocao de ruido", "mm.measure",
"tabulacao"]
cases['description'] = [{"text": latex_to_text(texto)}]
cases['input']       = np.array(inplist).tolist()
cases['output']      = np.array(outlist).tolist()

moodle_cases = json.dumps(cases)
caso0_inp = cases['input'][0]
caso0_out = cases['output'][0]
]]
```

Abaixo, a fotografia da questão efetivamente gerada e impressa para uma das 110 variações do Simulado

4, mostrando o enunciado, a imagem de entrada ruidosa e o exemplo de saída esperado (imagem filtrada e tabela de medidas):

1. Uma imagem binária corrompida por ruído sal e pimenta contém no mínimo dois objetos geométricos isolados.

Escreva um programa que: 1. Leia dois inteiros na primeira linha representando a altura (H) e largura (W) da imagem. 2. Leia as H linhas seguintes da matriz binária (valores 0 e 1 separados por espaço) usando mm.readImg(H, W). 3. Aplique a filtragem morfológica para eliminar o ruído sal e pimenta. 4. Imprima a matriz limpa diretamente em valores 0 e 1 (0s e 1s separados por espaço). 5. Extraia as medições geométricas dos objetos usando mm.measure(imgLimpa). 6. Ordene a lista de objetos e imprima a tabela formatada das medidas.

Observação sobre a área: A área calculada pelo OpenCV (cv2.contourArea) mede o polígono contínuo delimitado pelos centros dos pixels de borda, resultando em um valor menor do que a simples contagem discreta de pixels 1 (np.sum).

Regra de Ordenação: A lista de objetos deve ser ordenada em ordem crescente pela posição X do retângulo delimitador (bbox[0]), utilizando a posição Y (bbox[1]) como critério de desempate caso dois objetos comecem na mesma coluna (medidas.sort(key=lambda m: (m['bbox'][0], m['bbox'][1]))). Em seguida, reatribua os IDs sequencialmente de 1 a N.

Exemplo de Entrada:

```
12 43
0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Exemplo de Saída:

```
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

id	area	perimeter	cx	cy	x	y	w	h	circularity	solidity	vertices
1	20	18.0	6.5	2.0	4	0	6	5	0.78	1.000	4
2	20	18.0	26.0	2.5	24	0	5	6	0.78	1.000	4

Figura A.2: Questão real gerada pelo MCTest para o Simulado 4 — tópico "Operadores Morfológicos".

A.5 Considerações finais

Este apêndice resumiu os passos para instalar o MCTest, criar exames e construir questões parametrizadas com integração ao VPL do Moodle. O uso combinado de variáveis sorteadas na descrição (`[[code:...]]`) e de código Python embutido na própria questão (`[[def: ... ]]`) permite gerar centenas de variações de prova a partir de um único modelo, cada uma corrigida automaticamente e de forma consistente — com a variável `texto`, em particular, garantindo que o mesmo enunciado seja exibido tanto no PDF impresso quanto na avaliação da questão pelo estudante no Moodle. O exemplo real do Simulado 4 (Seção A.4) ilustra como esse mecanismo é usado na prática para gerar questões de programação com cenários visuais distintos por estudante. O Apêndice B detalha como essas variações são publicadas e corrigidas no Moodle, e o Apêndice C descreve como restringir o acesso às provas por meio do SEB.

Appendice B

Sobre o Moodle (VPL)

Este apêndice descreve a configuração de uma atividade **VPL (*Virtual Programming Lab*)** no Moodle, *plugin* utilizado para a submissão e a correção automática dos Exercícios de Programação (EPs), dos simulados quinzenais e das provas descritas neste livro. A versão do Moodle utilizada foi a **4.5**.

B.1 Configurando uma atividade VPL

A criação de um exame no MCTest com integração ao VPL (ver Apêndice A) gera dois arquivos que devem ser renomeados para `linker.json` e `students_variations.csv`. Esses arquivos, juntamente com o `morph.py`, devem ser adicionados aos **Arquivos de Execução** da atividade VPL, além de todos os arquivos contidos em `VPL_modification/V14-AI-feedback.zip`, disponível no repositório do MCTest em https://github.com/fzampirolli/mctest. Todos eles devem ser marcados com a opção **"*Files to keep when running*"**.

Em síntese, a atividade VPL passa a conhecer, para cada estudante, qual variação da questão foi sorteada (por meio de `students_variations.csv`) e qual é o conjunto de casos de teste correspondente (por meio de `linker.json`). Isso permite que a correção seja individualizada, mesmo quando dezenas de estudantes resolvem a mesma atividade com variações distintas.

B.2 Publicando os EPs como atividades VPL

Os Exercícios de Programação (EPs) apresentados ao final de cada capítulo deste livro também podem ser publicados como atividades VPL, seguindo o mesmo mecanismo. O fluxo típico de uso em sala de aula foi:

1. Abertura dos dois *notebooks* Colab do capítulo (teórico e prático), com destaque para os simuladores interativos;
2. Execução dos blocos de código, alterando parâmetros para reforçar a compreensão dos conceitos;
3. Nas aulas em laboratório, submissão das respostas dos EPs diretamente na atividade VPL correspondente, com retorno imediato sobre a aprovação ou reprovação nos casos de teste.

Caso a atividade VPL não seja gerada pelo MCTest, os arquivos `linker.json` e `students_variations.csv` não serão necessários. Entretanto, para utilizar o *feedback* por IA (ver Apêndice D), devem ser adicionados os arquivos contidos em `VPL_modification/V14-EP0_VPL-TEMPLATE.zip`, disponível no mesmo repositório (https://github.com/fzampirolli/mctest), seguindo o procedimento descrito na Seção B.1.

> **Reaproveitamento dos casos de teste**
>
> Os arquivos `.cases` utilizados na validação local (classe `TestSuite`, descrita no corpo deste livro) são compatíveis com o formato esperado pelo VPL, permitindo reaproveitar o mesmo conjunto de testes tanto no desenvolvimento do EP quanto na correção automatizada no Moodle.

Nos simulados quinzenais — aplicados com o SEB (ver Apêndice C) e com bônus de 5% sobre a nota final —, foram utilizadas atividades VPL com EPs semelhantes aos das listas regulares, porém corrigidas sob restrição de acesso à internet, reforçando o caráter avaliativo da atividade.

B.3 Considerações finais

Este apêndice descreveu a configuração de atividades VPL no Moodle para a correção automática de EPs, simulados e provas parametrizadas. A combinação entre MCTest (geração de variações e casos de teste), VPL (submissão e correção) e SEB (restrição de acesso) forma o tripé sobre o qual se apoia a avaliação automatizada e individualizada utilizada ao longo da disciplina.

Appendice C

Uso do SEB nos simulados quinzenais e avaliações

Além das duas provas com questões parametrizadas (ver Apêndice A), o ***Safe Exam Browser*** **(SEB)** também foi utilizado para restringir o acesso aos **simulados quinzenais** — atividades VPL (*Virtual Programming Lab*) com Exercícios de Programação (EPs) semelhantes aos das listas regulares, valendo um bônus de até 5% sobre a nota final —, bem como às demais avaliações práticas.

O SEB garante um ambiente seguro de avaliação ao bloquear o acesso a recursos externos não autorizados nas máquinas do laboratório. Para a implementação adotada neste trabalho, utilizou-se o sistema operacional **Windows 10** e a versão **3.7.1.704** do SEB, disponível em https://safeexambrowser.org/, que deve estar previamente instalada em todas as estações do laboratório.

A seguir, descreve-se o procedimento para configurar o arquivo de ambiente do SEB e vinculá-lo à atividade VPL no Moodle.

C.1 Configurando a aplicação no *SEB Tool*

Para gerar o arquivo de configuração `.seb` que será distribuído aos estudantes, abra o utilitário *SEB Tool* e siga os passos descritos a seguir.

1. ***General***
   - No campo ***Start URL***, insira a URL direta da atividade VPL no Moodle.
   - *Nota:* caso seja necessário navegar por mais de uma atividade dentro do mesmo curso, insira a URL principal do curso e, na aba ***Browser***, marque a opção ***Allow navigation back/forward in exam***.
2. ***Network***
