



# Visione

## Appunti:

### INDICE

[Appunti:](#)

[Dispense:](#)

1. Funzioni

2. Python

Note

Operatori

Espressioni Condizionali

Cicli: while e for

Liste

Tuple

Insiemi

Dizionari

Slicing

Funzioni

Unpacking

3. NumPy

Importazione

Attributi di ogni oggetto ndarray:

Creare un array: la funzione array()

Matrici

Individuazione e slicing su array

Individuare con array di booleani

4. Immagini

Immagini grayscale

Immagini a colori

Rappresentazioni HSV e HSL

Istogramma

LUT(Look Up Table)

Operazioni aritmetiche fra immagini

Differenza tra immagini

Operatori bitwise

Alpha blending

Binarizzare un'immagine grayscale

**Soglia globale**

Soglia locale

Binarizzazione canale HSL

## Dispense:

VA tutte le slide.pdf

 [https://drive.google.com/open?id=1KrAvAkMvUhUIUgcOXiwLoifIKPXxrvBx&authuser=zavattaelia%40gmail.com&usp=drive\\_fs](https://drive.google.com/open?id=1KrAvAkMvUhUIUgcOXiwLoifIKPXxrvBx&authuser=zavattaelia%40gmail.com&usp=drive_fs)

[https://s3-us-west-2.amazonaws.com/securere.notion-station.com/3d28f770-15bf-4ee7-b774-e5543847dc5/VA\\_1\\_intro.pdf](https://s3-us-west-2.amazonaws.com/securere.notion-station.com/3d28f770-15bf-4ee7-b774-e5543847dc5/VA_1_intro.pdf)

VA\_2\_Python.pdf

 [https://drive.google.com/open?id=10uBi-NjKzyXFKRYz3rvIM7cnjTEPZ&authuser=zavattaelia%40gmail.com&usp=drive\\_fs](https://drive.google.com/open?id=10uBi-NjKzyXFKRYz3rvIM7cnjTEPZ&authuser=zavattaelia%40gmail.com&usp=drive_fs)

Contrast stretching  
Equalizzazione dell'istogramma  
Trasformazioni geometriche  
Ingrandimento di un'immagine  
Trasformazioni affini  
Rotazione rispetto a un punto diverso dall'origine  
Trasformazione affine da coppie di punti corrispondenti  
Trasformazione proiettiva

5. Calibrazione  
Calibrazione camera in OpenCV  
Linee

6. Filtri  
Gestione bordi  
Filtri Lineari  
Filtri Lineari Separabili  
Sfocatura  
Regolarizzazione con un filtro gaussiano (Gaussian blur)  
Sharpening  
Bordi  
Canny edge detector  
Filtri derivativi e gradiente  
Median Filter

7. Analisi  
Trasformata distanza  
Estrazione del contorno  
Etichettatura delle componenti connesse  
Morfologia Matematica  
Dilatazione  
Erosione  
Apertura  
Chiusura

8. Movimento  
Estrazione dei frame da un video e calcolo differenza  
Esaminiamo la distribuzione dei valori di alcuni pixel di background  
L'algoritmo BackgroundSubtractorMOG2  
Miglioramento immagine binaria con le tecniche viste in precedenza  
Abbassare risoluzione immagine  
Rimuove piccole componenti connesse  
Chiude piccoli buchi e concavità  
Riempie buchi più grandi  
Rimuove piccole protuberanze  
Tracking di oggetti con Mean-shift  
Esempi di tracking di oggetti

9. Riconoscimento  
Template matching  
Machine Learning  
Deep Learning  
Utilizzare CNN(Convolutional Neural Network)  
Ottenere informazioni sui livelli che compongono la rete

Esami

Teoria

Altra teoria

Suggerimenti della simulazione

Esercizi dei vecchi esami

Binarizzazione  
Componenti connesse  
Operazioni morfologiche (apertura, chiusura, erosione, dilatazione)  
Estrarre bordi con morfologia matematica

VA\_3\_Numpy.pdf

 [https://drive.google.com/open?id=10OAm\\_YUTjQd02YiPYduBd6E5V23wFxzj&authuser=zavattaelia%40gmail.com](https://drive.google.com/open?id=10OAm_YUTjQd02YiPYduBd6E5V23wFxzj&authuser=zavattaelia%40gmail.com)

m&usp=drive\_fs  
VA\_4\_Immagini.pdf

 [https://drive.google.com/open?id=10MyycnPw0KQql9Bh9EL1K6ZjOHU7co4&authuser=zavattaelia%40gmail.com&usp=drive\\_fs](https://drive.google.com/open?id=10MyycnPw0KQql9Bh9EL1K6ZjOHU7co4&authuser=zavattaelia%40gmail.com&usp=drive_fs)

[https://s3-us-west-2.amazonaws.com/securer.notion-static.com/199c00-96a8-45a0-bf3b-f3e03b2b5b32/VA\\_5\\_Calibrazione.pdf](https://s3-us-west-2.amazonaws.com/securer.notion-static.com/199c00-96a8-45a0-bf3b-f3e03b2b5b32/VA_5_Calibrazione.pdf)

VA\_6\_Filtri.pdf

 [https://drive.google.com/open?id=10lyxQAOOkgy4KRzAOwBEtl940FavRzAn&authuser=zavattaelia%40gmail.com&usp=drive\\_e](https://drive.google.com/open?id=10lyxQAOOkgy4KRzAOwBEtl940FavRzAn&authuser=zavattaelia%40gmail.com&usp=drive_e)

[https://s3-us-west-2.amazonaws.com/securer.notion-static.com/b2d355e9-7fb2-4f60-9612-f3ee4b1a0c80/VA\\_7\\_Analisi.pdf](https://s3-us-west-2.amazonaws.com/securer.notion-static.com/b2d355e9-7fb2-4f60-9612-f3ee4b1a0c80/VA_7_Analisi.pdf)

Determinare Pixel di background di img con distanza minore di 9 pixel (secondo la metrica d8)  
dal foreground.

Contrast stretching

Gradiente

Somma righe

Media valori

Equalizzazione dell'istogramma di un'immagine grayscale

Restituire come output un'immagine

#### Esercitazioni

##### 2. Immagini

Es. 1

Es. 2

Es. 3

Es. 4

Es. 5

Es. 6

##### 3. Calibrazione

Es. 1

Es. 2

Es. 3

Es. 4

Es. 5

##### 4. Filtri

Es. 1

Es. 2

Es. 3

Es. 4

Es. 5

Es. 6

##### 5. Analisi immagini binarie

Es. 1

Es. 2

Es. 3

Es. 4

Es. 5

Es. 6

##### 6. Movimento

Es. 1

Es. 2

Es. 3

Es. 4

Es. 5

Es. 6

##### 7. Template matching

Es. 1

Es. 2

Es. 3

Es. 4

Es. 5

Es. 6

##### 8. Deep Learning

Es. 1

Es. 2

Es. 3

Es. 4

Es. 5

Es. 6

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/950746b9-a0cd-4abb-8cbe-4420eb7754ae/VA\\_8\\_Movimento.pdf](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/950746b9-a0cd-4abb-8cbe-4420eb7754ae/VA_8_Movimento.pdf)

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b185e8af-3535-46c0-b5a4-be370552ceae/VA\\_9\\_Riconoscimento.pdf](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/b185e8af-3535-46c0-b5a4-be370552ceae/VA_9_Riconoscimento.pdf)

#### Bigliettino

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/a9057fc8-d629-4c31-a292-8e930d3297a5/Bigliettino\\_Visione.docx](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/a9057fc8-d629-4c31-a292-8e930d3297a5/Bigliettino_Visione.docx)

<https://s3-us-west-2.amazonaws.com/secre.notion-statice.com/00e7190a-72eb-4910-8fea-bb37dbaaabea/Ripasso.ipynb>

# 1. Funzioni

Comandi Jupyter:

- Shift + Tab → apri documentazione
- Tab → autocompletamento

Appunti di funzioni specifiche.

```
#prendere specifico elemento di una funzione
cap.read()[1] #prende il secondo elemento della tupla restituita

#Riempie array con val
np.full_like(array,val)

#Riempie array con val random (da, a, dimArray)
np.random.randint(0 , 256 , (15 , 16))

#Carica immagini bianco e nero
cv.imread('immagini/img.png',cv.IMREAD_GRAYSCALE) #imread(percorso, filtro):

#Conversione in HSV
cv.cvtColor(img, cv.COLOR_BGR2HSV) #cvtColor(img, conversione): es. da BGR a HSV

#Separare canali
h,s,v=cv.split(img)

#Unire canali
cv.merge(h,s,v)

#Unire maschere
cv.bitwise_or(mask1, mask2) #or bit a bit
cv.bitwise_and(mask1, mask2) #intersezione con and bit a bit

#Negativo, inverti colori maschera
cv.bitwise_not(mask1)

#Trova in insieme
np.isin(trova, insieme) # true se presente

#passare nelle stringhe
immagini = [cv.imread(f'movimento/{i}.png') for i in sprite_names]

#media array
media = np.mean(img[:Ym+1,...]) #...-> prende tutto di tutte le altre dimensioni

#where(condizione, xtrue, yfalse)
np.where(cv.distanceTransform(255-img3, cv.DIST_C, 3)>4, 255, 0) #se vero -> mette 255; altrimenti 0

#eliminazione componenti connesse TODO
res[np.isin(cc,small_area)]=0 #da capire
```

## 2. Python

### Note

```
istruzione_molto_lunga =3 + 4 + 5 + 6 \ #va a capo
+ 9 + 11 + 13

#mette variabili in stringhe
img = cv.imread(f"calibrazione/sample_{i+1}.jpg")
```

### Operatori

```
#Aritmetici
** Esponenziale (x ** y)
// Divisione intera (x // y)

#bit a bit (bitwise)
& AND
| OR
^ XOR
~ NOT: Inverts all the bits
<< Zero fill left shift: Shift left by pushing zeros in from the right and let the leftmost bits fall off
>> Signed right shift: Shift right by pushing copies of the leftmost bit in from the left , and let the rightmost bits fall off
```

### Espresioni Condizionali

```
if b > a:
    print("b>a")
elif b == a:
    print("a=b")
else:
    print("b<a")

print("b>a") if b>a else print("b<=a")
```

### Cicli: while e for

```
i=0
while True:
    i+=1
    if i==4:
        continue #Interrompe l'iterazione corrente e passa alla successiva
    print(i)
    if i==6:
        break #Interrompe il ciclo
else: #Blocco di codice eseguito alla fine del ciclo (anche se non ci sono iterazioni) a meno che non si interrompa con "break"
    print("Fine del ciclo")

colori = ["rosso", "verde", "blu"]
for x in colori:
    print(x)

for x in "Visione Artificiale": #lettera per lettera
    print(x)

for i in range(6): # da 1 a 5
    print(i)
```

### Liste

elenco ordinati e modificabili

```

vuota = []
colori = ['Rosso','Verde','Blu']
colori.append('Viola')
colori.remove('Blu')
colori.extend(['Giallo', 'Azzurro'])
print(colori)

#list comprehension (output: c, collezione: for c in colori, condizione: if c[0]=='A')
coloriA = [c for c in colori if c[0]=='A']
quadrati = [x*x for x in range(1,10)]

#funzioni
len(), sort(), copy(), clear()
+ concatenazione, * moltiplicazione

```

## Tuple

come le liste ma **non** modificabili

```

colori = ('Rosso','Verde','Blu')
t1 = (3) == t1 = 3, != t1 = 3
t2 = (3,4) == t2 = 3,4

```

## Insiemi

Collezione non ordinata e non indicizzata, non può contenere duplicati, elementi non modificabili

```

colori = {'Rosso','Verde','Blu'}

#set comprehension
coloriA = {c for c in colori if c[0]=='A'}

```

## Dizionari

Collezione non ordinata e modificabile di coppie chiave=valore , non può contenere chiavi duplicate

```

studenti = {101:"C.Rossi", 103:"R.Bianchi", 112:"M.Neri"}

for mat in studenti:
    print(mat, studenti[mat])
for mat, nome in studenti.items():
    print(mat, nome)
for nome in studenti.values():
    print(nome)
rimosso = studenti.pop(103)
studenti[103] = rimosso
# Esempi di "dictionary comprehension"
studenti_105 = {m: studenti[m] for m in studenti if m<105}
somme = {(k, v): k+v for k in range(4) for v in range(4)}

```

## Slicing

a[i:j] —> seleziona gli elementi di a con indice k tale che  $i \leq k < j$  ( $j$  è escluso)

-1 —> ultimo elemento

a[::2] —> passo di 2 (uno sì uno no, parte prendendo il primo elemento)

```

a = "Python!"
print(a[4:6], a[0:2], a[:2]) # on Py Py
print(a[4:len(a)], a[4:100], a[4:]) # on! on! on!
print(a[:-3], a[-3:]) # Pyth on!
print(a[1:-1], a[:]) # ython Python!
print(a[::2]) # Pto!
print(a[1::2]) # yhn
print(a[::-1]) # !nohtyP

```

```

print(a[-3:-2]) # o!
print(a[-1:-4:-2]) # !o
a = [1, 2, 3, 4, 5, 6, 7, 8, 9]
print(a[-2:-2]) # [8, 6, 4, 2]
a[3:] = a[3:] #?
print(a) # [4, 5, 6, 7, 8, 9, 4, 5, 6, 7, 8, 9]
a[::2] = [0]*6
print(a) # [0, 5, 0, 7, 0, 9, 0, 5, 0, 7, 0, 9]
a[3:-3] = [] #?
print(a) # [0, 5, 0, 7, 0, 9]
a[:] = a[::-1]
print(a) # [9, 0, 7, 0, 5, 0]

```

## Funzioni

```

def calcola(x, y):
    return x * y

#lambda consente di definire semplici funzioni anonime costituite da una singola espressione
f = lambda x, y: x ** y
print(f(2,3))

def prodotto(x, z = 1, *argom, **dizio):# z=1 valore di default, *argom consente di ottenere una tupla contenente i valori di eventuali
                                                # argomenti aggiuntivi passati alla funzione;
                                                # **dizio come *argom ma dizionario con argomenti <nome>=<valore>
    p = x
    for f in altri_fattori:
        p *= f
    for mat in studenti:
        print(mat, studenti[mat])
    return p + z

print(prodotto(2), prodotto(6,7), prodotto(2,2,2,2,2), prodotto(5,4,3,,5,M101="C.Rossi", M103="M.Bianchi", M111="L.Verdi"))

```

## Unpacking

Nel passaggio dei parametri:

- L'operatore \* consente di passare una sequenza (lista, tupla, ...) a una funzione che richiede un elenco di argomenti separati
- L'operatore \*\* consente di passare un dizionario a una funzione che richiede un elenco separato di argomenti <nome>=<valore>

Nell'assegnamento:

- Consente assegnamento multiplo in una sola linea
- L'operatore \* permette di catturare tutti gli elementi restanti in una lista

```

#parametri
a = ["Python", "NumPy", "OpenCV"]
s = {"M101":"C.Rossi",
      "M103":"M.Bianchi",
      "M111":"L.Verdi"}
prodotto(2, *a, **s)

#assegnamento
x, y, z = 2, 3, 4 # (x, y, z) = (2, 3, 4)
a1, a2, a3 = a
a1, *r = a
primo, secondo, *altri, ultimo = range(10)

```

## 3. NumPy

NumPy (Numerical Python): package fondamentale per il calcolo scientifico in Python.

- Consente di lavorare in modo efficiente su grandi quantità di dati memorizzati come vettori e matrici.

