

# Introdução a Visão Computacional com Python e OpenCV

Adaptado de Ricardo Antonello

# Requisitos:

Softwares Necessários:

Python 3.9.1 – disponível em <https://www.python.org/downloads/>

Tesseract-OCR para Windows (verificar se é 32 ou 64 bit's) – disponível em <https://github.com/UB-Mannheim/tesseract/wiki>

Na pasta aonde foi instalada a IDLE do Python

Pelo PROMPT DE COMANDO do Windows, digitar :

cd C:\Users\celso\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.9 (depende do seu computador)

```
> py -m pip install pillow
```

```
> py -m pip install matplotlib
```

```
> py -m pip install opencv-python
```

```
> py -m pip install pytesseract
```

# Sistema de coordenadas e manipulação de pixels

## Exemplo

```
# Importação das bibliotecas
import cv2

# Leitura da imagem com a função imread()
imagem = cv2.imread('entrada.jpg')
print('Largura em pixels: ', imagem.shape[1]) #largura da imagem
print('Altura em pixels: ', imagem.shape[0]) #altura da imagem
print('Qtde de canais: ', imagem.shape[2])

# Mostra a imagem com a função imshow
cv2.imshow("Nome da janela", imagem)
cv2.waitKey(0) #espera pressionar qualquer tecla
# Salvar a imagem no disco com função imwrite()
cv2.imwrite("saida.jpg", imagem)
```

# Conceitos

A importação da biblioteca padrão da OpenCV é obrigatória para utilizar suas funções. A primeira função usada é para abrir a imagem através de `cv2.imread()` que leva como argumento o nome do arquivo em disco.

A imagem é lida e armazenada em „imagem“ que é uma variável que dará acesso ao objeto da imagem que nada mais é que uma matriz de 3 dimensões (3 canais) contendo em cada dimensão uma das 3 cores do padrão RGB (red=vermelho, green=verde, blue=azul). No caso de uma imagem preto e branca temos apenas um canal, ou seja, apenas uma matriz de 2 dimensões.

Para facilitar o entendimento podemos pensar em uma planilha eletrônica, com linhas e colunas, portanto, uma matriz de 2 dimensões. Cada célula dessa matriz é um pixel, que no caso de imagens preto e brancas possuem um valor de 0 a 255, sendo 0 para preto e 255 para branco. Portanto, cada célula contém um inteiro de 8 bits (sem sinal) que em Python é definido por „uint8“ que é um unsigned integer de 8 bits.

# Resultados

```
===== RESTART: C:/Users/Celso/Desktop/2020/UNISAL 2020-1/PDI/Aula1.py =====  
Largura em pixels: 810  
Altura em pixels: 620  
Qtde de canais: 3  
|
```



# Exemplo

```
import cv2
```

```
imagem = cv2.imread('entrada.jpg')
```

```
(b, g, r) = imagem[0, 0] #veja que a ordem BGR e  
não RGB
```

```
print('O pixel (0, 0) tem as seguintes cores:')
```

```
print('Vermelho:', r, 'Verde:', g, 'Azul:', b)
```

`imagem[y, x]`

# Conceitos

Imagens são matrizes Numpy neste caso retornadas pelo método “imread” e armazenada em memória através da variável “imagem” conforme acima.

Lembre-se que o pixel superior mais a esquerda é o (0,0). No código é retornado na tupla (b, g, r) os respectivos valores das cores do píxel superior mais a esquerda.

Veja que o método retorna a sequência BGR e não RGB como poderíamos esperar

# Resultados

```
===== RESTART: C:/Users/Celso/Desktop/2020/UNISAL 2020-1/PDI/Aula2.py =  
O pixel (0, 0) tem as seguintes cores:  
Vermelho: 255 Verde: 255 Azul: 255
```



# Exercício 1:

- Crie a seguinte imagem (pode usar o paint)
- Leia um pixel vermelho, um verde, um azul, um amarelo e um preto. E informe seus valores em RGB

