

# Complemento Teórico sobre Imagens Digitais

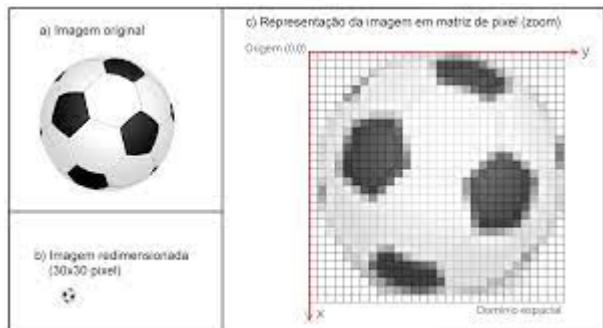
Teoria aplicada ao processamento digital de Imagens utilizado no sistema OpenPDI

Imagen Digital	3
Visão Computacional	4
Python	5
Numpty	5
ndarray	6
arange e linspace	6
Fatiamento (Slicing)	7
Copy	7
operações	7
meshgrid	8
nonzero	9
tile	9
clip	9
sort	10
argsort	10
OpenCV	11
Open>Show Images	11
BGR	11
RGB	11
Grayscale	11
Brilho e Contraste	12
Rotate	14
Contando o número de pixels em preto e branco na imagem	15
Subtração de Fundo	18
Encontre e desenhe contornos usando OpenCV	20
Canny Edge Detector	22
Detecção de contorno com sementes personalizadas usando Python – OpenCV	27
Matplotlib	31
Pandas	32
PIL	32
Resize com PIL	33

Detecção de borda com Pillow	33
<b>Pré processamento de imagens</b>	<b>38</b>
Redimensionamento	38
Redimensionamento no openCV	39
Segmentação	40
Limiarização	40
Limiarização Simples	40
Teste Limiar Simples	41
Método de Otsu	44
Limiarização Adaptativa	45
Primeira atividade de Limiarização Adaptativa	46
imagem de entrada	48
Limiar Ordinário	49
CLAHE Aplicado	49
Limiarização Adaptativa Gaussiana	49
Operações Morfológicas	50
Dilatação e Erosão	50
Erosão	50
Dilatação	52
Abertura e Fechamento	52
Abertura	53
Fechamento	54
Operações morfológicas no processamento de imagens (gradiente)	55
Remoção de Ruído	57
Convolução	57
Remoção de Ruído com Desfoque	59
Desfoque com Média	60
Desfoque Gaussiano	60
Desfoque com Mediana	61
Filtro Bilateral	62

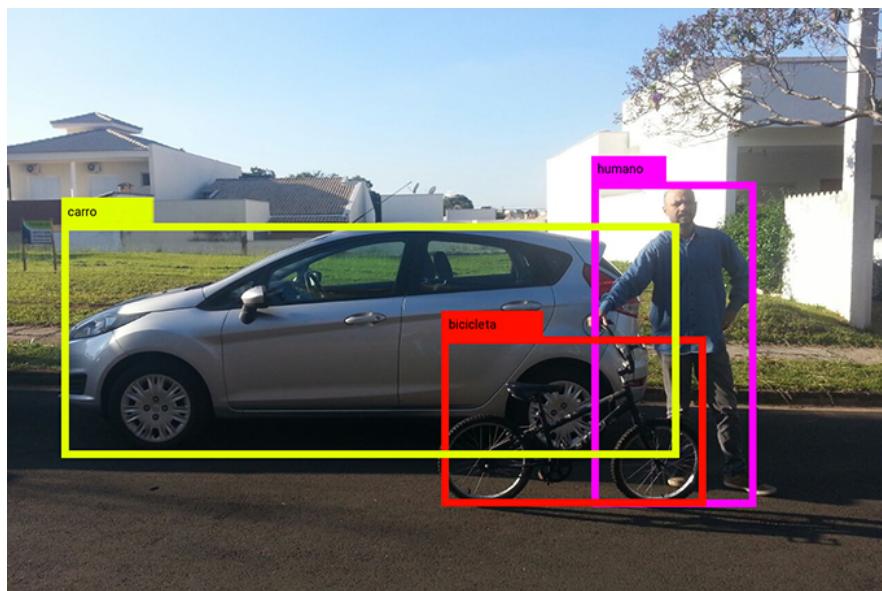
## Imagen Digital

### Pixels

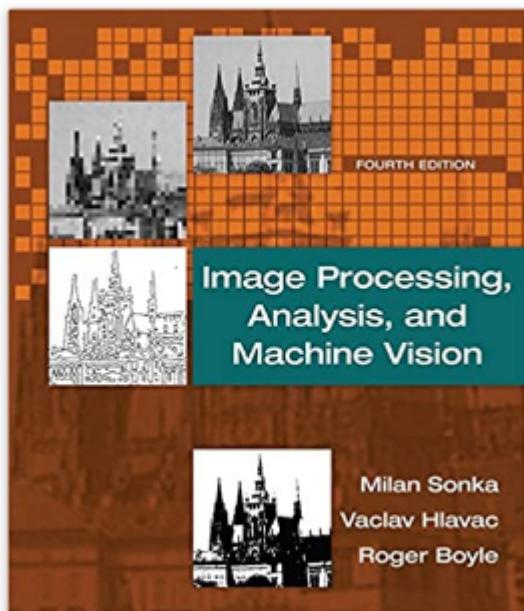


caixa delimitadora mínima (em inglês "minimum bounding box", ou MBB)  
region of interest (roi)