   - Inclua as URLs permitidas para navegação. Para restringir o acesso à atividade desejada, utilize uma expressão de filtro no padrão `/mod/vpl/*?id=4624*`, em que o número final corresponde ao identificador da atividade VPL no Moodle. Na Tabella C.1, a última linha contém a URL da página principal do curso (identificador `475`), mas ela também pode ser substituída pela URL de uma seção que contenha várias atividades. Essa configuração é opcional e útil quando há mais de uma atividade VPL. Nesse caso, a primeira linha da tabela deve ser repetida para cada atividade avaliativa que deverá ficar acessível durante o *exame*.
   - **Configurações gerais da aba *Filter***:
     ☒ *Activate URL filtering*
     ☐ *Filter also embedded content*

Tabella C.1: Expressões de filtro de URL utilizadas na configuração do SEB.

Active	*Regex*	*Expression*	*Action*
[x]	[ ]	`https://moodle.ufabc.edu.br/mod/vpl/*?id=4624*`	*Allow*
[x]	[ ]	`https://moodle.ufabc.edu.br/login/index.php*`	*Allow*
[x]	[ ]	`https://moodle.ufabc.edu.br/course/switchrole.php*`	*Allow*

Active	*Regex*	*Expression*	*Action*
[x]	[ ]	`https://mctest.ufabc.edu.br/compare`	*Allow*
[x]	[ ]	`https://moodle.ufabc.edu.br/enrol/index.php?id=475`	*Allow*

3. ***Security***
   - Caso o laboratório utilize um projetor ou monitor secundário, ajuste o campo ***Maximum allowed number of connected displays*** para um valor maior que 1, evitando que o SEB bloqueie a exibição.
4. ***Config File***
   - Clique em ***Save Settings As...*** e salve o arquivo com a extensão `.seb`.
5. ***Exam***
   - Marque a opção ***Use Browser Exam Key and Configuration Key***.
   - Copie o código gerado no campo ***Browser Exam Key***. *Importante:* copie essa chave somente **após** salvar o arquivo no passo anterior.

C.2 Vinculando a chave do SEB à atividade VPL no Moodle

Após gerar o arquivo e obter a *Browser Exam Key*, acesse o Moodle para aplicar as restrições.

1. Na página da atividade VPL, clique no ícone de engrenagem (**Configurações**).
2. Acesse ***Configuration*** → ***Submission restrictions*** e clique em ***Show more***.
3. Altere a opção ***SEB browser required*** para ***Yes***.
4. Cole a chave copiada no campo ***SEB exam key(s)***.

C.3 Restrição por endereço IP (opcional)

Para garantir que as submissões ocorram exclusivamente a partir dos computadores do laboratório, é possível combinar o SEB com o filtro de rede do Moodle, preenchendo o campo ***Allowed submission from net***.

- **Faixas consecutivas:** utilize um hífen para definir o intervalo. Exemplo: `111.11.11.1-62` (permite os IPs de `111.11.11.1` a `111.11.11.62`).
- **IPs não consecutivos:** informe os endereços individuais separados por vírgulas.

C.4 Considerações finais

A integração do SEB com o VPL no Moodle cria um fluxo simples para o estudante: basta dar um duplo clique no arquivo `.seb` disponibilizado pelo docente para que o navegador seguro seja aberto diretamente na atividade configurada.

A combinação da *Browser Exam Key* com a restrição por faixa de endereços IP das máquinas do laboratório fornece uma solução robusta para avaliações de programação *online*, dificultando acessos indevidos e contribuindo para a integridade do processo de avaliação.

Appendice D

Geração de material didático com o *Gemini Notebook*

Este apêndice descreve o uso do ***Gemini Notebook*** (***NotebookLM***) como ferramenta de apoio à preparação do material didático da disciplina, complementando o uso de IA na revisão de textos e códigos, mencionado no prefácio deste livro.

D.1 Vídeos conceituais e de resolução de EPs

Para cada capítulo, foi criado um projeto no *Gemini Notebook* tendo como fontes o PDF completo do livro e o arquivo `morph.py`. A partir dessas fontes, o *Gemini Notebook* gerou automaticamente um vídeo curto (em média, 7 minutos), utilizado na abertura das aulas. Os capítulos alternaram dois tipos de vídeo:

- **Vídeo conceitual**, apresentando os fundamentos teóricos do capítulo, geralmente exibido na primeira aula dedicada ao tema;
- **Vídeo de resolução de EPs**, apresentando uma estratégia de solução para os Exercícios de Programação, exibido na aula seguinte, quando a maior parte dos EPs era resolvida em conjunto com a turma.

> **Papel do vídeo na aula**
>
> O vídeo gerado pelo *Gemini Notebook* não substitui a explicação do professor. Ele funciona como uma breve introdução e motivação ao tema, contextualizando o estudante antes da apresentação dos *slides* e da abertura dos *notebooks Colab*.

D.2 *Slides* de apoio

Complementando os vídeos, o *Gemini Notebook* também gerou, a partir das mesmas fontes (o PDF do livro e o arquivo `morph.py`), um conjunto de aproximadamente 12 *slides* por capítulo, abrangendo tanto os conceitos teóricos quanto a parte prática. A geração utilizou um *prompt* específico para cada capítulo, delimitando o escopo do conteúdo a ser sintetizado e evitando tanto a antecipação de assuntos de capítulos posteriores quanto a reprodução de trechos extensos do livro sem adaptação didática.

Após a apresentação dos *slides*, a aula prosseguia com a abertura dos dois *notebooks Colab* do capítulo (teórico e prático), enfatizando os simuladores interativos e a execução dos blocos de código com alteração de parâmetros.

D.3 Material principal e considerações finais

A geração de vídeos e *slides* com o *Gemini Notebook*, a partir do próprio livro e da biblioteca `morph.py`, permitiu padronizar a abertura das aulas e reduzir o tempo de preparação do material didático, sem prescindir da curadoria do professor na elaboração dos *prompts* e na condução das atividades em sala de aula.

Os vídeos e *slides* produzidos pelo *Gemini Notebook* têm caráter **motivacional** e servem como ponto de partida para a discussão dos conceitos apresentados em cada capítulo. Como qualquer conteúdo gerado por IA, eles podem conter imprecisões ou erros ocasionais. Em alguns casos, essas imprecisões são exploradas intencionalmente durante a aula para verificar se os estudantes estão acompanhando criticamente a apresentação e conseguem identificar inconsistências conceituais.