### Importazione

```
import numpy as np
print(np.__version__)
```

### Attributi di ogni oggetto ndarray:

- **ndim** - Il numero di dimensioni, chiamati anche assi (axes)
- **shape** - Una tupla di interi che indica il numero di elementi lungo ciascuna dimensione
- **size** - Il numero totale di elementi dell'array
- **dtype** - Il tipo degli elementi
- **itemsize** - La dimensione (in byte) di ogni elemento

### Creare un array: la funzione array()

```
a = np.array([2,3,5])
a = np.array([2,3,5], np.uint8)

a = np.array([[ 0,  1,  1,  2,  3],
              [ 5,  8, 13, 21, 34],
              [ 55, 89, 144, 233, 377]])

a = np.array([ [ [1,2] ], [ [3,4] ] ])

a = np.empty((2,7)) # 2 righe, 7 colonne

# zeros() e ones(): valori a zero/uno
print(np.zeros(5))
print(np.ones(3, np.int))

print(np.zeros((2,3)))

# arange() è simile a range() di Python
a = np.arange(100, 110, 2)
#Stampa: [100 102 104 106 108]

# identity() crea una matrice identità
a = np.identity(3)
```

## Matrici

```
A = np.array( [[1,1],
               [0,1]] )
B = np.array( [[2,0],
               [3,4]] )
print(A * B) # Prodotto elemento-per-elemento (ris. [[2 0], [0 4]] )
print(A @ B) # Prodotto tra matrici (ris. [[5 4], [3 4]] )

b = a.reshape(2, 6) # 2 righe, 6 colonne

# Scambiare le dimensioni con transpose() o .T
d = a.T
e = a.transpose()
print(d.shape, e.shape)
# squeeze() elimina eventuali dimensioni a uno
f = c.squeeze()
```

### Indicizzazione e slicing su array

- monodimensionali

```

a = np.arange(10)**3
print(a) #[ 0 1 8 27 64 125 216 343 512 729]
print(a[2]) # Accesso a un elemento
print(a[2:5]) # Slicing: dall'elemento 2 al 4 -> [ 8 27 64]

a[::6:2] = 42 # Modifica elementi di posto 0, 2, 4
print(a) # [42 1 42 27 42 125 216 343 512 729]
print(a[::-1]) # Step negativo: ordine inverso.
#->[729 512 343 216 125 42 27 42 1 42]

a[::2] += a[1::2] # a[i] += a[i+1], i=0,2,4,6,8
print(a) # [43 1 69 27 167 125 559 343 1241 729]

a[:] = -1 # Modifica tutti gli elementi ->[-1 -1 -1 -1 -1]

```

- multidimensionali

```

# Riga 2, Colonna 3
print(a[2, 3])
# Righa 0 e 1, colonna 2
print(a[:, 2])
# La colonna 1
print(a[:, 1])
# La riga 1
print(a[1, :])
# Righa 0 e 2, colonne 0 e 3
print(a[:, 2, ::3])
# Righa 1 e 2
print(a[1:3])
# Righa 0 e 1, colonne 0, 1 e 2
print(a[:, 2, :3])
# Ultime due righe
print(a[-2:])

```

## Indicizzare con array di booleani

Nell'indicizzazione con array di interi si specificano gli indici da considerare, invece in questo caso si scelgono esplicitamente quali elementi si vogliono e quali no. Il modo più semplice è utilizzare array booleani con la stessa forma dell'array originale.

```

a = np.arange(12).reshape(3,4)
print(a)
b = a > 4
print(b) # array booleano con la stessa forma di a
print(a[b]) # un array 1D con gli elementi considerati

# La selezione con elementi booleani è molto utile per
# modificare solo gli elementi che soddisfano un certo criterio.
criterio = a%3 == 0
print(criterio)
print(a[criterio])

```

## 4. Immagini

Sono matrici di valori.

### Immagini grayscale

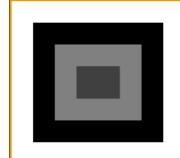
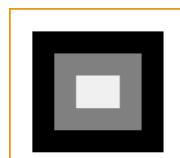
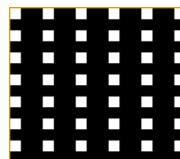
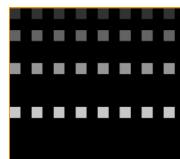
Array numpy bidimensionali. 255 = bianco, 0 = nero → 1 byte di valori

```

# Creazione di un'immagine grayscale 16x15 con un byte per pixel
img1 = np.zeros((15, 16), dtype=np.uint8) # Tutti i pixel a 0
img2 = np.full((15, 16), 255, dtype=np.uint8) # Tutti i pixel a 255

```

```

```

```

a = np.full((15, 16), 255, dtype=np.uint8)
# Slicing e broadcasting
a[2:-2, 2:-2] = 0
a[4:-4, 4:-4] = 128
a[6:-6, 6:-6] = 64

```

```

b = a.copy()
# Boolean indexing e broadcasting
mask = b==64 # mask è un array di bool
b[mask] = 240

```

```

c = np.zeros_like(a)
# Slicing e broadcasting
c[::-2, ::3] = 255

```

```

d = np.zeros((15, 16), dtype=np.uint8)
# Integer array indexing e broadcasting
y = np.array([0, 2, 5, 9, 14])
x = np.arange(0, 16, 2)
d[y[:, np.newaxis], x] = np.arange(50, 255, 50)[:, np.newaxis]

```

## Immagini a colori

3 canali dove ognuno contiene la luminosità di un colore primario (modello additivo)

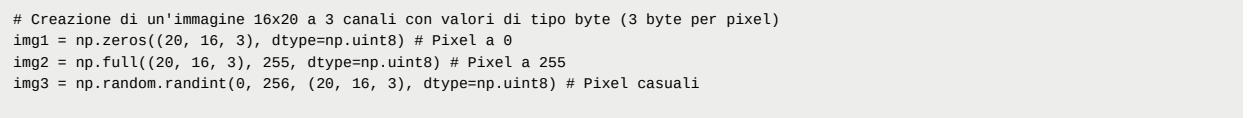
Esistono 2 Modelli RGB:

- **Modello additivo: somma dei tre canali** (il più utilizzato)
- Spazio RGB: ogni colore è un punto in uno spazio a 3 dimensioni(R,G,B) (poco idoneo alla percezione dei colori dell'uomo)

OpenCV utilizza **BGR** e non RGB

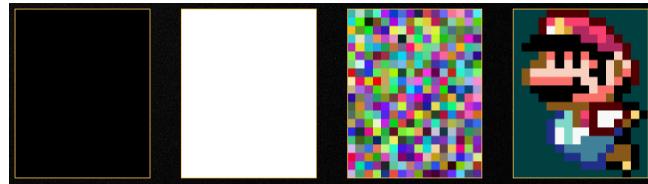
immagine come array NumPy tridimensionali

```

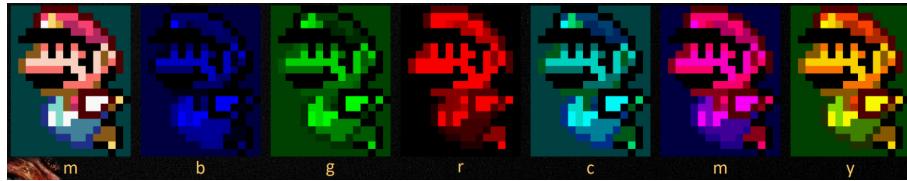
# Creazione di un'immagine 16x20 a 3 canali con valori di tipo byte (3 byte per pixel)


```

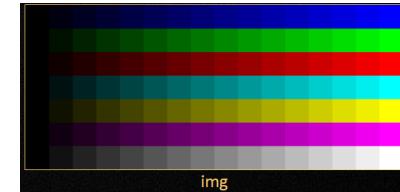
```
# Caricamento di un'immagine da file (formato BGR)
img4 = cv.imread('esempi/mario-c.png')
```



```
# Caricamento di un'immagine da file (formato BGR)
m = cv.imread('esempi/mario-c.png')
# Crea tre immagini BGR ciascuna con valori solo in un canale e gli altri due a zero
b, g, r = m.copy(), m.copy(), m.copy()
b[...,1:3], g[...,0:3:2], r[...,0:2] = 0, 0, 0
# Crea tre immagini BGR ciascuna corrispondente alla somma di due canali
c, m, y = b+g, b+r, g+r
```



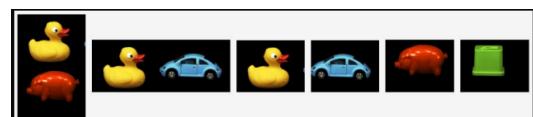
```
# Crea l'array [0 17 34 51 68 85 102 119 136 153 170 187 204 221 238 255]
a = np.arange(0, 256, 17, dtype=np.uint8)
# Immagine BGR con 7 righe e tante colonne quanti sono gli elementi di a
img = np.zeros((7, a.size, 3), dtype=np.uint8)
# Combinazione di intere array indexing, slicing e broadcasting
img[[0,3,5,6],:,0] = a
img[[1,3,4,6],:,1] = a
img[[2,4,5,6],:,2] = a
```



```
t = cv.imread('immagini/toys.png')
f = (t[::-1], t[:,::-1], t[:-1,:,:-1], t[...,:-1])

h, w = t.shape[:2]
h2, w2 = h//2, w//2
s = (t[:,::w2], t[:h2], t[:h2,:w2], t[:h2,w2:], t[h2:,:w2], t[h2:,w2:])

r = [t[::k,:,:k] for k in range(2,30,2)]
```



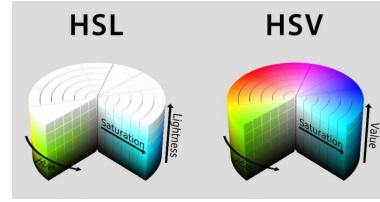
## Rappresentazioni HSV e HSL

Più vicine al modo con cui gli esseri umani percepiscono i colori.

Basate su:

- Tinta (Hue: angolo  $0^\circ$  rosso,  $120^\circ$  verde,  $240^\circ$  blu )
  - Saturazione
  - Luminosità (Value o Lightness; asse verticale da 0 nero a 1 bianco )
- In **HSV** questi colori saturi hanno **luminosità 1** mentre in **HSL** hanno luminosità **0.5**

Vantaggi: Possibilità di specificare i colori in modo intuitivo, Possono essere utilizzate più efficacemente per localizzazione e riconoscimento i oggetti nelle immagini.



Per il passaggio da RGB a HSV/HSL bisogna trasformare il cubo RGB in un cilindro

Anche in questo caso le immagini sono tensori 3D: cambia il significato dei 3 canali.

`cvtColor` per convertire da RGB a HSV/HSL e viceversa:

- COLOR\_HSV2BGR, COLOR\_BGR2HSV
- COLOR\_HLS2BGR, COLOR\_BGR2HLS

Attenzione al caso HSL: in OpenCV l'ordine è HLS (H=0, L=1, S=2)

Intervalli di valori per immagini di byte:

- H: [0..179], corrispondente a [0..2pi]
- S, V/L: [0..255], corrispondente a [0..1]

Per visualizzare o salvare le immagini, è necessario convertirle in formato BGR.

```
originale = cv.imread('immagini/toys.png')
# Converte in HLS
hls = cv.cvtColor(originale, cv.COLOR_BGR2HLS)

# Separa i tre canali in 3 immagini grayscale
h, l, s = cv.split(hls)

# Modifica dei valori HLS. Dimezza la luminosità di tutti i pixel
l1 = l//2

# Valore di saturazione 64 per tutti i pixel
s1 = np.full_like(s, 64)

# Riunisce i canali e converte in BGR
risultato = cv.cvtColor(cv.merge((h, l1, s1)), cv.COLOR_HLS2BG
R)
```

```
s = [0, 64, 128, 192, 255] # 5 livelli di saturazione
# 18 valori di luminosità in ogni colonna
v = np.tile(np.arange(255, -1, -15, np.uint8)[ :, np.newaxis], (1, 18))
# 18 valori di hue in ogni riga
h = np.tile(np.arange(0, 180, 10, np.uint8), (18, 1))
# Combina i 3 canali (si poteva usare anche cv.merge)
hsv = [np.dstack((h, np.full_like(h, x), v)) for x in s]
hls = [i[..., [0, 2, 1]] for i in hsv] # Scambia gli ultimi due canali per ottenere 2 risultati uguali (uno HSL e una HSV)
```

## Istogramma

Indica il **numero di pixel** dell'immagine per ciascun livello di grigio

Dall'istogramma si possono estrarre informazioni interessanti:

- se la maggior parte dei valori solo "condensati" in una zona, l'immagine ha uno scarno **contrasto**
- se nell'istogramma sono predominanti le basse intensità, l'immagine è molto **scura**
- puo indicare la **presenza di oggetti** nel caso di sfondi omogenei con livelli di grigio differenti dagli oggetti

```
#calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
hist = cv.calcHist([img], [0], None, [256], [0, 256]).squeeze()
#squeeze rimuove gli assi di lunghezza 1, a cosa serve qua?
```



## LUT(Look Up Table)

Precalcola i risultati di una funzione per ogni valore che un pixel dell'immagine può assumere.

Se il numero di colori o livelli di grigio è inferiore al numero di pixel nell'immagine risulta utile, dato che avremo più volte da calcolare lo stesso valore sfrutteremo la LUT al posto di calcolarlo.

```
lut = img_lut[0] #array di 1 riga
img = cv.imread('immagini/radio1.png',cv.IMREAD_GRAYSCALE)

resNP = lut[img] #numpy
#oppure
resCV = cv.LUT(img,lut) #OpenCV(migliore)
```

```
# Un esempio di funzione f (Gamma correction per un certo valore di γ)
γ = 0.5
# Calcolo di un singolo valore di f
f = lambda p: 255 * (p/255.0)**γ

# Calcolo di f su tutti i valori di un array NumPy
f_np = lambda a: f(a).astype(np.uint8)

# Calcolo dell'array LUT
lut = f_np(np.arange(256))

# Una semplice implementazione Python
def applica_py_f(img):
    res = np.empty_like(img)
    h, w = res.shape
    for y in range(h):
        for x in range(w):
            res[y,x] = f(img[y,x])
    return res
```

## LUT da grayscale a RGB

```
imgs = [cv.imread('immagini/'+n, cv.IMREAD_GRAYSCALE) for n in ('tbbt.jpg', 'radio1.png', 'radio2.png')]
maps = (cv.COLORMAP_AUTUMN, cv.COLORMAP_JET, cv.COLORMAP_HSV)

colored_imgs = [cv.applyColorMap(i, m) for m in maps for i in imgs]
```

## Operazioni aritmetiche fra immagini

### Differenza tra immagini

La sottrazione dello "sfondo" può consentire di individuare gli oggetti di interesse

```
img, back = cv.imread('esempi/mario-game.png'), cv.imread('esempi/mario-back.png')
diff = img - back # N.B. diff = cv.subtract(img, back) è più efficiente
mask = diff!=0
res = np.zeros_like(img)
res[mask] = img[mask]
```



### Operatori bitwise

Analogamente alle maschere di bit, l'operatore AND consente di azzerare selettivamente alcuni i pixel, l'operatore OR consente di impostarne il valore, etc.

```
sprite = cv.imread('esempi/sprite.png')
mask = cv.imread('esempi/mask.png')
back = cv.imread('esempi/hill.jpg')
res = back.copy()
```

```

x, y = 200, 100
h, w = sprite.shape[:2]
roi = res[y:y+h,x:x+w]
roi &= mask
roi |= sprite

#riscritto (da me) in OpenCV
roi = cv.bitwise_and(roi, mask)
roi = cv.bitwise_or(roi, sprite)

```



## Alpha blending

Combinazione fra uno sfondo (RGB) e un'immagine (RGB) con abbinato un valore di "trasparenza" per ciascun pixel (fra 0 e 1)

```

img1 = cv.imread('esempi/hill.jpg')
img2 = cv.imread('esempi/study.png')
img2_alpha = cv.imread('esempi/study-alpha.png')
h, w = img2.shape[:2]
a = img2_alpha / 255 # Trasforma il canale alpha da [0,255] a [0,1]
x, y = 200, 100
res = img1.copy()
res[y:y+h,x:x+w] = img1[y:y+h,x:x+w]*(1.0-a) + img2*a

```

## Binarizzare un'immagine grayscale

### Soglia globale

Modi per scegliere la soglia:

- Manualmente
- Osservando l'istogramma
- Metodo di Otsu: in automatico cerca sull'istogramma la soglia che minimizza la varianza intra-classe dell'intensità dei pixel delle due classi (foreground e background) determinate dalla soglia stessa.

```

# Soglia globale (128) -> ciò che è più scuro (cioè minore) viene ignorato (seleziona i valori tra 128 e 255)
img = cv.imread('esempi/bolts.png', cv.IMREAD_GRAYSCALE)
_, res = cv.threshold(img, 128, 255, cv.THRESH_BINARY) #cv.THRESH_BINARY_INV posso invertire bianco e nero (utilità?)

# Soglia globale determinata dall'algoritmo di Otsu
img = cv.imread('immagini/torre.jpg', cv.IMREAD_GRAYSCALE)
t, res = cv.threshold(img, -1, 255, cv.THRESH_OTSU)

```

### Soglia locale

Per adattare **immagini non uniformi**, determinata per ogni pixel considerando una piccola regione dell'immagine attorno ad esso (media dei pixel nella regione meno un valore costante)

```

# Soglia locale (media su intorno 11x11 meno il valore 10)
img = cv.imread('immagini/sudoku.jpg', cv.IMREAD_GRAYSCALE)
res = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 11, 10)

```

## Binarizzazione canale HSL

```

# Cosa succede binarizzando il canale saturazione di un'immagine HSL?
img = cv.imread('esempi/ferrari.jpg')
hls = cv.cvtColor(img, cv.COLOR_BGR2HLS)
_, hls[...,2] = cv.threshold(hls[...,2], 150, 255, cv.THRESH_BINARY)
res = cv.cvtColor(hls, cv.COLOR_HLS2BGR)

```



## Contrast stretching

Espansione dei livelli di grigio per aumentare il contrasto.

scegliere  $\alpha$  e  $\beta$  sull'istogramma ad esempio in corrispondenza del 5° e 95° percentile, e non valori minimo e massimo.

```
def contrast_stretching(img, a, b):
    # Converte in floating point
    n = 255*(img.astype(float)-a)/(b-a)
    # Forza il range [0,255] e converte in byte
    return np.clip(n, 0, 255).astype(np.uint8)

img = cv.imread('immagini/rice.png', cv.IMREAD_GRAYSCALE)

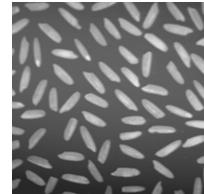
# Contrast stretching con  $\alpha=\min(I)$  e  $\beta=\max(I)$ 
res = contrast_stretching(img, img.min(), img.max())

img = cv.imread('esempi/kernel.png', cv.IMREAD_GRAYSCALE)

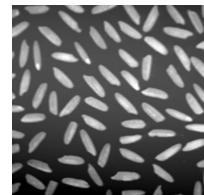
# Contrast stretching con  $\alpha=5(I)$  e  $\beta=95(I)$  (Consigliato*)
res = contrast_stretching(img, np.percentile(img, 5), np.percentile(img, 95))

a = np.percentile(img, 5)
b = np.percentile(img, 95)

n = 255*(img.astype(float)-a)/(b-a)
res = np.clip(n, 0, 255).astype(np.uint8)
```



originale



con contrast stretching

## Equalizzazione dell'istogramma

Migliora il contrasto e rende confrontabili immagini catturate in condizioni diverse di illuminazione.