## Visão Computacional



Livro: Image Processing, Analysis, and Machine Vision



<https://www.casadocodigo.com.br/products/livro-visao-computacional>

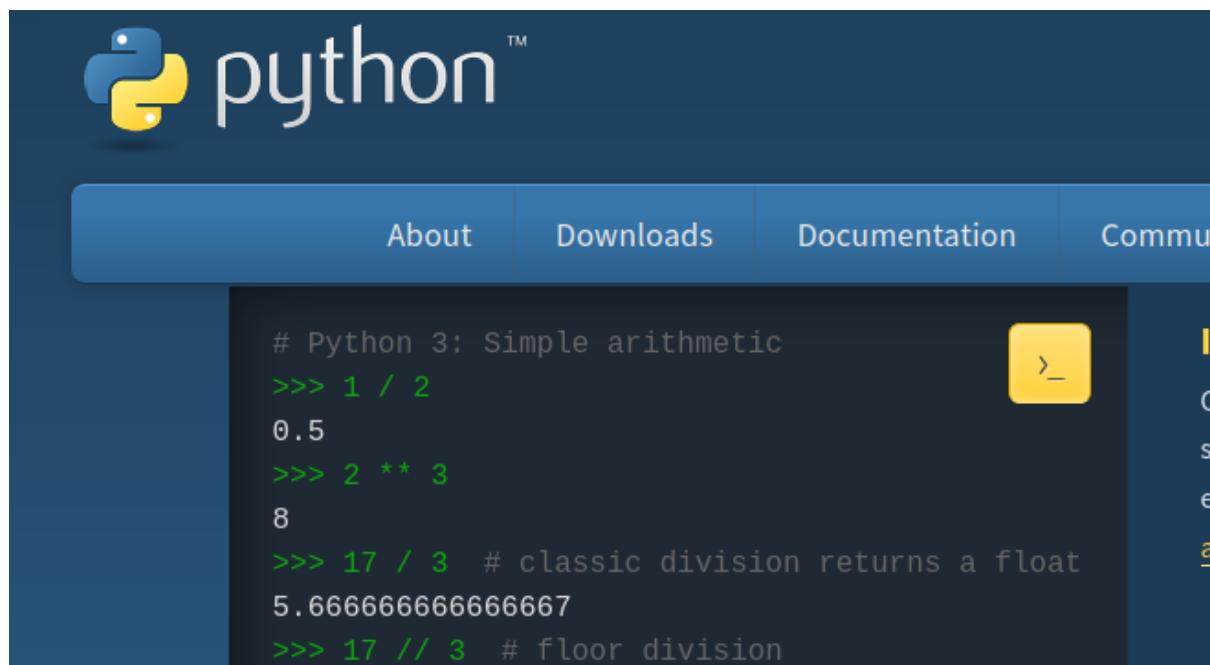
The image displays the book cover for 'Introdução à Visão Computacional' by Felipe Barelli. The cover features a large blue eye icon with circuit board patterns extending from it. The title 'Introdução à Visão Computacional' is at the top, followed by the subtitle 'Uma abordagem prática com Python e OpenCV'. The author's name 'Felipe Barelli' is below the subtitle. To the right, there are three purchase options: 'E-book\*' for R\$ 39,90, 'Impresso' for R\$ 69,90 + frete, and 'E-book + Impresso' for R\$ 84,90 + frete, each with a 'COMPRAR' button.

**Introdução à Visão Computacional**  
Uma abordagem prática com Python e OpenCV  
Felipe Barelli

E-book*	por R\$ 39,90	COMPRAR
Impresso	R\$ 69,90 + frete	COMPRAR
E-book + Impresso	R\$ 84,90 + frete	COMPRAR

Python

<https://www.python.org/>



## Numpy

<https://numpy.org/>



The fundamental package for scientific computing with Python

[GET STARTED](#)

Biblioteca para trabalhar com vetores e matrizes.  
Imagens digitais são matrizes.

ndarray

ndarray

```
# ndarray
#a = np.array([2,4,6,8,10])
#print("a:\n", a)
#print("a.ndim:\n", a.ndim)
#print("a.shape:\n", a.shape)
#print("a.dtype:\n", a.dtype)
#a.dtype = np.uint8
#print("a.dtype:\n", a.dtype)

np.zeros(3,3)
np.ones(2,3)
```

### arange e linspace

```
# arange e linspace
#print(np.arange(5))
#print()
#print(np.arange(2,6))
#print()
#print(np.arange(0,2,0.5))
#print()
#print(np.linspace(0,2,5)) #define o inicial, o final e a
#quantidade de elementos entre os dois
```

### Fatiamento (Slicing)

```
#slice vetor
#a = np.arange(15) * 10
#print(a)
#print()
#print(a[2:5])
#print()
```

```
#slice matriz
#a = np.arange(49)
#a = a.reshape(7,7)
#print(a)
#print()
#print(a[::-2,::2])
```