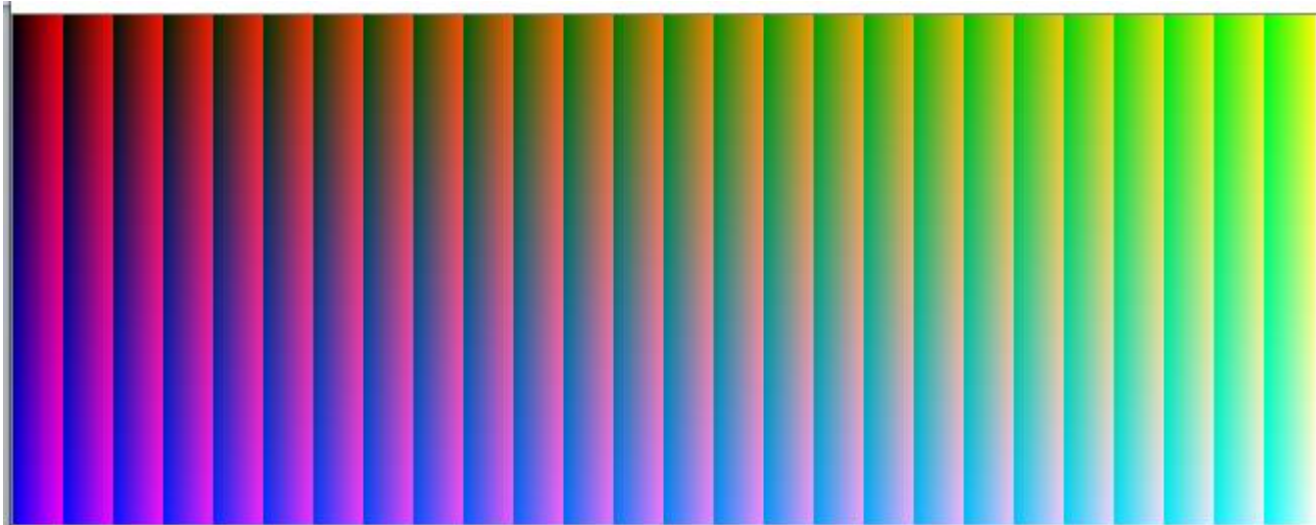


# Exercício 2:

- Faça uma escala de azul
- Pode utilizar a imagem *entrada.jpg*
- Dica:

```
for b in range(0, 255, 1): #percorre as linhas
    for g in range(0, 255, 10): #percorre as colunas
        for r in range(0, 255, 10): #percorre as colunas
            imagem[b, int(25*g/10+r/10)] = (b,g,r)
```

- Não se esqueça de mostrar a imagem



# Exercício 3

- Gere uma imagem com quadrados verdes
- Pode utilizar a imagem *entrada.jpg*
- Dicas:

```
for x in range(0, imagem.shape[1], 20): #percorre as linhas  
    for y in range(0, imagem.shape[0], 20): #percorre as colunas
```

```
        for ax in range (0,5,1):  
            for ay in range (0,5,1):  
                imagem[y+ay, x+ax] = (0,255,0)
```