Obiettivo (ideale):

- Distribuire l'istogramma uniformemente sui livelli di grigio
- Zone dell'istogramma "scarsamente popolate" vengono "compresse"
- Zone "con molti pixel" vengono "allargate"

Equalizzazione immagini a colori:

- Non è corretto applicare separatamente l'equalizzazione a ciascun canale RGB
- Si può convertire in HSL per applicare l'equalizzazione al solo canale L

```
#Equalizzazione di un'immagine grayscale
img = cv.imread('esempi/kernel.png', cv.IMREAD_GRAYSCALE)
res = cv.equalizeHist(img)

# Equalizzazione di un'immagine a colori. Esempio di come NON fare: equalizzazione di ciascun canale RGB
bgr = cv.imread('immagini/tbbt.jpg')
b, g, r = cv.split(bgr)
b_eq, g_eq, r_eq = [cv.equalizeHist(x) for x in (b, g, r)] #non equalizzare tutti i canali
res_wrong = cv.merge((b_eq, g_eq, r_eq))

# Esempio di come procedere convertendo in HSL ed equalizzando L
hls = cv.cvtColor(bgr, cv.COLOR_BGR2HLS) #converti in HSL
h, l, s = cv.split(hls) #separa
l_eq = cv.equalizeHist(l) #equalizza L
```

```

hls_eq = cv.merge((h, l_eq, s)) #unisci
res_ok = cv.cvtColor(hls_eq, cv.COLOR_HLS2BGR) #converti in RGB per vedere il risultato

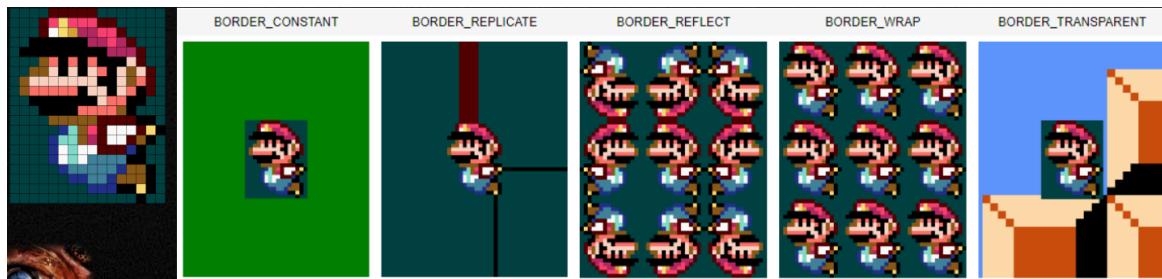
```

## Trasformazioni geometriche

Trasformazioni che si applicano alle coordinate dei pixel, non ai valori.

Uno dei problemi da risolvere sono i pixel al di fuori dei bordi, approcci comuni:

- Colore di background costante (cv.BORDER\_CONSTANT) valori a 0
- Copia del pixel più vicino (cv.BORDER\_REPLICATE)
- Riflessione dei pixel dell'immagine (cv.BORDER\_REFLECT)
- Replica dell'immagine (cv.BORDER\_WRAP)
- Nessuna modifica: lascia il valore già presente nell'immagine destinazione (cv.BORDER\_TRANSPARENT)



Un altro problema sono le coordinate non intere per cui bisogna stimare un valore, il metodo più semplice è scegliere il pixel più vicino (interpolazione nearest-neighbor).

## Ingrandimento di un'immagine

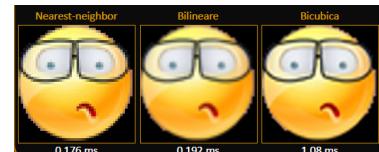
```

img = cv.imread('esempi/study.png')
h, w = img.shape[:2]
scale = 7.5
size = (int(w*scale), int(h*scale))

img_nn = cv.resize(img, size, interpolation=cv.INTER_NEAREST)
img_bl = cv.resize(img, size, interpolation=cv.INTER_LINEAR) #efficienza e velocità
img_bc = cv.resize(img, size, interpolation=cv.INTER_CUBIC) #risultati migliori ma lenta

#rimpicciolire img (?)
img_a = cv.resize(img, size, interpolation=cv.INTER_AREA)

```



## Trasformazioni affini

Le trasformazioni avvengono rispetto all'origine dell'immagine (in alto a sinistra).

warpAffine() applica una trasformazione affine a un'immagine.

```

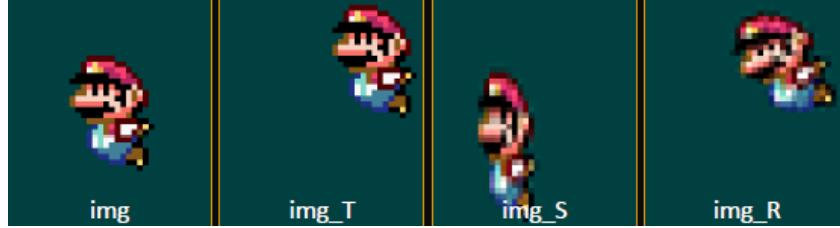
import math
img = cv.imread('esempi/mario-c.png'); b = img.shape[0]//2;
bc = img[0,0].tolist()
img = cv.copyMakeBorder(img, b, b, b, b, cv.BORDER_CONSTANT, value = bc);
h, w, _ = img.shape
trasf = lambda M: cv.warpAffine(img,M,(w,h),None,cv.INTER_CUBIC,cv.BORDER_CONSTANT,bc)
# Traslazione di 9 pixel verso destra e 9 verso l'alto
M_T = np.array([ [1., 0., 9],
                 [0., 1., -9] ])

```

```

img_T = trasf(M_T)
# Riduzione di scala x e aumento y
M_S = np.array([ [0.7, 0.0, 0],
                 [0.0, 1.3, 0] ])
img_S = trasf(M_S)
# Rotazione di - π/8 rispetto l'origine
θ = -math.pi/8; cos_θ, sin_θ = math.cos(θ), math.sin(θ)
M_R = np.array([ [cos_θ,-sin_θ, 0],
                 [sin_θ, cos_θ, 0] ])
img_R = trasf(M_R)

```



### Rotazione rispetto a un punto diverso dall'origine

```

# Rotazione rispetto al centro dell'immagine cx,cy
cx, cy = w//2, h//2
# Trasla il centro dell'immagine nell'origine
M_T1 = np.array([ [1., 0., -cx],
                  [0., 1., -cy],
                  [0., 0., 1]]) 
img_T1 = trasf(M_T1[:2])

# Rotazione di π/4
θ = math.pi/4
cos_θ, sin_θ = math.cos(θ), math.sin(θ)
M_R = np.array([ [cos_θ,-sin_θ, 0],
                 [sin_θ, cos_θ, 0],
                 [0., 0., 1]]) 
img_R = trasf(M_R[:2])

# Traslazione inversa
M_T2 = np.array([ [1., 0., cx],
                  [0., 1., cy],
                  [0., 0., 1]]) 
img_T2 = trasf(M_T2[:2])

# Compone le tre trasformazioni in una singola
M_Rc = M_T2 @ M_R @ M_T1
img_Rc = trasf(M_Rc[:2])

# restituisce direttamente la matrice a partire dalle coordinate del punto, dall'angolo (verso antiorario) e da una scala (opzionale)
res = cv.getRotationMatrix2D((x,y), angolo, scala)

```

### Trasformazione affine da coppie di punti corrisponde

`getAffineTransform()`: Date tre coppie di punti corrispondenti, calcola la matrice di trasformazione affine

$M = \begin{bmatrix} a_{00} & a_{01} & tx \\ a_{10} & a_{11} & ty \end{bmatrix}$

che mappa ciascun punto nel suo corrispondente.

```

back = cv.imread('esempi/cake.png')
m = cv.imread('esempi/mario.png')
h, w = m.shape[:2]
m_pts = np.float32([[1,1],[w-2,1],[1,h-2]])
c_pts = np.float32([
    [[ 68, 35],[106, 35],[ 68, 66]],
    # ... 3 punti per ogni quadrato nell'immagine
    [[ 4,116],[ 38,140],[ 4,163]] ])

```

```

def drawPoints(img, lp):
    for pts in lp:
        for i in range(3):
            p = tuple(pts[i].round().astype(int))
            cv.drawMarker(img, p, (255,0,0), i, 3)
    return img

back_points = drawPoints(back.copy(), c_pts)
m_points = drawPoints(m.copy(), [m_pts])

for pts in c_pts:
    M = cv.getAffineTransform(m_pts,pts)
    cv.warpAffine(m, M, back.shape[1::-1], back, cv.INTER_LINEAR, cv.BORDER_TRANSPARENT)

```

## Trasformazione proiettiva

A differenza delle trasformazioni affini, non preserva il parallelismo fra rette.

In OpenCV sono disponibili le funzioni:

- `getPerspectiveTransform`
- `warpPerspective`

```

img = cv.imread('esempi/building.jpg')
pts1 = np.float32([[195,16], [466,183], [485,269], [181,293]])
w, h = 400, 120 # Dimensioni dell'immagine destinazione
pts2 = np.float32([[0,0], [w-1,0], [w-1, h-1], [0, h-1]])
Mp = cv.getPerspectiveTransform(pts1, pts2)
res = cv.warpPerspective(img, Mp, (w,h), flags = cv.INTER_CUBIC)

```



```

m = cv.imread('esempi/sprite.png')
h, w = m.shape[:2] # Dimensioni di m
pts_m = np.float32([[0,0], [w-1,0], [w-1, h-1], [0, h-1]])
pts_b = np.float32([[243,111], [269,122], [273,220], [244,217]])
Mp = cv.getPerspectiveTransform(pts_m, pts_b)
res2 = cv.warpPerspective(m, Mp, img.shape[1::-1], img.copy(),
cv.INTER_LINEAR, cv.BORDER_TRANSPARENT)

```



## 5. Calibrazione

Consiste nel determinare i parametri estrinseci ed intrinseci di una camera.

### Parametri estrinseci:

- Matrice di rotazione  $R$
- Vettore di traslazione  $t$

### Parametri intrinseci:

- Lunghezza focale  $f$ ;
- Parametri del modello di deformazione:  $k_1, k_2, k_3, \dots$
- Coordinate del punto principale  $cx, cy$ ;
- Dimensioni dei pixel  $s_x, s_y$ ;

Disponendo di un numero sufficiente di **corrispondenze** fra punti 3D della scena e punti 2D dell'immagine, è possibile stimare i vari parametri

### Calibrazione camera in OpenCV

Funzionalità offerte:

- Proiezione di punti 3D sull'immagine (conversione da  $Pw$  a  $p$ ) dati i parametri intrinseci ed estrinseci: `cv.projectPoints()`;
- Individuazione dei vertici interni di un pattern scacchiera su un'immagine, con possibilità di raffinare la ricerca con precisione sub-pixel e di disegnare i punti trovati: `cv.findChessboardCorners()`, `cv.cornerSubPix()`, `cv.drawChessboardCorners()`;
- Stima dei parametri intrinseci ed estrinseci da un insieme di corrispondenze  $Pw$   $i$ ,  $p$   $i$  individuate in una serie di immagini di un pattern di calibrazione: `cv.calibrateCamera()`;
- Rettifica della deformazione di un'immagine dati i parametri intrinseci della camera: `cv.undistort()`, `cv.getOptimalNewCameraMatrix()`;
- Funzioni per lavorare con matrici di rotazione 3D: `cv.RQDecomp3x3()`, `cv.Rodrigues()`.

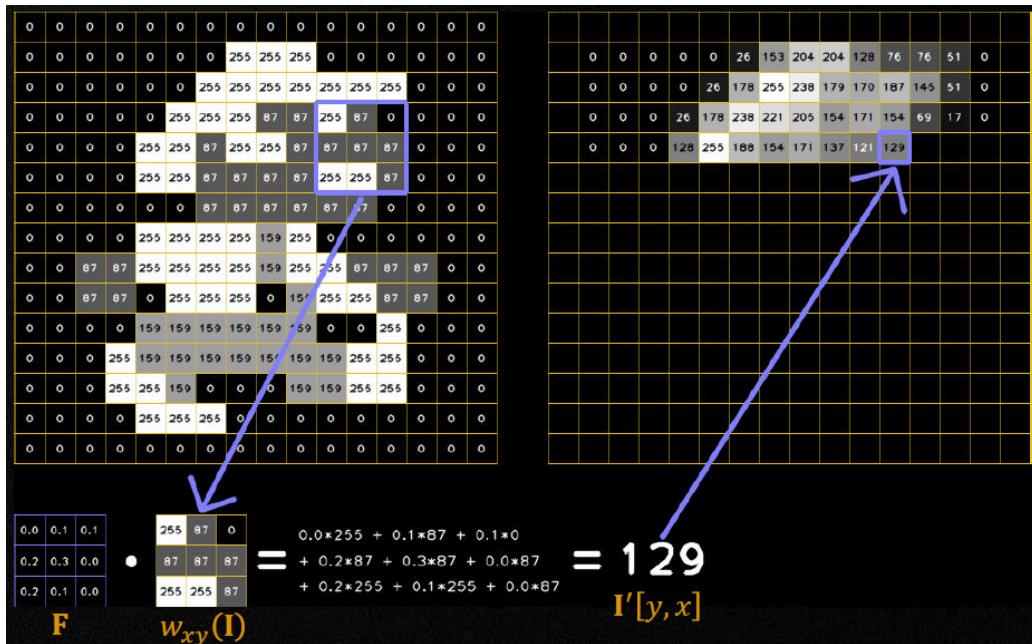
## Linee

```
#linea(img, punto partenza, punto arrivo, colore, spessore linea)
cv.line(test_image_with_lines,(498,406),(460,54),(0,0,255),1)

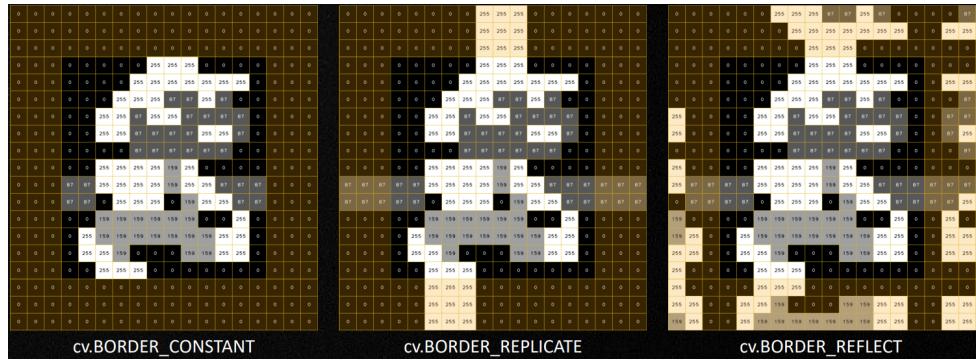
#poligono(img, [punti], isClosed, color, thickness)
image= cv.polylines(timg, np.array([punti_test]), True, (0,0,255), 2)
```

## 6. Filtri

Di norma è una matrice quadrata, con dimensioni  $m \times m$ ,  $m$  dispari. Viene applicato ad un solo pixel alla volta e il suo valore è calcolato in base ai valori dei pixel adiacenti e se stesso x il valore del filtro corrispondente.