Copy

```
## Copy
#a = np.arange(15)
#b = a.copy()
#c = a
#print(a)
#print()
#print(b)
#print()
#print(c)
```

operações

```
## Operacoes
#a = np.arange(6)
#a = a.reshape(2,3)
#b = np.array([1,3,2,4,2,1])
#b = b.reshape(3,2)
#print(a)
#print()
#print(b)
#print()
#print(b.T)
#print(a.dot(b) )
#print()
#print(a * b.T)

#a =
np.array([[1,8,9,16],[2,7,10,15],[3,6,11,14],[4,5,12,13]])
#print(a)
#print()
#print(a.max())
#print(a.min())
#print(a.sum())
#print(a.prod())
#print(a.cumsum())
#print(a.cumprod())
```

meshgrid

```
## meshgrid
#a = np.array([-1.5,-1.0,-0.5,0.0,0.5])
#b = np.array([-20,-10,0,10,20,30])
# pode-se usar linspace para gerar os parametros 'a' e 'b'
#l,c = np.meshgrid(a,b,indexing='ij')
#l,c = np.meshgrid(a,b)
#print(a)
#print()
#print(b)
#print()
#print(l)
#print()
#print(c)
```

nonzero

```
## nonzero
#a = np.array([0,32,0,14,0,0,83,0,110,0,0,21,0,0,0,0])
#(b,) = a.nonzero()
#print()
#print(a)
#print()
#print(b)
#print()
#print(a[b])
#print()
```

tile

```
# tile
#a = np.array([1,2,3])
#print(a)
#print()
#print(np.tile(a,5))
#print()
```

```
#print(np.tile(a, (2,3)))
#print()
#a = np.array([[1,2],[3,4]])
#print(a)
#print()
#print(np.tile(a,2))
#print()
#print(np.tile(a, (2,2)))
```

clip

```
## clip
#a = np.arange(20)
#print(a)
#print()
#print(np.clip(a,5,12))
#print()
#a = a.reshape(4,5)
#print(a)
#print()
#print(np.clip(a,5,10))
#print()
```

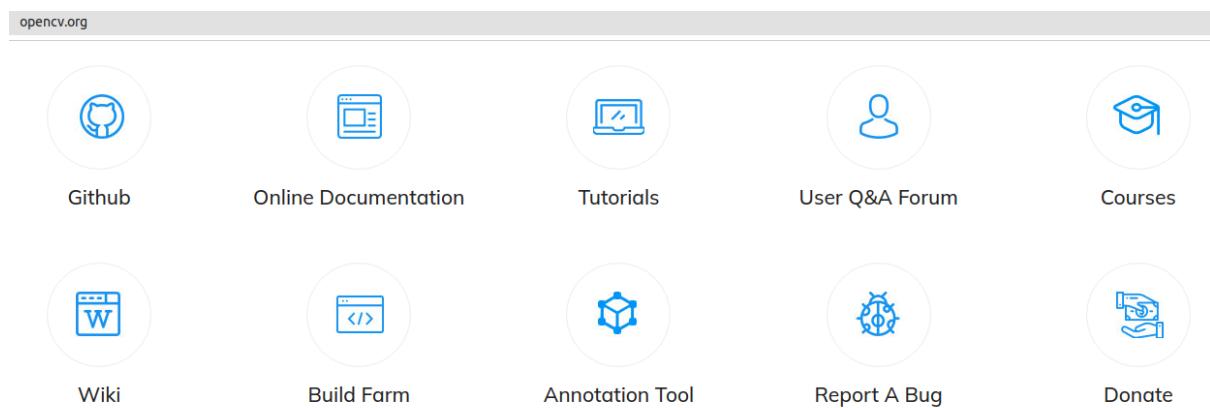
sort

```
# sort
#a = np.array([8,5,3,1,7,4,9,2])
#print(a)
#print()
#print(np.sort(a))
#a =
#np.array([[3,15,6,12],[8,10,4,14],[5,1,9,16],[2,7,11,13]])
#print(a)
#print()
#print(np.sort(a))
#print()
#print(np.sort(a, axis=0))
```

```
#print()  
  
argsort  
  
# argsort  
#a = np.array([8,5,3,1,6,4,7,2])  
#idx = np.arange(8)  
#print(idx)  
#print()  
#print(a)  
#print()  
#print(np.argsort(a))  
#print()  
#print(a[np.argsort(a)])
```

## OpenCV

<https://opencv.org/>



## Open/Show Images

```
img = cv2.imread('letra-m.jpg')  
cv2_imshow(img)
```

## BGR

Padrão utilizado pelo OPENCV

```
img = cv2.imread('/content/teste01.jpg')
cv2_imshow(img) # BGR
```

## RGB

Red Green Blue

```
rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
cv2_imshow(rgb)
```

## Grayscale

escala de cinza

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```

## Brilho e Contraste

- **Brilho:** quando o brilho é ajustado, toda a gama de tons na imagem é aumentada ou diminuída de acordo.
- **Contraste:** Quando o ajuste de contraste é aumentado, os tons médios são eliminados. A imagem terá uma porcentagem maior de tons escuros ou pretos e brancos ou realces com tons médios mínimos.