- Não se esqueça de mostrar a imagem



# Fatiamento e desenho sobre a imagem

## Exemplo:

```
import cv2
image = cv2.imread('entrada.jpg')

#Cria um retangulo azul por toda a
largura da imagem
image[30:50, :] = (255, 0, 0)

#Cria um quadrado vermelho
image[100:150, 50:100] = (0, 0, 255)

#Cria um retangulo amarelo por toda a
altura da imagem
image[:, 200:220] = (0, 255, 255)

#Cria um retangulo verde da linha 150 a
300 nas colunas 250 a 350
image[150:300, 250:350] = (0, 255, 0)
```

```
#Cria um quadrado ciano da linha 150 a
300 nas colunas 250 a 350
image[300:400, 50:150] = (255, 255, 0)

#Cria um quadrado branco
image[250:350, 300:400] = (255, 255,
255)

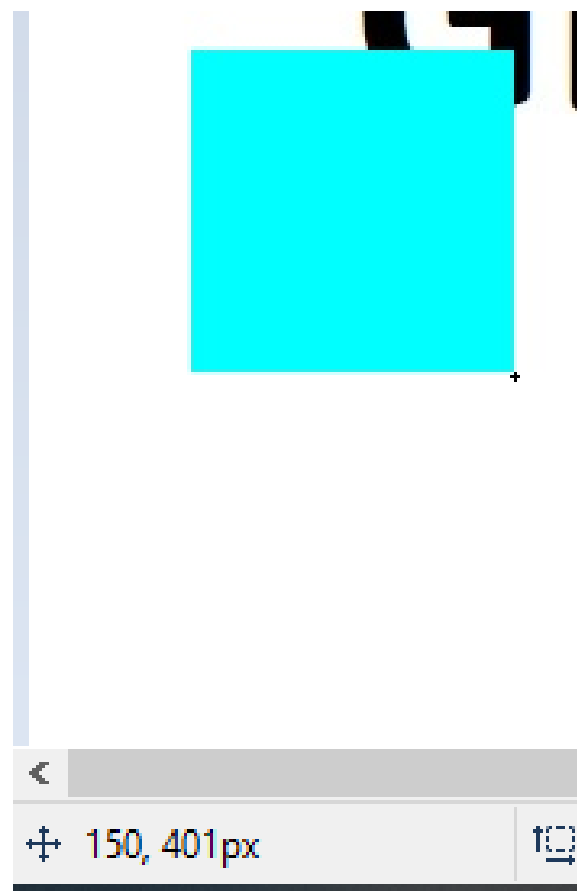
#Cria um quadrado preto
image[70:100, 300: 450] = (0, 0, 0)

cv2.imshow("Imagem alterada", image)
cv2.imwrite("alterada.jpg", image)
cv2.waitKey(0)
```

# Conceito

```
#image [yi:yf,xi:xf]
```

```
image[300:400, 50:150] = (255, 255, 0)
```



X,Y

# Resultado



# Exercícios

1. Faça uma moldura ao redor da imagem
2. Escreva seu nome no rodapé
3. Considerando a imagem entrada.jpg, faça um programa que troque o branco pelo verde claro (0,200,0) e o preto pelo verde escuro (0,50,0)
4. Considerando a imagem entrada.jpg, faça um programa que inverta as cores (branco/preto)

Aplicações



# 1. Troca de cores:

```
# Importação das bibliotecas
import cv2
# Leitura da imagem com a função imread()
imagem = cv2.imread('entrada.jpg')
print('Largura em pixels: ', imagem.shape[1]) #largura da imagem
print('Altura em pixels: ', imagem.shape[0]) #altura da imagem
print('Qtde de canais: ', imagem.shape[2])

for x in range(0, imagem.shape[1], 1): #percorre as linhas
    for y in range(0, imagem.shape[0], 1): #percorre as colunas
        (b, g, r) = imagem[y, x]
        if ((b>200)&(g>200)&(r>200)):
            imagem[y,x] = (0,60,0)
        if ((b<20)&(g<20)&(r<20)):
            imagem[y,x] = (0,220,0)

# Mostra a imagem com a função imshow
cv2.imshow("Nome da janela", imagem)
cv2.waitKey(0) #espera pressionar qualquer tecla
# Salvar a imagem no disco com função imwrite()
cv2.imwrite("saida.jpg", imagem)
```

**TESTE  
GECO9AN**



## 2. Reconhecimento de formas

```
# Importação das bibliotecas
import cv2
# Leitura da imagem com a função imread()
imagem = cv2.imread('vermelho.jpg')
print('Largura em pixels: ', imagem.shape[1]) #largura da imagem
print('Altura em pixels: ', imagem.shape[0]) #altura da imagem
print('Qtde de canais: ', imagem.shape[2])
xi= imagem.shape[1]
xf=0
yi= imagem.shape[0]
yf=0

for y in range(0, imagem.shape[0], 1): #percorre as linhas
    for x in range(0, imagem.shape[1], 1): #percorre as colunas
        (b, g, r) = imagem[y, x]
        if ((b==0)&(g==0)&(r>220)):
            if (x<xi): xi=x
            if (x>xf): xf=x
            if (y<yi): yi=y
            if (y>yf): yf=y

print ('xi= ',xi, 'xf= ',xf)
print ('yi= ',yi, 'yf= ',yf)

cv2.waitKey(0) #espera pressionar qualquer tecla
```

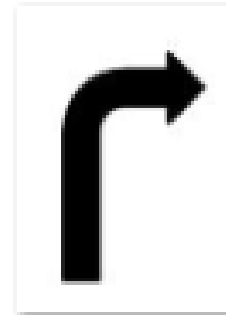


```
----- RESISTIR. C. (0301)
Largura em pixels:  440
Altura em pixels:  318
Qtde de canais:  3
xi=  118 xf=  194
yi=  74 yf=  245
>>> |
```