## Gestione bordi



## Filtri Lineari

### ▼ Implementazioni python e numpy ( meno efficienti )

```
# Applicazione di un filtro lineare a un'immagine con un solo canale. Implementazione esemplificativa in Python. Il risultato è memorizzato su un'immagine di interi con segno a 16 bit
def filter2D_py(img, f):
    h, w = img.shape
    kh, kw = f.shape
    dy, dx = kh//2, kw//2
    imgb = cv.copyMakeBorder(img, dy, dy, dx, dx, cv.BORDER_DEFAULT)
    res = np.zeros(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            s = 0
            for i in range(kh):
                for j in range(kw):
                    # N.B. sarebbe imgb[(dy+y)+(i-dy), (dx+x)+(j-dx)]
                    s += imgb[y+i, x+j]*f[i, j]
            res[y, x] = s
    return res
```

```
# Applicazione di un filtro lineare a un'immagine con un solo canale Implementazione esemplificativa in Python con calcolo del valore di ciascun pixel mediante operazioni NumPy Il risultato è memorizzato su un'immagine di interi con segno a 16 bit
def filter2D_np(img, f):
    h, w = img.shape
    kh, kw = f.shape
    dy, dx = kh//2, kw//2
    imgb = cv.copyMakeBorder(img, dy, dy, dx, dx, cv.BORDER_DEFAULT)
    res = np.zeros(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            res[y, x] = (imgb[y:y+kh, x:x+kw]*f).sum()
```

```
# Misura del tempo di esecuzione delle tre implementazioni con un filtro 9x9 con tutti i coefficienti pari a uno.
img = cv.cvtColor(cv.imread('esempi/bolts.png'), cv.COLOR_BGR2GRAY)
f = np.ones((9,9))
implementazioni = [filter2D_py, filter2D_np, filter2D_cv]
for filter2D in implementazioni:
    print(filter2D.__name__, end=' ')
    %timeit filter2D(img, f)
```

```
# Applicazione di un filtro lineare a un'immagine con OpenCV. Il risultato è memorizzato su un'immagine di interi con segno a 16 bit
cv.filter2D(img, cv.CV_16S, filtro)
```

## Filtri Lineari Separabili

al posto di una matrice  $mxm \rightarrow$  due vettori  $mx1$  e  $1xm$

▼ Implementazioni python e numpy ( meno efficienti )

```
# Applicazione di un filtro lineare separabile a un'immagine con un solo canale
# Implementazione esemplificativa in Python
# Il risultato è memorizzato su un'immagine di interi con segno a 16 bit
def sepFilter2D_py(img, fx, fy):
    fx, fy = fx.ravel(), fy.ravel()
    h, w = img.shape
    kw, kh = fx.size, fy.size
    dy, dx = kh//2, kw//2
    imgb = cv.copyMakeBorder(img, 0, 0, dx, dx, cv.BORDER_DEFAULT)
    t = np.empty(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            t[y, x] = sum(imgb[y, x+j]*fx[j] for j in range(kw))
    tb = cv.copyMakeBorder(t, dy, dy, 0, 0, cv.BORDER_DEFAULT)
    res = np.empty(img.shape, dtype=np.int16)
    for y in range(h):
        for x in range(w):
            res[y, x] = sum(tb[y+i, x]*fy[i] for i in range(kh))
    return res

# Misura del tempo di esecuzione delle tre implementazioni
# con un filtro 9x9 con tutti i coefficienti pari a uno.
img = cv.cvtColor(cv.imread('esempi/bolts.png'), cv.COLOR_BGR2GRAY)
f = np.ones((9,9))
fx, fy = np.ones((9,1)), np.ones((1,9))
print("Verifica: ", np.array_equal(fx*fy, f))
implementazioni = [sepFilter2D_py, sepFilter2D_np, sepFilter2D_cv]
for sepFilter2D in implementazioni:
    print(sepFilter2D.__name__, end=': ')
    %timeit sepFilter2D(img, fx, fy)
```

```
# Applicazione di un filtro lineare separabile a un'immagine con OpenCV. Il risultato è memorizzato su un'immagine di interi con segno
# a 16 bit
cv.sepFilter2D(img, cv.CV_16S, filtrox, filtroy)
```

Di seguito alcune applicazioni dei filtri lineari.

## Sfocatura

`cv.boxFilter()`: attenua il contrasto locale, riduce il rumore

```
# Nelle immagini a colori, OpenCV applica lo stesso filtro a ciascun canale. Box filter con normalizzazione
smoothed = cv.boxFilter(img, -1, (7,7), normalize = True)
# La funzione cv.blur equivale a chiamare boxFilter con normalize=True
test = [cv.blur(img, (sx, sy)) for sx in range(1, 10, 2) for sy in range(1, 10, 2)]
```

## Regolarizzazione con un filtro gaussiano (Gaussian blur)

Per sfocatura più uniforme (no effetto griglia), produce filtri separabili

```
#apposita funzione GaussianBlur
img_blurred2 = cv.GaussianBlur(img, (5,5), 1) # m=5, σ=1

#altra applicazione. Crea un filtro gaussiano 1D e lo passa a sepFilter2D
g = cv.getGaussianKernel(5, 1) # m=5, σ=1
img_blurred = cv.sepFilter2D(img, -1, g, g)

# N.B. se σ<=0, la funzione calcola σ da m come σ = 0.3*((m-1)*0.5 - 1) + 0.8
list_img_blurred = [cv.GaussianBlur(img, (blur_size,blur_size), 0) for m in range(3,17,2)]
```

## Sharpening

Aumenta contrasto e rumore. Ci sono metodi più semplici??

```
# Metodo 1: filtro di blur e operazioni aritmetiche fra immagini
def sharpen(img, gaussian_blur, m, k):
    blurred = cv.GaussianBlur(img, (m,m), 0) if gaussian_blur else cv.blur(img, (m,m))
    mask = img.astype(np.int16) - blurred
    return np.clip(np.round(img.astype(float)+k*mask), 0, 255).astype(np.uint8)

# Metodo 2: creazione di un unico filtro che fa tutto
def create_sharpen_filter(gaussian_blur, m, k):
    if gaussian_blur:
        g = cv.getGaussianKernel(m, 0); F_b = g.reshape(1,-1)*g
    else:
        F_b = np.ones((m,m), np.float32)
    F_id = np.zeros_like(F_b); F_id[m//2, m//2] = 1
    F_b /= F_b.sum() # Normalizza il filtro di blur
    return F_id + k*(F_id - F_b)

img = cv.imread('esempi/kernel.png', cv.IMREAD_GRAYSCALE)
f = create_sharpen_filter(False, 5, 2)
res1 = sharpen(img, False, 5, 2)
res2 = cv.filter2D(img, -1, f) #migliore
```

## Bordi

### Canny edge detector

mostra i bordi, mediante isteresi

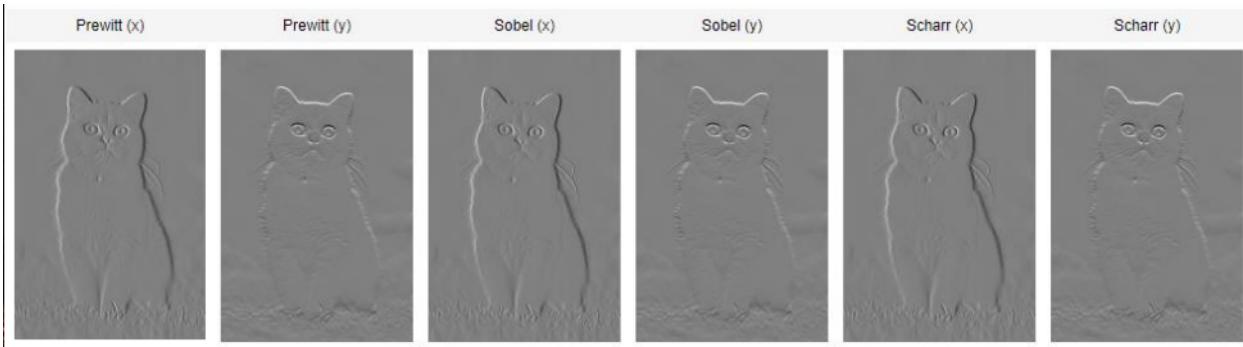
```
#Smooth gaussiano con filtro 3x3 e sigma calcolato da OpenCV
blurred = cv.GaussianBlur(cv.cvtColor(img, cv.COLOR_BGR2GRAY), (3, 3), 0)
# Algoritmo di Canny con soglie t1 e t2, t1>t2
edges = cv.Canny(blurred, 200, 100)
img_e = img.copy()
img_e[edges!=0] = (0,255,255)
```

## Filtri derivativi e gradiente

- Per applicare i filtri di Sobel e Scharr utilizzare `cv.Sobel()` e `cv.Scharr()`

```
# Calcolo derivate parziali per ogni pixel
dx, dy = cv.Sobel(img, cv.CV_32F, 1, 0, scale=1/8), cv.Sobel(img, cv.CV_32F, 0, 1, scale=1/8)
# Calcolo modulo e orientazione del gradiente per ogni pixel
mod = cv.magnitude(dx, dy)
ang = cv.phase(dx, dy, angleInDegrees=True)
```

- Si possono ottenere i filtri di Sobel e Scharr con `cv.getDerivKernels()`
- `cv.spatialGradient()` restituisce entrambe le derivate parziali calcolate con i filtri di Sobel non normalizzati



Il gradiente è perpendicolare al bordo

```
#calcola componenti x e y del gradiente
dx, dy = cv.spatialGradient (img )
dx, dy = dx.astype(np.float32), dy.astype(np.float32)
# Calcola il modulo del gradiente per ciascun pixel
m = cv.magnitude(dx, dy)
# Calcola l'angolo del gradiente per ciascun pixel (in radianti)
a = cv.phase(dx, dy)
# Converte i moduli nel range [0,255] NORMALIZZAZIONE
m = cv.normalize(m, None, 0, 255, cv.NORM_MINMAX, cv.CV_8U)
# Converte gli angoli da [0,2pi] a [0,255]
a = (a/(2*np.pi)*255).round().astype(np.uint8)
a = cv.applyColorMap(a, cv.COLORMAP_HSV) # Angolo -> Hue
# Visualizza solo i gradienti con modulo superiore a una soglia
t = 40
a1 = a.copy(); a1[m<t] = 0
m1 = m.copy(); m1[m<t] = 0
```

## Median Filter

Filtro non lineare, operazione locale in cui il valore di ciascun pixel è la mediana dei valori dei pixel nell'intorno considerato.

Partiamo da un'immagine con rumore nello sfondo (alcuni piccoli punti più chiari)

```
# Applica un filtro mediano della dimensione minima per poter eliminare il rumore
noise_reduction_median_filter = cv.medianBlur(img, 9)

# Applica un filtro gaussiano della dimensione minima per poter eliminare il rumore (solo per confronto)
noise_reduction_gaussian_filter = cv.GaussianBlur(img, (25,25), 0)
```



## 7. Analisi

### Trasformata distanza

```

dt4 = cv.distanceTransform(img, cv.DIST_L1, 3) #distanza d4 -> considera solo oriz e verticali
dt8 = cv.distanceTransform(img, cv.DIST_C, 3) #distanza d8 -> anche diagonali
#3 -> maskSize. Useremo 3 perchè lavoriamo con filtri 3x3

```

## Estrazione del contorno

Non confondere con l'estrazione del bordo che veniva fatta sulle immagini grayscale, qui lavoriamo su immagini binarie su cui è facile riconoscere il bordo. Vogliamo inseguire il contorno.

```

esempi = ('labirinto', 'leaf', 'butterfly', 'bat')
for n in esempi:
    img = cv.imread('esempi/' + n + '.png', cv.IMREAD_GRAYSCALE)
    c, _ = cv.findContours(img, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE) # RETR_EXTERNAL = sono contorni esterni (no buchi interni)
    #cv.CHAIN_APPROX_NONE = lista ordinata dei pixel;
    #cv.CHAIN_APPROX_SIMPLE = Lista dei vertici dei segmenti che lo compongono
    #cv.CHAIN_APPROX_TC89_L1 e cv.CHAIN_APPROX_TC89_KCOS = Approssimandolo con una serie di curve

    res = cv.drawContours(cv.cvtColor(img, cv.COLOR_GRAY2BGR), c, -1, (255,0,0), 2) # (img, contorni, quali vuoi stampare , colore, spesso
    re)
    # -1 signifies drawing all contours, (255,0,0) disegna contorno blu

```

### Esempio completo di estrazione del contorno

```

img = cv.imread('esempi/donuts.png')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
# Binarizza con soglia 253 (lo sfondo è bianco)
_, binarized = cv.threshold(gray, 253, 255, cv.THRESH_BINARY_INV)

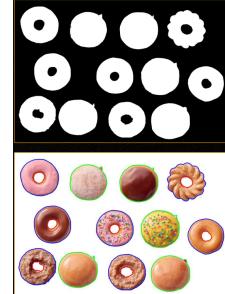
# La funzione restituisce anche una struttura dati ad albero che contiene eventuali relazioni fra contorni
# (un contorno "padre" contiene uno o più "figli")
contours, tree = cv.findContours(binarized, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

# Disegna tutti i contorni trovati
allc = cv.drawContours(img.copy(), contours, -1, (255,0,0), 5)

# Esclude i contorni con area piccola e abbina due flag booleani per sapere se il contorno ha un padre e/o figli
info = [(c, tree[0,i,2]>=0, tree[0,i,3]>=0) for i, c in enumerate(contours)
         if cv.contourArea(c)>100]

# Colora diversamente i contorni selezionati che hanno "padri" o "figli"
res = img.copy()
for c, f, p in info:
    cv.drawContours(res, [c], -1, (0,0,255) if p else (255,0,0) if f else (0,255,0), 5)

```

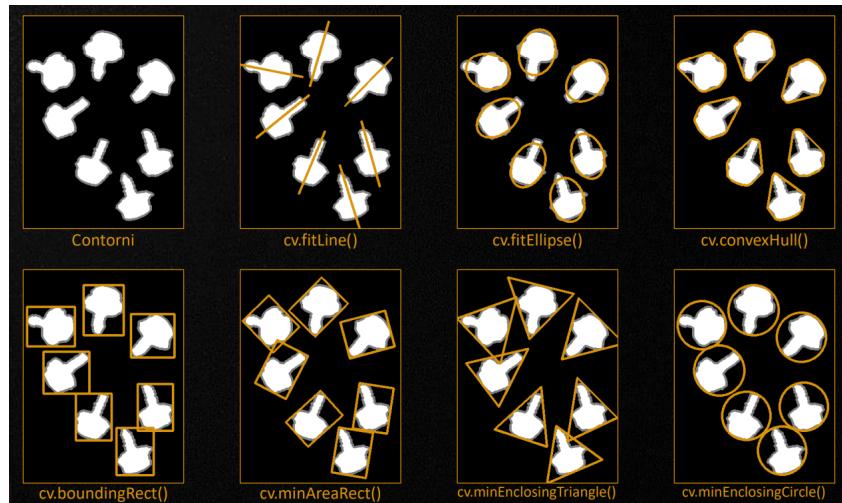


### Funzioni che si possono applicare ai contorni

```

cv.fitLine()
cv.fitEllipse()
cv.convexHull()
cv.boundingRect()
cv.minAreaRect()
cv.minEnclosingTriangle()
cv.minEnclosingCircle()

```



## Etichettatura delle componenti connesse

Individuare automaticamente le diverse componenti connesse in un'immagine, assegnando loro etichette (tipicamente numeriche)

Partiamo da un'immagine binarizzata

```
n, cc = cv.connectedComponents(img_bw)
# n è il numero di componenti connesse
# cc è un'immagine di interi in cui il valore di ogni pixel è l'indice della componente connessa a cui appartiene
print(f'Componenti connesse: {n}'')
```

con Statistiche (area, dimensioni, posizione)

```
#lettura e binarizzazione dell'immagine
img = cv.imread('esempi/bolts.png', cv.IMREAD_GRAYSCALE)
_, bw = cv.threshold(img, 152, 255, cv.THRESH_BINARY)

# La funzione cv.connectedComponentsWithStats, oltre all'immagine con le etichette, restituisce, per ogni componente connessa, le coordinate del baricentro, l'area e il bounding box.
n, cc, stats, centroids = cv.connectedComponentsWithStats(bw) # n=num componenti trovate; cc= ?; stats=statistiche; centroids= ?.

@interact(i=(0,n-1))
def show_stats(i=0):
    res = cv.cvtColor(img, cv.COLOR_GRAY2BGR)
    res[cc==i,1] = res[cc==i,1]//4 + 255*3//4

    x, y = stats[i, cv.CC_STAT_LEFT], stats[i, cv.CC_STAT_TOP]
    w, h = stats[i, cv.CC_STAT_WIDTH], stats[i, cv.CC_STAT_HEIGHT]
    area = stats[i, cv.CC_STAT_AREA]

    cv.rectangle(res, (x,y), (x+w,y+h), (255,0,0), 3)
    va.center_text(res, f'A[{i}]= {area}', centroids[i].astype(int), (0,0,255))
```

## Morfologia Matematica

applichiamo queste operazioni a immagini binarizzate

### Dilatazione

una parte dell'elemento strutturante è contenuta nell'oggetto

```
# Elemento strutturante: quadrato 15x15
se = cv.getStructuringElement(cv.MORPH_RECT, (15,15))

d = cv.morphologyEx(img, cv.MORPH_DILATE, se) # dilatazione

# Colora i pixel aggiunti dalla dilatazione
d1 = cv.cvtColor(d, cv.COLOR_GRAY2BGR)
d1[d-img!=0]=(0,255,0)
```

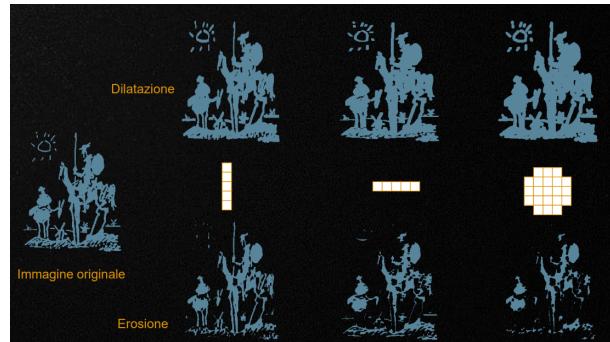
## Erosione

l'intero elemento strutturante è contenuto nell'oggetto

```
# Elemento strutturante: quadrato 15x15
se = cv.getStructuringElement(cv.MORPH_RECT, (15,15))

e = cv.morphologyEx(img, cv.MORPH_ERODE, se)#erosione

# Colora i pixel rimossi dall'erosione
e1 = cv.cvtColor(img, cv.COLOR_GRAY2BGR)
e1[e-img!=0]=(0,128,255)
```