Execute no spyder:

```
import cv2

def BrightnessContrast(brightness=0):

    brightness = cv2.getTrackbarPos('Brightness',
                                    'GEEK')

    contrast = cv2.getTrackbarPos('Contrast',
                                 'GEEK')
```

```
effect = controller(img, brightness,
                    contrast)

cv2.imshow('Effect', effect)

def controller(img, brightness=255,
               contrast=127):

    brightness = int((brightness - 0) * (255 - (-255)) / (510 - 0)
+ (-255))

    contrast = int((contrast - 0) * (127 - (-127)) / (254 - 0) +
(-127))

    if brightness != 0:
        if brightness > 0:
            shadow = brightness
            max = 255
        else:
            shadow = 0
            max = 255 + brightness
        al_pha = (max - shadow) / 255
        ga_mma = shadow
        cal = cv2.addWeighted(img, al_pha,
                             img, 0, ga_mma)

    else:
        cal = img

    if contrast != 0:
        Alpha = float(131 * (contrast + 127)) / (127 * (131 -
contrast))
        Gamma = 127 * (1 - Alpha)
        cal = cv2.addWeighted(cal, Alpha,
                             cal, 0, Gamma)

cv2.putText(cal, 'B:{} ,C:{}' .format(brightness,
                                         contrast), (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
```

```
return cal

if __name__ == '__main__':
    original = cv2.imread("pic.jpeg")

    img = original.copy()

    cv2.namedWindow('GEEK')

    cv2.imshow('GEEK', original)

    cv2.createTrackbar('Brightness',
                      'GEEK', 255, 2 * 255,
                      BrightnessContrast)

    cv2.createTrackbar('Contrast', 'GEEK',
                      127, 2 * 127,
                      BrightnessContrast)

    BrightnessContrast(0)

cv2.waitKey(0)
```

## Rotate

`cv2.rotate()` método é usado para girar uma matriz 2D em múltiplos de 90 graus.

A função `cv :: rotate` gira o array de três maneiras diferentes.

**Sintaxe:** `cv2.cv.rotate (src, rotateCode [, dst])`

**Parâmetros:**

**src:** É a imagem cujo espaço de cores deve ser alterado.

**rotateCode:** É um enum para especificar como girar a matriz.

**DST:** é a imagem de saída do mesmo tamanho e profundidade da imagem src. É um parâmetro opcional.

**Valor de retorno:** ele retorna uma imagem.

Imagen usada para todos os exemplos abaixo:

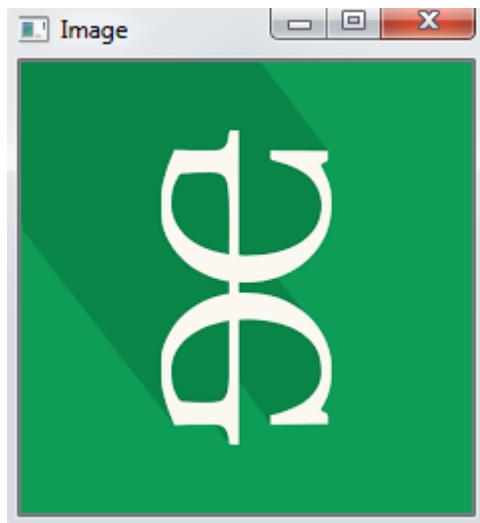


**Exemplo # 1:** girar 90 graus no sentido horário

```
import cv2
path = r'.\geeks14.png'
src = cv2.imread(path)
window_name = 'Image'

image = cv2.rotate(src, cv2.cv2.ROTATE_90_CLOCKWISE)
cv2.imshow(window_name, image)
cv2.waitKey(0)
```

## Resultado:

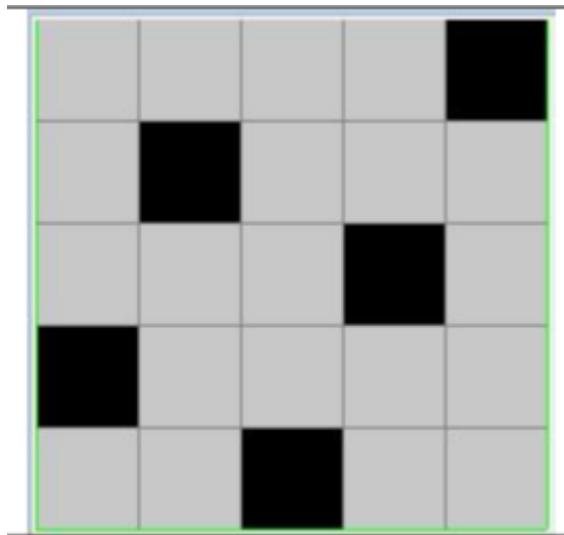


Contando o número de pixels em preto e branco na imagem

## Pré-requisitos:

- OpenCV
- NumPy

A imagem que estamos usando para a demonstração é mostrada abaixo:



Para exibir o OpenCV fornece o método `imshow()` para exibir a imagem que lemos recentemente.

## Contando Pixels

NumPy fornece uma função `sum()` que retorna a soma de todos os elementos do array NumPy. Esta função `sum()` pode ser usada para contar o número de pixels com base nos critérios exigidos.

Agora, um pouco de conhecimento dos padrões de pixel entra em cena. Como sabemos que cada pixel em uma imagem colorida varia de [0-255] com tudo incluído, o valor do pixel para a cor preta sendo 0 e para a cor branca é 255. Isso nos dá uma certa condição fixa de diferenciação para os pixels pretos e brancos, respectivamente, dos pixels de outras cores, respectivamente.