O conteúdo oficial da disciplina, entretanto, é o material produzido pelo método descrito no prefácio deste livro, disponibilizado em três formatos complementares: **HTML**, para navegação com simuladores interativos; **PDF**, para leitura e impressão; e ***notebooks Jupyter*** (`.ipynb`), para execução e experimentação dos códigos. Todo o material gerado pelo *Gemini Notebook* deve ser entendido como um recurso complementar a esse conteúdo principal.

Assim como nos demais usos de IA generativa descritos neste livro, a responsabilidade pela qualidade técnica, pela correção conceitual e pela adequação pedagógica do material permanece com o professor.

Appendice E

Uso de IA na Avaliação de Estudantes: o Projeto vpl-ai-feedback

Este apêndice descreve o **vpl-ai-feedback**, sistema de código aberto desenvolvido para apoiar a correção de submissões de código enviadas ao VPL (Moodle) por meio de *feedback* gerado por Inteligência Artificial (IA). Diferentemente de um corretor automático tradicional — que apenas compara saídas com casos de teste —, o **vpl-ai-feedback** envia o código do estudante, junto com a rubrica da questão, a um modelo de linguagem (LLM), que devolve uma análise qualitativa por critério, no estilo de um ***feedback* socrático**: o estudante recebe comentários que apontam o que foi feito corretamente, o que pode ser melhorado e por quê, tanto em atividades **formativas** (exercícios de prática) quanto em atividades **avaliativas** (simulados) do VPL.

Esse mecanismo de *feedback* socrático em atividades VPL foi proposto por Zampirolli et al. (2026). O estudo descreve a integração de IA ao processo de avaliação de exercícios de programação parametrizados no VPL/Moodle, utilizando um conjunto de sete LLMs adotados para analisar o código dos estudantes e gerar *feedback* automatizado a cada solicitação de correção, com resultados preliminares indicando percepção positiva dos estudantes quanto à geração de ideias, clareza das explicações e autonomia de aprendizagem.

O restante deste apêndice detalha a implementação prática dessas ideias no projeto **vpl-ai-feedback** — cujo código está disponível em https://github.com/fzampirolli/vpl-ai-feedback — e descreve especificamente seu uso nas três turmas de PDI-VC entre maio e agosto de 2026.

> ⚠ Atenção: IA não substitui o julgamento docente
>
> O uso de IA para **avaliar** estudantes **não é recomendável como fonte única de nota**, pois modelos de linguagem podem **alucinar** — isto é, produzir avaliações, critérios ou justificativas plausíveis, porém incorretas, mesmo diante de um código correto ou incorreto. Um LLM pode atribuir nota alta a um código com erro sutil, ou nota baixa a um código correto mas escrito de forma incomum, simplesmente porque a explicação gerada "parece" coerente. **A IA deve ser usada apenas como um complemento à avaliação**, nunca como substituta do professor. O Apêndice A e o Apêndice B descrevem os mecanismos de correção objetiva (casos de teste) que continuam sendo a base da nota oficial; este apêndice trata exclusivamente do uso da IA como camada adicional de *feedback* qualitativo.

E.1 Contexto de uso

O **vpl-ai-feedback** foi utilizado em três turmas da disciplina **PDI-VC**, ministradas entre maio e agosto de 2026, totalizando aproximadamente uma centena de estudantes. O *feedback* por IA foi empregado de três formas complementares:

1. ***Feedback* contínuo em atividades formativas** — exercícios de prática enviados ao VPL ao longo do curso, nos quais o estudante podia reenviar o código quantas vezes desejasse. A cada submissão, recebia um *feedback* qualitativo gerado por IA, além da correção objetiva realizada pelo VPL, descrita em Zampirolli et al. (2026).

2. ***Feedback* complementar em simulados (atividades avaliativas bônus)** — quatro simulados realizados ao longo do quadrimestre, previstos no plano de ensino como atividades bônus, cada um

valendo até 5% da nota final. Aplicados aproximadamente 30 minutos antes do término das aulas práticas quinzenais em laboratório, os simulados utilizavam o **vpl-ai-feedback** para gerar uma rubrica individual de avaliação, enviada por e-mail juntamente com um resumo comparativo entre a nota atribuída pelo Moodle/VPL e a avaliação produzida pela IA.

3. ***Feedback* complementar nas Provas 1 e 2 (avaliações oficiais)** — as duas provas formais do quadrimestre, cada uma valendo uma parcela maior da nota final da disciplina, também passaram a utilizar o **vpl-ai-feedback** após sua aplicação. Diferentemente dos simulados (bônus), aqui a nota oficial já é definida pela correção objetiva do VPL e/ou avaliação manual do professor antes da divulgação; a rubrica gerada pela IA é enviada ao estudante apenas posteriormente, como material de apoio para revisão do próprio desempenho — reforçando, também nas avaliações de maior peso, a separação entre *nota oficial* e *feedback* complementar detalhada na Seção D.7.

Uma pesquisa de opinião, respondida por 102 estudantes antes da adoção do sistema, indicou elevada aceitação da proposta. Apenas 2 estudantes atribuíram nota 1 (rejeição) e 7 atribuíram nota 2 ao interesse em receber *feedback* automático de código por IA. Outros 19 declararam-se indiferentes (nota 3), enquanto a maioria — 38 e 37 estudantes — atribuiu notas 4 e 5 (alta aprovação), respectivamente, totalizando os 102 respondentes. Esse resultado motivou a adoção do sistema como complemento ao processo formativo, e não como substituto da correção humana.

! A nota oficial nunca foi a nota da IA

É fundamental destacar que, conforme definido no plano de ensino, **a nota utilizada tanto para o cálculo do bônus de cada simulado quanto para a nota oficial das Provas 1 e 2 foi sempre a nota atribuída pelo VPL** (baseada nos casos de teste) e/ou pela avaliação manual do professor — e **não** a nota sugerida pela IA. A rubrica gerada pelo **vpl-ai-feedback** foi enviada ao estudante apenas como material de apoio ao aprendizado — nunca como critério de atribuição de nota, seja em atividades bônus ou em avaliações oficiais. Essa separação entre *nota oficial* (VPL/professor) e *feedback complementar* (IA) é o princípio central deste apêndice e foi retomada na Seção D.8.

E.2 Arquitetura do projeto

O **vpl-ai-feedback** é um *script* Python assíncrono organizado da seguinte forma:

```
.
  config.yaml              ← configuração (credenciais, provedor, pesos) - NUNCA versionado
  config.yaml.example      ← template público de configuração
  main.py                  ← script principal (async), orquestra todo o processo
  run.sh                   ← wrapper de execução (bash run.sh config.yaml)
  gerar_relatorio.py       ← converte o consolidado *_ALL.txt em CSV
  enviar_email.py          ← envia as rubricas por e-mail a cada estudante
  providers/                ← clientes das APIs de LLM
      __init__.py          ← fábrica de clientes (Factory)
      base.py               ← lógica de retry, backoff e fallback entre modelos
      groq.py
      deepseek.py
      gemini.py
  core/                      ← lógica de negócio
      grader.py              ← identifica arquivos por questão, monta o prompt, chama a LLM
      utils.py
  <pasta_da_turma>/          ← submissões baixadas do Moodle VPL
      Nome estudante - login/
          <timestamp>/                 ← última submissão do estudante
              Q1.py                    ← padrão para provas com múltiplas questões
              Q2.py
              rubrica.txt              ← gerado pela LLM (somente se avaliação válida)
```

```
        <timestamp>.ceg/
            execution.txt            ← nota/objetiva atribuída pelo VPL
    ...
```

O sistema aceita três provedores de LLM — **Groq**, **DeepSeek** e **Gemini** — configuráveis em `config.yaml` por meio da chave `llm.provider`. Cada provedor pode ter **múltiplos modelos** cadastrados; a cada chamada, a lista é **embaralhada**, distribuindo a carga entre os estudantes processados em paralelo e reduzindo a chance de erros de limite de requisições (HTTP 429). Quando um modelo falha, o sistema tenta até três vezes antes de passar ao próximo da lista, respeitando o tempo de espera sugerido pelo provedor. Erros de autenticação (401) ou de saldo insuficiente (402) interrompem imediatamente a tentativa naquele provedor.

Um aspecto importante do projeto diz respeito à privacidade dos dados. **Nome, e-mail, login, RA e CPF do estudante nunca são enviados à API do LLM**. São transmitidos para processamento apenas: o nome do arquivo contendo o código-fonte; o próprio código (com os comentários removidos); o *prompt* da rubrica — que não contém dados pessoais —; e, quando disponíveis no `execution.txt` gerado pelo VPL, o enunciado oficial da questão sorteada para aquele estudante e o resultado bruto dos casos de teste executados (entrada, saída produzida pelo código e saída esperada). Esses dois últimos itens, embora extraídos de um arquivo específico de cada estudante, também não contêm identificação pessoal — apenas o texto da questão e os valores de entrada/saída dos testes automáticos —, e são usados como evidência objetiva para reduzir o risco de a IA identificar incorretamente o tipo de questão ou avaliar a lógica do código apenas por leitura estática (Seção D.3).

Ainda assim, é importante reconhecer que os três provedores atualmente suportados — Groq, DeepSeek e Gemini — são serviços comerciais operados por empresas estrangeiras, cujos modelos são executados em infraestrutura fora do país e sujeitos a políticas de uso, disponibilidade e custo que fogem ao controle da instituição de ensino. Essa dependência de provedores externos traz riscos que vão além da correção de código pontual: mudanças de preço, descontinuação de modelos, indisponibilidade temporária do serviço, alterações unilaterais nos termos de uso e, em casos mais sensíveis, restrições geopolíticas de acesso podem comprometer a continuidade de um sistema de avaliação que passe a depender estruturalmente dessas APIs. Por isso, é estratégico que Instituições de Ensino Superior (IES) como a UFABC invistam no desenvolvimento e na hospedagem de **provedores próprios de IA** — por exemplo, modelos abertos (*open-weight*) executados em infraestrutura local ou em nuvem soberana —, reduzindo a dependência de serviços externos, especialmente de outros países, e ampliando o controle institucional sobre custos, disponibilidade, privacidade dos dados de estudantes e continuidade pedagógica de longo prazo. O desenho do **vpl-ai-feedback** já favorece esse caminho: a arquitetura de `providers/` foi construída como uma fábrica (*factory*) extensível, na qual um novo provedor — inclusive um modelo hospedado localmente pela própria instituição, com interface compatível — pode ser adicionado com baixo esforço de integração.

E.3 O *prompt* universal e a rubrica em duas etapas

Cada tipo de prova é descrito em um arquivo de *prompt* (por exemplo, `Simulado4.txt`), referenciado em `grading.prompt_file` no `config.yaml`. Esse arquivo instrui a LLM a realizar a avaliação em **duas etapas**:

1. **Identificação da questão** — a LLM determina o tipo específico de operação sorteado para aquele estudante (útil quando há questões paramétricas sorteadas e embaralhadas por estudante, como descrito no Apêndice A);
2. **Aplicação da rubrica correspondente** — a LLM avalia o código segundo critérios pontuados, cada um com uma faixa máxima de pontos, retornando ao final uma nota no formato `NOTA FINAL: X/PESO`.

Em provas parametrizadas, o arquivo de *prompt* costuma conter **várias rubricas paralelas**, uma para cada tipo de questão possível, delimitadas por marcadores como `[START_RUBRICA_TIPO_A]` / `[END_RUBRICA_TIPO_A]`. A LLM identifica primeiro qual tipo foi sorteado para aquele estudante e aplica **apenas** a rubrica correspondente, ignorando as demais — o que evita que critérios de um tipo diferente contaminem a nota.

Quando a prova possui mais de uma questão por atividade VPL, os arquivos de código devem seguir o

padrão `Q1.*`, `Q2.*` etc. — um arquivo por questão, com extensão livre conforme a linguagem escolhida pelo estudante —, e cada questão tem peso configurável individualmente em `grading.weights` no `config.yaml`. Quando a prova tem uma única questão e não foi gerada pelo MCTest, o sistema aceita qualquer nome de arquivo com extensão suportada, desde que seja o único arquivo de código presente na pasta de submissão.

No caso do Simulado 4 (tópico *Operadores Morfológicos*), por exemplo, um dos tipos possíveis definia três critérios, com peso total de 100 pontos para aquela questão:

Critério	Descrição	Pontuação máxima
1 — Entrada de dados	Leitura de `H`, `W` e da matriz binária (`mm.readImg`)	25 pts
2 — Processamento morfológico	Abertura + Fechamento (`mm.sebox()`), `mm.measure`, ordenação por `bbox[0]`/`bbox[1]` e reatribuição de IDs	50 pts
3 — Saída e formatação	Exibição da imagem limpa (`mm.drawImg`) e tabela de medidas formatada	25 pts

O *prompt* exige ainda um **formato de resposta obrigatório** (largura de linha, ordem das seções, marcação `NOTA FINAL:`), o que torna a extração automática da nota e a montagem do relatório consolidado mais confiáveis — embora, como discutido na Seção D.8, isso não elimine o risco de erro semântico na avaliação do conteúdo.

E.4 Duas camadas adicionais de evidência

Para reduzir o risco de a LLM alucinar o tipo de questão ou a corretude do código — o risco central discutido na Seção D.7 —, o **vpl-ai-feedback** passou a fornecer à IA duas fontes de evidência objetiva, extraídas automaticamente do próprio `execution.txt` gerado pelo VPL, além do código-fonte:

- **Enunciado oficial da questão.** Como as questões costumam ser sorteadas e embaralhadas por estudante no MCTest (Apêndice A), o `execution.txt` de cada estudante já traz, no campo "Descrição resumida da questão", o texto exato da questão que caiu para ele. O sistema extrai esse trecho automaticamente e o envia à LLM como evidência **primária** para identificar o tipo/variação da questão — mais confiável do que inferir apenas pela leitura estática do código, especialmente em submissões incompletas ou ambíguas entre dois tipos próximos (por exemplo, erosão "plana" *versus* "com pesos"). Quando o código entregue diverge do que o enunciado pede, o sistema não ignora o enunciado nem "corrige" a leitura silenciosamente: mantém a identificação pelo enunciado, reduz a confiança da avaliação para "Baixa" e explicita a divergência no *feedback* enviado ao estudante.
- **Resultado real da execução no VPL.** Quando disponível, o sistema também envia à LLM os casos de teste efetivamente executados sobre aquele código — entrada, saída produzida e saída esperada —, como evidência objetiva de corretude, a ser usada para confirmar ou refutar a leitura da lógica antes de pontuar os critérios de processamento e de saída, em vez de a IA depender apenas de simular mentalmente o código (o que é especialmente falho em laços com `min`/`max`, deslocamento de índice ou cálculo de limiar de Otsu, onde erros sutis passam despercebidos numa leitura estática).

Além disso, a LLM é instruída a declarar explicitamente seu **nível de confiança** (Alta, Média ou Baixa) na identificação de cada tipo de questão, com justificativa quando a confiança não for Alta. Esse valor é extraído automaticamente e alimenta uma coluna `Revisar_Manualmente` no relatório CSV consolidado (Seção D.4), permitindo ao professor priorizar exatamente os casos em que a própria IA reconhece maior incerteza — em vez de tratar todas as avaliações de uma turma como igualmente confiáveis.

E.5 Fluxo de execução

A execução típica, feita por `bash run.sh config.yaml`, segue os passos:

1. Carrega `config.yaml` e localiza a pasta de submissões (`paths.student_base_dir`);
2. Para cada estudante, identifica a **última submissão** (pasta com o *timestamp* mais recente);
3. Extrai a nota objetiva do Moodle a partir de `*.ceg/execution.txt`;
4. Para cada questão da prova, localiza o arquivo de código (`Q1.*`, `Q2.*`, ou o único arquivo, no caso de prova que não foi gerada pelo MCTest), monta o *prompt* (código + rubrica) e envia a um modelo sorteado da lista configurada;
5. Processa todos os estudantes **em paralelo**, com controle de concorrência;
6. Gera, por estudante, o arquivo `rubrica.txt` — **somente se a IA retornar ao menos uma avaliação válida** para as questões que possuem código; caso contrário, o arquivo não é salvo, forçando nova tentativa na próxima execução;
7. Consolida todos os `rubrica.txt` em um único `*_ALL.txt` e gera um relatório `*_relatorio.csv`, comparando nota do Moodle, nota da IA e a diferença entre ambas, por questão e no total.

Situação	`rubrica.txt` é salvo?
Todas as questões com código avaliadas com sucesso	✅ Sim
IA falhou em ao menos uma questão com código	❌ Não — reprocessa na próxima execução
estudante sem nenhum arquivo de código enviado	❌ Não
Tipo de questão identificado como DESCONHECIDO	❌ Não

E.6 Exemplo ilustrativo: rubrica de uma prova com três questões

ℹ Sobre a origem deste exemplo

O trecho a seguir **não reproduz o código de nenhum estudante específico**. Trata-se de um exemplo reconstruído pelo autor, que reproduz um *padrão* de erro observado repetidamente ao longo das turmas de PDI-VC — a confusão entre uma função morfológica 2D (`mm.ero`/`mm.ero0`) e uma operação 1D sobre sinais — sem citar ou identificar qualquer submissão real. Essa opção evita qualquer risco de violação do direito autoral do estudante sobre seu próprio código-fonte, ainda que anonimizado, já que a titularidade da obra permanece do autor original mesmo quando nome e *login* são removidos.

O trecho a seguir ilustra a saída gerada pelo **vpl-ai-feedback** para um estudante de uma prova com **três questões** (Q1, Q2, Q3), usando o modelo `deepseek-chat`. O resumo comparativo aparece no topo do arquivo, seguido do enunciado oficial de cada questão, do código submetido e da avaliação por critério.