## Apertura

erosione + dilatazione

```
#elemento strutturante: ellissi 5x5
s = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))

# L'apertura permette di risolvere alcune piccole componenti connesse dovute al rumore
res1 = cv.morphologyEx(bw, cv.MORPH_OPEN, s)
```

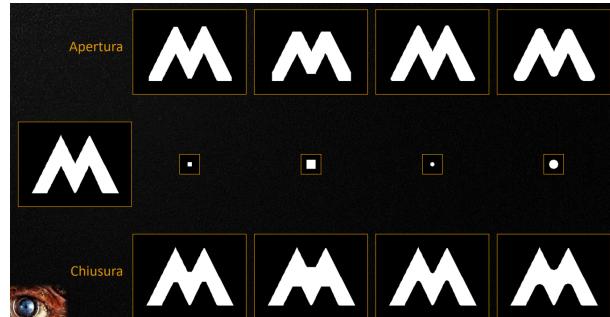
## Chiusura

dilatazione + erosione

```
#elemento strutturante: ellissi 5x5
s = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))

# La chiusura permette di risolvere alcuni "buchi" all'interno degli oggetti
res2 = cv.morphologyEx(bw, cv.MORPH_CLOSE, s)

#possiamo fare apertura + chiusura partendo dall'output dell'apertura e queste due righe qui sotto per avere l'img iniziale perfettamente ritagliata
res = img.copy()
res[res2==0]=0
```



## 8. Movimento

Approccio generale:

- costruisco il modello del background
- rilevo ciò che cambia significativamente dal background e lo identifico come foreground

### Estrazione dei frame da un video e calcolo differenza

```
# Costruisce un oggetto VideoCapture che permette di estrarre i frame da un file video
cap = cv.VideoCapture('esempi/simpsons_abbey_road.mp4')

thr = 50 # La soglia che si intende utilizzare
# Per ogni frame, convertito in grayscale, crea una maschera dei pixel la cui differenza dal precedente è maggiore di thr
fm = [] # Lista di tuple (frame, mask)
prev = None
while True:
    ret, frame = cap.read()
    if not ret: # ret è False quando non ci sono più frame
        break
    f = cv.cvtColor(frame, cv.COLOR_BGR2GRAY).astype(np.int16)
    if prev is not None:
        mask = np.zeros_like(f)
        mask[np.abs(f - prev) > thr] = 255 # A 255 i pixel la cui differenza supera thr
        fm.append( (frame, mask) ) # Lista di tuple (frame, mask)
    prev = f
cap.release() # Rilascia le risorse (in questo caso il file)
```

### Esaminiamo la distribuzione dei valori di alcuni pixel di background

cambiamenti nel background

```
# Calcolo dell'istogramma della luminosità di un pixel (x,y) lungo un video
gray_frames = [cv.cvtColor(frame, cv.COLOR_BGR2GRAY) for frame in frames]
frames3d = np.dstack(gray_frames) # Tensor 3D con tutti i frame del video
x, y = 486, 305
hist, _ = np.histogram(frames3d[y,x], 256, (0,255)) # Slicing per prendere i valori
```

### L'algoritmo BackgroundSubtractorMOG2

In caso di **variazioni "periodiche"** la distribuzione può essere multimodale: in tal caso può essere utile modellare ciascun pixel con una **Mixture of Gaussians (MoG)**. Cosa fa??

```
mog = cv.createBackgroundSubtractorMOG2(detectShadows=False)

cap = cv.VideoCapture('esempi/simpsons_abbey_road.mp4')
fm = [] # Lista di tuple (frame, mask)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    f = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    mask = mog.apply(f) # Utilizza l'algoritmo MOG2 sui frame grayscale
    fm.append( (frame, mask) )
cap.release()
```

Al metodo **apply()** è possibile passare un parametro learningRate fra 0 e 1 che controlla in che misura il modello del background è aggiornato dal nuovo frame: 0 significa che il modello non viene aggiornato, 1 che il modello è reinizializzato dal nuovo frame. Se non specificato o negativo, è scelto in automatico.

## Miglioramento immagine binaria con le tecniche viste in precedenza

### Abbassare risoluzione immagine

Per la maschera binaria è normalmente sufficiente una risoluzione più bassa.

Ridurre la risoluzione del frame prima di passarlo all'algoritmo di estrazione del foreground riduce la complessità computazionale dell'algoritmo stesso e di eventuali successive altre operazioni sulla maschera.

```
MASK_SF = 2.0
...
mask = mog.apply(cv.resize(frame, None, fx = 1/MASK_SF, fy = 1/MASK_SF))
# ... eventuali modifiche alla maschera (a risoluzione ridotta) ...
# Riscalza la maschera alle dimensioni originali
mask = cv.resize(mask, None, fx = MASK_SF, fy = MASK_SF, interpolation = cv.INTER_NEAREST)
fm.append( (frame, mask) )
```



### Rimuove piccole componenti connesse

Se è possibile fare un'ipotesi sulla dimensione minima degli oggetti in movimento, si può supporre che tutte le componenti con area minore di una certa soglia siano dovute al rumore e possano essere eliminate.

```
n, cc, stats, _ = cv.connectedComponentsWithStats(mask, connectivity=4)
small = [i for i in range(1,n) if stats[i, cv.CC_STAT_AREA]<70]
mask[np.isin(cc, small)] = 0
```

### Chiude piccoli buchi e concavità

Chiude piccoli buchi e concavità utilizzando l'operazione di chiusura della morfologia matematica. La dimensione dell'elemento strutturante va scelta con attenzione. (come si sceglie??)

```
se = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))
mask = cv.morphologyEx(mask, cv.MORPH_CLOSE, se)
```

### Riempie Buchi più grandi

Riempie eventuali altri buchi eseguendo l'estrazione delle componenti connesse sul "negativo" della maschera (ossia sul background). Componenti connesse con area inferiore a una soglia ragionevole sono normalmente "buchi" che possono essere "chiusi".

```
n, cc, stats, _ = cv.connectedComponentsWithStats(255-mask, connectivity=4) #255-mask -> inverso
holes = [i for i in range(1,n) if stats[i, cv.CC_STAT_AREA]<2000]
mask[np.isin(cc, holes)] = 255
```

### Rimuove piccole protuberanze

Rimuove eventuali piccole protuberanze mediante l'operazione di apertura della morfologia matematica.

Questo può aiutare a rimuovere eventuali artefatti dovuti al rumore che si sono "fusi" con le componenti connesse principali.

```

se = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))
mask = cv.morphologyEx(mask, cv.MORPH_OPEN, se)

```

## Tracking di oggetti con Mean-shift

Mean-shift è un algoritmo iterativo per calcolare i massimi locali, può essere usato per il tracking di un oggetto.

Partendo dal frame precedente con una regione  $W$  dove era contenuto l'oggetto, utilizza una mappa di confidenza  $C$  che indica, per ogni pixel del nuovo frame, la probabilità che tale pixel appartenga all'oggetto.

Ad ogni iterazione si calcola la media pesata di  $C$  in  $W$  e si sposta  $W$  di conseguenza; fermandosi quando la media converge.

La mappa di confidenza  $C$  può essere ottenuta analizzando il colore o altre caratteristiche dell'oggetto, come in questo caso la luminosità.

```

def get_wnd_and_hist(i, frame, stats, mask):
    """Date le statistiche delle componenti connesse stats, calcola l'istogramma della luminosità del bounding box della componente di indice i sui soli pixel indicati da mask. Restituisce bounding box e istogramma normalizzato in [0,255]."""
    x, y = stats[i, cv.CC_STAT_LEFT], stats[i, cv.CC_STAT_TOP]
    w, h = stats[i, cv.CC_STAT_WIDTH], stats[i, cv.CC_STAT_HEIGHT]
    roi = cv.cvtColor(frame[y:y+h, x:x+w], cv.COLOR_BGR2GRAY)
    hist = cv.calcHist([roi],[0],mask[y:y+h, x:x+w],[256],[0,255])
    return (x, y, w, h), hist

def get_new_wnd(frame, mask, hist, wnd):
    """Esegue meanShift sulla luminosità di frame ponendo a zero i pixel di background secondo mask, utilizzando hist come istogramma e wnd come regione di partenza."""
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # La funzione cv.calcBackProject calcola la mappa di confidenza a partire dall'istogramma e dai valori dei pixel
    conf_map = cv.calcBackProject([gray],[0],hist,[0,255],1)
    conf_map[mask==0] = 0 # Confidenza a zero fuori dalla maschera
    _, new_wnd = cv.meanShift(conf_map, wnd, term_crit)
    return new_wnd

```

## Esempi di tracking di oggetti

```

# Data la sequenza fm di tuple (frame, mask)
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )
tracked_objects = [] # (window, hist, label)
next_label = 1
res_tracking = [] # Sequenza di immagini risultato
for frame, mask in fm[1:]:
    n, cc, stats, _ = cv.connectedComponentsWithStats(mask)
    cc_used, next_tracked_objects = set(), []
    if n > 1: # Cerca di continuare a inseguire ciascuno degli oggetti
        for wnd,hist,label in tracked_objects:
            x,y,w,h = get_new_wnd(frame, mask, hist, wnd) # Applica meanShift
            # Quanti pixel per ogni etichetta ci sono dentro la nuova finestra?
            cc_hist, _ = np.histogram(cc[y:y+h, x:x+w], n, (0,n))
            index = np.argmax(cc_hist[1:])+1 # La più rappresentata (escluso back)
            if index not in cc_used and cc_hist[index]>=MIN_AREA_IN_WND:
                wnd, hist = get_wnd_and_hist(index, frame, stats, mask)
                next_tracked_objects.append( (wnd, hist, label) )
            cc_used.add(index)

    # Controlla se ci sono componenti connesse non abbinate a oggetti
    for i in range(1,n):
        if (stats[i, cv.CC_STAT_AREA] >= MIN_AREA) and (i not in cc_used):
            wnd, hist = get_wnd_and_hist(i, frame, stats, mask)
            next_tracked_objects.append( (wnd, hist, next_label) )
            next_label += 1

    tracked_objects = next_tracked_objects

    # Disegna informazioni sugli oggetti nell'immagine risultato
    res = frame.copy()
    for (x,y,w,h),hist,label in tracked_objects:

```

```

cv.rectangle(res, (x,y), (x+w,y+h), 255, 2)
cv.rectangle(res, (x,y-22), (x+w,y), 255, -1)
cv.putText(res, f'{label}', (x+5, y-5), cv.FONT_HERSHEY_PLAIN, 1, 0, 1)
# ... altre informazioni, ad es. linea che collega tutti i baricentri di ogni oggetto, contatore che indica da quanti frame è inseguito ciascun oggetto, ...
res_tracking.append(res)

```

## 9. Riconoscimento

Invece di confrontare direttamente due immagini, si definiscono uno più pattern modello (template): piccole immagini che vengono poi "cercate" all'interno dell'immagine, misurandone il grado di "somiglianza" (matching) in tutte le possibili posizioni.

La tecnica più semplice consiste nel confrontare i pixel del template con quelli dell'immagine

### Template matching

```

I = cv.imread('esempi/sheldon.jpg', cv.IMREAD_GRAYSCALE)
T = cv.imread('esempi/cat-template.png', cv.IMREAD_GRAYSCALE)
h, w = T.shape
metodi = ['cv.TM_SQDIFF', 'cv.TM_SQDIFF_NORMED',
          'cv.TM_CCORR', 'cv.TM_CCORR_NORMED',
          'cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED']

@interact(methodo=metodi)
def test_templateMatching(methodo):
    metodo = eval(methodo)
    # Confronta T con tutte le posizioni (x,y) di I, risultato in R
    R = cv.matchTemplate(I, T, metodo)
    # Cerca il minimo e il massimo di R
    r_min, r_max, pos_min, pos_max = cv.minMaxLoc(R)
    if metodo in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
        pos, val = pos_min, r_min
    else:
        pos, val = pos_max, r_max
    # Disegna un rettangolo nella miglior posizione trovata
    res = cv.rectangle(I.copy(), pos, (pos[0]+w, pos[1]+h), (0,255,255), 2)

```

## Machine Learning

Estrazione **caratteristiche** (feature) è un modo per ridurre la quantità di informazioni di un'immagine, rappresentando le parti più interessanti sotto forma di vettori numerici compatti.

Una volta che le feature sono state estratte, possono essere utilizzate per individuare e riconoscere **oggetti** con varie tecniche.

Un sistema di machine learning (apprendimento automatico) durante la fase di «training» (addestramento), apprende a partire da una serie **esempi**; dopodichè ci sarà una **scelta manuale delle features**. Successivamente, in fase di inference, è in grado di generalizzare e gestire anche nuovi dati nello stesso dominio applicativo.

### Deep Learning

Una tipologia di machine learning in cui sia la **scelta delle feature** che la **classificazione** viene appresa direttamente dagli esempi.

L'addestramento avviene utilizzando un grande set di dati etichettati e reti neurali «profonde».

### Utilizzare CNN(Convolutional Neural Network)

Reti neurali profonde specializzate per immagini e video: sono fra le tecniche di deep learning più utilizzate.

A ciascuna immagine, a diverse risoluzioni, vengono applicati filtri, il cui output viene utilizzato come input per il livello seguente. I filtri possono catturare inizialmente **feature** molto semplici, come la luminosità e i bordi, per assumere via via forme più complesse, fino a definire l'oggetto da riconoscere.

Esempio di utilizzo di una rete (semplice e già addestrata) per classificare immagini di cifre numeriche scritte a mano.  
La rete riceve in ingresso un'immagine grayscale di 28x28 pixels e restituisce un vettore di 10 elementi (corrispondenti alle cifre [0,9]): la classe più probabile secondo la rete è quella corrispondente al valore più alto nel vettore.

```
# Carica la rete da file
net = cv.dnn.readNet('minst/model.onnx')

# Carica un'immagine di test
img = cv.imread('minst/test_6.png', cv.IMREAD_GRAYSCALE)

# Fornisce l'immagine di test in ingresso alla rete
net.setInput(cv.dnn.blobFromImage(img))

# Esegue la rete per generare l'output (inferenza)
out = net.forward()
classe = np.argmax(out) # Risultato della classificazione (0..9)
```

## Ottenere informazioni sui livelli che compongono la rete

?

```
# Stampa tipo e forma dell'input e output di ogni livello
for i, l_in, l_out in zip(*net.getLayerShapes((1,1,28,28))):
    i = int(i[0]); l = net.getLayer(i)
    print(i, l.type, l_in[0].squeeze()[1:], l_out[0].squeeze()[1:])

# Esegue la rete e restituisce l'output di tutti i livelli
all_outs = net.forward(net.getLayerNames())
va.show(*all_outs[1][0]) # Visualizza l'output dei filtri del livello 1 come 8 immagini
```

# Esami

## Teoria

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/ff11b7d6-7989-4e82-b4b1-bd9324159b34/Untitled.pdf>

### 1. Cosa si intende per **LookupTable (LUT)**? A cosa può servire?

è una **struttura dati di supporto** che è utile utilizzare quando il set di valori che i pixel di un'immagine possono assumere è significativamente inferiore al numero di pixel dell'immagine stessa. Nel caso di immagine a livelli di grigio è un array lungo 255.

Si vanno a **memorizzare i risultati pre-calcolati per ogni possibile valore di input dei pixel** in modo da rendere l'elaborazione dell'immagine più efficiente, perché al posto di elaborare ogni pixel con ad esempio il valore 2, possiamo controllare il valore già elaborato di 2 sulla LUT.

### 2. Quali problemi presenta l'applicazione affine a un'immagine tramite **mapping diretto**? Com'è possibile risolverli?

Con il mapping diretto si vanno a mappare ogni pixel della vecchia immagine nella nuova immagine

Con il mapping diretto si hanno i seguenti problemi:

- Valori dei nuovi pixel non necessariamente interi (approssimazione);

- Alcuni pixel vengono mappati al di fuori della nuova immagine;
- Alcuni pixel della nuova immagine non sono coperti (“buchi”)

soluz ??