Esta condição pode ser escrita no NumPy como:

```
number_of_white_pix = np.sum (img == 255) # extraindo apenas pixels  
brancos
```

```
number_of_black_pix = np.sum (img == 0) # extraindo apenas pixels  
pretos
```

A primeira linha diz para extrair e contar todos os pixels do objeto de imagem cv2 “`img`” cujo valor de pixel é 255, ou seja, pixels brancos. Da mesma forma, a segunda linha diz para extrair e contar todos os pixels do objeto de imagem cv2 “`img`” cujo valor de pixel é 0, ou seja, pixels pretos.

A condição dentro de `sum()` extrai apenas os pixels da imagem que seguem essa condição e atribui a eles o valor True (1) e o restante dos pixels é atribuído a False (0). Assim, a soma de todos os True (1s) nos dá a contagem dos pixels que estão satisfazendo a condição fornecida entre parênteses.

### Código: implementação Python para contar o número de pixels em preto e branco na imagem usando OpenCV

```
import cv2  
import numpy as np  
img = cv2.imread("chess5.png")  
cv2.imshow('Image',img)  
number_of_white_pix = np.sum(img == 255)  
number_of_black_pix = np.sum(img == 0)
```

```
print('Number of white pixels:', number_of_white_pix)
print('Number of black pixels:', number_of_black_pix)
```

Resultado:

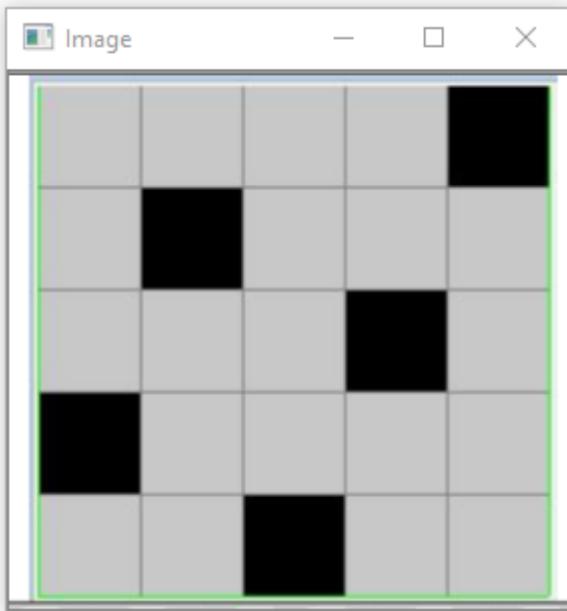


Imagen para processar

```
Number of white pixels: 11745
Number of black pixels: 30892
```

Contagem de pixels em preto e branco da imagem acima

Assim, podemos inferir dos resultados acima que a imagem pode ser processada para recuperar os pixels de qualquer cor desejada com a ajuda de seu código de cores ou obter os pixels com base em qualquer outra condição conforme necessário.

Referências:

- <https://numpy.org/doc/stable/reference/generated/numpy.sum.html>
- <https://note.nkmk.me/en/python-numpy-count>

## Subtração de Fundo

A subtração do fundo é uma forma de eliminar o fundo da imagem. Para conseguir isso, extraímos o primeiro plano em movimento do fundo estático.

Background Subtraction tem vários casos de uso no dia a dia, está sendo usado para segmentação de objetos, reforço de segurança, rastreamento de pedestres, contagem do número de visitantes, número de veículos no trânsito etc. É capaz de aprender e identificar a máscara de primeiro plano.

No OpenCV, temos 3 algoritmos para fazer esta operação –

**BackgroundSubtractorMOG** – É um algoritmo de segmentação de fundo / primeiro plano baseado em mistura gaussiana.

**BackgroundSubtractorMOG2** – também é um algoritmo de segmentação de fundo / primeiro plano baseado em mistura gaussiana. Fornece melhor adaptabilidade a cenas variadas devido a mudanças de iluminação, etc.

**BackgroundSubtractorGMG** – Este algoritmo combina estimativa estatística de imagem de fundo e segmentação bayesiana por pixel.

Como aplicar as funções embutidas do OpenCV para subtração de fundo –

**Etapa # 1** – Criar um objeto para representar o algoritmo que estamos usando para subtração de fundo.

**Etapa 2** – Aplicar `backgroundsubtractor.apply()` função na imagem.

```
import numpy as np

import cv2

fgbg1 = cv2.bgsegm.createBackgroundSubtractorMOG();

fgbg2 = cv2.createBackgroundSubtractorMOG2();
```

```
fgbg3 = cv2.bgsegm.createBackgroundSubtractorGMG();

cap = cv2.VideoCapture(0);

while(1):

    ret, img = cap.read();

    fgmask1 = fgbg1.apply(img);

    fgmask2 = fgbg2.apply(img);

    fgmask3 = fgbg3.apply(img);

    cv2.imshow('Original', img);

    cv2.imshow('MOG', fgmask1);

    cv2.imshow('MOG2', fgmask2);

    cv2.imshow('GMG', fgmask3);

    k = cv2.waitKey(30) & 0xff;

    if k == 27:

        break;

cap.release();

cv2.destroyAllWindows();
```

Encontre e desenhe contornos usando OpenCV

**Os contornos** são definidos como a linha que une todos os pontos ao longo do limite de uma imagem que possuem a mesma intensidade. Os contornos são úteis na análise de formas, na localização do tamanho do objeto de interesse e na detecção do objeto.

