Roteiro de Processamento de Imagens

Este roteiro trata dos principais comandos relacionados ao Processamento de Imagens. A etapa de processamento contempla as etapas anteriores às etapas de reconhecimento, classificação, etc. O processo de aquisição da imagem nem sempre é controlado e, por muitas vezes, processar as imagens é fundamental para o bom funcionamento dos modelos com imagem.

**Pacotes Python**

Um pacote simples e muito utilizado é o Scikit-Image.

**Carregar imagem**

import cv2

image = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)

**Visualizar imagem**

import matplotlib.pyplot as plt

plt.imshow(image, cmap='gray')

plt.show()

fig, axes = plt.subplots(linhas, colunas, figsize=(X,Y)) #para plotar mais de uma imagem

axes[0].imshow...

axes[1].imshow...

**Transformações**

- Negativo da imagem

USAR a imagem: person.jpg

from skimage.util import invert

image\_negative = invert(image\_original)

- Transformação Logaritmica

USAR a imagem: person.jpg

from skimage.exposure import adjust\_log

image\_log = adjust\_log(image\_original, c) #default valor para c=1

- Transformação de Potência (gamma)

USAR a imagem: Forest.jpg

from skimage.exposure import adjust\_gamma

image\_gamma = adjust\_gamma(image\_original, gamma) #ajustar o melhor valor

- Transformação Linear por partes – Alargamento de Contraste

USAR a imagem: polen.jpg

def pixelVal(img, r1, s1, r2, s2):

if (0 <= img and img <= r1):

return (s1 / r1) \* img

elif (r1 < img and img <= r2):

return ((s2 - s1) / (r2 - r1)) \* (img - r1) + s1

else:

return ((255 - s2) / (255 - r2)) \* (img - r2) + s2

r1 = minimo

s1 = 0

r2 = maximo

s2 = 255

pixelVal\_vec = np.vectorize(pixelVal)

image\_contr = pixelVal\_vec(image\_original, r1, s1, r2, s2)

image\_contr = np.array(image\_contr, dtype=np.uint8) #mostrar o min e max após!

- Equalização do Histograma

image1 = cv2.imread("images\\polenEscuro.jpg", cv2.IMREAD\_GRAYSCALE)

image2 = cv2.imread("images\\polenClaro.jpg", cv2.IMREAD\_GRAYSCALE)

image3 = cv2.imread("images\\polenBaixoContraste.jpg", cv2.IMREAD\_GRAYSCALE)

image4 = cv2.imread("images\\polenAltoContraste.jpg", cv2.IMREAD\_GRAYSCALE)

bins = 100

hist1= **cv2.calcHist**([image1],[0],None,[bins],[0,255])

hist2= **cv2.calcHist**([image2],[0],None,[bins],[0,255])

hist3= **cv2.calcHist**([image3],[0],None,[bins],[0,255])

hist4= **cv2.calcHist**([image4],[0],None,[bins],[0,255])

fig, axes = plt.subplots(2, 4, figsize=(20,10))

axes[0,0].imshow(image1, cmap='gray', vmin=0, vmax=255)

axes[0,1].imshow(image2, cmap='gray', vmin=0, vmax=255)

axes[0,2].imshow(image3, cmap='gray', vmin=0, vmax=255)

axes[0,3].imshow(image4, cmap='gray', vmin=0, vmax=255)

axes[1,0].plot(hist1)

axes[1,1].plot(hist2)

axes[1,2].plot(hist3)

axes[1,3].plot(hist4)

plt.tight\_layout()

plt.show()

image\_equalized = **cv2.equalizeHist**(image\_original) #Altere o nome da imagem

hist\_equalized = cv2.calcHist([image\_equalized],[0],None,[bins],[0,255])

fig, axes = plt.subplots(1, 2, figsize=(10,5))

axes[0].imshow(image\_equalized, cmap='gray', vmin=0, vmax=255)

axes[1].plot(hist\_equalized)

plt.tight\_layout

plt.show()

- Filtros Espaciais de Suavização – Filtro da Média (Blur)

image\_blur = cv2.blur(image\_original, ksize=(5,5), borderType=cv2.BORDER\_REPLICATE)

- Filtros Espaciais de Suavização – Filtro da Mediana (medianBlur)

image\_medianBlur = cv2.medianBlur(image\_original, ksize=5)

- Filtros Espaciais de Suavização – Filtra Gaussiano

from skimage.filters import gaussian

image\_medianBlur = cv2.GaussianBlur(image\_original, ksize=(0,0), sigmaX=X)

image\_medianBlur = guassian(image\_original, sigma=X, preserve\_range=True)