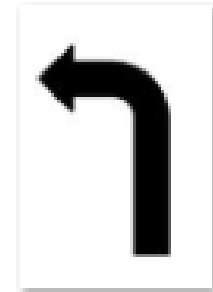
# 3. Reconhecimento de imagens

```
# Importação das bibliotecas
import cv2
# Leitura da imagem com a função imread()
imagem = cv2.imread('Direita.jpg')
print('Largura em pixels: ', imagem.shape[1]) #largura da imagem
print('Altura em pixels: ', imagem.shape[0]) #altura da imagem
print('Qtde de canais: ', imagem.shape[2])
xi= imagem.shape[1]
xf=0
```

```
for y in range(0, imagem.shape[0], 1): #percorre as linhas
    for x in range(0, imagem.shape[1], 1): #percorre as colunas
        (b, g, r) = imagem[y, x]
        if ((b==0)&(g==0)&(r==0)):
            if (x<xi): xi=x
            if (x>xf): xf=x
            if (x<xf): xm=x
print ('xi= ',xi, 'xf= ',xf, 'xm= ',xm)
if(xm<(xi+xf)/2): texto = 'DIREITA'
if(xm>(xi+xf)/2): texto = 'ESQUERDA'
print (texto)
cv2.imshow(texto, imagem)
cv2.waitKey(0) #espera pressionar qualquer tecla
```



Direita



Esquerda

```
Largura em pixels: 105
Altura em pixels: 148
Qtde de canais: 3
xi= 22 xf= 89 xm= 39
DIREITA
Largura em pixels: 101
Altura em pixels: 149
Qtde de canais: 3
xi= 11 xf= 78 xm= 77
ESQUERDA
```

# Com base nos exemplos...

1. Informar se a figura é um quadrado ou um retângulo
2. Completar o programa anterior, informando a área e o perímetro da figura
3. Completar os anteriores, informando a cor da figura (considere que a figura seja apenas preta ou azul ou verde ou vermelha)

