



UNIVERSITÀ  
DEGLI STUDI  
**FIRENZE**

Scuola di Scienze Matematiche, Fisiche e Naturali  
Corso di Laurea in Informatica

Tesi di Laurea

**RAFFINAMENTO DI UNA LIBRERIA  
PYTHON PER L'OCR DI DOCUMENTI  
D'IDENTITÀ**

**REFINEMENT OF A PYTHON LIBRARY TO  
PERFORM OCR ON IDENTITY DOCUMENTS**

ALESSIO FALAI

Relatore: *Maria Cecilia Verri*

Anno Accademico 2018-2019



---

## INDICE

---

<b>Introduzione</b>	7
<b>1 Alcuni prerequisiti matematici nell'ambito della computer vision</b>	9
1.1 Immagini digitali	9
1.2 Convoluzione	9
1.3 Elementi di morfologia	12
<b>2 Binarizzazione di immagini</b>	19
2.1 Introduzione	19
2.2 Sogliatura globale	20
2.3 Sogliatura adattativa (o locale)	21
2.4 Sogliatura di Otsu	22
2.5 Sogliatura proposta	23
<b>3 Image matching</b>	29
3.1 Introduzione	29
3.2 Template matching	30
3.3 Feature matching	31
3.4 Shape detection	32
3.5 Analisi comparativa	36
<b>4 Text detection</b>	39
4.1 Introduzione	39
4.2 MSER	39
4.3 EAST	41
4.4 Ritaglio dinamico dei campi di un documento	42
<b>Conclusioni e sviluppi futuri</b>	47
<b>Ringraziamenti</b>	49
<b>Bibliografia</b>	50



---

## ELENCO DELLE FIGURE

---

1	Applicazione di un filtro di <i>Gauss</i> di dimensione $10 \times 10$	12
2	Erosione (dilatazione) morfologica con elemento strutturante rettangolare	13
3	Apertura morfologica con elemento strutturante circolare	14
4	<i>Dilatazione con ricostruzione</i>	15
5	<i>Apertura con ricostruzione</i>	16
6	Input per le operazioni di sogliatura	19
7	Output sogliatura globale con $\text{thresh} = 20$	20
8	Output sogliatura locale <i>Gaussian</i> a con $c = 2$ e $\text{window} = 11 \times 11$	22
9	Iistogramma bimodale	23
10	Output sogliatura di <i>Otsu</i>	23
11	Output <i>trasformazione comune</i> sogliatura proposta	25
12	Output <i>Canny edge detector</i> sogliatura proposta	25
13	Output <i>bounding boxes</i> caratteri sogliatura proposta	25
14	Output <i>trasformazione semplice</i> sogliatura proposta	26
15	Output <i>trasformazione complessa</i> sogliatura proposta	27
16	Trasformazioni geometriche (originale, traslazione, rotazione, affine e prospettica)	30
17	Esempio di <i>template matching</i>	31
18	Rappresentazione di alcune tipologie di <i>keypoints</i>	32
19	Operatore <i>Sobel</i> per <i>edge detection</i>	34
20	Operatore <i>Canny</i> con diversi parametri	35
21	Output della rete <i>HED</i>	36
22	<i>Text detection</i> con <i>MSER</i>	40
23	<i>Text detection</i> con <i>EAST</i>	41
24	Campi vecchia carta d'identità italiana (retro)	42
25	Vecchia carta d'identità italiana (fronte) non standard	43
26	Campi vecchia carta d'identità italiana (fronte) non standard, ritagliati in modo statico	43
27	Output di <i>EAST</i> su una vecchia carta d'identità italiana (fronte) non standard	44
28	Campi vecchia carta d'identità italiana (fronte) non standard, ritagliati in modo dinamico	44
29	Tessera sanitaria retro con <i>barcode</i>	47

4 Elenco delle figure

30 Passaporto italiano con <i>MRZ</i> . . . . .	47
---	----

*"If a machine is expected to be infallible, it cannot also be intelligent"*  
— Alan Turing



---

## INTRODUZIONE

---

Questo lavoro di tesi vuole essere un riassunto delle attività svolte durante il tirocinio curriculare presso l’azienda QI-LAB di Firenze nel periodo *marzo - giugno 2019*. Il progetto di tirocinio prevedeva l’apporto di migliorie, in termini di accuratezza ed efficienza, riguardanti una libreria per il riconoscimento ottico dei caratteri di alcuni campi di documenti d’identità, attualmente solo italiani. La libreria, denominata QI-OCR, era inizialmente il risultato del lavoro compiuto da un precedente tirocinante, *Emilio Cecchini*, che si era occupato dell’implementazione del framework di base, sul quale ho avuto il piacere di lavorare io stesso.

QI-OCR è stata distribuita come pacchetto *pip* ed è scritta in linguaggio Python, in versione *3.6.8*, anche se compatibile con la versione minor successiva, ovvero la *3.7.x*, nonchè l’ultima versione stabile rilasciata, nella data in cui mi trovo a scrivere questo testo. QI-OCR ha alcune dipendenze esterne, tra cui la libreria *OpenCV*, per la manipolazione di immagini, e il software *Tesseract*, per le funzionalità di OCR.

QI-OCR prevedeva la suddivisione in 4 moduli principali:

1. **Preprocessing:** Si occupa della localizzazione del documento all’interno dell’immagine, del ritaglio dei contorni e del raddrizzamento, tramite l’algoritmo di *feature matching SIFT*.
2. **Docfields:** Si occupa di effettuare un ritaglio statico, mediante coordinate predeterminate, dei campi d’interesse del documento già centrato.
3. **OCR:** Si occupa di effettuare una binarizzazione dei campi dell’immagine mediante un algoritmo di sogliatura globale *ad-hoc* e di fornire le immagini prodotte in input a un motore di OCR.
4. **Postprocessing:** Si occupa di applicare tecniche di analisi sintattica e semantica sui risultati restituiti dal software OCR.

Il lavoro da me svolto riguarda il miglioramento della libreria QI-OCR, mediante tecniche che verranno descritte successivamente. A tal proposito, la tesi è suddivisa in quattro capitoli. Il capitolo 1 presenta una descrizione dei prerequisiti matematici necessari a comprendere correttamente le spiegazioni dei metodi presenti nei capitoli successivi. Il capitolo

2 riguarda lo studio e l'implementazione di tecniche di binarizzazione e segmentazione di immagini, che nel nostro caso vengono utilizzate per la "pulizia" di una generica immagine contenente testo, in modo da ottenere, idealmente, il solo testo di colore nero e tutto il resto di colore bianco. Il capitolo 3 presenta una panoramica di algoritmi di *image matching* e descrive metodi alternativi al ben noto *SIFT* per l'individuazione della posizione di un documento all'interno di un'immagine. Infine, il capitolo 4 introduce l'applicazione di tecniche avanzate di *text-detection* con lo scopo di colmare problemi derivanti da documenti che non rispettano alcun tipo di template prefissato, come ad esempio la vecchia carta d'identità cartacea italiana, che risulta comunque rilasciabile, in alternativa alla nuova carta d'identità elettronica italiana, in casi di estrema esigenza o per tutti i cittadini che non possono recarsi per motivi di handicap presso il municipio.

In questa tesi vengono descritte le principali migliorie apportate alla libreria, anche se ne sono state introdotte di ulteriori, sia per quanto riguarda il lato sviluppo/distribuzione (containerizzazione con *Docker*, *benchmarks*, ...), sia per quanto riguarda l'utilizzo finale del prodotto (OCR parametrizzabile sui campi del documento, implementazione di diversi motori di OCR, ...).

# 1

---

## ALCUNI PREREQUISITI MATEMATICI NELL'AMBITO DELLA COMPUTER VISION

---

### 1.1 IMMAGINI DIGITALI

Un'immagine digitale  $I$  con risoluzione  $m \times n$  è una matrice di valori interi di dimensione  $n \times m$ , che può essere matematicamente interpretata come una funzione semplice (nè iniettiva, nè suriettiva)

$$I : \mathbb{N} \times \mathbb{N} \supseteq X \rightarrow P^c \subseteq \mathbb{N}^c, \quad (1.1)$$

che si occupa di mappare una coppia ordinata  $(u, v) \in \mathbb{N} \times \mathbb{N}$ , con  $u, v \in \mathbb{N}$ , in una *c-upla*  $p = (p_1, p_2, \dots, p_c) \in \mathbb{N}^c$ , in cui ciascun  $p_i$  di  $p$  rappresenta il valore del pixel in posizione  $(u, v)$  nel canale  $i$  dell'immagine, con  $0 \leq p_i \leq 2^b - 1$ , dove  $b$  è il numero fissato di bit utilizzati per rappresentare ciascun pixel e  $2^b$  indica il massimo numero di colori o di livelli di grigio rappresentabili.

Per esempio, ipotizzando di utilizzare canali colore standard, nel caso in cui si lavori con immagini in scala di grigi si avrà  $c = 1$ , mentre nel caso in cui si lavori con immagini *RGB* si avrà  $c = 3$  ( $c = 4$  per immagini *RGBA* o *CMYK*). Inoltre, supponendo di utilizzare una profondità pari a  $b = 8$  sarà possibile specificare interi compresi tra 0 e 255 per i valori di ciascun pixel.

### 1.2 CONVOLUZIONE

La *convoluzione* è un'operazione alla base dell'*image processing*, grazie alla quale è possibile analizzare ed eventualmente accentuare o alleviare diversi aspetti di una data immagine, come la sfocatura, la nitidezza, i contorni e molto altro. L'operazione, che viene descritta nella forma discreta, prende in input due immagini digitali, ovvero l'immagine originale  $I$  (normalmente un singolo canale) e l'immagine  $K$ , denominata *kernel*, e

intuitivamente si occupa di far scorrere la matrice  $K$  sull'immagine  $I$ , generalmente a partire dal pixel in alto a sinistra, effettuando una somma pesata dei valori di  $I$  dati dalle proiezioni delle posizioni correntemente analizzate dalla matrice  $K$ , in cui i pesi sono dati proprio dagli elementi di  $K$ . Solitamente la dimensione della matrice  $K$  è molto minore della dimensione dell'immagine originale e spesso  $K$  è una matrice quadrata  $n \times n$ , con  $n$  dispari.

Più formalmente, l'operazione di *convoluzione*  $I * K$  in un punto  $(i, j)$  è data da:

$$\begin{aligned} I^*(i, j) &= \sum_{x=-n}^n \sum_{y=-n}^n (I(i-x, j-y) \cdot K(x, y)) = \\ &= \sum_{x=1}^n \sum_{y=1}^n (I(i + \lceil \frac{n}{2} \rceil - x, j + \lceil \frac{n}{2} \rceil - y) \cdot K(x, y)) \end{aligned} \quad (1.2)$$

Nel caso in cui nella formula 1.2 i segni - e + venissero invertiti si otterrebbe la formulazione della *correlazione*  $I \otimes K$  in un punto  $(i, j)$ :

$$\begin{aligned} I^\otimes(i, j) &= \sum_{x=-n}^n \sum_{y=-n}^n (I(i+x, j+y) \cdot K(x, y)) = \\ &= \sum_{x=1}^n \sum_{y=1}^n (I(i - \lceil \frac{n}{2} \rceil + x, j - \lceil \frac{n}{2} \rceil + y) \cdot K(x, y)) \end{aligned} \quad (1.3)$$

*Convoluzione* e *correlazione* possono risultare equivalenti se per passare da un'operazione all'altra si effettua un ribaltamento orizzontale e verticale del filtro (graficamente  $K$  viene ruotata di  $180^\circ$ ). Dunque le due operazioni risultano identiche nel caso in cui la matrice  $K$  sia simmetrica rispetto ai due assi. Nella pratica, si preferisce utilizzare la *convoluzione* quando sono necessarie le proprietà commutativa, associativa e distributiva. Inoltre, dato che spesso il risultato delle operazioni descritte viene utilizzato come valore di intensità di pixel, tale risultato viene normalizzato, dividendolo per la somma dei pesi del filtro.

La complessità computazionale delle operazioni, data ad esempio un'immagine di dimensione  $m \times m$  e un kernel di dimensione  $n \times n$ , è piuttosto elevata, dato che richiede un numero di moltiplicazioni pari a  $n^2 \cdot m^2$  e altrettante somme.