### 3. Che cos'è l'istogramma di un'immagine digitale? A cosa può servire?

indica il numero di pixel dell'immagine per ciascun livello di grigio

serve a vedere se un'immagine:

- ha basso contrasto (valori condensati in una zona)
- è scura o chiara

e può evidenziare oggetti

### 4. Cosa si intende con “convoluzione di un'immagine con un filtro digitale”? Elencare alcune applicazioni

La convoluzione è una operazione di media pesata locale, utilizza il filtro specchiato (spiegazione di filtro) e lo applica ad un'immagine

Commutativa e associativa

Applicazioni:

- produrre sfocature per nascondere imperfezioni
- evidenziare dettagli
- estrazione dei contorni

### 5. Descrivere gli spazi colore HSL/HSV e illustrarne brevemente vantaggi e svantaggi

- Tinta (Hue) → H
- Saturazione → S
- Luminosità → L oppure Valore (Value) → V

Vantaggi HSV/HSL:

- Possibilità di specificare i colori in modo intuitivo
- Possono essere utilizzate più efficacemente per localizzazione e riconoscimento di oggetti nelle immagini

Sebbene HSL e HSV servano per scegliere un singolo colore, ignorano gran parte della complessità dell'aspetto del colore ???.

in HSL il valore più alto sull'asse della luminosità corrisponde al bianco, in HSV corrisponde al colore (value)

### 6. Che cosa si intende con segmentazione degli oggetti dal background? In quali casi è un'operazione semplice? In quali casi può essere molto complicata?

La segmentazione è la separazione di uno o più oggetti di interesse dallo sfondo.

- **caso semplice:** sfondo e oggetto sono omogenei e hanno variazioni di grigio diverse
- **caso complicato:** Quando oggetti e sfondo non sono uniformi, la scelta della soglia globale è un'operazione molto critica  
→ Soglie locali

### 7. Descrivere gli operatori di Roberts e illustrarne brevemente le applicazioni.

Vengono impiegati per il calcolo del gradiente allo scopo di estrarre contorni di oggetti da un'immagine, utilizzati anche per il calcolo dei massimi locali in direzione del gradiente ad esempio nel metodo Canny edge detector.

(convoluzione di quali filtri serve saperla?)

Caratteristiche

- Pro: possono essere calcolati in modo rapido ed efficiente
- Contro: sono molto sensibili al rumore

### 8. Descrivere gli operatori di Prewitt ed illustrarne brevemente alcune applicazioni

Impiego come Roberts

Caratteristiche

- Meno sensibili a variazioni di luce e rumore
- Simmetrici rispetto al punto di applicazione
- Assi x e y orientati in modo tradizionale

9. **Descrivere gli operatori di Sobel ed illustrarne brevemente alcune applicazioni.**

impiego come roberts e prewitt

Caratteristiche

- come Prewitt
- Danno un peso maggiore al pixel centrale

10. **Descrivere sinteticamente il metodo “Canny edge detector”.**

Impiegato per il riconoscimento di contorni in un'immagine.

Prevede:

- **Smoothing gaussiano dell'immagine** (gli elementi del filtro sono pesati secondo una funzione gaussiana)
- **Calcolo del gradiente** (con operatori di roberts più efficiente, con prewitt più semplice)
- **Soppressione dei non-massimi in direzione ortogonale all'edge** (eliminati dall'immagine i pixel che non sono massimi locali rispetto l'orientaz del gradiente (?))
- **Selezione degli edge significativi mediante isteresi** (obiettivo: selezionare solo gli edge significativi; mezzo: isteresi, impiegate le soglie T1 e T2 ( $T1 > T2$ ), validi pixel con il modulo del gradiente  $> T1$ , se  $T1 > \text{val} > T2$  valido solo se adiacente a pixel validi)

11. **Descrivere sinteticamente il funzionamento della Trasformata Distanza e alcune possibili applicazioni.**

La trasformata distanza di F rispetto a  $F^*$  è una replica di F in cui i pixel sono etichettati con il valore della loro distanza da  $F^*$ , calcolata secondo una data metrica. (definizione da capire, pensavo fosse la distanza dal background)

Nel caso della metrica d4 possiamo calcolare la trasformata con due scansioni (diretta e inversa ) ... (?)

Utilizzata per:

- la misura geometrica di lunghezze e spessori ad esempio;
- calcolo dello scheletro di un oggetto;
- template matching;
- robotica (ad esempio ricerca direzione di movimento ottimale e aggiramento ostacoli)

12. **Descrivere il funzionamento dell'algoritmo di etichettatura delle componenti connesse.**

Obiettivo: individuare le componenti connesse di un'immagine assegnandogli un'etichetta numerica

Prima scansione: troviamo un pixel p di foreground e:

- se nessun pixel vicino è etichettato mettiamo a p una nuova etichetta
- se uno è etichettato mettiamo la stessa etichetta a p
- se più di uno è etichettato si assegna a p una delle etichette (e si annotano le equivalenze(?))

Seconda scansione per assegnare le etichette finali

13. **Descrivere l'algoritmo di Hilditch**

L'algoritmo di Hilditch consiste nell'eseguire più passaggi sul pattern e su ogni passaggio, l'algoritmo controlla tutti i pixel e decide di cambiare un pixel da nero a bianco se soddisfa delle determinate condizioni.

...

NON E' SULLE SLIDE

14. **Com'è possibile, con la morfologia matematica, rimuovere piccole componenti connesse da un'immagine?**

tramite l'operazione di apertura (erosione seguita da dilatazione) se si utilizza come elemento strutturante un elemento che è più grande della componente connessa che si intende eliminare

15. Com'è possibile, con la morfologia matematica, chiudere piccoli "buchi" in un'immagine binaria?

tramite l'operazione di chiusura (dilatazione seguita da erosione) se si utilizza come elemento strutturante un elemento che è più grande dei "buchi" che si intendono chiudere

16. Definire formalmente i due operatori di base della morfologia matematica e illustrarne brevemente il funzionamento.

Erosione: l'intero elemento strutturante è contenuto nell'oggetto

Dilatazione: almeno una parte dell'elemento strutturante è contenuta nell'oggetto

17. Che cosa è la "hit-or-miss transform"? A cosa può servire?

rileva una data configurazione in una immagine binaria, usando l'operatore di erosione e due elementi strutturanti disgiunti, localizza i punti in cui S1 è contenuto nel foreground e S2 nello sfondo (?)

Può servire per il pattern matching.

NON E' SULLE SLIDE

18. Si dia una definizione di "template matching rigido" e se ne discutano pregi ed difetti.

"ricerca" di un template all'interno di un'immagine con l'obiettivo di determinare se l'immagine contiene l'oggetto (match) e in quale posizione appare.

**Funzionamento:**

1. il template T è costituito da un oggetto rigido;
2. T viene sovrapposto all'immagine in tutte le possibili posizioni (rispetto agli assi X e Y);
3. Per ogni istanza  $T_i$  viene calcolato il grado di similarità massimizzando la correlazione con la porzione di immagine "coperta" dal template.

**Svantaggi:**

- difficile gestione di pattern deformabili (esempio: localizzazione di un componente elettronico che può subire diverse deformazioni);
- complessità computazionale: il numero di operazioni richieste cresce linearmente con il numero di istanze e con il numero di pixel dell'immagine e del template;

Adottando però un approccio multi-risoluzione(?) si riescono ad ottenere i seguenti vantaggi:

- "Scrematura" ai livelli iniziali;
- Una perfezione della localizzazione;
- Filtrazione delle "false somiglianze" ai livelli successivi;
- La complessità computazionale si riduce notevolmente.

19. Perché il confronto "pixel a pixel" generalmente non è efficace in applicazioni reali di ricerca di un oggetto all'interno di un'immagine?

Il confronto diretto di immagini, mediante distanza fra tali vettori nello spazio, in generale non funziona per:

- differenze di **traslazione, rotazione, scala e prospettiva**;
- **deformazione e variabilità dei pattern**;
- cambiamenti di **illuminazione**;
- presenza di **rumore** nelle immagini e utilizzo di **tecniche di acquisizione diverse**.

## Altra teoria

- **Machine/Deep Learning e CNN**

machine learning (apprendimento automatico) durante la fase di «training», apprende a partire da una serie esempi.

Successivamente, in fase di inference è in grado di gestire nuovi dati

Deep learning: l'addestramento avviene utilizzando un grande set di dati etichettati e reti neurali «profonde», ossia contenenti un numero elevato di livelli hidden (?). addestramento più lungo

Convolutional Neural Network (CNN): Reti neurali profonde specializzate per immagini e video: sono fra le tecniche di deep learning più

utilizzate. A ciascun immagine sono applicati filtri a ripetizione che catturano le funzioni più semplici (bordi, luminosità) fino a quelle più complesse.

## Suggerimenti della simulazione

```
#Promemoria: alcune funzioni che potrebbero essere utili
cv.threshold(src, thresh, maxval, type) -> retval, dst      #_, res = cv.threshold(InputImage, 128, 255, cv.THRESH_BINARY)
cv.morphologyEx(src, op, kernel) -> dst                      #c = cv.morphologyEx(res, cv.MORPH_OPEN, elemStrut)
cv.getStructuringElement(shape, ksize) -> retval            #s = cv.getStructuringElement(cv.MORPH_ELLIPSE, (7,7))
cv.distanceTransform(src, distanceType, maskSize) -> dst    #resImg = np.where(cv.distanceTransform(255-img, cv.DIST_C, 3) > 9, 255, 0).astype(np.uint8)

#Promemoria: alcune costanti che potrebbero essere utili
cv.THRESH_BINARY
cv.MORPH_ELLIPSE
cv.MORPH_ERODE, cv.MORPH_DILATE
cv.DIST_C
```

## Esercizi dei vecchi esami

### Binarizzazione

```
_, res = cv.threshold(InputImage, 128, 255, cv.THRESH_BINARY)
#può chiedere operazioni sulla soglia.

#soglia la media dei livelli di grigio dei pixel con coordinata y minore o uguale a Ym.
_, res = cv.threshold(InputImage, np.mean(img[:Ym+1,...]), 255, cv.THRESH_BINARY)

#Binarizzare InputImage utilizzando una soglia globale pari al livello medio di grigio dell'immagine.
_, res = cv.threshold(InputImage, np.mean(img), 255, cv.THRESH_BINARY)

#binarizzi un'immagine grayscale utilizzando come soglia locale il valore corrispondente al minimo fra la media dei livelli di grigio in un intorno di 5x5 pixel e la media dei livelli di grigio dell'intera immagine.
_, res = cv.threshold(InputImage, np.argmin([intorno?, np.mean(ImageInput)]), 255, cv.THRESH_BINARY) #??
res = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C, cv.THRESH_BINARY, 5, 0) #soglia locale?
```

### Componenti connesse

```
#Etichettare le componenti connesse del risultato ottenuto al passo precedente usando la metrica D4; eliminare poi le componenti connesse con un perimetro superiore a 25 pixel.
n, cc, stats, _ = cv.connectedComponentsWithStats(mask, connectivity=4) #connectivity 4 per d4?? senza il parametro connectivity è d8?
small = [i for i in range(1,n) if stats[i, cv.CC_STAT_AREA]<25] #non AREA ma perimetro (da calcolare)
mask[np.isin(cc, small)] = 0

#Etichettare le componenti connesse dell'immagine binarizzata, utilizzando 255 come valore di foreground.
n, cc, stats, _ = cv.connectedComponentsWithStats(mask) #foreground??
```

```
#Individuare eventuali componenti connesse che soddisfino la seguente condizione: almeno il 15% dei pixel ha orientazione del gradiente
(così come calcolata su InputImage) compresa nell'intervallo (-π/4, π/4).
n, cc = cv.connectedComponents(img)

dx = cv.Sobel(img, cv.CV_32F, 1, 0)
dy = cv.Sobel(img, cv.CV_32F, 0, 1)
angle = cv.phase(dx, dy, angleInDegrees = True)

for comp in range(1, n):
    ang_comp = ang[comp==cc]
    if( ( (ang_comp <= 45) | (ang_comp >= 335)).mean() < 0.15): #-pi/4 = 335, pi/4= 45
        cc[comp==cc]=0

#conta il numero di componenti connesse (considerando il foreground pari a 255) con perimetro inferiore a metà della propria area.
n, cc, stats, centroids = cv.connectedComponentsWithStats(img)
c=0
for i in range(i, n):
    if(2*stats[1:, cv.CC_STATS_WIDTH] + 2*stats[1:, cv.CC_STATS_HEIGHT] < stats[1:, cv.CC_STATS_AREA]/2)
        cc+=1
```

## Operazioni morfologiche (apertura, chiusura, erosione, dilatazione)

```
C = cv.morphologyEx(res, cv.MORPH_OPEN, cv.getStructuringElement(cv.MORPH_ELLIPSE, (7,7)))
# OPEN / CLOSE / ERODE / DILATE
# ELLIPSE / RECT
```

## Estrarre bordi con morfologia matematica

```
#Costruire un'immagine contenente solo i bordi (con uno spessore di 2 pixel) delle componenti connesse dell'immagine ottenuta al punto precedente.
img4 = img3 - cv.morphologyEx(img3, cv.MORPH_ERODE, cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5)))
#elemento strutturante di dim (5,5) -> 5 = 2*2+1 ?
```

## Determinare Pixel di background di img con distanza minore di 9 pixel (secondo la metrica d8) dal foreground.

```
resImg = np.where(cv.distanceTransform(255-img, cv.DIST_C, 3) > 9, 255, 0).astype(np.uint8)
#      DIST_C -> d8          DIST_L1 -> d4
```

Da foreground a background TODO

```
resImg = np.where(cv.distanceTransform(img, cv.DIST_C, 3)>9, 255, 0).astype(np.uint8) #tolgo il 255- ??
```

## Contrast stretching

```
#Eseguire l'operazione di contrast stretching

# Converte in floating point
n = 255*(img.astype(float)-min)/(max-min)
# Forza il range [0,255] e converte in byte
return np.clip(n, 0, 255).astype(np.uint8) #np.clip limita i valori nel range specificato
```

## Gradiente

```
#Calcolare, per ogni pixel di InputImage (esclusi eventualmente quelli di bordo), le componenti x e y del gradiente mediante convoluzione con opportuni filtri.
dx, dy = cv.spatialGradient(InputImage)

#Calcolare, per ogni pixel di InputImage (esclusi eventualmente quelli di bordo), l'orientazione del gradiente, esprimendola come angolo in radianti in [-π, π].
ang = cv.phase(dx, dy)
```

## Somma righe

```
#Calcolare, per ciascuna riga dell'immagine, la somma dei valori dei pixel: sia Ym la coordinata y della riga dell'immagine con la somma minore.
Ym = np.argmin(np.sum(img, 1)) #argmax -> se chiede la somma maggiore
```

## Media valori

```
#media dei livelli di grigio dei pixel con coordinata y minore o uguale a Ym
np.mean(img[:Ym+1,...])

#media pixel intorno 51x51, considerando eventuali pixel fuori dal bordo come aventi livello di grigio pari a zero.
mean = cv.blur(img, (51,51), borderType= cv.BORDER_CONSTANT) # cv.BORDER_CONSTANT per bordi a zero
#oppure
f = np.ones((51, 51))
f/=f.sum()
mean2 = cv.filter2D(img, cv.CV_8U , f, borderType= cv.BORDER_CONSTANT)
```

## Equalizzazione dell'istogramma di un'immagine grayscale

```
res = cv.equalizeHist(img)
```

## Restituire come output un'immagine

- BGR

```
return cv.merge(blu,green,red)
```

- Grayscale

```
#Restituire come output un'immagine grayscale (Result) in cui i pixel appartenenti alle componenti connesse individuate al passo precedente hanno valore 255, mentre tutti gli altri pixel hanno valore 0.
Result = np.zeros_like(img)
Result[mask==255]=255
return Result
```

- Int

```
#Restituire come output un'immagine di int (Result) in cui i pixel di bordo individuati al passo 4 hanno valore pari a -1
Result[C2==255]=-1
# i pixel individuati al punto 5 hanno valore pari a 0
Result[img5==255]=0
# i restanti pixel hanno valore pari al quadrato della corrispondente luminosità in InputImage.
```

```

for i in range(img.shape()):#devo scorrere i valori, come lo faccio?
    if(Result[i]!=0 & Result[i]!= 1):
        Result[i] = InputImage[i] * InputImage[i];

#oppure più semplicemente setto i restanti per primi
Result = InputImage * InputImage;#?
Result[C2==255]=-1
Result[img5==255]=0

```

- metà (arrotondata all'intero inferiore) della corrispondente luminosità in img

```
(img & (-(img4 | img5)))//2 #??
```

es completo, restituire immagine dove img4 è il canale B, img5 il canale G

```
cv.merge((img4, img5, (img & (-(img4 | img5)))//2))
```

- Calcolare l'immagine Diff in cui ciascun pixel è il valore assoluto della differenza fra il pixel corrispondente nell'immagine originale e il valore medio determinato al passo precedente.

```
Diff = np.abs(InputImage - res)
```

## Esercitazioni

### 2. Immagini

#### Es. 1

```

#immagine1 greyscale dimensione 200x256, dall'alto verso il basso deve contenere tutte le possibili sfumature
np.arange(256, dtype=np.uint8) //vettore (0,1,2,...,255)
np.arange(256, dtype=np.uint8)[ :,np.newaxis] #come reshape (255 righe e 1 colonna)
immagine1 = np.tile(np.arange(256, dtype=np.uint8)[ :,np.newaxis], (1, 200)) #ripete per 200 colonne
#tile -> costruisce un array ripetendo il primo argomento il numero di volte del secondo ar

#immagine2 come immagine1 ruotata di 90° senso antiorario (trasposta)
immagine2 = immagine1.T.copy()

#immagine3 copia di immagine1 con pixel con valore multiplo di 3 modificati in 0
immagine3 = immagine1.copy()
immagine3[immagine3 % 3 == 0] = 0

#immagine4 sotto-immagine di immagine1 con origine alle coordinate (100,150) e dimensioni 20x30
immagine4 = immagine1[150:180, 100:120].copy()