# OCR

<https://nanonets.com/blog/ocr-with-tesseract/>

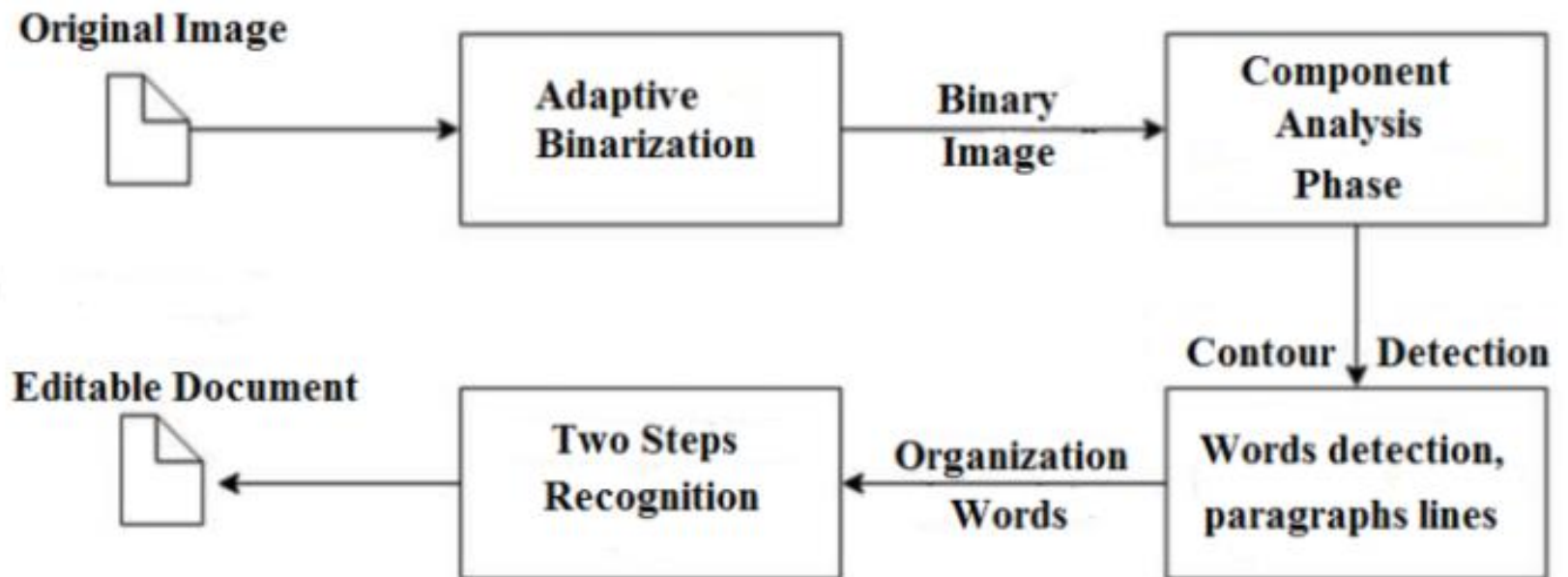
## Introduction

OCR = Optical Character Recognition.

In other words, OCR systems transform a two-dimensional image of text, that could contain machine printed or handwritten text from its image representation into machine-readable text. OCR as a process generally consists of several sub-processes to perform as accurately as possible. The subprocesses are:

- Preprocessing of the Image
- Text Localization
- Character Segmentation
- Character Recognition
- Post Processing

# Tesseract OCR



Tesseract 3 OCR process from [paper](#)

# OCR with Pytesseract and OpenCV

```
import cv2
```

```
import pytesseract img = cv2.imread('image.jpg')
```

```
# Adding custom options
```

```
custom_config = r'--oem 3 --psm 6'
```

```
pytesseract.image_to_string(img, config=custom_config)
```

# Preprocessing for Tesseract

```
import cv2
import numpy as np

try:
    from PIL import Image
except ImportError:
    import Image

import pytesseract
pytesseract.pytesseract.tesseract_cmd =
r'C:\Program Files (x86)\Tesseract-OCR\tesseract'

# get grayscale image
def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# noise removal
def remove_noise(image):
    return cv2.medianBlur(image,5)

#thresholding
def thresholding(image):
    return cv2.threshold(image, 0, 255,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
```

```
#dilation
def dilate(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.dilate(image, kernel, iterations = 1)

#erosion
def erode(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.erode(image, kernel, iterations = 1)

#opening - erosion followed by dilation
def opening(image):
    kernel = np.ones((5,5),np.uint8)
    return cv2.morphologyEx(image,
cv2.MORPH_OPEN, kernel)

#canny edge detection
def canny(image):
    return cv2.Canny(image, 100, 200)
```



```
#skew correction
def deskew(image):
    coords = np.column_stack(np.where(image
> 0))
    angle = cv2.minAreaRect(coords)[-1]
    if angle < -45:
        angle = -(90 + angle)
    else:
        angle = -angle
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center,
angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w,
h), flags=cv2.INTER_CUBIC,
borderMode=cv2.BORDER_REPLICATE)
    return rotated
```

```
#template matching
def match_template(image, template):
    return cv2.matchTemplate(image,
template, cv2.TM_CCOEFF_NORMED)
```

```
#####
```

```
image1 = cv2.imread('placa.jpg')
```

```
gray = get_grayscale(image1)
thresh = thresholding(gray)
opening = opening(gray)
canny = canny(gray)
```

```
cv2.imshow("Original", image1)
cv2.imshow("Gray", gray)
cv2.imshow("thresh", thresh)
cv2.imshow("opening", opening)
cv2.imshow("canny", canny)
```

```
print ("Altura (height): %d pixels" %
(image1.shape[0]))
print ("Largura (width): %d pixels" %
(image1.shape[1]))
print ("Canais (channels): %d" %
(image1.shape[2]))
```

```
# Salva a imagem
cv2.imwrite("image2.jpg", gray)
print(pytestesseract.image_to_string(Image.o
pen('image2.jpg')))
```

# Na prática:

- 1. Comparar as imagens após o reproprocessamento (ocrteste1.py+placa.jpg)
- 2. Reconhecer a placa de um automóvel (CancelaInicial+placa1.jpg)