Di seguito un esempio di *convoluzione* e *correlazione*:

$$I = i \begin{pmatrix} & j \\ & \dots \\ 30 & 28 & 32 \\ \vdots & 27 & 26 & 10 & \vdots \\ 29 & 22 & 18 \\ & \dots \end{pmatrix}, K = \begin{pmatrix} 4 & -2 & 1 \\ -1 & 5 & -3 \\ -6 & 0 & 4 \end{pmatrix}$$

Calcoliamo  $I * K$  nella posizione  $(i, j)$  come

$$\begin{aligned} I^*(i, j) = & 18 \cdot 4 - 22 \cdot 2 + 29 \cdot 1 - 10 \cdot 1 + \\ & + 26 \cdot 5 - 27 \cdot 3 - 32 \cdot 6 + 28 \cdot 0 + 30 \cdot 4 = 24, \end{aligned} \quad (1.4)$$

mentre calcoliamo  $I \otimes K$  nella posizione  $(i, j)$  come

$$\begin{aligned} I^\otimes(i, j) = & 30 \cdot 4 - 28 \cdot 2 + 32 \cdot 1 - 27 \cdot 1 + \\ & + 26 \cdot 5 - 10 \cdot 3 - 29 \cdot 6 + 22 \cdot 0 + 18 \cdot 4 = 67 \end{aligned} \quad (1.5)$$

Un problema che può risultare evidente riguarda il modo di trattare i bordi, nel caso in cui l'intorno di un pixel non sia disponibile. Per ovviare a tale problema esistono diverse tecniche valide, come ipotizzare che i pixel non disponibili abbiano intensità zero, oppure prolungare i pixel di bordo supponendo intensità costante in quelli non disponibili.

Alcuni esempi di filtri classici:

- Sfocatura (filtro di *Gauss*):  $\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$

- Nitidezza:  $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

- Contorni (filtro di *Laplace*):  $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$

In figura 1 è possibile osservare un esempio di sfocatura di un'immagine, tramite l'applicazione di un filtro di *Gauss*.



Figura 1: Applicazione di un filtro di *Gauss* di dimensione  $10 \times 10$

### 1.3 ELEMENTI DI MORFOLOGIA

#### 1.3.1 *Basi della matematica morfologica*

La matematica morfologica è una collezione di operatori, basata sulla teoria degli insiemi e definita su una struttura astratta<sup>1</sup>. Il suo obiettivo è quello di analizzare le forme e gli oggetti di un'immagine digitale, attraverso trasformazioni quali erosione, dilatazione, apertura, chiusura e altre. Nella loro forma originaria le operazioni morfologiche prevedono l'utilizzo di un'*immagine binaria* a singolo canale, composta da pixel che possono assumere solo due valori, 1 e 0, bianco e nero, *foreground* e *background*, oltre che di un *elemento strutturante (SE)*, anch'esso sotto forma di immagine binaria a singolo canale, generalmente di dimensione molto minore di quella dell'immagine originale. Per correttezza, da ora in poi, in questa sezione, considereremo un'immagine binaria  $I$  anche come un insieme di punti  $Q_I$  contenente tutte le coppie  $(u, v)$  di pixel *foreground* di  $I$ , ovvero tali che  $I(u, v) = 1$  e considereremo i termini in **grassetto** come coppie ordinate  $(u, v) \in \mathbb{N} \times \mathbb{N}$ .

Il risultato dell'operazione di *erosione* di un'immagine mostra dove lo SE viene contenuto negli oggetti dell'immagine, ovvero, intuitivamente

---

<sup>1</sup> La struttura astratta alla quale si fa riferimento è un *reticolo* infinito, un'estensione della teoria degli insiemi di Minkowski.

permette di rimuovere pixel dal contorno di *componenti connesse*<sup>2</sup> di colore bianco, in quantità dipendente dalla grandezza dello SE. In formule:

$$[\epsilon_{Q_{SE}}(I)](\mathbf{x}) = \bigwedge_{\mathbf{s} \in Q_{SE}} (I(\mathbf{x} + \mathbf{s})) \quad (1.6)$$

Il risultato dell'operazione di *dilatazione* di un'immagine mostra dove lo SE tocca gli oggetti dell'immagine, ovvero, intuitivamente permette di aggiungere pixel al contorno di componenti connesse di colore bianco, in quantità dipendente dalla grandezza dello SE. In formule:

$$[\delta_{Q_{SE}}(I)](\mathbf{x}) = \bigvee_{\mathbf{s} \in Q_{SE}} (I(\mathbf{x} + \mathbf{s})) \quad (1.7)$$

In figura 2 è possibile osservare l'applicazione degli operatori di *erosione* e *dilatazione*, con input-output in ordine sinistra-destra e destra-sinistra, rispettivamente.

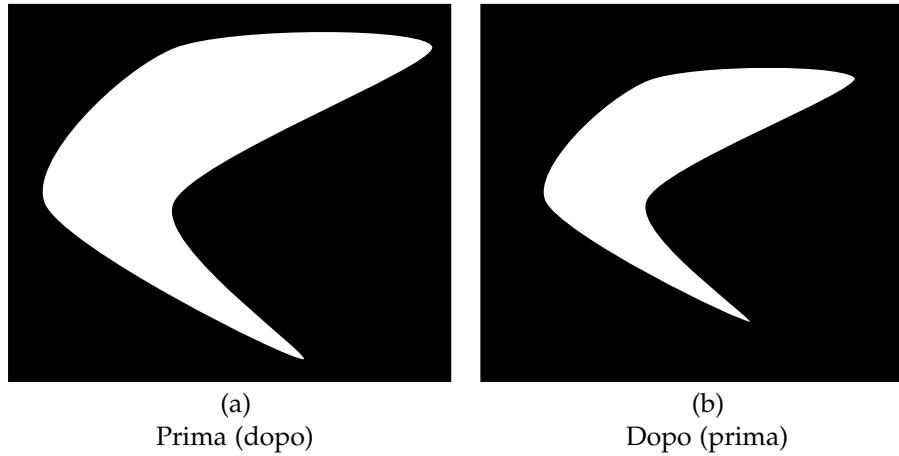


Figura 2: Erosione (dilatazione) morfologica con elemento strutturante rettangolare

Le operazioni di *apertura* (*chiusura*) sono invece la combinazione in sequenza di erosione (dilatazione) e dilatazione (erosione) dello SE con la propria *riflessione*<sup>3</sup>. Queste due operazioni vengono molto utilizzate per

<sup>2</sup> Una *componente连通*  $C$  di un insieme di punti *foreground*  $P$  di un'immagine binaria  $I$  è un insieme  $C \subseteq P$  tale che  $\forall \mathbf{p}_i, \mathbf{p}_j \in C$  esiste un percorso  $\mathbf{p}_i, \dots, \mathbf{p}_j$  in cui ogni  $\mathbf{p}_h$  è  $k$ -vicino (4-vicino o 8-vicino) a  $\mathbf{p}_{h-1}$  e  $\neg \exists \mathbf{q} \notin P$   $k$ -adiacente a un punto di  $P$  [7].

<sup>3</sup> La *riflessione* di un insieme di punti  $P$  di un'immagine binaria  $I$  rispetto all'origine viene definita come  $\check{P} = \{(-u, -v) : (u, v) \in P\}$ .

rimuovere "sporco" (*noise*) dalle immagini e per "chiudere" piccoli "buchi" all'interno di oggetti che si trovano nel *foreground*. In formule:

$$\gamma_{Q_{SE}}(I) = \delta_{Q_{SE}^c}(\epsilon_{Q_{SE}}(I)), \quad (1.8)$$

$$\phi_{Q_{SE}}(I) = \epsilon_{Q_{SE}^c}(\delta_{Q_{SE}}(I)) \quad (1.9)$$

In figura 3 è possibile osservare la rimozione delle aree *foreground* di piccola dimensione, grazie all'utilizzo dell'*apertura* morfologica.

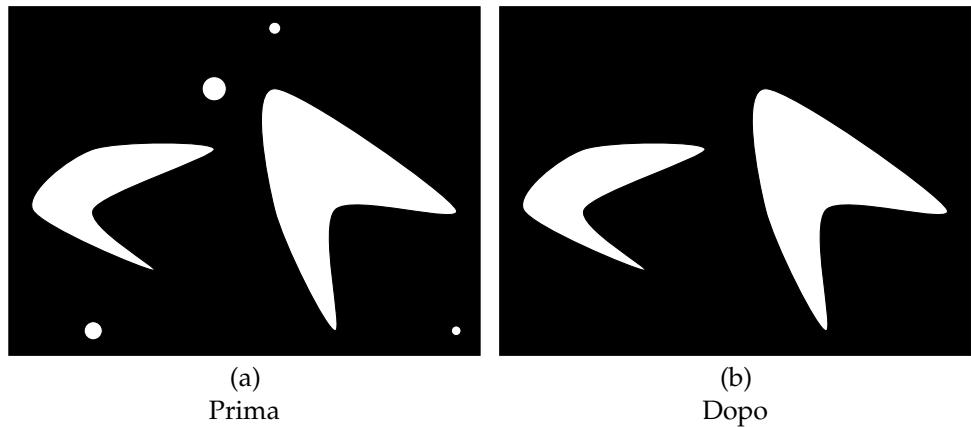


Figura 3: Apertura morfologica con elemento strutturante circolare

### 1.3.2 Ricostruzione morfologica

La *ricostruzione* è un'operazione molto utile nella matematica morfologica, che viene spesso presentata come un insieme di operatori *geodetici*, che consentono di estrarre le componenti connesse dell'immagine originale, marcate da un'altra immagine *marker*. Gli operatori di *erosione* e *dilatazione* possono dunque essere ridefiniti in senso *geodetico*. In particolare, consideriamo al solito un'immagine binaria  $I$  (*mask*), definita dal suo insieme di punti *foreground*  $Q_I$ , un elemento strutturante (SE), che definisce la connettività da rispettare, e un'immagine *marker*  $F$  tale che  $Q_F \subseteq Q_I$ , che permette di definire quali componenti connesse devono subire i risultati delle operazioni morfologiche. A questo punto, possiamo definire l'*erosione condizionale* come:

$$[\epsilon_{Q_{SE}}(F)]_I^{(0)}(\mathbf{x}) = [\epsilon_{Q_{SE}}(F)](\mathbf{x}) \vee I(\mathbf{x}) \quad (1.10)$$

e la *dilatazione condizionale* come:

$$[\delta_{Q_{SE}}(F)]_I^{(0)}(\mathbf{x}) = [\delta_{Q_{SE}}(F)](\mathbf{x}) \wedge I(\mathbf{x}) \quad (1.11)$$

Le varianti *geodetiche* vengono definite in modo iterativo a partire da quelle *condizionali*. Dunque, si avrà che l'*erosione geodetica* può essere espressa come:

$$[\epsilon_{Q_{SE}}(F)]_I^{(i)}(\mathbf{x}) = [\epsilon_{Q_{SE}}([\delta_{Q_{SE}}(F)]_I^{(i-1)}(\mathbf{x}))]_I^{(0)}(\mathbf{x}), i = 1, 2, 3, \dots, \quad (1.12)$$

mentre la *dilatazione geodetica*:

$$[\delta_{Q_{SE}}(F)]_I^{(i)}(\mathbf{x}) = [\delta_{Q_{SE}}([\epsilon_{Q_{SE}}(F)]_I^{(i-1)}(\mathbf{x}))]_I^{(0)}(\mathbf{x}), i = 1, 2, 3, \dots \quad (1.13)$$

Concettualmente, le iterazioni sopra definite potrebbero continuare indefinitamente, ma per ragioni pratiche si sceglie di farle terminare per un intero  $n$  tale che nessun cambiamento avvenga  $\forall n' > n$ . L'output stabile

$$R_I^\epsilon(F) = [\epsilon_{Q_{SE}}(F)]_I^{(n)}(\mathbf{x}), \quad (1.14)$$

$$R_I^\delta(F) = [\delta_{Q_{SE}}(F)]_I^{(n)}(\mathbf{x}) \quad (1.15)$$

così definito viene denominato rispettivamente *erosione* e *dilatazione con ricostruzione* (vedi figura 4).