*#Importante: ksize é o tamanho do filtro. Se 0, os dois usam o valor do sigma!*

- Filtros Espaciais de Aguçamento – Gradiente – Máscaras de Sobel

USAR a imagem: pcb.jpeg

from skimage.filters import gaussian

image\_sobel\_x = cv2.Sobel(image\_original, ddepth=-1, dx=1, dy=0, ksize=X)

image\_sobel\_y = cv2.Sobel(image\_original, ddepth=-1, dx=0, dy=1, ksize=X)

*#Importante: ddepth=-1 mantém a profundidade da imagem original*

from skimage.filters import sobel, sobel\_h, sobel\_v

image\_sobel = sobel(image\_original)

image\_sobel\_h = sobel\_h(image\_original)

image\_sobel\_v = sobel\_v(image\_original)

- Filtros Espaciais de Aguçamento – Gradiente – Máscaras de Prewitt

USAR a imagem: pcb.jpeg

from skimage.filters import prewitt, prewitt\_h, prewitt\_v

image\_prewitt = prewitt(image\_original)

image\_prewitt\_h = prewitt\_h(image\_original)

image\_prewitt\_v = prewitt\_v(image\_original)

- Filtros Espaciais de Aguçamento – Laplaciano

USAR a imagem: pcb2.tif, pcb.jpeg, pcb1.png

image\_lap = cv2.Laplacian(image\_original, ddepth=cv2.CV\_64F, ksize=1)

image\_lapl\_a = np.uint8(np.absolute(image\_lap)) #ou

image\_lapl\_a = cv2.Laplacian(image\_original, ddepth=-1, ksize=1)

- Filtros Espaciais de Aguçamento – Detector de Bordas de Canny

image\_canny = cv2.Canny(image\_original, X/2, X, apertureSize=”3 a 7”)

*#importante: os valores dos thresholds determinam o que “fica” e o que “sai” da imagem*

- Filtragem – Definindo as máscaras (Kernel)

kernel = (1/9)\*np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1],]) # FiltroPassaAlta

kernel = (1/9)\*np.array([[1, 1, 1], [1, 1, 1], [1, 1, 1],]) # FiltroPassaBaixa

image\_filtrada = cv2.filter2D(image\_original, ddepth=-1, kernel)

*#Importante: essa é a forma de definir TODAS as filtragens anteriores, criando o seu kernel*

**Transformações**

- Segmentação por limiarização/binarização – Inspeção de Histograma

hist\_image = cv2.calcHist([image\_original], [0], None, [256], [0, 255])

(T, image\_binario) = cv2.threshold(image\_original, limiar, 255, cv2.THRESH\_BINARY)

*#Importante: variar os valores de limiar, baseado no histograma*

- Segmentação por limiarização/binarização – Método de Otsu

(T, image\_binario) =

cv2.threshold(image\_original, limiar, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

*#Importante: definir limiar=0. T é a saída*

- Segmentação por limiarização/binarização – Método de multi-Otsu

from skimage.filters import threshold\_multiotsu

limiares = threshold\_multiotsu(image\_original)

regioes = np.digitalize(image\_original, bins=limiares)

# um plot para imagem, um para histograma e um para binarizado

# o histograma, usar

# for thres in limiares: axes[1].acvline(thres, color='r')

**Atividade**

- Com a imagem *noise.tif*, faça o seguinte:

1. Abra a imagem
2. Calcule e apresente o histograma
3. Determine o limiar de binarização (qualquer método que desejar)
4. Faça uma suavização utilizando filtro Gaussiano
5. Calcule e apresente o histograma
6. Determine o limiar de binarização (qualquer método que desejar)
7. Apresente e compare as imagens

- Segmentação por limiarização/binarização – Utilizando Informação de Borda

Avalie o código e os resultados. Para esta atividade, utilize a imagem *levedura.tif*

image\_histo = cv2.calcHist([image], [0], None, [256], [0,255])

(T, image\_binario\_seg) =

cv2.threshold(image, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

print(T)

fig, axes = plt.subplots(1, 3, figsize=(20,5))

axes[0].imshow(image, cmap='gray')

axes[1].imshow(image\_binario\_seg, cmap='gray')

axes[2].plot(image\_histo)

plt.tight\_layout

plt.show()

image\_lap\_seg = cv2.Laplacian(image, ddepth=-1, ksize=3)

(T, image\_binario\_seg) =

cv2.threshold(image\_lap\_seg, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

print(T)

image\_binario\_seg = np.uint8(np.absolute(image\_binario\_seg))

imm = image\_lap\_seg \* 2 \* image

image\_histo\_lap = cv2.calcHist([imm], [0], None, [256], [1,255])

(T, image\_binario\_seg\_1) =

cv2.threshold(imm, 0, 255, cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