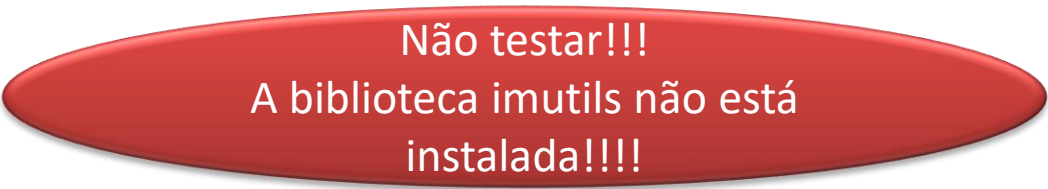
Manipulação de imagens  
livro pág. 15

# Cortando uma imagem

```
import cv2  
imagem = cv2.imread('ponte.jpg')  
recorte = imagem[100:200, 100:200]  
cv2.imshow("Recorte da imagem", recorte)  
cv2.imwrite("recorte.jpg", recorte) #salva no  
disco
```

# Redimensionando

```
import numpy as np
import imutils
import cv2
```

A red oval with a slight gradient and a drop shadow, containing a warning message in white text.

Não testar!!!  
A biblioteca imutils não está  
instalada!!!!

```
img = cv2.imread('ponte.jpg')
cv2.imshow("Original", img)
proporcao = 100.0 / img.shape[1]
tamanho_novo = (100, int(img.shape[0] * proporcao))
img_redimensionada = cv2.resize(img, tamanho_novo, interpolation =
cv2.INTER_AREA)
cv2.imshow("Imagem redimensionada", img_redimensionada)
cv2.waitKey(0)
```

# Espelhando uma imagem

```
import cv2
img = cv2.imread('entrada.jpg')
cv2.imshow("Original", img)

flip_horizontal = img[::-1,:] #comando equivalente abaixo #flip_horizontal =
cv2.flip(img, 1)
cv2.imshow("Flip Horizontal", flip_horizontal)

flip_vertical = img[:,::-1] #comando equivalente abaixo #flip_vertical = cv2.flip(img, 0)
cv2.imshow("Flip Vertical", flip_vertical)

flip_hv = img[::-1,::-1] #comando equivalente abaixo #flip_hv = cv2.flip(img, -1)
cv2.imshow("Flip Horizontal e Vertical", flip_hv)

cv2.waitKey(0)
```

# Rotacionando

```
import cv2

img = cv2.imread('entrada.jpg')
(alt, lar) = img.shape[:2] #captura altura e largura
centro = (lar // 2, alt // 2) #acha o centro

M = cv2.getRotationMatrix2D(centro, 30, 1.0) #30 graus
img_rotacionada = cv2.warpAffine(img, M, (lar, alt))

cv2.imshow("Imagem rotacionada em 45 graus", img_rotacionada)

cv2.waitKey(0)
```

# Máscaras

```
import cv2
import numpy as np
img = cv2.imread('entrada.jpg')
cv2.imshow("Original", img)
mascara = np.zeros(img.shape[:2], dtype = "uint8")
(cX, cY) = (img.shape[1] // 2, img.shape[0] // 2)
cv2.circle(mascara, (cX, cY), 100, 255, -1)
img_com_mascara = cv2.bitwise_and(img, img, mask = mascara)
cv2.imshow("Máscara aplicada à imagem", img_com_mascara)
cv2.waitKey(0)
```



# Sistemas de Cores

```
import cv2
import numpy as np
img = cv2.imread('entrada.jpg')
cv2.imshow("Original", img)

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow("Gray", gray)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2.imshow("HSV", hsv)

lab = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
cv2.imshow("L*a*b*", lab)
cv2.waitKey(0)
```

# Canais da imagem colorida

```
import cv2
import numpy as np
img = cv2.imread('cores.jpg')

(canalAzul, canalVerde, canalVermelho) = cv2.split(img)

cv2.imshow("Vermelho", canalVermelho)
cv2.imshow("Verde", canalVerde)
cv2.imshow("Azul", canalAzul)
cv2.waitKey(0)
```