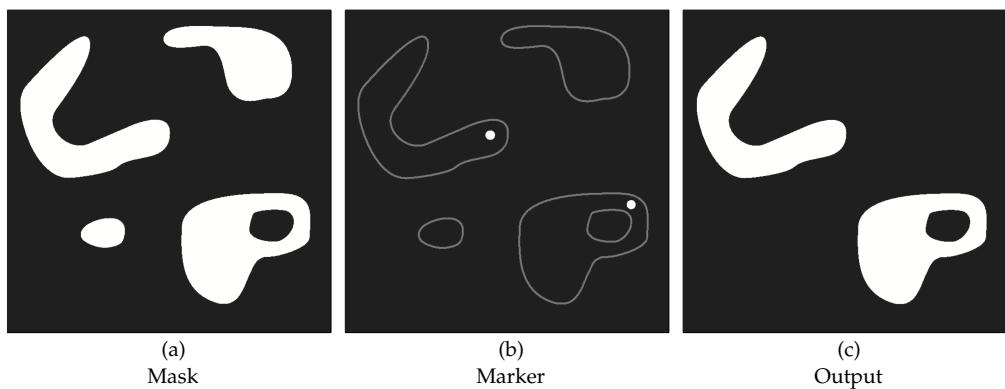


Figura 4: *Dilatazione con ricostruzione*<sup>4</sup>

---

<sup>4</sup> Fonte: *Advanced morphological image processing* [8]

Definiamo adesso le operazioni di *apertura* e *chiusura* con *ricostruzione*: l'operazione di apertura permette di rimuovere oggetti in base allo SE usato, ma ha il limite di non preservare le altre strutture, che vengono comunque modificate. Questo limite viene completamente superato dall'operatore di *apertura con ricostruzione*, che opera nel seguente modo: l'immagine viene erosa con lo SE come per l'apertura normale, ma invece di far seguire una dilatazione, si opera una ricostruzione usando l'immagine originale come *mask*, e l'immagine erosa come *marker*. L'operazione di *chiusura con ricostruzione* opera in modo analogo. In formule:

$$\tilde{\gamma}_R(I) = \gamma_R^{(n)}(I) = R_I^\delta[\epsilon^{(n)}(I)], \quad (1.16)$$

$$\tilde{\phi}_R(I) = \phi_R^{(n)}(I) = R_I^\epsilon[\delta^{(n)}(I)] \quad (1.17)$$

In figura 5 è possibile osservare un confronto fra le operazioni di *apertura morfologica "standard"* e con *ricostruzione*, in cui l'obiettivo è quello di estrarre le lettere con altezza superiore a 50px, tramite l'utilizzo di uno SE di dimensione  $51 \times 1$ .

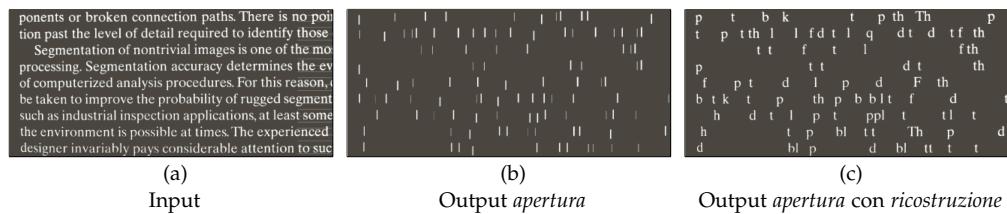


Figura 5: *Apertura con ricostruzione*<sup>5</sup>

### 1.3.3 Altri operatori morfologici

Gli operatori fin'ora descritti permettono di estrarre caratteristiche importanti all'interno delle immagini, ma, ad esempio, consentono difficilmente di effettuare un'equalizzazione di parametri dell'immagine, oppure di estrarre piccoli particolari d'interesse, oppure ancora di rilevare i contorni di un oggetto. Per ovviare a queste problematiche, si può ricorrere all'utilizzo di trasformazioni più specifiche, derivate da quelle di base.

---

<sup>5</sup> Fonte: *Advanced morphological image processing* [8]

In particolare, possiamo definire l'operatore *top-hat*, nelle due versioni *bianca* e *nera*, rispettivamente:

$$g_{Q_{SE}}^w(I) = I - \gamma_{Q_{SE}}(I), \quad (1.18)$$

$$g_{Q_{SE}}^b(I) = \phi_{Q_{SE}}(I) - I \quad (1.19)$$

Entrambi ritornano un'immagine contenente gli elementi dell'immagine originale che sono più piccoli dello SE. Il primo ritorna inoltre solo gli elementi più *luminosi* degli oggetti a loro vicini, mentre il secondo solo gli elementi più *scuri* degli oggetti a loro vicini. Questo ci fa capire che un buon utilizzo per questo operatore riguarda la correzione di condizioni di luce non uniforme all'interno di un'immagine, per consentire una separazione tra *foreground* e *background* più marcata, grazie al conseguente incremento di contrasto.

Altri operatori morfologici sono quello *hit-or-miss*, che consente di estrarre pixel i cui vicini hanno una specifica configurazione (*pattern*), in base allo SE utilizzato; oppure quello del *gradiente*, che consiste nella differenza tra dilatazione e erosione di un'immagine, e restituisce, ad esempio, i contorni di oggetti.



# 2

---

## BINARIZZAZIONE DI IMMAGINI

---

### 2.1 INTRODUZIONE

Nel campo dell'*image processing* ricopre un ruolo fondamentale la possibilità di distinguere diversi oggetti, forme e contorni presenti nell'immagine in analisi. Per far ciò è possibile ricorrere a svariate metodologie, che sono strettamente dipendenti dal contesto in cui si intende operare. Per esempio, nel caso del *riconoscimento ottico dei caratteri* (OCR), le operazioni che andranno descritte assumono importanza assoluta, dato che questo tipo di applicazione, per operare al meglio, ha spesso bisogno di ricevere in input un'immagine in bianco e nero, in cui lo sfondo è di colore bianco e il testo è di colore nero. Gli argomenti affrontati, nel caso di un'applicazione OCR, saranno soprattutto utili nel caso in cui il motore utilizzato sia *open-source* (vedi *Tesseract*), ovvero nel caso in cui un buon *pre-processing* dell'immagine possa fare la differenza. In questo capitolo andremo quindi ad analizzare gli *algoritmi* più ricorrenti nell'ambito del *thresholding* di immagini e nella sezione 2.5 affronteremo un approccio specificamente studiato per i casi d'uso della libreria QI-OCR.

Da qui in avanti ipotizzeremo di lavorare con immagini originali in *scala di grigi* con profondità 8 bit, anche se i metodi descritti possono facilmente essere generalizzati per immagini con più di un canale colore.

Consideriamo l'immagine riportata in figura 6 come esempio per l'analisi delle operazioni che andremo a descrivere.

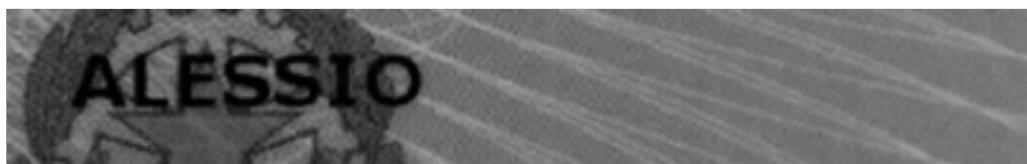


Figura 6: Input per le operazioni di sogliatura

## 2.2 SOGLIATURA GLOBALE

È il più semplice metodo di *thresholding*, che prevede la scelta di un determinato valore di soglia, compreso tra 0 e 255. L'algoritmo di sogliatura prevede quindi la scansione, pixel per pixel, dell'immagine e, nel caso in cui il valore del pixel esaminato sia minore del valore di soglia, tale pixel assumerà valore zero, che corrisponde al colore nero. Altrimenti, nel caso in cui il valore del pixel esaminato sia maggiore o uguale del valore di soglia, tale pixel assumerà valore 255, che corrisponde al colore bianco. Risulta facile intuire quale possa essere il problema principale di questo tipo di algoritmo di sogliatura, ovvero la scelta del valore di soglia. Nel caso in cui il contesto in cui si opera sia relativamente statico, la *sogliatura globale* risulta comunque l'opzione più consigliata, per la sua facilità di implementazione. In particolare, il metodo descritto risulta adatto se le immagini in analisi presentano all'incirca caratteristiche uniformi in termini di illuminazione e contrasto. Se così non fosse, sarebbe necessario valutare l'implementazione di algoritmi leggermente più complessi.

In figura 7 è possibile osservare come l'attenta scelta del corretto valore di soglia possa portare a risultati eccellenti.

---

### Algoritmo 1 Sogliatura globale

---

```

1: function GLOBAL-THRESHOLDING
2:   image  $\leftarrow$  input image
3:   thresh  $\leftarrow$  threshold value
4:
5:   for i  $\in$  ROWS(image) do
6:     for j  $\in$  COLUMNS(image) do
7:       if image[i][j]  $\geq$  thresh then
8:         image[i][j]  $\leftarrow$  255
9:       else
10:        image[i][j]  $\leftarrow$  0
11:   return image

```

---



ALESSIO

Figura 7: Output sogliatura globale con  $\text{thresh} = 20$

### 2.3 SOGLIATURA ADATTATIVA (O LOCALE)

Questo metodo di *thresholding* consente di sopperire alle mancanze della sogliatura globale, nel caso in cui le immagini in analisi presentino illuminazione e/o contrasto non uniforme. In particolare, questa tecnica dinamica computa automaticamente differenti valori di soglia per diverse aree dell'immagine. Dunque, l'immagine viene suddivisa in tante *sotto-immagini*, abbastanza piccole da poter ipotizzare che in ciascuna di esse illuminazione e contrasto siano sufficientemente uniformi. Una volta partizionata l'immagine, un valore di soglia viene calcolato per ciascuna *finestra*. Il calcolo di ogni valore di soglia dipende dall'implementazione specifica dell'algoritmo, ma genericamente si ricorre all'utilizzo, per ogni sotto-immagine, di semplici operatori statistici, come la media, la mediana o la media fra massimo e minimo.

Alcune tecniche di sogliatura locale efficaci sono implementate da algoritmi quali quello di *Niblack* [9] e *Sauvola* [10]. In particolare, l'algoritmo di *Niblack* prevede l'utilizzo delle metriche di media e deviazione standard, per una specifica finestra centrata su ciascun pixel dell'immagine. La formula utilizzata per calcolare un generico valore di soglia è data da:

$$T(x, y) = m(x, y) + k \cdot s(x, y), \quad (2.1)$$

dove  $T(x, y)$  indica il valore di soglia per il pixel in posizione  $(x, y)$ ,  $m(x, y)$  ( $s(x, y)$ ) indica la media (deviazione standard) dei valori dei pixel vicini a quello centrato sulla finestra corrente e  $k$  è un coefficiente, che può essere determinato empiricamente<sup>1</sup>.

L'utilizzo di parametri configurabili è molto comune per algoritmi di questo tipo, in quanto questi risultano essere fortemente dipendenti dal contesto di utilizzo. Un esempio pratico riguarda la libreria *OpenCV*, che nella funzione `ADAPTIVETHRESHOLD` consente di selezionare un valore  $c$ , che viene sottratto da ciascun valore di soglia di ogni sotto-immagine. Per esempio, nel caso in cui il valore di soglia venga calcolato utilizzando la media `mean`, il limite scelto non sarà esattamente `mean`, ma `mean - c`.

Purtroppo, il problema principale della sogliatura adattativa è che tende a valorizzare tutti gli elementi presenti nell'immagine, non consentendo di differenziare al meglio le componenti d'interesse da quelle che invece si vorrebbero scartare.

---

<sup>1</sup> *Niblack* consiglia l'impostazione del parametro  $k$  al valore  $-0.2$ .

---

**Algoritmo 2** Sogliatura adattativa

---

```

1: function LOCAL-THRESHOLDING
2:   image  $\leftarrow$  input image
3:
4:   for pixel in image do
5:     thresh  $\leftarrow$  STATISTICAL-OPERATOR(NEIGHBORS(PIXEL))
6:     if pixel  $\geqslant$  thresh then
7:       pixel  $\leftarrow$  255
8:     else
9:       pixel  $\leftarrow$  0
10:    return image

```

---

In figura 8 è possibile osservare un esempio di sogliatura adattativa di *Gauss*, in cui il valore di soglia di ciascuna sotto-immagine è dato dalla somma pesata dei valori vicini, dove i pesi sono dati da una *finestra Gaussiana*.