```
RESUMO - IA (deepseek-chat)  x  MOODLE

Peso total : 100 pts

Q1 (IA) :  3.3 / 33 pts
Q2 (IA) :  6.7 / 33 pts
Q3 (IA) : 33.3 / 34 pts

Moodle : (Q1=33 + Q2=0 + Q3=34) = 67 pts
IA     : 43.3 / 100 pts

Diferença (IA - Moodle): -23.7 pts

Q1: tipo identificado = B  | confiança = Baixa (código não
```

```
    corresponde ao enunciado da questão)  REVISAR
  Q2: tipo identificado = A  | confiança = Alta (enunciado oficial
    confirma o tipo e a variação)
  Q3: tipo identificado = C1 | confiança = Alta (enunciado oficial
    confirma erosão 2D com OpenCV integrada)
```

Q1 — código não corresponde ao enunciado (confiança baixa): o enunciado oficial da questão, extraído do `execution.txt`, pedia uma erosão morfológica **1D** sobre um sinal (leitura de N, M, o sinal e o elemento estruturante B, seguida do mínimo dos vizinhos ativos). O código submetido, no entanto, lê apenas dois inteiros e invoca `mm.ero0` — uma função de erosão **2D** para imagens, não relacionada à operação pedida.

```
Tipo identificado: TIPO B (Erosão 1D - plana)
Confiança: Baixa - código não corresponde ao enunciado da questão (usa
mm.ero0, função de erosão 2D, em vez de implementar erosão 1D plana
sobre um sinal)

Critério 1 - [3.33/10.00 pts]:
- O código lê dois inteiros em uma única linha (que corresponderiam a
  N e M), mas não lê o sinal nem o elemento estruturante B como vetores
  1D. Em vez disso, trata ambos como imagens via mm.readImg, o que não
  corresponde à leitura esperada de um sinal 1D e de B em linhas
  separadas.

Critério 2 - [0.00/16.67 pts]:
- A operação aplicada é mm.ero0(img, element), erosão 2D para imagens,
  não a erosão 1D de sinais pedida no enunciado. A lógica de mínimo dos
  vizinhos ativos com bordas por recorte não está implementada.

Critério 3 - [0.00/6.67 pts]:
- A saída é uma imagem 2D via mm.drawImg, não o sinal 1D resultante em
  uma única linha com valores inteiros separados por espaço, como
  exigido.

NOTA FINAL: 3.33/33.33

Comentários para o estudante: O código apresentado não corresponde ao que o
enunciado pede. A questão sorteada solicita uma erosão morfológica 1D
sobre um sinal unidimensional; o código usa uma função de erosão 2D
para imagens. Releia atentamente o enunciado sorteado antes de escolher
a implementação.
```

Q2 — erro de sintaxe impede a execução (confiança alta, nota baixa por falha real): aqui a IA identificou corretamente o tipo (limiarização com Otsu), com apoio do enunciado oficial, mas o código continha um erro de indentação que impedia a execução — evidenciado tanto pela leitura estática quanto pela ausência de saída válida nos testes do VPL.