O OpenCV possui `findContour()` função que auxilia na extração dos contornos da imagem. Funciona melhor em imagens binárias, então devemos primeiro aplicar técnicas de limiar, bordas de Sobel, etc.

Abaixo está o código para encontrar contornos -

```
import cv2
import numpy as np
image = cv2.imread('./shapes.jpg')
cv2.waitKey(0)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edged = cv2.Canny(gray, 30, 200)
cv2.waitKey(0)

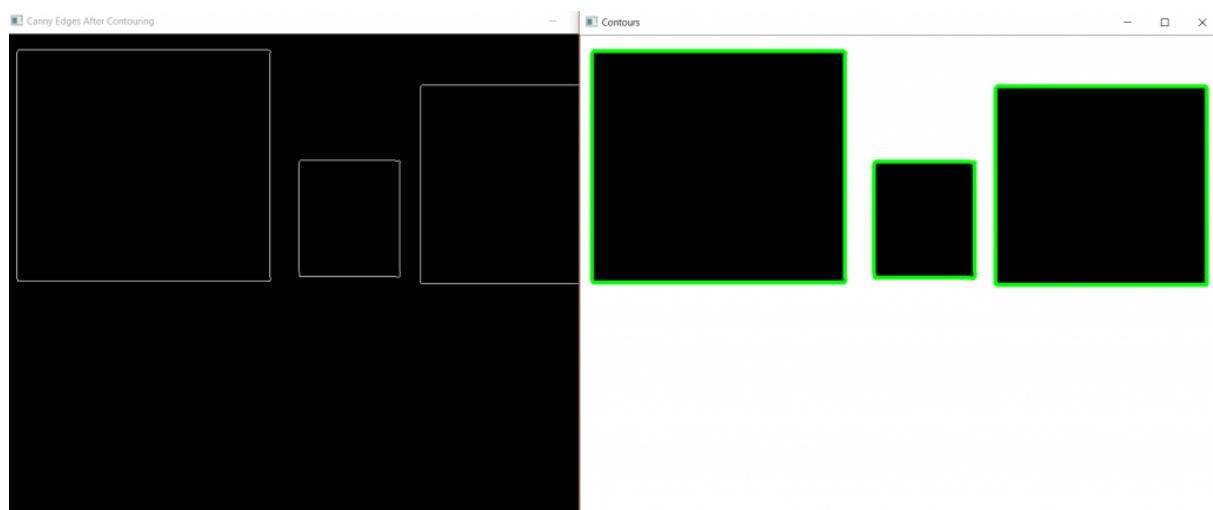
contours, hierarchy = cv2.findContours(edged,
                                         cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

cv2.imshow('Canny Edges After Contouring', edged)
cv2.waitKey(0)

print("Number of Contours found = " + str(len(contours)))
cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Saída:



vemos que existem três argumentos essenciais na `cv2.findContours()` função. O primeiro é a imagem de origem, o segundo é o modo de recuperação de contorno, o terceiro é o método de aproximação de contorno e produz a imagem, contornos e hierarquia. 'contours' é uma lista Python de todos os contornos da imagem. Cada contorno individual é um array Numpy de coordenadas (x, y) de pontos de fronteira do objeto.

## Canny Edge Detector

Canny desenvolvido por John F. Canny em 1986. Normalmente, em Matlab e OpenCV usamos a detecção de bordas astutas para muitas tarefas populares na detecção de bordas, como detecção de pista, esboços , remoção de bordas, agora aprenderemos o funcionamento interno e a implementação desse algoritmo do zero.

### Compreensão Teórica

As etapas básicas envolvidas neste algoritmo são:

- Redução de ruído usando filtro Gaussiano
- Cálculo do gradiente ao longo dos eixos horizontal e vertical
- Supressão não máxima de bordas falsas
- Limiar duplo para segregar bordas fortes e fracas
- Rastreamento de borda por histerese

conceitos:

#### 1. Redução de ruído usando filtro Gaussiano

Esta etapa é de extrema importância na detecção de bordas do Canny. Ele usa um filtro Gaussiano para a remoção do ruído da imagem, pois esse ruído pode ser assumido como bordas devido à mudança repentina de intensidade pelo detector de bordas. A soma dos elementos no kernel gaussiano é 1, portanto, o kernel deve ser normalizado antes de aplicar como convolução à imagem. Neste tutorial, usaremos um kernel de tamanho 5 X 5 e sigma = 1,4, que borrará a imagem e removerá o ruído dela. A equação para o kernel do filtro gaussiano é

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

## 2. Cálculo do gradiente

Quando a imagem é suavizada, as derivadas  $I_x$  e  $I_y$  são calculadas em relação aos eixos x e y. Pode ser implementado usando a convolução de kernels Sobel-Feldman com a imagem fornecida:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

### Sobel Kernels

depois de aplicar esses kernel, podemos usar as magnitudes do gradiente e o ângulo para processar ainda mais esta etapa. A magnitude e o ângulo podem ser calculados como

$$|G| = \sqrt{I_x^2 + I_y^2},$$

$$\theta(x, y) = \arctan \left( \frac{I_y}{I_x} \right)$$

### Magnitude e ângulo do gradiente

## 3. Supressão Não Máxima

Esta etapa visa reduzir os pixels de mesclagem duplicados ao longo das bordas para torná-los irregulares. Para cada pixel, encontre dois vizinhos nas direções do gradiente positiva e negativa, supondo que cada vizinho ocupe o ângulo  $\pi / 4$  e 0 seja a direção diretamente para a direita. Se a magnitude do pixel atual é maior que a magnitude dos vizinhos, nada muda, caso contrário, a magnitude do pixel atual é definida como zero.

#### 4. Limiar duplo

As magnitudes do gradiente são comparadas com dois valores de limite especificados, o primeiro é menor que o segundo. Os gradientes que são menores que o valor de limite baixo são suprimidos, os gradientes mais altos que o valor de limite alto são marcados como fortes e os pixels correspondentes são incluídos no mapa de borda final. Todos os gradientes restantes são marcados como fracos e os pixels correspondentes a esses gradientes são considerados na próxima etapa.

#### 5. Rastreamento de borda usando histerese

Uma vez que um pixel de borda fraca causado por bordas verdadeiras será conectado a um pixel de borda forte, o pixel W com gradiente fraco é marcado como borda e incluído no mapa de borda final se e somente se estiver envolvido no mesmo componente conectado que algum pixel S com forte gradiente. Em outras palavras, deve haver uma cadeia de pixels fracos vizinhos conectando W e S (os vizinhos têm 8 pixels ao redor do considerado). Vamos construir e implementar um algoritmo que encontra todos os componentes conectados do mapa de gradiente considerando cada pixel apenas uma vez. Depois disso, você pode decidir quais pixels serão incluídos no mapa de borda final.

Abaixo está a implementação.