Figura 8: Output sogliatura locale *Gaussiana* con  $c = 2$  e  $window = 11 \times 11$

#### 2.4 SOGLIATURA DI OTSU

Questo tipo di sogliatura utilizza tecniche di analisi dell'*istogramma*<sup>2</sup> dell'immagine ed è particolarmente adatto per immagini *bimodali* (vedi figura 9), ovvero per immagini che presentano istogrammi con una netta separazione tra due picchi principali. La binarizzazione di *Otsu* si occupa esattamente di trovare la "valle" di scissione fra tali picchi, che risulta essere proprio il valore di soglia ottimale per l'applicazione di un'operazione di sogliatura globale.

Nel caso in cui l'immagine in analisi non presenti esattamente due massimi locali nella funzione istogramma, il procedimento di *Otsu* po-

---

<sup>2</sup> L'*istogramma*  $h$  di un'immagine  $I$  in scala di grigi, con valori d'intensità  $\in [0, K - 1]$  e definiti nell'immagine della funzione  $I$ , è una funzione tale che  $h(i)$  è uguale al numero di pixel di  $I$  con valore di intensità  $i$ , per  $0 \leq i < K$ . Più formalmente  $h(i) = |(u, v) : I(u, v) = i|$ .

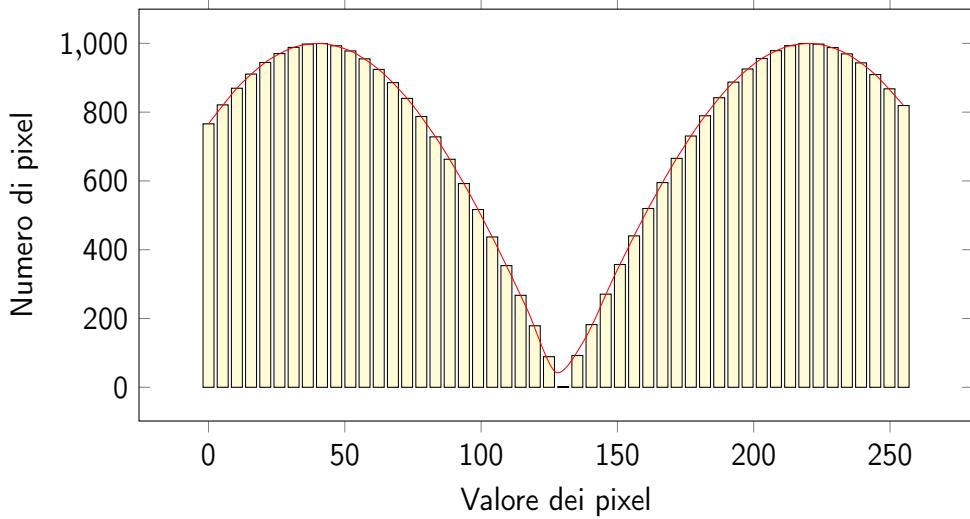


Figura 9: Istogramma bimodale

trebbe però portare a risultati indesiderati, come è possibile osservare in figura 10.

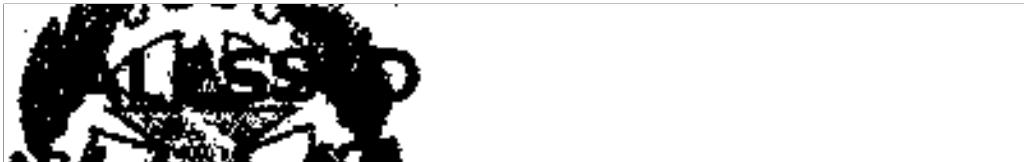


Figura 10: Output sogliatura di *Otsu*

## 2.5 SOGLIATURA PROPOSTA

In questa sezione andiamo a descrivere un algoritmo pensato e implementato per la *segmentazione* di immagini contenenti *testo nero su uno sfondo colorato*, arricchito di decorazioni, simboli e varie grafiche, per ottenere un'immagine correttamente processabile da motori OCR (in particolare da quelli *open-source*, come *Tesseract*).

L'approccio precedentemente adottato dalla libreria QI-OCR prevedeva la scelta mirata di diversi livelli di sogliatura per ciascun campo del documento in analisi, effettuando così una serie di operazioni di sogliatura globale e di conseguenti chiamate al motore OCR, per poi scegliere il risultato migliore a posteriori. Il problema principale di questo metodo riguarda la scarsa efficienza dal punto di vista computazionale, in quanto il tempo di esecuzione per effettuare un'operazione di sogliatura globale

e per ricevere l'output dal motore OCR è di circa 1s<sup>3</sup>. Considerando che, in alcuni casi, il numero di soglie da applicare a un particolare campo poteva arrivare anche fino a 10, il tempo di esecuzione per la sola operazione di OCR sarebbe stato all'incirca 10s.

Il percorso che ha portato alla soluzione descritta in seguito è passato da due strade fallimentari: la prima prevedeva l'analisi delle componenti connesse dell'immagine in input, come descritto in [12], mentre la seconda prevedeva l'implementazione di una rete neurale convoluzionale (CNN - *Convolutional Neural Network*), allenata a partire da associazioni *campo - soglia* prestabilite. Purtroppo, entrambi i tentativi non hanno ritornato i risultati sperati: nel primo caso per una mancanza di strumenti adatti, mentre nel secondo per la mancanza di una conoscenza approfondita del settore.

### 2.5.1 Trasformazione comune

Il primo passaggio riguarda un'insieme di operazioni che restituiranno in output un'immagine che verrà utilizzata come input sia per la fase descritta in 2.5.2 che per quella in 2.5.3.

In particolare, prima di tutto l'immagine originale viene *ridimensionata* di un determinato fattore di scala, che empiricamente è stato posto pari a 3. Dopodichè viene applicato un *clip* superiore dell'immagine al valor medio assunto dai pixel (mean). Questa operazione permette di equalizzare l'istogramma dell'immagine, andando immediatamente a rimuovere varie componenti di *background*, limitando superiormente i pixel dell'immagine al valore mean – c, con c costante arbitraria, che nel nostro caso d'uso è stata posta pari a 30.

Il problema principale di questa operazione riguarda l'eliminazione di informazione utile nel caso in cui il testo presente sia sufficientemente sbiadito.

Come si può notare osservando la figura 11, l'output di questa fase ha ridotto notevolmente la presenza dominante dello sfondo, producendo un'immagine con istogramma molto vicino alla definizione di *bimodale*. L'immagine prodotta può dunque essere processata più correttamente dall'algoritmo di sogliatura di Otsu (vedi 2.4).

---

<sup>3</sup> L'elevato tempo di esecuzione è dato principalmente dalla chiamata al motore OCR. Per esempio, nel caso di *Tesseract*, in Python non esiste una vera e propria API ufficiale ed è dunque necessario utilizzare un *wrapper* che effettua chiamate al software di sistema, introducendo *overhead*.



Figura 11: Output *trasformazione comune sogliatura proposta*

### 2.5.2 *Trasformazione semplice*

Il secondo passaggio applica le seguenti operazioni, nell'ordine riportato:

1. *Sogliatura di Otsu* (vedi 2.4)
2. *Apertura morfologica* (vedi 1.3.1)
3. *Chiusura morfologica* (vedi 1.3.1)
4. *Ritaglio del testo*

Il punto 1 viene utilizzato per convertire l'immagine in input in bianco e nero. I punti 2 e 3 vengono utilizzati per rimuovere *noise* e riempire "buchi" nelle lettere del testo di interesse e richiedono la definizione di un elemento strutturante, o kernel, che, come già anticipato, dipende strettamente dal contesto di applicazione.

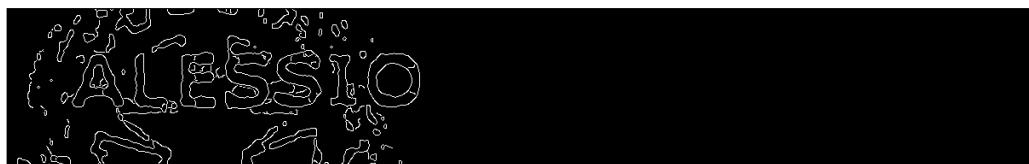


Figura 12: Output *Canny edge detector sogliatura proposta*



Figura 13: Output *bounding boxes caratteri sogliatura proposta*

Il punto 4 utilizza una sorta di strategia di *clustering*, che prevede prima di tutto l'individuazione dei contorni presenti nell'immagine (vedi figura 12), tramite l'algoritmo *Canny edge detector*, descritto in 3.4.2, e

da questi la definizione dei vari *bounding boxes* (vedi figura 13), ovvero dei minimi rettangoli contenenti ciascuna forma riconosciuta. A questo punto i *bounding boxes* vengono filtrati in base alla dimensione, eliminando quelli troppo piccoli (*noise*) e mantenendo solamente quelli della giusta dimensione, pari all'incirca alla dimensione del font utilizzato. A partire dai *bounding boxes* viene poi calcolato il valore di ordinata più frequente, arrotondato alla decina più vicina, e il massimo valore di altezza fra tutti, che vengono utilizzati per effettuare un taglio preciso dell'immagine al rettangolo che si suppone contenga il testo di interesse, per ottenere un risultato come quello riportato in figura 14. Ovviamente l'approccio di ritaglio utilizzato è efficace solamente nel caso in cui il testo sia l'elemento preminente nell'immagine.



Figura 14: Output *trasformazione semplice* sogliatura proposta

### 2.5.3 Trasformazione complessa

Il terzo passaggio applica le seguenti operazioni, nell'ordine riportato:

1. Operazione *top-hat modificata*
2. *Blurring* con filtro mediano (vedi 1.2)
3. *Chiusura morfologica* (vedi 1.3.1)
4. *Ritaglio* del testo
5. *Sogliatura di Otsu* (vedi 2.4)
6. *Apertura morfologica* (vedi 1.3.1)

Percorriamo le operazioni svolte a ritroso. In particolare, i punti 6, 5, 4, 3 equivalgono a quelli effettuati nella *trasformazione semplice*, con l'unica differenza riguardante il tipo di *chiusura morfologica* utilizzata, poichè in questo caso l'operazione risulta essere definita per un'immagine in scala di grigi, anzichè binaria.

Il punto 2 applica una *sfocatura* dell'immagine, utilizzando un filtro che scorre su ciascun pixel e sostituisce ciascun valore con la *mediana* degli elementi "vicini", ovvero degli elementi che si trovano in una determinata sotto-immagine quadrata, centrata sul pixel valutato. La sfocatura è

necessaria per mettere in risalto gli elementi importanti dell'immagine in esame.

Infine, il punto 1 applica l'operazione morfologica *top-hat*, nella versione *bianca*, descritta in 1.3.3, con l'eccezione di sostituire nella formula l'utilizzo dell'*apertura* con la combinazione di *apertura con ricostruzione* e *chiusura con ricostruzione* [4]. In formule:

$$\tilde{g}_{QSE}^w(I) = I - \tilde{\phi}_R(\tilde{\gamma}_R(I)), \quad (2.2)$$

Questo tipo di operazione (vedi figura 15) consente di correggere efficacemente condizioni di luce e/o contrasto non uniforme e di recuperare testi leggermente sbiaditi, che venivano invece penalizzati dalla prima trasformazione descritta.



Figura 15: Output *trasformazione complessa* sogliatura proposta

A questo punto, il numero di chiamate OCR effettuabili viene fissato a un massimo di 3 per campo, ovvero una chiamata per ciascun output prodotto dall'algoritmo di sogliatura proposto, riducendo notevolmente il tempo di esecuzione, senza compromettere l'accuratezza.

Per valutare effettivamente i risultati del lavoro prodotto, sono stati effettuati diversi test con campioni casuali di 50 tessere sanitarie fronte, scelte da una distribuzione uniforme.