```
Tipo identificado: TIPO A (Limiarização - T manual + Otsu)
Confiança: Alta - enunciado oficial confirma o tipo e a variação

Critério 1 - [6.67/10.00 pts]:
- Lê corretamente os três parâmetros L, C e T em linhas separadas e a
  imagem de entrada. Porém, a definição de uma função auxiliar de
  limiarização está indentada incorretamente dentro do escopo global,
  causando um erro de indentação que impede a execução completa do
  programa.

Critério 2 - [0.00/16.67 pts]:
- A binarização com limiar fixo T está corretamente estruturada, mas o
  cálculo do limiar de Otsu depende da função auxiliar malformada, que
```

```
  referencia uma variável fora de escopo. O erro de indentação impede
  qualquer processamento.

Critério 3 - [0.00/6.67 pts]:
- Devido ao erro de indentação, nenhuma saída é produzida.

NOTA FINAL: 6.67/33.33

Comentários para o estudante: A estrutura geral está correta (leitura de L,
C, T e da imagem), mas há um erro de indentação que impede a execução.
Revise a sintaxe Python antes de submeter e teste localmente o código
antes do envio ao VPL.
```

Q3 — divergência de função, mas equivalência funcional confirmada por evidência real (confiança alta, nota justa): o enunciado pedia explicitamente o uso de `mm.ero` (versão integrada ao OpenCV), mas o código usa `mm.ero0` (versão plana implementada manualmente). Como o `execution.txt` mostrava os 10 casos de teste do VPL passando integralmente, a IA não penalizou a divergência de função — reconheceu a equivalência funcional demonstrada pelos testes reais, mas ainda assim registrou a observação no *feedback*.

```
Tipo identificado: TIPO C1 (Erosão 2D - plana sem pesos)
Confiança: Alta - enunciado oficial confirma erosão 2D com OpenCV
integrada (mm.ero), e o código implementa erosão plana via mm.ero0

Critério 1 - [10.00/10.00 pts]:
- Lê corretamente altura, largura, a matriz da imagem e as dimensões e
  valores do elemento estruturante B.

Critério 2 - [16.67/16.67 pts]:
- O enunciado pedia o uso de mm.ero (OpenCV integrada), mas o código
  usa mm.ero0 (plana sem pesos). Apesar da divergência de função, a
  operação foi implementada corretamente com o elemento estruturante
  lido: os 10 testes executados no VPL passaram integralmente (10/10),
  confirmando equivalência funcional para os casos testados.

Critério 3 - [6.67/6.67 pts]:
- Imprime a imagem resultante no formato correto; os testes confirmaram
  a saída correta.

NOTA FINAL: 33.34/33.33

Comentários para o estudante: O código está correto e funcional, com todos
os 10 testes passando. Observação: o enunciado pedia explicitamente
mm.ero (versão OpenCV), mas você usou mm.ero0 (versão plana). Embora os
resultados tenham sido equivalentes nos testes, siga exatamente a
função solicitada no enunciado em provas futuras.
```

Note que, nesse caso composto, a IA atribuiu **23,7 pontos a menos** do que a nota objetiva do VPL, puxada principalmente pela Q1. A diferença agregada por si só já seria um sinal para investigação (Seção D.7), mas o próprio sistema entrega ao professor a causa provável de cada questão: a coluna `Revisar_Manualmente` do CSV é marcada "SIM" sempre que ao menos uma questão do estudante recebe confiança baixa, com o motivo específico registrado na coluna de confiança correspondente — reduzindo o tempo que o professor precisa gastar para localizar, entre uma centena de estudantes, os poucos casos que realmente merecem atenção manual antes da atribuição da nota oficial. O caso da Q3, por sua vez, ilustra o cuidado inverso: a divergência entre enunciado e código **não** resultou em penalização indevida, porque a evidência real de execução confirmou a equivalência funcional — exatamente o comportamento que a Seção D.3 descreve para essas duas camadas de evidência.

E.7 Envio do *feedback* por e-mail

Após a geração das rubricas, o *script* `enviar_email.py` envia a cada estudante um e-mail com o `rubrica.txt` em anexo. O corpo do e-mail é definido em `config.yaml`, em `templates.corpo`, e é interpolado com `{nome_pasta}` e `{login}`:

```yaml
email:
  smtp_server: smtp.ufabc.edu.br
  smtp_port: 587
  from_address: professor@ufabc.edu.br
  password: "SUA_SENHA_AQUI"     # nunca versionar credenciais reais
  use_tls: true