```
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt

def Canny_detector(img, weak_th = None, strong_th = None):

    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    img = cv2.GaussianBlur(img, (5, 5), 1.4)

    gx = cv2.Sobel(np.float32(img), cv2.CV_64F, 1, 0, 3)
    gy = cv2.Sobel(np.float32(img), cv2.CV_64F, 0, 1, 3)

    mag, ang = cv2.cartToPolar(gx, gy, angleInDegrees = True)

    mag_max = np.max(mag)
    if not weak_th:weak_th = mag_max * 0.1
    if not strong_th:strong_th = mag_max * 0.5
```

```

height, width = img.shape

for i_x in range(width):
    for i_y in range(height):
        grad_ang = ang[i_y, i_x]
        grad_ang = abs(grad_ang-180) if abs(grad_ang)>180 else
abs(grad_ang)
            if grad_ang<= 22.5:
                neighb_1_x, neighb_1_y = i_x-1, i_y
                neighb_2_x, neighb_2_y = i_x + 1, i_y
            elif grad_ang>22.5 and grad_ang<=(22.5 + 45):
                neighb_1_x, neighb_1_y = i_x-1, i_y-1
                neighb_2_x, neighb_2_y = i_x + 1, i_y + 1
            elif grad_ang>(22.5 + 45) and grad_ang<=(22.5 +
90):
                neighb_1_x, neighb_1_y = i_x, i_y-1
                neighb_2_x, neighb_2_y = i_x, i_y + 1
            elif grad_ang>(22.5 + 90) and grad_ang<=(22.5 +
135):
                neighb_1_x, neighb_1_y = i_x-1, i_y + 1
                neighb_2_x, neighb_2_y = i_x + 1, i_y-1
            elif grad_ang>(22.5 + 135) and grad_ang<=(22.5 +
180):
                neighb_1_x, neighb_1_y = i_x-1, i_y
                neighb_2_x, neighb_2_y = i_x + 1, i_y
                if width>neighb_1_x>= 0 and height>neighb_1_y>=
0:
                    if mag[i_y, i_x]<mag[neighb_1_y, neighb_1_x]:
                        mag[i_y, i_x]= 0
                        continue
                    if width>neighb_2_x>= 0 and height>neighb_2_y>= 0:
                        if mag[i_y, i_x]<mag[neighb_2_y, neighb_2_x]:
                            mag[i_y, i_x]= 0

weak_ids = np.zeros_like(img)
strong_ids = np.zeros_like(img)
ids = np.zeros_like(img)

for i_x in range(width):
    for i_y in range(height):
        grad_mag = mag[i_y, i_x]

```

```
        if grad_mag<weak_th:  
            mag[i_y, i_x]= 0  
        elif strong_th>grad_mag>= weak_th:  
            ids[i_y, i_x]= 1  
        else:  
            ids[i_y, i_x]= 2  
  
    return mag  
  
frame = cv2.imread('food.jpeg')  
canny_img = Canny_detector(frame)  
  