Numero di campi con accuratezza pari al 100%	Accuratezza media	Tempo totale
168/300	73%	820s ≈ 13.66m

Tabella 1: Benchmark senza modifiche (campione 1)

Numero di campi con accuratezza pari al 100%	Accuratezza media	Tempo totale
182/300	75%	360s = 6.00m

Tabella 2: Benchmark con modifiche (campione 1)

Per dare un'idea più mirata delle migliorie relative ai tempi di esecuzione abbiamo suddiviso i *benchmark* in base alla funzione svolta dalla libreria, ottenendo i seguenti risultati:

Numero di campi con accuratezza pari al 100%	Accuratezza media	Tempo di OCR	Tempo di rotazione	Tempo totale
193/264	86%	331s ≈ 5.51m	221s ≈ 3.68m	573s ≈ 9.55m

Tabella 3: Benchmark senza modifiche (campione 2)

Numero di campi con accuratezza pari al 100%	Accuratezza media	Tempo di OCR	Tempo di rotazione	Tempo totale
182/264	86%	89s ≈ 1.48m	209s ≈ 3.48m	342s ≈ 5.7m

Tabella 4: Benchmark con modifiche (campione 2)

Dunque, possiamo notare che l'accuratezza è rimasta la stessa, il tempo totale impiegato è effettivamente dimezzato, mentre il tempo necessario a effettuare le operazioni di *OCR* è diminuito di circa 4 volte.

I *benchmark* effettuati sono stati prodotti utilizzando *SIFT* come algoritmo di localizzazione del documento e l'accuratezza media è stata valutata utilizzando un algoritmo per il calcolo della similitudine tra due stringhe<sup>4</sup>. In particolare, l'accuratezza media è data dalla media delle accuratezze di ogni singolo documento processato, dove l'accuratezza di un documento è data dalla media delle accuratezze su ogni singolo campo e l'accuratezza su un campo è la misura della distanza della stringa prodotta dal motore *OCR* e la stringa effettiva, derivata da osservazioni dirette (*ground-truth*).

---

<sup>4</sup> L'algoritmo menzionato è quello di *Ratcliff/Obershelp*.

# 3

---

## IMAGE MATCHING

---

### 3.1 INTRODUZIONE

Quando confrontiamo due immagini, una domanda che sorge spontanea è la seguente: quando e sotto quali condizioni due immagini possono essere definite uguali o simili?

Questo capitolo affronta l'argomento del confronto d'immagini e in particolare quello che consente di localizzare una data sotto-immagine (*template*) all'interno di un'immagine più grande (*scene*). La difficoltà di queste operazioni sta nel tipo di trasformazione geometrica subita dal template all'interno dell'immagine in esame, che può essere di vari tipi, come mostrato in figura 16:

- *Scala* ← Ridimensionamento dell'immagine
- *Traslazione* ← Spostamento dell'immagine, in direzione  $(x, y)$  di un determinato *shift*  $(t_x, t_y)$
- *Rotazione* ← Rotazione dell'immagine di un determinato angolo  $\theta$
- *Trasformazione affine* ← Trasformazione geometrica in cui tutte le linee parallele dell'immagine originale rimangono parallele nell'immagine trasformata, ma possono cambiare angoli e lunghezze
- *Trasformazione prospettica* ← Trasformazione geometrica 3D non affine, che conserva solo la rettilinearità, ovvero trasforma linee in linee e figure concave (convesse) in figure concave (convesse)

Possiamo affermare che le *trasformazioni affini* sono un sottoinsieme delle *trasformazioni prospettiche* (*omografie*), in quanto quest'ultime consentono di mappare linee parallele in non parallele e viceversa.

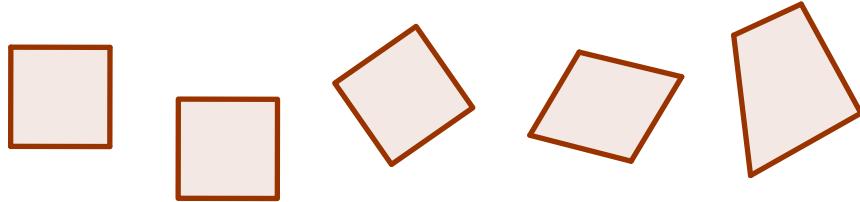


Figura 16: Trasformazioni geometriche (originale, traslazione, rotazione, affine e prospettica)

Le sezioni seguenti descrivono vari metodi testati nella libreria QI-OCR per la localizzazione di un documento, a partire da una data immagine *template*.

### 3.2 TEMPLATE MATCHING

L'approccio più semplice al quale si può pensare, molto simile a quello utilizzato nella *convoluzione*, riguarda lo scorrimento dell'immagine *template* sull'immagine di input, per confrontare il template e l'area dell'immagine originale coperta dal template stesso (vedi figura 17). L'output di questa operazione è ancora un'immagine, in scala di grigi, in cui ogni pixel denota il grado di similitudine dei suoi "vicini" con il template. La scelta dell'area intorno al pixel di intensità massima, nell'immagine prodotta, darà così il risultato cercato.

Il punto cruciale di questa tecnica sta nel metodo di confronto fra pixel scelto, che potrebbe essere, ad esempio, la misura di *distanza euclidea*. Consideriamo  $d_E(r, s)$  come la distanza euclidea tra l'immagine template  $T$  e l'immagine in input  $I$ , nella posizione  $(r, s)$ :

$$d_E(r, s) = \sqrt{\sum_{(i,j) \in T} (I(r+i, s+j) - T(i, j))^2} \quad (3.1)$$

A questo punto, per trovare la posizione di *match* migliore, è necessario minimizzare il quadrato di  $d_E(r, s)$ :

$$\begin{aligned} d_E^2(r, s) &= \sum_{(i,j) \in T} (I(r+i, s+j) - T(i, j))^2 = \\ &= \underbrace{\sum_{(i,j) \in T} I^2(r+i, s+j)}_{A(r,s)} + \underbrace{\sum_{(i,j) \in T} T^2(i, j)}_B - 2 \cdot \underbrace{\sum_{(i,j) \in T} I(r+i, s+j) \cdot T(i, j)}_{C(r,s)} \quad (3.2) \end{aligned}$$

Possiamo notare che ci siamo appena ricondotti alla risoluzione di un problema di *correlazione lineare*, come descritto in 1.2. In particolare, considerando i due termini  $A(r, s)$  e  $B$  come costanti<sup>1</sup>, otteniamo proprio l'equazione 1.3.

Il problema principale del *template matching* è la poca resistenza a trasformazioni geometriche, anche semplici. Un modo per evitare il presentarsi di questi problemi è quello di utilizzare diverse immagini *template*, scalate e ruotate, a discapito dell'efficienza di esecuzione.

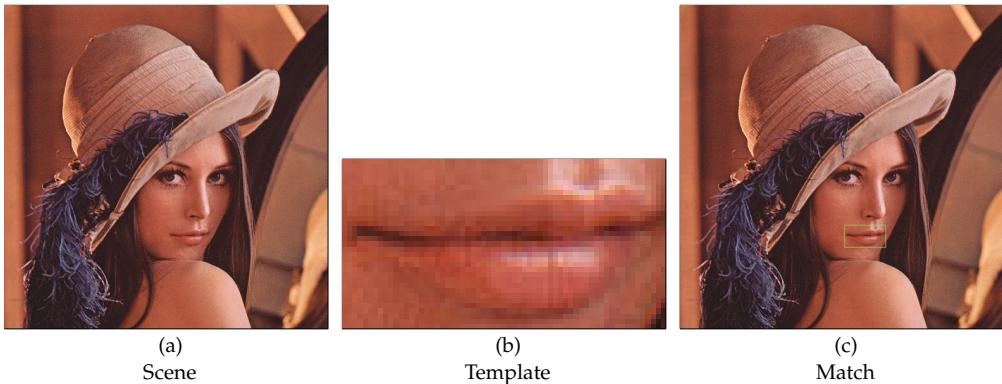


Figura 17: Esempio di *template matching*

### 3.3 FEATURE MATCHING

Questo approccio si basa sull'utilizzo di particolari punti di un'immagine, denominati *keypoints* (letteralmente *punti chiave*), che possono godere di elevata robustezza e stabilità per quanto riguarda trasformazioni geometriche e fotometriche (illuminazione e contrasto). I *keypoints* vengono identificati da specifiche *feature* dell'immagine, quali bordi e contorni, e vengono associati a particolari *descrittori locali*, che codificano informazioni di interesse intorno a ciascun *keypoint*. Una volta individuati *keypoints* e *descrittori* del template e dell'immagine in input è possibile computare un *matching*, che consiste nell'individuare l'associazione fra le varie *feature* delle due immagini.

Esistono vari algoritmi di *feature matching*, come *KNN* (*K-Nearest-Neighbors*), che prevedono l'utilizzo dei due seguenti metodi principali:

---

<sup>1</sup> Sfortunatamente, l'assunzione che il termine  $A(r, s)$  sia costante non vale generalmente ed è per questo necessario l'utilizzo di nozioni più avanzate, come il *coefficiente di correlazione*.

- *Esaustivo*  $\leftarrow$  Ritorna il miglior *match* (*brute force*) o i migliori K *match*, per ciascuna *feature*, in senso assoluto
- *Approssimato*  $\leftarrow$  È basato sull'utilizzo di algoritmi efficienti di ricerca *nearest neighbor* approssimata<sup>2</sup>, che vengono utilizzati solo per insiemi di *feature* di dimensioni elevate

Alcuni degli algoritmi più conosciuti e utilizzati per il calcolo di *keypoints* e *descrittori* sono *SIFT* (*Scale Invariant Feature Transform*) [13], *SURF* (*Speeded Up Robust Feature*) [14], *AKAZE* (*Accelerated KAZE* [15]) [16], *BRI-SK* (*Binary Robust Invariant Scalable Keypoints*) [17], *ORB* (*Oriented FAST and Rotated BRIEF*) [18] e altri.

In figura 18 alcuni esempi di *keypoints* ricavati da un'immagine messa a disposizione dal *vision toolbox* del software *MATLAB*.

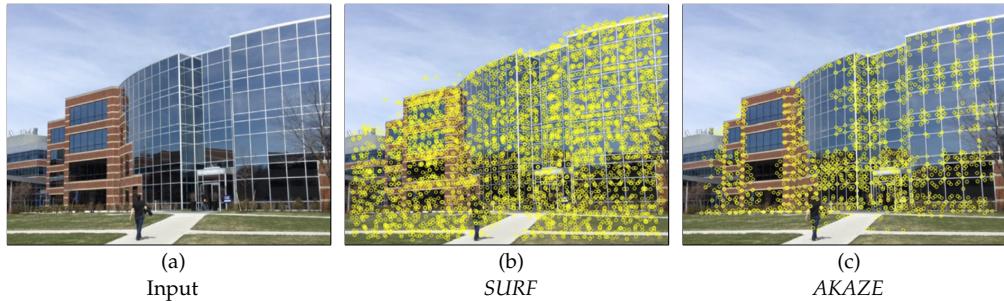


Figura 18: Rappresentazione di alcune tipologie di *keypoints*<sup>3</sup>

### 3.4 SHAPE DETECTION

Nel caso specifico della libreria QI-OCR, le operazioni descritte di *template matching* e *feature matching* non sono le sole applicabili per la localizzazione del documento d'identità nell'immagine in input. Infatti, considerando le varie tipologie di documenti presenti in territorio internazionale, è possibile osservare una caratteristica comune a tutti, ovvero la loro *forma rettangolare*, più o meno stondata ai vertici. Questo *pattern* consente quindi l'esplorazione di strategie alternative a quelle già menzionate, permettendo l'utilizzo di alcuni algoritmi di *shape detection*.

<sup>2</sup> La libreria *OpenCV* si appoggia alla libreria esterna *FLANN* (*Fast Library for Approximate Nearest Neighbors*) per questo tipo di operazioni.

<sup>3</sup> Fonte: *A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK* [19]

In particolare, la procedura utilizzata prevede l'individuazione dei bordi presenti nell'immagine e la selezione del contorno di forma approssimativamente rettangolare di area massima, supponendo, appunto, che tale forma sia proprio il documento cercato. La supposizione appena effettuata è consistente, poiché è possibile presumere che anche nel caso in cui il documento d'identità abbia subito svariate trasformazioni geometriche e la qualità della scansione/fotografia non sia eccelsa, questo sia comunque l'oggetto prevalente all'interno dell'immagine.

Dunque, nelle seguenti sezioni analizzeremo vari metodi di *edge detection*.