templates:
  assunto: "Rubricas e Correções geradas por LLM - Simulado - {login}@aluno.ufabc.edu.br"
  corpo: |
    Prezado(a) {nome_pasta},
    ...
```

! Segurança de credenciais

O `config.yaml` contém segredos reais (senha de SMTP, chaves de API). Ele deve constar do `.gitignore` do projeto e **jamais** ser publicado, compartilhado ou versionado — inclusive em anexos de e-mail, capturas de tela ou repositórios públicos.

Um exemplo real de e-mail enviado a um estudante (dados **anonimizados**, com nome e login substituídos por um caso fictício) é reproduzido a seguir, para ilustrar o tom didático e as ressalvas explicitadas ao estudante:

! Exemplo de mensagem individual (anonimizada)

Prezado(a) [Nome do Estudante] - [login],
A sua nota do Simulado 4 está disponível no Moodle.
Envio abaixo as competências avaliadas e uma correção detalhada do seu código, gerada automaticamente por Inteligência Artificial (IA).
No anexo "rubrica.txt" está o retorno completo da IA (recomendo baixar e abrir com um editor de texto ou bloco de notas).
Ressalto que essa correção gerada por IA PODE conter imprecisões ou erros, mas serve como um excelente apoio ao seu processo de aprendizagem na disciplina.
Uma sugestão pedagógica é submeter os seus códigos (contidos neste arquivo), juntamente com a RUBRICA abaixo, a outros modelos de LLM para comparação. Isso pode ajudar a identificar possíveis divergências, além de oferecer diferentes perspectivas sobre o seu código e sobre os critérios de avaliação.
Caso tenha dúvidas ou note alguma inconsistência gritante na correção apresentada no Moodle, estou à disposição para esclarecimentos.
Atenciosamente,
Prof. Francisco Zampirolli
PS.: Explicando o processo: a última versão do seu código salva no Moodle foi anexada ao prompt e enviada para um dos LLMs escolhidos de forma integrada ao nosso sistema de avaliação (modelos = ("deepseek-chat")). O processo é repetido até que seja obtida uma resposta com nota válida (entre 0 e 100).

Três elementos didáticos merecem destaque nesse texto: (i) o alerta explícito de que a correção **pode conter erros**; (ii) o convite para que o próprio estudante **contraste a avaliação com outros LLMs**, transformando a IA em ferramenta de estudo ativo em vez de veredito passivo; e (iii) a explicação transparente do processo automatizado, incluindo o(s) modelo(s) usado(s) e a política de reprocessamento até obter uma nota válida.

E.8 Riscos, limites éticos e responsabilidade docente

! O professor não pode simplesmente adotar a nota da IA

Este é o ponto central deste apêndice: **em nenhuma hipótese a nota sugerida pela IA deve ser atribuída automaticamente ao estudante como nota oficial**. Modelos de linguagem podem alucinar — atribuir pontos a critérios não atendidos, "inventar" que uma função foi chamada corretamente quando não foi, ou penalizar um código correto por má interpretação da rubrica. A nota final de qualquer atividade avaliativa deve **sempre** resultar de avaliação manual do professor (ou da correção objetiva por casos de teste, como no VPL/MCTest), com a IA cumprindo, no máximo, o papel de **primeira leitura** ou de **material de apoio ao estudante**, nunca de instância decisória.

Alguns cuidados práticos adotados no uso do **vpl-ai-feedback**, e recomendados a qualquer professor que deseje adaptar o sistema:

- **Separação clara entre nota oficial e *feedback* de IA.** No caso relatado, a nota do bônus dos simulados sempre veio do VPL, nunca da IA (Seção D.1). A rubrica de IA foi comunicada ao estudante como material de apoio, com aviso explícito de que pode conter erros.
- **Transparência com o estudante.** O e-mail enviado (Seção D.6) explicita que a avaliação foi gerada por IA, informa o(s) modelo(s) utilizados e convida o estudante a comparar com outros LLMs — reduzindo a assimetria de informação e incentivando o pensamento crítico sobre a própria correção recebida.
- **Reprocessamento em vez de *cache* inválido.** O sistema só grava `rubrica.txt` quando obtém avaliação válida (Seção D.4); isso evita que uma resposta malformada, incompleta ou visivelmente inconsistente da IA seja apresentada ao estudante como se fosse definitiva.
- **Monitoramento das divergências.** O relatório CSV consolidado (coluna `Diferenca = Total_IA - Total_Moodle`) permite ao professor identificar rapidamente os casos em que IA e correção objetiva mais discordam — como no exemplo da Seção D.5 (+20 pontos) —, priorizando esses casos para revisão manual.
- **Canal aberto para contestação.** O e-mail final oferece explicitamente ao estudante a possibilidade de reportar inconsistências, mantendo o professor como autoridade final sobre a nota.
- **Dados pessoais fora do *prompt*.** Como descrito na Seção D.2, nome, e-mail, login, RA e CPF nunca são enviados à API do LLM, reduzindo riscos de privacidade no uso de provedores externos de IA.

E.9 Considerações finais

O **vpl-ai-feedback** mostra como a IA pode enriquecer o ciclo de *feedback* em disciplinas de programação com alto volume de submissões, oferecendo a cada estudante uma leitura qualitativa e individualizada do próprio código — algo inviável de se fazer manualmente para uma centena de estudantes, em múltiplas questões e provas ao longo do quadrimestre. A pesquisa de opinião citada na Seção D.1 sugere que esse tipo de retorno é bem recebido pelos estudantes. Ainda assim, o uso responsável do sistema depende de um princípio inegociável, reiterado ao longo deste apêndice: **a IA complementa, mas não substitui, a avaliação do professor**, e a nota oficial de qualquer atividade avaliativa deve continuar sendo definida por correção objetiva (VPL/MCTest, Apêndices A e B) e/ou julgamento humano — nunca pela nota bruta devolvida por um modelo de linguagem.

IL PERCORSO DI EDI & VC

SISTEMATIZZARE LA CONOSCENZA, DAL PIXEL ALL'INTELLIGENZA

Parte I — Fondamenti di EDI

- 1. Fondamenti e Primi Passi
- 2. Campionamento, Quantizzazione e Connettività
- 3. Operazioni Spaziali e Filtraggio
- 4. Morfologia Matematica e Segmentazione
- 5. Trasformate e Compressione

Parte II — Visione Computazionale

- 6. Analisi e Ispezione dei Documenti
- 7. Classificazione e Riconoscimento dei Pattern
- 8. Comprensione delle Scene e Segmentazione
- 9. Deep Learning (Apprendimento Profondo)

Appendici Essenziali

- **A:** MCTest
- **B:** Moodle/VPL
- **C:** SEB/Simulatore
- **D:** Materiale Didattico su Gemini
- **E:** IA nel feedback del VPL
- **F:** Percezione e Valutazione

IMMAGINE DI INPUT (PIXEL) → **EDI** (ELABORAZIONE) → **VC** (VISIONE) → OUTPUT INTELLIGENTE (SEMANTICO)

DOMINARE I PIXEL, COSTRUIRE IL FUTURO.

Dai semplici istogrammi alla comprensione complessa delle scene con il deep learning, questo lavoro offre una roadmap completa.

Trasforma le immagini in dati e i dati in decisioni intelligenti.

La capacità di plasmare la percezione computazionale inizia qui. Rimani al passo, esplora e innova!

Disponibile in formato digitale (HTML, PDF e Notebook Jupyter), con simulazioni ed esercizi, inclusi casi di test con VPL e Moodle.

SCANSIONA E ACCEDI AI CONTENUTI EXTRA

AUTORE: FRANCESCO DE ASSIS ZAMPIROLLI

https://fzampirolli.github.io/edi-vc/