# Exibindo os canais nas cores originais

```
import numpy as np
import cv2
img = cv2.imread('entrada.jpg')

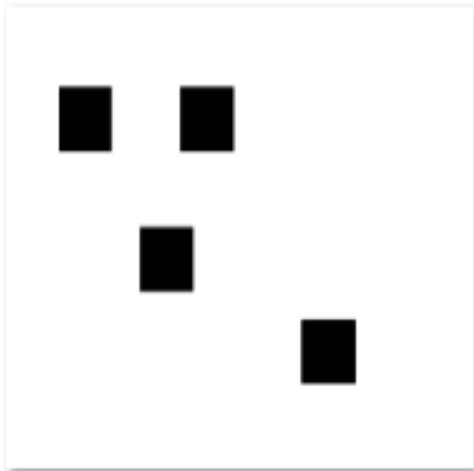
(canalAzul, canalVerde, canalVermelho) = cv2.split(img)

zeros = np.zeros(img.shape[:2], dtype = "uint8")

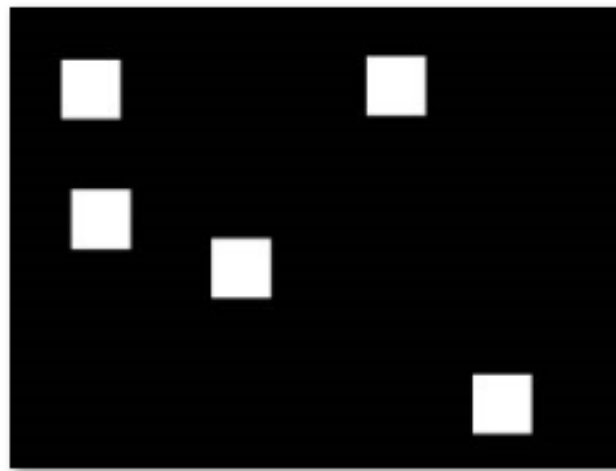
cv2.imshow("Vermelho", cv2.merge([zeros, zeros, canalVermelho]))
cv2.imshow("Verde", cv2.merge([zeros, canalVerde, zeros]))
cv2.imshow("Azul", cv2.merge([canalAzul, zeros, zeros]))

cv2.imshow("Original", img)
cv2.waitKey(0)
```

# Segmentação e Detecção de Bordas



Reconhecimento



Reconhecimento1



Reconhecimento3

# Exemplo de código

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from sys import argv

# mostra imagem na tela
def mostrar_imagem(nome, img):
    cv2.imshow(nome, img);
```

```
# pega os 4 vizinhos de uma coordenada
def vizinhos(img, y, x):
    vizinhos = [];
    if (y + 1 < len(img)):
        vizinhos.append((y + 1, x));
    if (y - 1 >= 0):
        vizinhos.append((y - 1, x));
    if (x + 1 < len(img[y])):
        vizinhos.append((y, x + 1));
    if (x - 1 >= 0):
        vizinhos.append((y, x - 1));
    return vizinhos;
```

```
# busca em largura na imagem
def bfs(img, ponto, pintado):
    y, x = ponto
    img[y][x] = pintado
    fila = [ponto];
    while fila:
        y, x = fila.pop()
        for vizinho in vizinhos(img, y, x):
            y_v, x_v = vizinho;
            cor = img[y_v][x_v];
            #if (cor > 0 and cor != pintado): # para objetos brancos
            if (cor == 0 and cor != pintado): # para objetos pretos
                img[y_v][x_v] = pintado;
                fila.append(vizinho);
```

```
# conta quantidade de objetos em uma imagem binaria
def contar_objetos(img):
    # cor usada para pintar
    pintado = 150
    total_objetos = 0;
    for y in range(0, len(img)):
        for x in range(0, len(img[y])):
            cor = img[y][x];
            #if cor == 255 and cor != pintado: # conta branco
            if cor == 0 and cor != pintado: # conta preto
                total_objetos += 1;
                bfs(img, (y, x), pintado);
            pintado = pintado+10
    print ('Quantidade de objetos:', total_objetos);
```



##### Corpo Principal #####