### 3.4.1 Edge detection basata sul gradiente

Questi metodi di rilevamento dei bordi si basano sul concetto di *derivata* di un'immagine, dato che questa consente l'analisi di cambiamenti in intensità all'interno di una funzione, ovvero la presenza di un bordo all'interno di un'immagine. Solitamente, la derivata parziale  $f_x = \frac{\partial f}{\partial x}(x, y)$  di una funzione  $f(x, y)$  può essere calcolata solamente se questa è definita nel *continuo*, mentre non è definita nel caso *discreto*. Dunque, data una funzione discreta  $h(x, y)$ , la sua derivata parziale  $\frac{\partial h}{\partial x}$  in un punto  $(u, v)$  può essere approssimata interpolando una retta tra i valori vicini  $(u+1, v)$  e  $(u-1, v)$ , ottenendo:

$$\frac{\partial h}{\partial x}(u, v) = \frac{h(u+1, v) - h(u-1, v)}{2} \quad (3.3)$$

In questo esempio, l'approssimazione della derivata riguarda la direzione orizzontale, lungo l'asse delle ascisse, ma lo stesso metodo può essere applicato in direzione verticale, per stimare la derivata lungo l'asse delle ordinate.

A questo punto, possiamo definire il gradiente di un'immagine  $I$  in posizione  $(u, v)$  come il vettore contenente le derivate parziali rispetto ai due assi  $x, y$ :

$$\nabla I(u, v) = \begin{bmatrix} I_x(u, v) \\ I_y(u, v) \end{bmatrix} \quad (3.4)$$

Dato che il gradiente è un vettore, questo avrà modulo e direzione<sup>4</sup>:

$$|\nabla I(u, v)| = \sqrt{I_x^2(u, v) + I_y^2(u, v)} \quad (3.5)$$

---

<sup>4</sup> La direzione del bordo in posizione  $(u, v)$  è perpendicolare alla direzione del gradiente nella stessa posizione.

$$\theta(u, v) = \arctan \frac{I_x(u, v)}{I_y(u, v)} \quad (3.6)$$

Per computare il gradiente di un'immagine, nella pratica viene effettuata una convoluzione con dei particolari filtri. Per esempio, utilizzando il filtro di *Sobel-Feldman* (vedi figura 19):

$$I_x = I * \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \quad (3.7) \quad I_y = I * \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}, \quad (3.8)$$

oppure quello di *Prewitt* [20]:

$$I_x = I * \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}, \quad (3.9) \quad I_y = I * \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}, \quad (3.10)$$

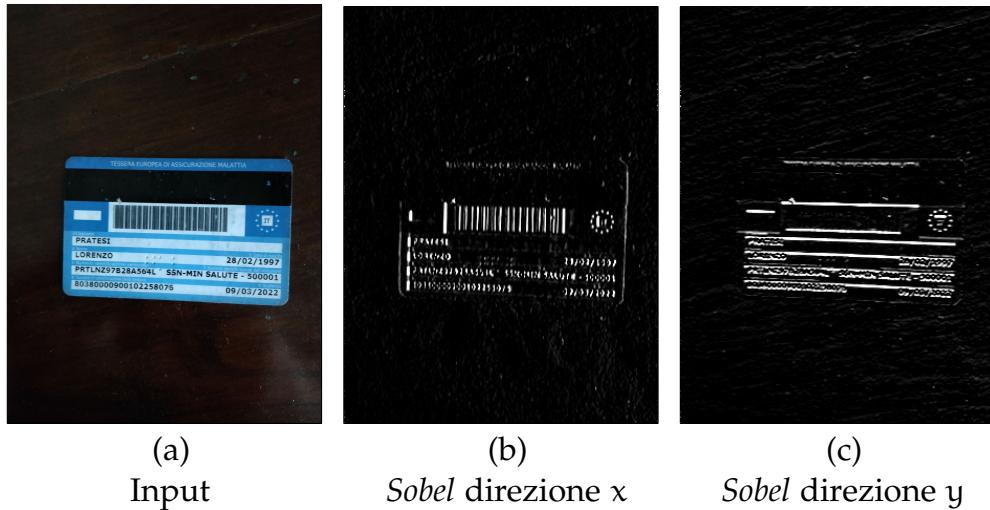


Figura 19: Operatore Sobel per edge detection

### 3.4.2 Canny edge detector

L'operatore proposto da Canny [21] per il rilevamento dei bordi è tutt'ora uno dei più utilizzati e viene considerato lo *stato dell'arte* in questo campo. L'algoritmo è suddiviso in 3 passi:

1. *Pre-processing*  $\leftarrow$  Rimozione del *noise* con un *filtro di Gauss* di dimensione  $\sigma$  e calcolo del gradiente tramite il *filtro di Sobel*
  2. *Localizzazione dei bordi*  $\leftarrow$  Assottigliamento dei bordi con una procedura di *nms (non-maxima-suppression)*. Vengono mantenuti solamente i punti di massimo locale, nella direzione del gradiente
  3. *Sogliatura con isterisi e tratteggio dei bordi*  $\leftarrow$  La *sogliatura con isterisi* prevede l'utilizzo di due valori di soglia, *thresh-low* e *thresh-high*. Per ciascun punto  $(u, v)$ , se  $|\nabla I(u, v)| \leq \text{thresh-low}$  allora il punto viene automaticamente scartato, mentre se  $|\nabla I(u, v)| \geq \text{thresh-high}$  allora il punto viene automaticamente accettato come parte di un contorno. Nel caso in cui  $\text{thresh-low} < |\nabla I(u, v)| < \text{thresh-high}$ , allora il punto viene accettato solo se *connesso* a un punto  $(u', v')$  tale che  $|\nabla I(u', v')| \geq \text{thresh-high}$

L'operatore *Canny* permette di ottenere dei buoni risultati (vedi figura 20), con un impiego di risorse relativamente basso, ma ha lo svantaggio di dipendere dai 3 parametri  $\sigma$ , `thresh-low` e `thresh-high`, strettamente dipendenti dal contesto di applicazione.

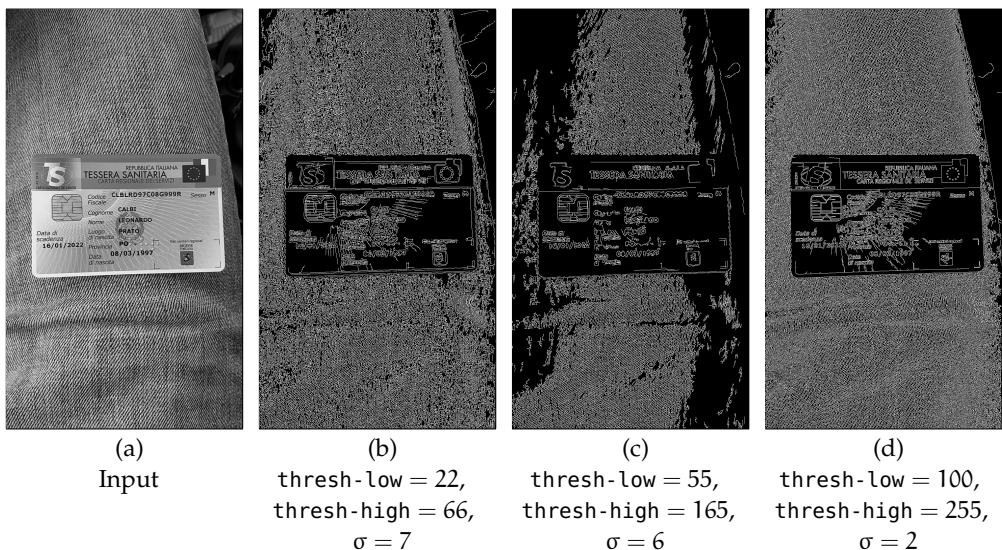


Figura 20: Operatore Canny con diversi parametri

### 3.4.3 HED

L'approccio ideale per i problemi di *edge detection* sarebbe quello di utilizzare un algoritmo che prende in input un'immagine e restituisce un'immagine binaria contenente informazioni sui bordi rilevanti dell'immagine in input, senza dover necessariamente impostare parametri diversi per contesti diversi. *HED* (*Holistically-nested Edge Detection*) [22] cerca di superare i limiti dell'operatore *Canny*, tramite l'implementazione di una rete neurale convoluzionale (*FCN - Fully Convolutional Network*). Le specifiche di questa soluzione non rientrano negli obiettivi di questa tesi, ma nelle figure 20 e 21 presentiamo un esempio di immagine processata sia dall'operatore *Canny* che dalla rete *HED*, per poter confrontare i risultati ottenuti.

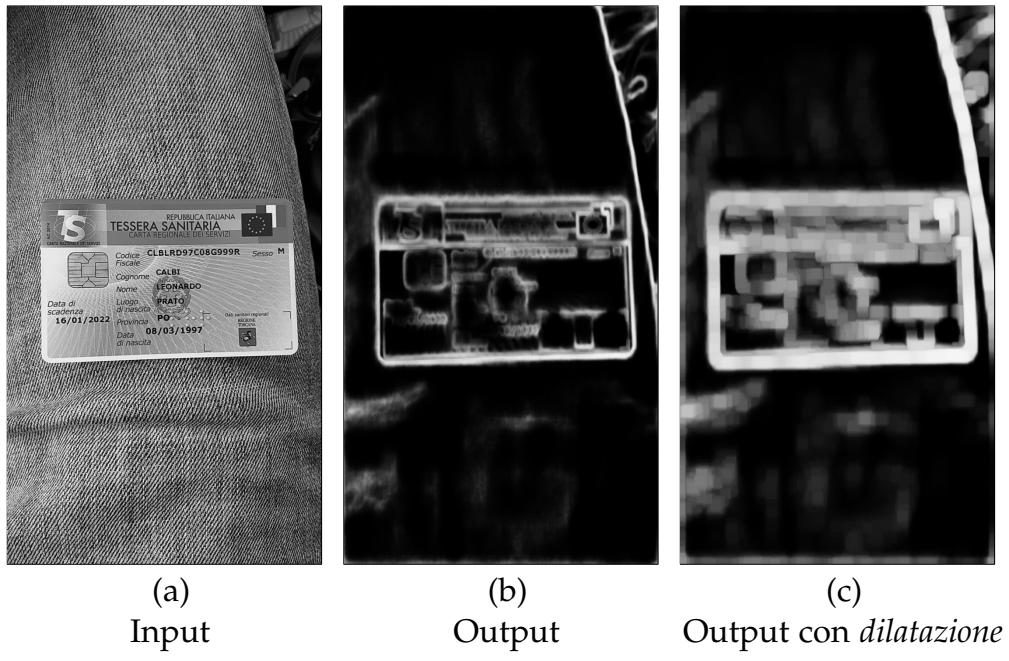


Figura 21: Output della rete *HED*

### 3.5 ANALISI COMPARATIVA

Inizialmente, la libreria QI-OCR prevedeva il solo utilizzo dell'algoritmo *SIFT* per la localizzazione del documento. A seguito di uno studio comparativo fra gli algoritmi di *image matching* sopra descritti, è stato deciso di scartare algoritmi di *template matching*, per favorire quelli di *feature*

*matching* e *shape detection*. In particolare, i primi vengono utilizzati perché invarianti per trasformazioni fotometriche e per tutte le trasformazioni geometriche menzionate, ad eccezione di quelle prospettiche; mentre i secondi vengono utilizzati perché computazionalmente efficienti e possibilmente invarianti anche per trasformazioni prospettiche, anche se non molto resistenti a cambi di illuminazione e/o contrasto dell'immagine.

Adesso, l'esecuzione della libreria prevede dunque l'utilizzo congiunto degli algoritmi *SIFT*, *SURF*, *AKAZE* e *BRISK* e della rete *HED*, tramite l'impiego di diversi *thread*. Non appena uno dei thread eseguiti torna il risultato prodotto al *processo padre*, l'esecuzione degli altri thread viene interrotta.

L'algoritmo che ha restituito i migliori risultati è *AKAZE*, che mantiene all'incirca l'accuratezza di *SIFT*, riducendo i tempi di esecuzione. L'algoritmo che invece ha restituito i peggiori risultati è *ORB*, implementato in fase preliminare e scartato successivamente. Per quanto riguarda *HED* invece, una realizzazione stabile è ancora in fase di lavorazione.