(T, image\_binario\_seg\_1) =

cv2.threshold(image, T, 255, cv2.THRESH\_BINARY)

print(T)

fig, axes = plt.subplots(1, 4, figsize=(20,5))

axes[0].plot(image\_histo\_lap)

axes[1].imshow(image\_lap\_seg, cmap='gray')

axes[2].imshow(imm, cmap='gray')

axes[3].imshow(image\_binario\_seg\_1, cmap='gray')

plt.tight\_layout

plt.show()

**Representação**

- Representação de Fronteiras

Usar a imagem *contorno.tif*

***Obs:*** se for usar imagem colorida, adicionar cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

resulting\_image = cv2.GaussianBlur(image\_original, ksize=(9, 9), sigmaX=9, sigmaY=9)

(T, imageBin) = cv2.threshold(  
 resulting\_image,   
 0,   
 255,  
 cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

contours, hierarchy = cv2.findContours(  
 imageBin,   
 mode=cv2.RETR\_EXTERNAL,   
 method=cv2.CHAIN\_APPROX\_NONE)

image\_copy = cv2.cvtColor(image.copy(), cv2.COLOR\_GRAY2BGR)

cv2.drawContours(image=image\_copy,   
 contours=contours,   
 contourIdx=0,   
 color=(0, 0, 255),   
 thickness=2,   
 lineType=cv2.LINE\_AA)

contours1, hierarchy1 = cv2.findContours(  
 imageBin,   
 mode=cv2.RETR\_EXTERNAL,  
 method=cv2.CHAIN\_APPROX\_SIMPLE)

image\_copy1 = cv2.cvtColor(image.copy(), cv2.COLOR\_GRAY2BGR)

cv2.drawContours(image=image\_copy1,   
 contours=contours1,   
 contourIdx=-1,  
 color=(0, 0, 255),   
 thickness=2,   
 lineType=cv2.LINE\_AA)

- Representação de Fronteiras – Comprimento e diâmetro

Usar a imagem *bone.tif*

contours, hierarchy = cv2.findContours(  
 image,   
 mode=cv2.RETR\_TREE,   
 method=cv2.CHAIN\_APPROX\_SIMPLE)

i = 0

cnt = contours[i]

leftmost = tuple(cnt[cnt[:, :, 0].argmin()][0])

rightmost = tuple(cnt[cnt[:, :, 0].argmax()][0])

topmost = tuple(cnt[cnt[:, :, 1].argmin()][0])

bottommost = tuple(cnt[cnt[:, :, 1].argmax()][0])

print("Coordenada mais a esquerda: ", leftmost[0], 'e', leftmost[1])

print("Coordenada mais a direita: ", rightmost[0], 'e', rightmost[1])

print("Coordenada mais acima: ", topmost[0], 'e', topmost[1])

print("Coordenada mais abaixo: ", bottommost[0], 'e', bottommost[1])

diametro = math.sqrt(  
 ((bottommost[0] - topmost[0]) \*\* 2) +   
 ((bottommost[1] - topmost[1]) \*\* 2))

perimetro = cv2.arcLength(cnt, True)

print("Perimetro da Fronteira:", perimetro)

print("Diametro - ponto mais alto - ponto mais baixo:", diametro)

x, y, w, h = cv2.boundingRect(cnt)

print("Ponto mais a esquerda, em x:", x)

print("Ponto mais alto, em y:", y)

print("Largura:", w)

print("Altura:", h)

cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)

rect = cv2.minAreaRect(cnt)

box = np.int64(cv2.boxPoints(rect))

image\_copy = cv2.cvtColor(image, cv2.COLOR\_GRAY2BGR)

cv2.drawContours(image\_copy, [box], 0, (0, 0, 255), 2)

(x1, y1), (MA, ma), angle = cv2.fitEllipse(cnt)

print("x do centro:", x1, "e y do centro:", y1)

print("Largura ou comprimento do eixo menor:", MA)

print("Altura ou comprimento do eixo maior:", ma)

print("Angulo de rotacao (clockwise):", angle)

excentricidade = ma / MA

print("Excentididadae:", excentricidade)

- Representação por Região

Usar a imagem *leaf.tif*

skel = cv2.ximgproc.thinning(image,thinningType=cv2.ximgproc.THINNING\_ZHANGSUEN)

**Descritores de Textura**

- Momentos HU

Usar as imagens *HU1.tif* até *HU6.tif*

from skimage.measure import moments\_hu, moments\_normalized, moments\_central

mu = moments\_central(image)

nu = moments\_normalized(mu)

momentos = moments\_hu(nu)

print("Momentos:", momentos))

**Atividade**

Para cada imagem, de HU1 a HU6, repita os comandos acima e guardem os valores de momento. Comparem os resultados. Como esses valores podem ser utilizados para descrever uma imagem?