```
cv2.destroyAllWindows()
```

```
img = cv2.imread('Reconhecimento.jpg',0) #monocromática = binária
```

```
ret, imgT = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)#230
```

```
mostrar_imagem('original-binaria', imgT);
```

```
contar_objetos(imgT);
```

```
mostrar_imagem('original-tons-cinza', img);
```

```
mostrar_imagem('pintada', imgT);
```

```
n,bins,patches = plt.hist(imgT.ravel(), 256, [1, 255])
```

```
plt.show()
```

```
cv2.waitKey()
```

# Exercícios

- 1-) Alterar a programação para contar imagens brancas em fundo preto. (arquivo Reconhecimento1.jpg)
- 2-) Alterar a programação para reconhecer imagens coloridas em fundo branco (arquivo Reconhecimento2.jpg)

# Contando elementos

# Comparando 2 imagens

- <https://pysource.com/2018/07/19/check-if-two-images-are-equal-with-opencv-and-python/>

Finding if two images are equal with Opencv, is a quite simple operation.

There are **2 fundamental elements to consider**:

1. The images have both the same size and channels
2. Each pixel has the same value

- We loaded the two images we can start making the comparison.  
**First we check if they have the same size and channels.** If they have the same size and channels, we continue further with the operation, if they don't then that they're not equal.
- If they have the same sizes and channels, **we proceed by subtracting them.** The operation `cv2.subtract(image1, image2)` simply subtract from each pixel of the first image, the value of the corresponding pixel in the second image.
- If for example the value of the pixel of the first image in the position (0, 0) is 255 and the value of the pixel in the corresponding position of the second image is also 255, it will be a simple subtraction:  $255 - 255$  which is equal to 0.
- That's why if the images are equal, the result will be a black image (which means each pixel will have value 0).

- **A colored image has 3 channels** (blue, green and red), so the `cv2.subtract()` operation makes the **subtraction for each single channel** and we need to check if all the three channels are black.

If they are, we can say that the images are equal.

# Exemplo

```
import cv2
import numpy as np

original =
cv2.imread("Componentes.jpg")
duplicate =
cv2.imread("ComponentesIguais.jpg"
)

# 1) Check if 2 images are equals
if original.shape == duplicate.shape:
    print("The images have same size
and channels")

difference = cv2.subtract(original,
duplicate)

b, g, r = cv2.split(difference)

if cv2.countNonZero(b) == 0 and
cv2.countNonZero(g) == 0 and
cv2.countNonZero(r) == 0:
    print("The images are completely
Equal")

if cv2.countNonZero(b) != 0 or
cv2.countNonZero(g) != 0 or
cv2.countNonZero(r) != 0:
    print("The images are not
completely Equal")

cv2.imshow("Original", original)
cv2.imshow("Duplicate", duplicate)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



# Destacando os pixels diferentes

```
import cv2
import numpy as np
original = cv2.imread("Componentes.jpg")
duplicate = cv2.imread("ComponentesDiferentes.jpg")
```

```
# 1) Check if 2 images are equals
if original.shape == duplicate.shape:
    print("The images have same size and channels")
```

```
difference = cv2.subtract(original, duplicate)
```

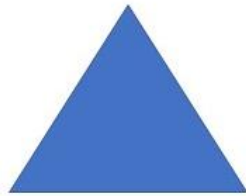
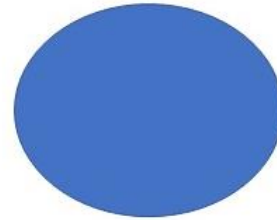
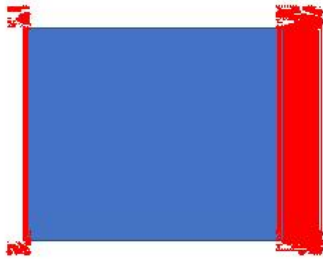
```
b, g, r = cv2.split(difference)
```

```
for y in range(0, original.shape[0], 1):
    for x in range(0, original.shape[1], 1):
        if b[y,x] != 0 or g[y,x] != 0 or r[y,x] != 0:
            duplicate[y,x] = (0, 0, 255)
```

```
if cv2.countNonZero(b) == 0 and cv2.countNonZero(g) == 0 and cv2.countNonZero(r) == 0:
    print("The images are completely Equal")
```

```
if cv2.countNonZero(b) != 0 or cv2.countNonZero(g) != 0 or cv2.countNonZero(r) != 0:
    print("The images are not completely Equal")
```

```
cv2.imshow("Original", original)
cv2.imshow("Duplicate", duplicate)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Resultado

# Exercícios