Infine, per consentire la produzione di *test massivi* e valutare l'accuratezza e la velocità ottenibile con ciascuno degli algoritmi implementati, la libreria risulta configurabile in base al tipo di localizzazione che si intende effettuare. I risultati riportati sono frutto dell'elaborazione da parte della libreria di un insieme di circa 400 documenti d'identità:

Feature matcher	Tempo medio di esecuzione	Accuratezza media
<i>AKAZE</i>	5.41s	27.24%
<i>SIFT</i>	6.94s	22.67%
<i>SURF</i>	8.68s	34.06%
<i>BRISK</i>	5.21s	26.17%
<i>All</i>	24.63s	77.49%

Tabella 5: Analisi comparativa algoritmi di *feature matching*

Come si può notare dai risultati in tabella 5, *SURF* risulta essere il miglior *feature matcher* in termini di accuratezza, mentre *BRISK* è quello che ha prestazioni migliori dal punto di vista temporale. Come già anticipato, in casi d'uso reali *AKAZE* restituisce il miglior rapporto qualità/tempo di esecuzione.

L'utilizzo congiunto dei vari *feature matcher* è descritto dalla parola chiave *All*: possiamo osservare come l'accuratezza incrementi notevolmente, con un ovvio decremento dell'efficienza della libreria.

È tutt'ora in fase di sviluppo una soluzione che permetta di ottenere dei buoni risultati in tempi ragionevoli. Questa soluzione dovrà prevedere una particolare condizione che suggerisca l'algoritmo di *image matching* più consono da utilizzare, per ogni esecuzione.

# 4

---

## TEXT DETECTION

---

### 4.1 INTRODUZIONE

Negli ultimi anni, con l'avvento delle tecnologie di intelligenza artificiale e machine learning, gli algoritmi di rilevamento e riconoscimento del testo (*text detection and recognition*) hanno subito un'evoluzione radicale, sia dal punto di vista dell'accuratezza dei risultati che dal punto di vista del tempo di esecuzione. Per quanto riguarda la *text recognition* basta osservare il salto di qualità subito da motori OCR open-source, quali *Tesseract*, a seguito dell'implementazione di reti neurali ricorrenti (*RNN*) di tipo *LSTM* (*Long Short Term Memory*), particolarmente adatte per questo tipo di applicazioni.

In questo capitolo ci concentriamo sull'introduzione di alcuni algoritmi di *text detection*, che vengono utilizzati per riconoscere regioni di testo presenti in un'immagine e presentiamo un'integrazione efficiente di questi metodi all'interno della libreria QI-OCR.

### 4.2 MSER

Il metodo presentato in [23] è un algoritmo di *blob<sup>1</sup> detection*, per la rilevazione di alcuni regioni distinguibili (*DRs - Distinguished Regions*) in un'immagine, ovvero regioni che posseggono proprietà di stabilità e invarianza, con un conseguente alto grado di ripetibilità nella rilevazione.

Introduciamo delle regioni che vengono denominate *ER* (*Extremal Regions*), nel senso che tutti i pixel interni hanno intensità esclusivamente maggiore o minore dei pixel nel contorno esterno della regione. Le *ER* godono di due importanti proprietà: sono invarianti per trasformazioni geometriche affini e per trasformazioni monotone di intensità dell'immagine.

---

<sup>1</sup> Un blob in un'immagine è un gruppo connesso di pixel che condividono una determinata proprietà.

gine. Le regioni di interesse sono un sottoinsieme delle *ER* e vengono denominate *MSER* (*Maximally Stable Extremal Regions*), poiché risultano *stabili* sotto una serie di operazioni di *thresholding* e permettono una rilevazione multi-scala delle regioni, nel senso che vengono individuate sia strutture molto piccole che molto grandi.

Il grande vantaggio di questo *blob detector* è la sua efficienza, in quanto l'enumerazione di tutte le *ER* ha una complessità temporale quasi lineare<sup>2</sup> di  $\mathcal{O}(n \log \log n)$ , data principalmente dall'algoritmo di enumerazione delle componenti connesse<sup>3</sup>, con  $n$  numero di pixel dell'immagine.

Nella pratica, l'algoritmo presentato può essere adattato per vari scopi e in particolare per il riconoscimento di testo in ambienti naturali, come descritto in [26].

In figura 22 un esempio di *text detection* con l'utilizzo dell'algoritmo *MSER* implementato nella libreria *OpenCV*:

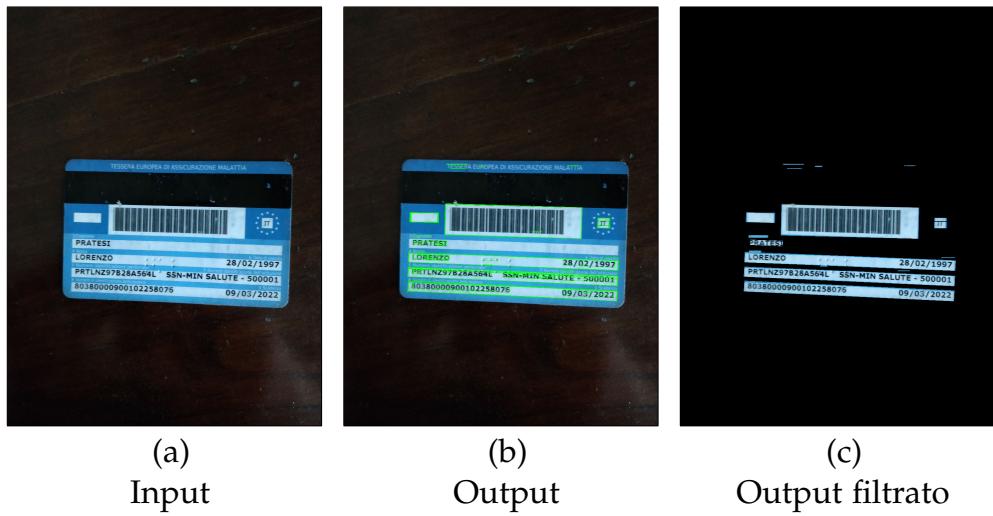


Figura 22: *Text detection* con *MSER*

<sup>2</sup> In [24] *Nistér* e *Stewénius* propongono un metodo per l'enumerazione delle *MSER* in tempo  $\mathcal{O}(n)$ , lineare nel numero di pixel dell'immagine.

<sup>3</sup> Esiste una versione più efficiente, ma più articolata, dell'algoritmo per il *labeling* delle componenti connesse (*Rosenfeld-Pfaltz* [25]) con complessità  $\mathcal{O}(n\alpha(n))$ , con  $\alpha$  inversa della funzione di *Ackermann*.

### 4.3 EAST

L'approccio ideale per i problemi di *text detection* sarebbe quello di utilizzare un algoritmo che prende in input un'immagine e restituisce una lista di rettangoli, in cui ciascun rettangolo identifica una regione di testo dell'immagine. In base al tipo di *text detection* che si intende effettuare, non orientata o orientata, ciascun rettangolo è identificato, rispettivamente, da una coppia  $(x, y)$ , lunghezza e altezza, oppure da 4 coppie  $(x, y)$  per ciascun vertice del rettangolo. Osservando l'output prodotto da MSER in figura 22, è possibile constatare che le regioni restituite non sono solamente quelle contenenti testo. Infatti, nell'esempio riportato, viene evidenziata anche la regione contenente il *codice a barre* del documento.

*EAST (Efficient and Accurate Scene Text detector)* [27] cerca di superare i limiti di *blob detector* come MSER, tramite l'implementazione di una rete neurale convoluzionale (*FCN - Fully Convolutional Network*). Le specifiche di questa soluzione non rientrano negli obiettivi di questa tesi, ma nelle figure 22 e 23 presentiamo un esempio di immagine processata sia da MSER che dalla rete *EAST*, per poter confrontare i risultati ottenuti.

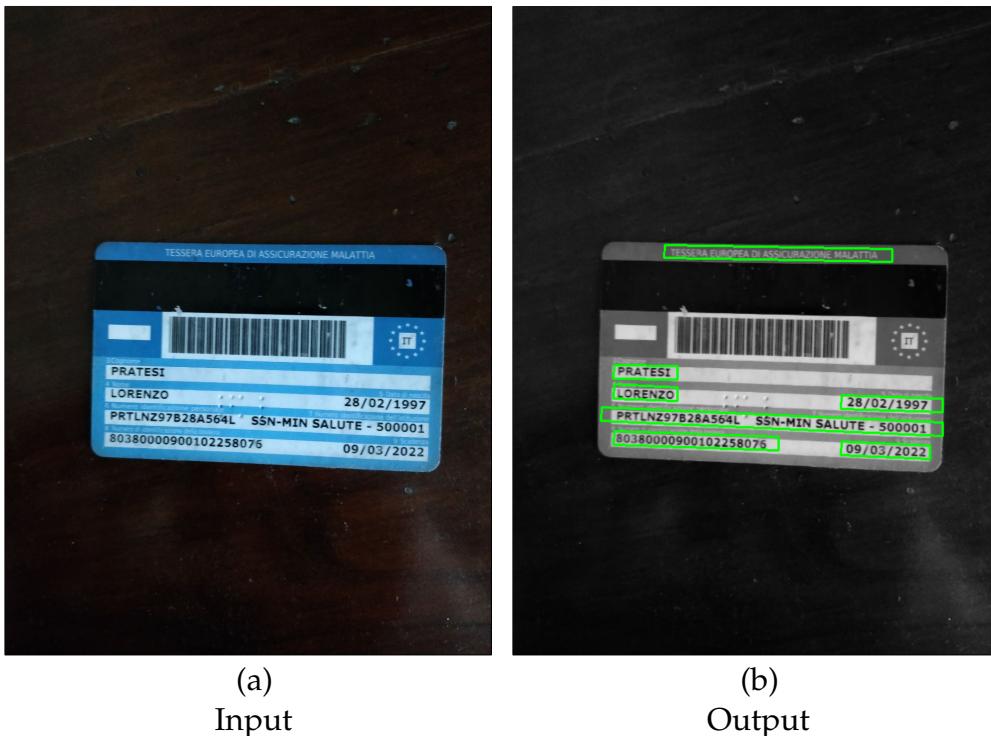


Figura 23: *Text detection* con *EAST*

#### 4.4 RITAGLIO DINAMICO DEI CAMPI DI UN DOCUMENTO

In questa sezione presentiamo un approccio dinamico per il ritaglio dei campi di un documento d'identità. Prima dell'introduzione di questo metodo, la libreria QI-OCR effettuava un ritaglio statico per ogni campo del documento in analisi, in base a coordinate predeterminate dal *template* del documento e ipotizzando un ritaglio e una centratura quasi perfetta a seguito degli algoritmi di localizzazione descritti nel capitolo 3.

In tabella 6 un esempio di coordinate definite per il retro della vecchia carta d'identità italiana, in base a una dimensione di  $700 \times 1000$  pixel, con campi e template visibili in figura 24.



Figura 24: Campi vecchia carta d'identità italiana (retro)

Campo	x	y	larghezza	altezza
Comune	100px	430px	500px	100px
Numero documento	160px	620px	470px	80px
Nome completo	100px	780px	480px	160px

Tabella 6: Coordinate campi vecchia carta d'identità italiana (retro)

Purtroppo, con il ritaglio statico utilizzato il rischio di perdita d'informazione aumenta quando la localizzazione del documento ritorna un risultato approssimativo, oppure quando si stanno estraendo campi da documenti che non rispettano esattamente i template prefissati, come la vecchia carta d'identità italiana. Per avere un'idea più concreta della

problematica menzionata, in figura 25 mostriamo un esempio di carta d'identità cartacea che non rispetta gli standard e in figura 26 i relativi campi ritagliati in modo statico.