```

#### Es.2

```

#Convertire immagine rgb in grayscale
gray_toys = cv.cvtColor(toys, cv.COLOR_BGR2GRAY)

#Crea immagine toys_s dimezzando luminosità con valore > 127 e raddoppiando il resto
toys_s = gray_toys.copy()
toys_s[gray_toys > 127] //= 2
toys_s[gray_toys <= 127] *= 2
va.show(toys, gray_toys, toys_s) #visualizza immagini

#Calcolare per ogni riga la somma dei valori dei pixel
somma_per_righe = toy_s.sum(axis = 1, dtype= np.uint32)

```

### Es. 3

```
#binarizzazione dati immagine e soglia  
cv.threshold(immagine, soglia, 255, cv.THRESH_BINARY)
```

### Es. 4

```
orig = cv.imread('immagini/tbbt.jpg')  
mod = cv.imread('immagini/tbbt-m.jpg')  
diff = mod - orig #differenza tra immagini  
m[...,:,1][diff[...,:,1] != 0] #pixel del canale verde dove c'è una differenza nel canale verde (?)
```

### Es. 5

```
img_lut = cv.imread('immagini/lut.png')  
lut = img_lut[0] #1° riga  
  
#Utilizzare array lut come lookup table da grayscale a BGR  
def test(n):  
    img = cv.imread(f'immagini/radio{n}.png', cv.IMREAD_GRAYSCALE)  
    bgr = lut[img]  
    return (img, 'Originale'), (bgr, 'Con lut')
```

### Es. 6

```
h, s, v = cv.split(cv.imread('immagini/toys.png'), cv.COLOR_BRG2HSV) #dividere immagine nei 3 canali hsv che corrispondono ai 3 canali  
r = lambda h, s, v, t: (cv.cvtColor(cv.merge((h,s,v)), cv.COLOR_HSV2BGR), t)  
  
f = lambda v: np.full_like(h, v)  
  
va.show(  
    r(h, s, v, 'Originale'), (h, 'H'), (s, 'S'), (v, 'V')  
    #Tutti i pixel del canale H a valore costante, S e V invariati - Hue (Tonalità)  
    r(f(0), s, v, 'H=0'), r(f(180//4), s, v, 'H=pi/4'), r(f(180//2), s, v, 'H=pi//2'), r(f(3*180//4), s, v, 'H=3pi/4'))  
  
    #Tutti i pixel del canale S a valore costante, H e V invariati - Saturazione  
    r(h, f(0), v, 'S=0'), r(h, f(255//4), v, 'S=1/4'), r(h, f(255//2), v, 'S=1/2'), r(h, f(255), v, 'S=1')  
  
    #Tutti i pixel del canale V a valore costante, S e H invariati - Luminosità  
    r(h, s, f(0), 'V=0'), r(h, s, f(255//4), 'V=1/4'), r(h, s, f(255//2), 'V=1/2'), r(h, s, f(255), 'V=1')  
  
    max_per_row=4 #+numero massimo di immagini per riga poi va a capo da solo  
)
```

## 3. Calibrazione

### Es. 1

```
punti_test = [(195,69), (155,407), (498,406), (460,54)]  
test_image = cv.imread('calibrazione/test.png')  
test_image_with_lines = test_image.copy()  
#Disegna poligono  
cv.polyline(test_image_with_lines, punti_lines, True, (0,0,255))
```

### Es. 2

```
VERTICI_SCACCHIERA = (7,6)  
def trova_scacchiera(img):  
    CRITERI_TERMINAZIONE = (cv.TER_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
```

```

img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
ret, corners = cv.findChessboardCorners(img, VERTICI_SCACCHIERA)
if ret:
    #migliora la precisione
    corners = cv.cornerSubPix(img, corners, (11,11), (-1,-1), CRITERI_TERMINAZIONE)
return ret, corners #ret True se ha avuto successo

```

### Es. 3

```

#array NumPy con shape (42,3) contenente le coordinate (x,y,z) dei 42 vertici
po = np.array([[x,y,z] for y in range(6) for x in range(7)], dtype=np.float32)

```

### Es. 4

```

#
for i in range(12):
    img = cv.imread(f'calibrazione/sample_{i+1}.jpg')
    ret, corners = trova_scacchiera(img) #funzione es.2
    if ret:
        immagini.append(img)
        punti_imagine.append(corners)
        punti Oggetto.append(po) #es.3

```

### Es. 5

```

img_ret = cv.undistort(img, matIntr, distCoeff, newCameraMatrix = matIntr2)

```

## 4. Filtri

### Es. 1

```

#crea filtro quadrato di dim size con tutti i coefficienti pari a zero tranne quelli lungo una linea orientata in base all'angolo "angle".
#Tali valori devono essere positivi, uguali tra loro e con somma 1 (filtro normalizzato).
def create_motion_blur_filter(size=7, angle=0):
    f=np.zeros((size,size))
    #converte da gradi a radianti
    a = math.radians(angle)
    r = size//2 #coordinate del centro in intero

    dx, dy= round(math.cos(a)*r*1.5), round(math.sin(a)*r*1.5)
    cv.line(f, (r+dx, r+dy), (r-dx, r-dy), 1) #disegno la linea

    f/=f.sum() #dividendo per la somma normalizzo il filtro

```

### Es. 2

```

# 1. convertire img in scala di grigi
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

# 2. applicare un blur gaussiano con filtro quadrato di dimensione blur_size e parametro determinato in automatico a partire dalla dim
# del filtro
img = cv.gaussianBlur(img, (blur_size, blur_size), 0) #se metto 0 sceglie in automatico in base alla dim del filtro

# 3. costruire un filtro quadrato con dim emboss_size con coeff di valore... diagonale sup. -1, diagonale 0, diagonale inf. 1
f = np.tri(emboss_size, k=-1) - np.tri(emboss_size, k=-1).T #se metto k=-1 mette gli 1 sotto alla diagonale

# 4. applica filtro memorizzando su un immagine di interi con segno a 16 bit
res = cv.filter2D(img, cv.CV_16S, f) # cv.CV_16S dice che vuole immagine a 16 bit con segno

```

```
#5. normalizza il risultato e restituisci sotto forma di immagine di byte a un solo canale
res = cv.normalize(res, None, 0, 255, cv.NORM_MINMAX, cv.CV_8U) #cv.CV_8U -> risultato in un byte senza sengno
return res
```

### Es. 3

```
#completare la funzione che consenta di rilevare i bordi dell'immagine
def find_edges(img):
    ing = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    cv.Canny(cv.GaussianBlur(ing, (3,3), 0, 120, 60)) #soglie trovate con prove
```

### Es. 4

```
#completare la funzione che riceve un'immagine binaria di tutti gli edge e restituisce un array NumPy monodimensionale contenente, per
ogni posizione di x, la coordinata y del primo edge che si incontra dall'alto verso il basso.
def find_top_edge(edges):

    #restituisce la prima coordinata massima che trova
    ypos = np.argmax(edges, 0)
    return ypos
```

### Es. 5

```
#completare la funzione che data un'immagine deve convertirla in scala di grigi, applicare un blur gaussiano con filtro di dimensione 1
5x15, calcolare l'orientazione del gradiente in ciascun punto tramite opportuni filtri derivativi.
#L'angolo deve sempre essere compreso tra -90° e 90°
def find_orientation(img):
    ing = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    gx, gy = cv.spatialGradient(cv.GaussianBlur(ing, (15,15), 0))
    gx, gy = gx.astype(np.float32), gy.astype(np.float32)
    a = cv.phase(gx, gy, angleInDegrees = True)
    return (-a % 180) -90 #verso dell'angolo cambiato e considero solo i primi due quadranti (da 0° a 180°)
```

### Es. 6

```
#completare la funzione che date due immagini e due interi applica i seguenti passi:
#1. applica un filtro gaussiano per ottenere un'immagine contenente le basse frequenze di img1, quindi sottrarre il risultato da img1 per
ottenere un'immagine delle alte frequenze
img1_lo = cv.GaussianBlur(img1, None, s1)
img1_hi = img1.astype(np.int16) - img1_lo
#2. applica un filtro gaussiano per ottenere un'immagine contenente le basse frequenze di img2, quindi sottrarre il risultato da img2 per
ottenere un'immagine delle alte frequenze
img2_lo = cv.GaussianBlur(img2, None, s2)
img2_hi = img2.astype(np.int16) - img2_lo
#3. ottiene una nuova immagine res sommando le alte frequenze di img1 alle basse di img2
res = np.clip(img_hi + img_lo, 0, 255).astype(np.uint8)
```

## 5. Analisi immagini binarie

### Es. 1

```
#trovare un metodo per separare gli oggetti dello sfondo
hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
mask1 = cv.inRange(hsv, (0,0,0), (179,255,102))
mask2 = cv.inRange(hsv, (12,0,0), (23,255,255))
mask = cv.bitwise_or(mask1, mask2)
```

## Es. 2

```
#utilizzare le tecniche di morfologia matematica per cercare di risolvere i problemi (buchi concavità, elementi non ben separati, alcuni pixel del background sono erroneamente assegnati al foreground)
mask1 = cv.morphologyEx(mask, cv.MORPH_CLOSE, cv.getStructuringElement(cv.MORPH_ELLIPSE, (3,3)))
mask1 = cv.morphologyEx(mask1, cv.MORPH_OPEN, cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5)))
```

## Es. 3

```
#costruire mask a partire da mask1, chiudendo eventuali buchi presenti all'interno delle componenti connesse di mask1
count, cc, stats, _ = cv.connectedComponentsWithStats(cv.bitwise_not(mask1))

buchi = [i for i in range(1,count) if stats[i, cv.CC_STAT_AREA]<200]

mask2 = mask1.copy()
mask2[np.isin(cc, buchi)] = 255 #se il valore appartiene a ciascun indice (cc) è presente in buchi
```

## Es. 4

```
#eseguire l'algoritmo di etichettatura sulle componenti connesse su mask2 e calcolare l'area (numero di pixel) di ciascuna componente connessa
count, cc, stats, _ = cv.connectedComponentsWithStats(mask2)

def create_cc_mask(index):
    m = np.zeros_like(mask2)
    m[cc==index] = 255
    return m

cca = [(create_cc_mask(i), stats[i, cv.CC_STAT_AREA]) for i in range(1, count)]
```

## Es. 5

```
#a partire dalla lista cca, costruire una nuova lista info che contenga una tupla (m, c, a, ((xc, yc), r)) per ciascun elemento di cca, dove:
#m è la stessa maschera contenuta nel corrispondente elemento di cca, ossia la maschera della corrispondente componente connessa
#c è il contorno della componente connessa, ottenuto mediante cv.findContours()
#a è l'area della componente connessa
#((xc, yc), r) è una tupla contenente le coordinate (xc, yc) e il raggio r del minimo cerchio che racchiude tutti i pixel del contorno c.

tmp = [(m, cv.findContours(m, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)[0][0], a) for m, a in cca] #lista di tuple di 3 elementi: maschera, contorno e area
info = [(m, c, a, cv.minEnclosingCircle(c)) for m, c, a in tmp]
```

## Es. 6

```
#a partire dalla lista info, costruire una nuova lista tondi che contenga i soli elementi di info che corrispondono a cioccolatini di forma circolare.
tondi = [(m, c, a, ((xc, yc), r) for (m, c, a, ((xc, yc), r)) in info if np.pi * r**2 - a < 200]
```

## 6. Movimento

### Es. 1

```
#Aprire il video "movimento/pacman.mp4"; creare una lista Python frames che contenga i frame (come matrici NumPy) di indice dispari dal 125 al 3523 (estremi compresi, numerando i frame del video a partire da 0)
cap = cv.VideoCapture('movimento/pacman')
for _ in range(125):
    ca.read() #salto i primi 124
frames = []
for _ in range(125, 3524, 2):
    frames.append(cap.read()[1]) #con [1] prendo direttamente il frame
    cap.read() #salto frame pari
cap.release()
```

### Es. 2

```
#Creare un array sprites che contenga le immagini caricate dai file il cui nome è nell'array sprite_names
#Una volta creato l'array sprites, procedere come segue:

#define una variabile OBJ_SIZE contenente la larghezza della prima immagine in sprites;
#creare una lista di istogrammi hists in cui, per ciascuna immagine in sprites, dopo averla convertita in HSV, si calcola l'istogramma dei valori H solo dei pixel con S>=20 e V>=40 .
#L'istogramma dovrà essere un array NumPy con 15 valori, suddividendo i 180 possibili valori di H in 15 gruppi da 12 valori ciascuno. Infine i valori degli istogrammi dovranno essere normalizzati fra 0 e 100.
#Ogni istogramma dovrà essere un array NumPy di 15 np.float32.
sprite_names=['F0','F1','F2','F3','P0']

sprites = [cv.imread(f'movimento/{n}.png') for n in sprite_name]
hists = []

for sprite in sprites:
    hsv = cv.cvtColor(sprite, cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv, (0,20,40), (179,155,255))
    hist = cv.calcHist([hsv], [0], mask, [15], [0, 180]).squeeze()
    cv.normalize(hist, hist, 0, 100, cv.NORM_MINMAX)
    hists.append(hist)
OBJ_SIZE = sprites[0].shape[1]
```

### Es. 3

```
#Completare la funzione create_frame_mask che, dato un frame del filmato, dopo averlo convertito in HSV, restituisca una tupla (hsv, mask), dove hsv è il frame convertito in hsv e mask una maschera binaria che escluda:
#tutti i pixel con coordinata y maggiore o uguale a 340; tutti i pixel con saturazione minore di 20; tutti i pixel con luminosità minore di 40.
#Completare anche la funzione create_confidence_map che, a partire da un frame in formato HSV, dalla maschera di pixel da considerare e dall'istogramma di uno degli sprite, deve costruire una mappa di confidenza (ossia un'immagine dove il valore di ciascun pixel è tanto più alto quanto è più probabile che tale pixel appartenga allo sprite in base al suo colore). In particolare tale funzione deve:
#utilizzare cv.calcBackProject per ottenere la mappa di confidenza su tutti i pixel;
#azzerare tutti i pixel che sono zero nella maschera;
#restituire l'immagine risultante.
def create_frame_mask(frame):
    hsv = cv.cvtColor(sprite, cv.COLOR_BGR2HSV)
    mask = cv.inRange(hsv, (0,20,40), (179,155,255))
    mask[340:] = 0
    return hsv, mask
```

```

def create_confidence_map(hsv, mask, hist):
    conf = cv.calcBackProject([hsv], [0], hist, [0, 180], 1)
    conf[mask==0] = 0
    return conf

```

## Es. 4

```

#Completare la funzione find_object che, data una mappa di confidenza, cerca la posizione più probabile dell'oggetto corrispondente. La funzione deve restituire una tupla (x,y) con le coordinate del centro dell'oggetto, oppure None se non lo trova. Una possibile strategia (ovviamente non è l'unica possibile) consiste nella serie seguente di operazioni:
#Costruire un'immagine binaria considerando solo i pixel con un valore di confidenza maggiore di una certa soglia (ad esempio 15 può essere una buona scelta);
#Eliminare tutte le componenti connesse inferiori a una certa area (ad esempio 130 pixel, il valore ottimale può dipendere dalla soglia al punto precedente);
#Chiudere eventuali piccoli buchi o concavità in tutte le componenti connesse rimaste, utilizzando la morfologia matematica;
#Eseguire nuovamente l'etichettatura delle componenti connesse e restituire il baricentro della componente con l'area massima, o None se non ci sono componenti connesse.
se = cv.getStructuringElement(cv.MORPH_ELLIPSE, (7, 7))

def find_object(conf):
    _, mask = cv.threshold(conf, 15, 255, cv.THRESH_BINARY)
    n, cc, stats, _ = cv.connectedComponentsWithStats(mask)
    i_small = [i for i in range(n) if stats[i, cv.CC_STAT_AREA] < 130]
    mask[np.isin(cc, i_small)] = 0
    mask = cv.morphologyEx(mask, cv.MORPH_CLOSE, se)

    n, cc, stats, centroids = cv.connectedComponentsWithStats(mask)
    if n > 1:
        iMax = np.argmax(stats[1:, cv.CC_STAT_AREA]) + 1 #slicing per ottenere il vettore di tutte le aree a partire dalla componente connessa di indice 1
        x, y = centroids[iMax]
        return int(round(x)), int(round(y))
    else:
        return None

