Codice Visione Artificiale

**Riempie array con val** np.full\_like(array,val)

**Carica immagini bianco e nero** cv.imread(‘img.png',cv.IMREAD\_GRAYSCALE)

**Conversione in HSV** cv.cvtColor(img, cv.COLOR\_BGR2HSV)

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

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

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

**Funzione Lambda**

f = lambda x, y: x \*\* y print(f(2,3)) #Semplici funzioni anonime

a = **np.empty**((2,7), np.int16) # 2 righe, 7 colonne vuote

a = **np.arange**(100, 110, 2) # [100 102 104 106 108]

a = **np.identity**(3) #Matrice identità

**Scambiare le dimensioni** d = a.T o e = a.transpose()

**Elimina eventuali dimensioni a uno** f = c.squeeze()

**Modificare solo gli elementi che soddisfano un certo criterio** criterio = a%3 == 0

**Tutti i pixel a 0** img1 = np.zeros((15, 16), dtype=np.uint8)

**Tutti i pixel a 255** img2 = np.full((15, 16), 255, dtype=np.uint8)

**Valori dei pixel casuali** img3 = np.random.randint(0, 256, (15, 16), dtype=np.uint8)

**Calcolo istogramma**

def calc\_hist\_cv(img):

return cv.calcHist([img], [0], None, [256], [0, 256]).squeeze()

**Soglia globale (128)**

\_, res = cv.threshold(img, 128, 255, cv.THRESH\_BINARY)

**Soglia globale determinata dall'algoritmo di Otsu**

t, res = cv.threshold(img, -1, 255, cv.THRESH\_OTSU)

**Soglia locale (media su intorno 11x11 meno il valore 10**

res = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE\_THRESH\_MEAN\_C, cv.THRESH\_BINARY, 11, 10)

**Contrast stretching** res = contrast\_stretching(img, img.min(), img.max())

**Equalizzazione di un'immagine grayscale** res = cv.equalizeHist(img)

**Ingrandimento immagine**

img\_nn = cv.resize(img, size, interpolation=cv.INTER\_NEAREST)

img\_bl = cv.resize(img, size, interpolation=cv.INTER\_LINEAR)

img\_bc = cv.resize(img, size, interpolation=cv.INTER\_CUBIC)

**Box filter con normalizzazione** smoothed = cv.boxFilter(img, -1, (7,7), normalize = True)

**Filtro gaussiano** img\_blurred2 = cv.GaussianBlur(img, (5,5), 1) # m=5, σ=1

**Estrazione Bordi**

blurred = cv.GaussianBlur(cv.cvtColor(img, cv.COLOR\_BGR2GRAY), (3, 3), 0)

edges = cv.Canny(blurred, 200, 100)

img [edges != 0 ] = (0,255,255)

**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)

**Trasformata distanza**

dt4 = cv.distanceTransform(img, cv.DIST\_L1, 3)

dt8 = cv.distanceTransform(img, cv.DIST\_C, 3)

**Estrazione contorno**

c, \_ = cv.findContours(img, cv.RETR\_EXTERNAL, cv.CHAIN\_APPROX\_NONE) # RETR\_EXTERNAL = sono contorni esterni (no buchi interni)

res = cv.drawContours(cv.cvtColor(img, cv.COLOR\_GRAY2BGR), c, -1, (255,0,0), 2) #disegna contorno (blu)

**Etichettatura delle componenti connesse** n, cc = cv.connectedComponents(bw)

**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)

**Elemento strutturante: quadrato 15x15**

se = cv.getStructuringElement(cv.MORPH\_RECT, (15,15))

**Morfologia matematica**

d = cv.morphologyEx(img, cv.MORPH\_DILATE, se) #dilatazione

e = cv.morphologyEx(img, cv.MORPH\_ERODE, se) #erosione

**Colora i pixel aggiunti dalla dilatazione**

d1 = cv.cvtColor(d, cv.COLOR\_GRAY2BGR)

d1[d-img!=0]=(0,255,0)

**Colora i pixel rimossi dall'erosione**

e1 = cv.cvtColor(img, cv.COLOR\_GRAY2BGR)

e1[e-img!=0]=(0,128,255)

**L'apertura permette di risolvere alcune piccole componenti connesse dovute al rumore**

res1 = cv.morphologyEx(bw, cv.MORPH\_OPEN, s)

**La chiusura permette di risolvere alcuni "buchi" all'interno degli oggetti**

res2 = cv.morphologyEx(res1, cv.MORPH\_CLOSE, s)

res = img.copy()

res[res2==0] = 0