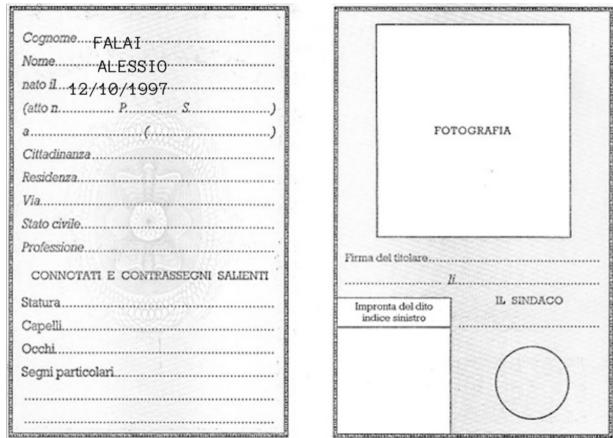


Figura 25: Vecchia carta d'identità italiana (fronte) non standard

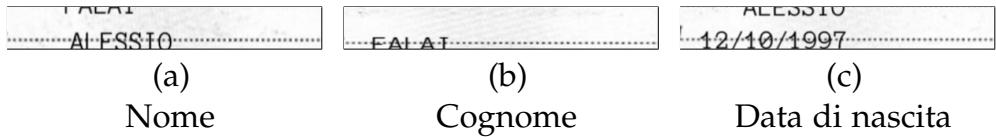


Figura 26: Campi vecchia carta d'identità italiana (fronte) non standard, ritagliati in modo statico

L'approccio proposto consente di risolvere le problematiche del ritaglio statico, quando lo scostamento delle coordinate reali da quelle predefinite non è troppo elevato. L'algoritmo in questione prevede l'individuazione delle regioni di testo in modo dinamico, tramite uno degli algoritmi di *text detection* descritti. Dopodichè, per ciascun campo, le informazioni sui *bounding boxes dinamici* vengono utilizzate per calcolare il valore di ordinata effettivo e la lunghezza corretta del campo in esame. Per svolgere questo compito, l'algoritmo computa l'intersezione del rettangolo statico con tutti i rettangoli dinamici e calcola l'area di intersezione massima. Le coordinate del *bounding box dinamico* che ha dato l'intersezione di area massima vengono utilizzate per determinare i nuovi valori di y e lunghezza del campo analizzato. Lo pseudocodice della procedura descritta è definito nell'algoritmo 3, mentre l'immagine 28 mostra i risultati ottenuti nell'esempio riportato in figura 25, grazie all'utilizzo della rete EAST (vedi figura 27).

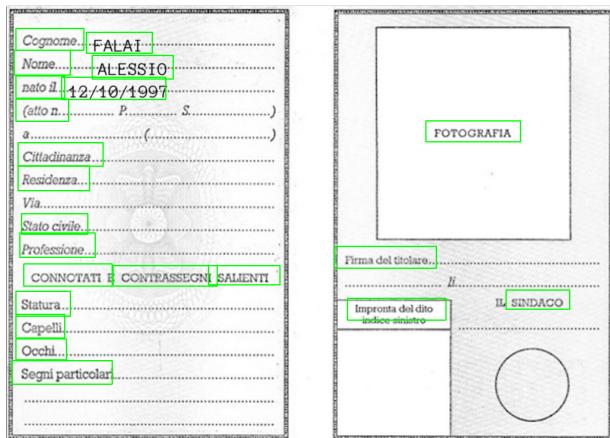


Figura 27: Output di *EAST* su una vecchia carta d'identità italiana (fronte) non standard

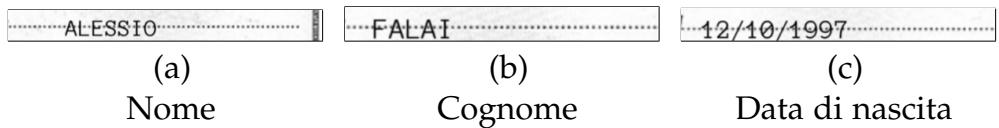


Figura 28: Campi vecchia carta d'identità italiana (fronte) non standard, ritagliati in modo dinamico

---

**Algoritmo 3** Ritaglio dinamico
 

---

```

1: function DYNAMIC-CUT
2:   image  $\leftarrow$  input image
3:   field  $\leftarrow$  rectangle(x,y,width,height)
4:   dynamic-fields  $\leftarrow$  list of rectangles(x,y,width,height)
5:
6:   max-bbox  $\leftarrow$  MAX-INTERSECTION-AREA-BBOX(field, dynamic-fields)
7:   field.y  $\leftarrow$  max-bbox.y
8:   field.width  $\leftarrow$  MAX(field.width, max-bbox.width)
9:   return image[field.y:field.y + field.height]
           [field.x:field.x + field.width]

10: function MAX-INTERSECTION-AREA-BBOX
11:   field  $\leftarrow$  rectangle(x,y,width,height)
12:   dynamic-fields  $\leftarrow$  list of rectangles(x,y,width,height)
13:
14:   max-area  $\leftarrow$  0, max-bbox  $\leftarrow$  field
15:   for dynamic-field in dynamic-fields do
16:     area  $\leftarrow$  INTERSECTION-AREA(field, dynamic-field)
17:     if area > max-area then
18:       max-area  $\leftarrow$  area
19:       max-bbox  $\leftarrow$  dynamic-field
20:   return max-bbox

21: function INTERSECTION-AREA
22:   rect-one  $\leftarrow$  rectangle(x,y,width,height)
23:   rect-two  $\leftarrow$  rectangle(x,y,width,height)
24:
25:   dx  $\leftarrow$  0, dy  $\leftarrow$  0
26:   dx = MIN(rect-one.x + rect-one.width,
              rect-two.x + rect-two.width)  $-$  MAX(rect-one.x, rect-two.x)
27:   dy = MIN(rect-one.y + rect-one.height,
              rect-two.y + rect-two.height)  $-$  MAX(rect-one.y, rect-two.y)
28:   if dx > 0  $\wedge$  dy > 0 then
29:     return dx  $\cdot$  dy
30:   return 0
  
```

---



## CONCLUSIONI E SVILUPPI FUTURI

Nel campo dell'*image processing* e del riconoscimento ottico dei caratteri (*OCR*) risulta difficile produrre risultati accurati in modo efficiente. Lo studio degli algoritmi descritti in questa tesi ha permesso l'implementazione di svariate soluzioni per apportare migliorie della libreria QIOCR in termini di correttezza e velocità di esecuzione.



Figura 29: Tessera sanitaria  
retro con *barcode*

Con il lavoro svolto, siamo riusciti a incrementare notevolmente la percentuale di campi esattamente uguali ai valori direttamente osservabili, passando da un'accuratezza del 5% a una pari a circa il 70% sul numero totale di campi processati, che sono sono alcuni dei campi dei documenti d'identità supportati. Per esempio, per la *tessera sanitaria* viene considerato solamente il *codice fiscale*, mentre per la *carta d'identità fronte* si considerano *note*. Nel caso di più campi il risultato passa il testano l'uguaglianza con il dato effettivo.

I tempi di esecuzione variano dai 5 ai 10 secondi per documento, a seconda della possibile difficoltà nelle operazioni di *image matching*. In particolare, riusciamo a ottenere i risultati migliori nel caso in cui l'immagine analizzata sia una scansione, oppure quando questa è una fotografia scattata in modo parallelo rispetto al documento inquadrato.

Per quanto riguarda il futuro, si ipotizza l'implementazione di nuovi documenti, quali il passaporto italiano, e si prevede l'aggiunta di nuove funzioni, come la lettura di codici a barre e di *MRZ* (*Machine Readable Zones*). Inoltre, si considera il supporto di documenti d'identità di altri paesi europei, in modo da poter estendere il mercato d'interesse del prodotto.



Figura 30: Passaporto italiano con *MRZ*<sup>4</sup>

<sup>4</sup> Fonte: [http://www.esteri.it/mae/doc/specifche\\_pass\\_italiani.pdf](http://www.esteri.it/mae/doc/specifche_pass_italiani.pdf)



---

## RINGRAZIAMENTI

---

Innanzitutto desidero ringraziare la mia famiglia. Ringrazio i miei genitori e i miei nonni per avermi sempre sostenuto, in ogni mia scelta. Allo stesso modo ringrazio mio fratello per essere sempre stato il mio punto di riferimento.

Ringrazio Silvia per essermi stata vicina in ogni momento e per avermi aiutato ad affrontare col sorriso qualunque difficoltà.

Ringrazio i miei compagni di studi e in particolare Leonardo, Jacopo e Lorenzo, con i quali ho felicemente condiviso tutte le mie esperienze scolastiche e di formazione più importanti.

Ringrazio i miei amici e in particolare Saul, che per me c'è sempre stato.

Infine ringrazio Michele Barbagli, per aver reso possibile la collaborazione che mi ha permesso di scrivere questa tesi e arricchire le mie conoscenze professionali, nonché di conoscere dei bravi colleghi, quali Yuri e Francesco, con i quali ho avuto, e ho tutt'ora, il piacere di lavorare.



---

## BIBLIOGRAFIA

---

- [1] Wilhelm Burger, Mark J. Burge - *Digital Image Processing - An Algorithmic Introduction Using Java, Second Edition*
- [2] Raffaele Cappelli - *Fondamenti di Elaborazione di Immagini - Operazioni sulle immagini* - Università degli Studi di Bologna
- [3] Wikipedia - <https://it.wikipedia.org>
- [4] Guocheng Wang, Yiwen Wang, Hui Li, Xuanqi Chen, Haitao Lu, Yanpeng Ma, Chun Peng, Yijun Wang, Linyao Tang - *Morphological Background Detection and Illumination Normalization of Text Image with Poor Lighting* - PLOS ONE (Cited on page 27.)
- [5] OpenCV documentation - <https://docs.opencv.org>
- [6] Simone Parca - *Studio e sviluppo di algoritmi per l'analisi di immagini della pelle acquisite tramite un sensore capacitivo* - Università degli Studi di Bologna
- [7] Francesco Tortorella - *Elementi di geometria delle immagini digitali binarie* - Università degli studi di Cassino e del Lazio Meridionale (Cited on page 13.)
- [8] *Advanced morphological image processing* - University of Groningen (Cited on pages 15 and 16.)
- [9] W. Niblack - *An introduction to Digital Image Processing* - Prentice-Hall (Cited on page 21.)
- [10] J. Sauvola, M. Pietikainen - *Adaptive document image binarization* - Pattern Recognition 33(2), pp. 225-236 (Cited on page 21.)
- [11] Deepanshu Tyagi - *Introduction To Feature Detection And Matching* - <https://medium.com>
- [12] P. Nagabhushan, S. Nirmala - *Text Extraction in Complex Color Document Images for Enhanced Readability* (Cited on page 24.)

- [13] David G. Lowe - *Distinctive Image Features from Scale-Invariant Keypoints* - University of British Columbia (Cited on page 32.)
- [14] Herbert Bay, Tinne Tuytelaars, Luc Van Gool - *SURF: Speeded Up Robust Features* (Cited on page 32.)
- [15] Pablo Fernàndez Alcantarilla, Adrien Bartoli, Andrew J. Davison - *KAZE Features* (Cited on page 32.)
- [16] Pablo F. Alcantarilla, Jesùs Nuevo, Adrien Bartoli - *Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces* (Cited on page 32.)
- [17] Stefan Leutenegger, Margarita Chli, Roland Y. Siegwart - *BRISK: Binary Robust Invariant Scalable Keypoints* (Cited on page 32.)
- [18] Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski - *ORB: an efficient alternative to SIFT or SURF* (Cited on page 32.)
- [19] Shaharyar Ahmed Khan Tareen, Zahra Saleem - *A Comparative Analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK* (Cited on page 32.)
- [20] J. Prewitt - *Object enhancement and extraction* - Picture Processing and Psychopictorics, pp. 415-431 (Cited on page 34.)
- [21] John Canny - *A Computational Approach to Edge Detection* (Cited on page 34.)
- [22] Saining Xie, Zhuowen Tu - *Holistically-Nested Edge Detection* (Cited on page 36.)
- [23] J. Matas, O. Chum, M. Urban, T. Pajdla - *Robust Wide Baseline Stereo from Maximally Stable Extremal Regions* (Cited on page 39.)
- [24] David Nistér, Henrik Stewénius - *Linear Time Maximally Stable Extremal Regions* (Cited on page 40.)
- [25] Azriel Rosenfeld, John L. Pfaltz - *Sequential Operations in Digital Picture Processing* (Cited on page 40.)
- [26] Huizhong Chen, Sam S. Tsai, Georg Schroth, David M. Chen, Radek Grzeszczuk and Bernd Girod - *Robust text detection in natural images with edge-enhanced Maximally Stable Extremal Regions* (Cited on page 40.)

- [27] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, Jiajun Liang - *EAST: An Efficient and Accurate Scene Text Detector* (Cited on page 41.)