```

## Es. 5

```

#Completare la funzione update_pos che, data la mappa di confidenza del nuovo frame e l'eventuale posizione dell'oggetto nel frame precedente, ne determina la nuova posizione. La funzione deve restituire una tupla (x,y) con le coordinate del centro dell'oggetto, oppure None se non lo trova. Nel dettaglio, la funzione deve comportarsi come descritto nei punti seguenti.
#Se pos è None, significa che nel frame precedente l'oggetto non era stato individuato: utilizzare la funzione find_object precedentemente implementata per trovarne la posizione e restituirla.
#Altrimenti, eseguire l'algoritmo Mean-shift con i criteri di terminazione TERM_CRIT già definiti, utilizzando come finestra un quadrato di lato OBJ_SIZE centrato nel punto pos; data la finestra restituita dall'algoritmo Mean-shift, verificare che al suo interno ci siano almeno 20 pixel con confidenza superiore a 15: in tal caso restituire come posizione il centro della finestra stessa, altrimenti supporre che l'oggetto non sia presente e restituire None.
#Dopo aver completato e verificato il corretto funzionamento di update_pos, nella stessa cella, o in una cella successiva, creare una lista all_tracking che contenga, per ogni frame, una lista con la posizione (x,y) di ciascuno sprite nel frame. A tal fine inizializzare all_tracking come lista vuota e una seconda lista tracking con 5 None (le posizioni degli sprite sono inizialmente sconosciute);
#procedere iterativamente sui frame richiamando opportunamente le funzioni create_frame_mask, create_confidence_map e update_pos su ciascun elemento di tracking, in modo da ottenere le 5 posizioni che corrispondono al frame corrente. Appendere di volta in volta il contenuto di tracking alla lista all_tracking in modo da ottenere, alla fine del ciclo, il risultato desiderato.

TERM_CRIT = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )

def update_pos(conf, pos):
    if pos is None:
        pos = find_object(conf)
    else:
        _, (x,y,w,h) = cv.meanShift(conf, (pos[0]-OBJ_SIZE//2, pos[1]-OBJ_SIZE//2, OBJ_SIZE; OBJ_SIZE), TERM_CRIT)
        if np.count_nonzero(conf[y:y+h, x:x+h] > 15) < 20:
            pos = None
        else:
            pos = x+w//2, y+h//2
    return pos

tracking = [None] / len(sprites)

```

```

all_tracking = []
for frame in frames:
    tracking = [update_pos(create_confidence_map("create_frame_mask(frame), h), p) for h, p in zip(hists, tracking)]
    all_tracking.append(tracking)

```

## Es. 6

```

#A partire dalla lista all_tracking, eseguire le operazioni seguenti:
#Costruire una lista contenente le stesse informazioni di all_tracking ma in cui tutti gli elementi pari a None sono sostituiti dalla tupla (np.nan, np.nan).
#Trasformare la lista create al passo precedente in un array NumPy mat, di tipo np.float32, con 3 dimensioni: la prima corrisponde al numero dei frame, la seconda al numero degli sprite, la terza è 2 (le coordinate x e y).
#Create un nuovo array NumPy dist contenente, per ogni frame eccetto il primo e per ogni sprite, la distanza euclidea rispetto alla posizione nel frame precedente. Tale array dovrà quindi avere due dimensioni: la prima pari al numero di frame meno uno e la seconda pari al numero di sprite. Se uno o entrambi i punti sono (np.nan, np.nan), il risultato del calcolo della distanza euclidea dovrà essere n p.nan (si noti che questo non richiede accorgimenti particolari, grazie al funzionamento della costante np.nan, in quanto qualsiasi operazione che coinvolge un operando np.nan restituisce np.nan).
#Modificare l'array dist sostituendo tutti i valori np.nan con zero. Suggerimento: un modo semplice consiste nell'utilizzare la funzione np.nan_to_num.
#A partire dall'array dist, stampare a video, per ciascuno sprite, il totale dei pixel percorsi nel video.
#1
tmp = [[(np.Nan, np.nan) if p is None else p for p in t] for t in all_tracking]
#2
mat = np.array(tmp, np.float32)# np.array data una lista di liste crea una matrice e posso specificare il tipo
#3
dist = np.sqrt(((mat[1:] - mat[:-1])**2).sum(2)) # facendo la differenza sto facendo tutte le differenze di tutte le distanze eucleede
#4
np.nan_to_num(dist, False)
#5
sum_dist = dist.sum(0).round().astype(int)
for i, n in enumerate(sprite_names):
    print(n, '->', sum_dist[i], 'pixel')

```

## 7. Template matching

### Es. 1

```

#Utilizzando il template matching, cercare all'interno dell'immagine img1, creata dalla cella precedente, il template contenuto nell'immagine "tm/t1.png".
#La misura del grado di somiglianza può essere effettuata con il metodo che si preferisce, purché sia efficace. Eventualmente eseguire dei test con più metodi. Completare la funzione FindWaldo1 che deve restituire una tupla (x, y, w, h), in cui x, y sono le coordinate del vertice superiore sinistro della posizione dell'oggetto dell'immagine e w, h sono le sue dimensioni (pari a quelle del template)
def FindWaldo1():
    template = cv.imread('tm/t1.png')# Caricare l'immagine del template
    res = cv.matchTemplate(img1, template, cv.TM_CCOEFF_NORMED)
    _, _, (x,y) = cv.minMaxLoc(res) #metto la pos del massimo in x,y

    h,w = template.shape[:2]
    return x, y, w, h

```

### Es. 2

```

#Creare una lista di immagini templates a partire da template2, utilizzando i seguenti fattori di scala: 100%, 120%, 140% e 160%.
#Prestare particolare attenzione alla dimensione delle immagini: larghezza e altezza dovranno essere arrotondate all'intero più vicino in base al corrispondente fattore di scala.
#Questo può essere ottenuto, ad esempio, sfruttando con gli appropriati parametri una certa funzione OpenCV. Eseguire poi la cella successiva per controllare che le dimensioni delle immagini nella lista siano quelle attese
templates = [cv.resize(template2, None, None, s, s) for s in (1.0, 1.2, 1.4, 1.6)]

```

### Es. 3

```
#Utilizzando il template matching, cercare, all'interno dell'immagine img2, ciascun template nella lista templates e restituire la posizione di quello che ha la massima somiglianza.  
#Si suggerisce di utilizzare un metodo normalizzato per misurare il grado di somiglianza, in modo da essere indipendenti dalle dimensioni del template.  
# Completare la funzione FindWaldo2, che deve restituire una tupla (x, y, w, h), in cui x, y sono le coordinate del vertice superiore sinistro della posizione dell'oggetto dell'immagine e w, h sono le sue dimensioni (pari a quelle del template, fra tutti quelli nella lista, che è risultato più simile)  
def FindWaldo2():  
  
    risultati = [cv.matchTemplate(img2, t, cv.TM_CCOEFF_NORMED), t.shape[:2]] for t in templates] #per ogni template vado a calcolare l'immagine con tutti i valori di somiglianza  
    mm = [(cv.minMaxLoc(r),(h, w)) for r,(h, w) in risultati] # mm contiene minMaxLoc e altezza e larghezza del template  
  
    (*_, (x,y)), (h, w) = max(mm, key = lambda x: x[0][1])#lista finale (quello con valore massimo)  
  
    return x, y, w, h
```

### Es. 4

```
#Utilizzando il template matching, cercare, all'interno dell'immagine img3, il template template3.  
#In primo luogo verificare che un semplice procedimento come quello del primo esercizio in questo caso non può portare al risultato desiderato; cercare quindi un modo di trasformare il template o l'immagine per poter applicare con successo il template matching.  
#Completare la funzione FindWaldo3, che deve restituire una tupla (x, y, w, h), in cui x, y sono le coordinate del vertice superiore sinistro della posizione dell'oggetto dell'immagine e w, h sono le sue dimensioni (pari a quelle del template)  
  
def FindWaldo3():  
  
    res = cv.matchTemplate(cv.Canny(img3, 200, 50), template3, cv.TM_CCOEFF_NORMED)  
    (*_, (x,y)) = cv.minMaxLoc(res) #con *_ scarto tutto tranne (x,y)  
    h, w = template3.shape[:2]  
  
    return x, y, w, h
```

### Es. 5

```
#Utilizzando il template matching, cercare, all'interno dell'immagine img, tutte le posizioni in cui compare il template template.  
#Completare il codice nella cella seguente che deve creare una lista di tuple (x, y, w, h) (rettangoli), in cui, per ogni oggetto individuato, x, y sono le coordinate del suo vertice superiore sinistro e w, h sono le sue dimensioni (pari a quelle del template).  
#In questo esercizio non è richiesto di trovare un solo rettangolo per ogni oggetto: è possibile che lo stesso oggetto sia trovato più volte, in posizioni molto vicine fra loro. Questo problema sarà poi affrontato nell'ultimo esercizio.  
res = cv.matchTemplate(img, template, cv.TM_CCOEFF_NORMED)  
h, w = template.shape[:2]  
thr = res.max() * 0.7 #70% del valore  
rettangoli = [(int(x), (int(y), h, w) for (y,x) in list(zip(*np.nonzero(res >= thr))) #indici degli elementi non zero, con zip ottengo una lista di coppia (y,x)
```

### Es. 6

```
#A partire dalla lista rettangoli, ottenere una nuova lista rettangoli_ok che contenga un solo rettangolo per ogni oggetto da individuare.  
#In altre parole, in caso di più rettangoli sovrapposti, è richiesto di considerarne solo uno: quello che corrisponde al valore di somiglianza più alto con il template  
scores = [... for x, y, _, _ in rettangoli]  
indices = cv.dnn.NMSBoxes(rettangoli, scores, 0, 0).squeeze()  
rettangoli_ok = [rettangoli[i] for i in indices]
```

## 8. Deep Learning

### Es. 1

```
#Utilizzando la funzione OpenCV cv.dnn.readNet(), caricare da file la rete neurale, passando il path "dnn/yolov3.weights" come model e  
il path "dnn/yolov3.cfg" come config: memorizzare il valore di ritorno (il riferimento alla rete in memoria) in una variabile net.  
#Leggere quindi i nomi delle 80 classi dal file di testo "dnn/yolov3.txt", che contiene un nome per ogni riga: memorizzare i nomi in un  
a lista Python di stringhe class_names: i nomi delle classi non devono contenere il carattere new line ('\n').  
  
path = "dnn/yolov3."  
net = cv.dnn.readNet(path + "weights", path + "cfg")  
class_names = open(path + "txt").read().splitlines()
```

### Es. 2

```
#In previsione di dover disegnare con colori diversi le posizioni di oggetti di classi diverse in un'immagine, creare una lista class_c  
olors di colori, dove il colore  $i$ - esimo è una tupla  $(Bi, Gi, Ri)$  corrispondente ai seguenti valori HSV:  $Hi=(i \cdot 29) \bmod 180, Si=200, Vi=25$   
5 .  
#I valori sono espressi negli stessi range utilizzati in OpenCV:  $H \in [0, 180], S \in [0, 255], V \in [0, 255]$  . In sostanza l'obiettivo dell'esercizio  
è ottenere una lista di colori in formato BGR, con saturazione e luminosità costanti e con lo hue di ogni colore ruotato di 58° rispetto a quello del precedente.  
hsv_images = [np.array(((i*29) % 100, 200, 255), np.uint8).reshape(1,1,3) for i in range(len(class_names))]  
class_colors = [tuple(cv.cvtColor(x, cv.COLOR_HSV2BGR).squeeze().tolist()) for x in hsv_images] #conversione in BGR, tolto le dimensioni  
inutili, trasformato in lista python e infine ottenuto la lista risultante
```

### Es. 3

```
#Come mostrato nelle dispense, per fornire un'immagine in ingresso alla rete, questa deve essere trasformata in un 'blob' mediante la funzione cv.dnn.blobFromImage(), che nel caso di Yolo dovrà essere chiamata con i parametri specificati nella cella seguente:  
#fattore di scala pari a 1.0/255, dimensioni 320x320 e scambio dei canali R e B (la nostra immagine è in formato BGR, mentre la rete si aspetta RGB). Nel caso l'immagine non abbia le dimensioni 320x320, cv.dnn.blobFromImage() provvede a ridimensionarla, ma senza preservare l'aspect ratio.  
#Al fine di evitare deformazioni che potrebbero penalizzare l'accuratezza del riconoscimento, completare la funzione nella cella seguente, calcolando l'eventuale bordo da aggiungere orizzontalmente (border_w) o verticalmente (border_h), in modo da rendere l'immagine img quadrata.  
#Si noti che uno dei due valori border_w o border_h dovrà sempre essere zero: saranno entrambi zero solo se l'immagine è già quadrata.  
#Creare quindi una nuova immagine padded_img, aggiungendo border_w pixel sia a sinistra che a destra e border_h pixel sia in alto che in basso: i pixel aggiunti dovranno essere neri.  
  
def prepare_input_blob(img):  
  
    height, width = img.shape[:2]  
    size = max(height, width)  
    border_w = (size - width) // 2  
    border_h = (size - height) // 2  
    padded_img = cv.copyMakeBorder(img, border_h, size - height - border_h, border_w, size - width - border_w, cv.BORDER_CONSTANT) #quando bordo aggiungere all'immagine  
  
    return border_w, border_h, padded_img.shape[0], cv.dnn.blobFromImage(padded_img, 1.0 / 255, (320, 320), swapRB = True)
```

### Es. 4

```
#Partendo dai risultati dell'esecuzione delle ultime celle, in particolare dal contenuto delle variabili border_w, border_h, size, output_values, si ottengano le seguenti informazioni dalla riga 231 del primo dei tre livelli di output della rete i cui valori sono già in output_values:  
  
#(x, y, w, h), coordinate in pixel del rettangolo relativamente alle dimensioni dell'immagine originale img;  
#class_index, indice della classe che ha ottenuto lo score più elevato (dovrà risultare 22);
```

```

#confidence, lo score della classe.
#Il procedimento è analogo a quanto mostrato nella figura precedente, con alcune differenze:

#la dimensione dell'immagine non è 320, ma è quanto contenuto nella variabile size;
#le coordinate x, y in questo caso non sono quelle del centro del rettangolo, ma del vertice in alto a sinistra;
#nell'ottenere il valore finale di x, y, si deve anche tenere conto che, per rendere l'immagine quadrata, era stato aggiunto un bordo
#(border_w, border_h) e le coordinate, per essere applicabili all'immagine originale, dovranno essere corrette di conseguenza.
#Una volta ottenuti i valori richiesti, copiare img in una nuova immagine res e disegnare le informazioni sull'oggetto chiamando la funzione draw_object() definita in precedenza.

box = out_values[0][231]
class_index = np.argmax(box[5:])#indice del massimo saltando i primi 5

x, y, w, h = (box[:4] * size).round().astype(int)
x -= (w // 2 + border_w)
y -= (h // 2 + border_h)
confidence = float(box[5 + class_index])
res = img.copy()
draw_object(res, class_index, confidence, (x, y, w, h), True)

```

## Es. 5

```

#Grazie all'esperienza maturata negli esercizi precedenti, completare la funzione yolo_detection(image, conf_threshold) che, ricevuti i
#n input un'immagine e una soglia fra 0 e 1:

#prepara l'input per la rete utilizzando prepare_input_blob();
#esegue la rete neurale su tale input;
#analizza quanto ottenuto dai 3 livelli di output e, restituisce tre liste class_indices, confidences, detected_objects, contenenti, rispettivamente,
#gli indici delle classi, gli score e i rettangoli (tuple x, y, w, h) di tutti i box che hanno uno score maggiore o uguale a conf_threshold. Per ogni box considerare solo lo score con valore massimo.

def yolo_detection(image, conf_threshold):
    border_w, border_h, size, blob = prepare_input_blob(img)
    border = np.array((border_w, border_h))
    net.setInput(blob)
    out_values = net.forward(out_names)

    class_indices, confidences, detected_objects = [], [], []
    for out in out_values:
        i_max = np.argmax(out[:,5:], axis=1) #ottengo un array contenente per ogni riga l'indice della classe
        row_indices = np.nonzero(out[np.arange(i_max.shape[0]), i_max + 5] >= conf_threshold)
        rects = out[row_indices, :4] * size.round().astype(int)
        rects[:,2:] -= (rects[:,2:]/2 + border)
        indices = i_max[row_indices]
        scores = out[row_indices, indices + 5]

        class_indices += indices.tolist()
        confidences += scores.tolist()
        detected_objects += rects.tolist()

    return class_indices, confidences, detected_objects

```

## Es. 6

```

#Implementare una nuova funzione yolo_detection_nms(image, conf_threshold, nms_threshold) che si comporti in modo analogo a yolo_detect
#ion(), ma applichi anche la funzione OpenCV cv.dnn.NMSBoxes() per filtrare i risultati di yolo_detection(), eliminando oggetti che si sovrappongono troppo (rispetto alla soglia nms_threshold) ad altri oggetti con valore di confidenza maggiore.
#Prestare attenzione al valore di ritorno di cv.dnn.NMSBoxes(): l'array di indici che restituisce ha una dimensione a 1 "di troppo". Inoltre è necessario gestire anche il caso in cui nessun oggetto soddisfi i criteri: in tal caso cv.dnn.NMSBoxes() restituisce una tupla vuota invece di un array NumPy: verificare che questo non crei problemi alla implementazione proposta.

def yolo_detection_nms(image, conf_threshold, nms_threshold):

    class_indices, confidences, detected_objects = yolo_detection(image, conf_threshold)
    indices = np.ravel(cv.dnn.NMSBoxes(detected_objects, confidences, conf_threshold, nms_threshold))

    class_indices = [class_indices[i] for i in indices]
    confidences = [confidences[i] for i in indices]
    detected_objects = [detected_objects[i] for i in indices]

```

```
return class_indices, confidences, detected_objects
```