plt.figure()  
f, plots = plt.subplots(2, 1)  
plots[0].imshow(frame)  
plots[1].imshow(canny_img)
```



Imagen de entrada



Imagen de saída

Detecção de contorno com sementes personalizadas usando Python – OpenCV  
detectar contornos dinamicamente clicando na imagem e atribuir cores diferentes a partes diferentes da imagem. Os contornos são uma ferramenta muito útil para análise de formas e detecção e reconhecimento de objetos. Este programa usa as bibliotecas OpenCV , Numpy e Matplotlib . Ele também usa um algoritmo integrado OpenCV Watershed para detectar contornos.

## Requisitos

- Python e OpenCV devem ser instalados na máquina local.
- Instale o Jupyter Notebook para depuração fácil.
- Aqui, o mapa de cores do Matplotlib é usado para obter cores diferentes. No exemplo abaixo, usaremos tab10 ([https://matplotlib.org/2.0.2/examples/color/colormaps\\_reference.html](https://matplotlib.org/2.0.2/examples/color/colormaps_reference.html)).

## Explicação

- Execute o programa.

- Clique na imagem onde deseja fazer contornos.
- Selecione uma cor diferente para uma parte diferente da imagem pressionando os números de zero a nove. (Um número para uma cor)

Abaixo está a implementação.

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

from matplotlib import cm

road = cv2.imread('road_image.jpg')

road_copy = np.copy(road)

marker_image = np.zeros(road.shape[:2], dtype=np.int32)

segments = np.zeros(road.shape, dtype=np.uint8)

def create_rgb(i):

    x = np.array(cm.tab10(i))[:3]*255

    return tuple(x)

colors = []

for i in range(10):

    colors.append(create_rgb(i))

no_markers = 10

current_marker = 1
```

```
marks_updated = False

def mouse_callback(event, x, y, flags, param):

    global marks_updated

    if event == cv2.EVENT_LBUTTONDOWN:

        cv2.circle(marker_image, (x, y), 5, (current_marker), -1)

        cv2.circle(road_copy, (x, y), 5, colors[current_marker], -1)

    marks_updated = True

cv2.namedWindow('Road Image')

cv2.setMouseCallback('Road Image', mouse_callback)

while True:

    cv2.imshow('WaterShed Segments', segments)

    cv2.imshow('Road Image', road_copy)

    k = cv2.waitKey(1)

    if k == 27:

        break

    elif k == ord('c'):

        road_copy = road.copy()

        marker_image = np.zeros(road.shape[0:2], dtype=np.int32)

        segments = np.zeros(road.shape, dtype=np.uint8)

    elif k > 0 and chr(k).isdigit():

        current_marker = int(chr(k))
```

```

current_marker = int(chr(k))

if marks_updated:

    marker_image_copy = marker_image.copy()

    cv2.watershed(road, marker_image_copy)

    segments = np.zeros(road.shape, dtype=np.uint8)

for color_ind in range(no_markers):

    segments[marker_image_copy == (color_ind)] =
colors[color_ind]

marks_updated = False

cv2.destroyAllWindows()

```

### Resultado:

Este código abre duas janelas. Um com a imagem original e outro preto. Clicar na imagem original criará pequenos círculos nela e os contornos serão exibidos na imagem preta. (Pressione os números de 0 a 9 para alterar as cores e produzir um gráfico de contorno diferente.)

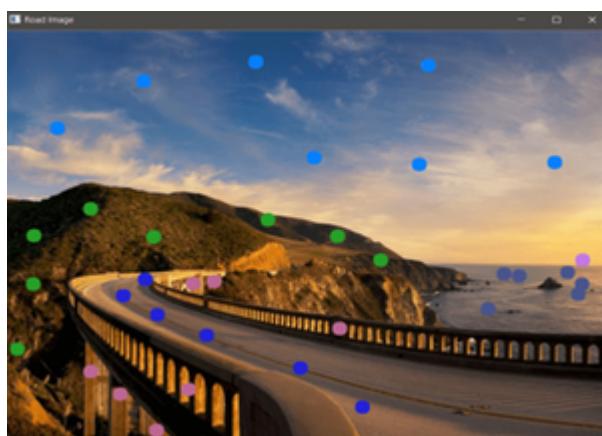
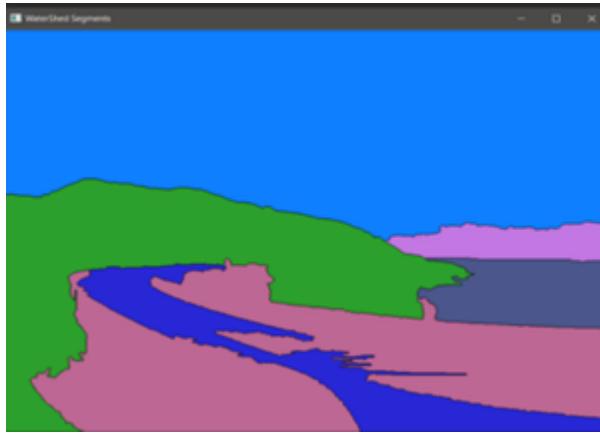


Imagen original



Matplotlib

<https://matplotlib.org/>

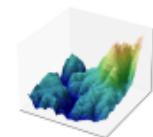
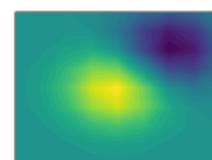
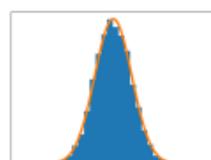
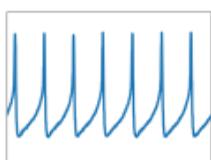


[Installation](#)   [Documentation](#)   [Examples](#)   [Tutorials](#)   [Contributing](#)

[home](#) | [contents](#) » Matplotlib: Python plotting

## Matplotlib: Visualization with Python

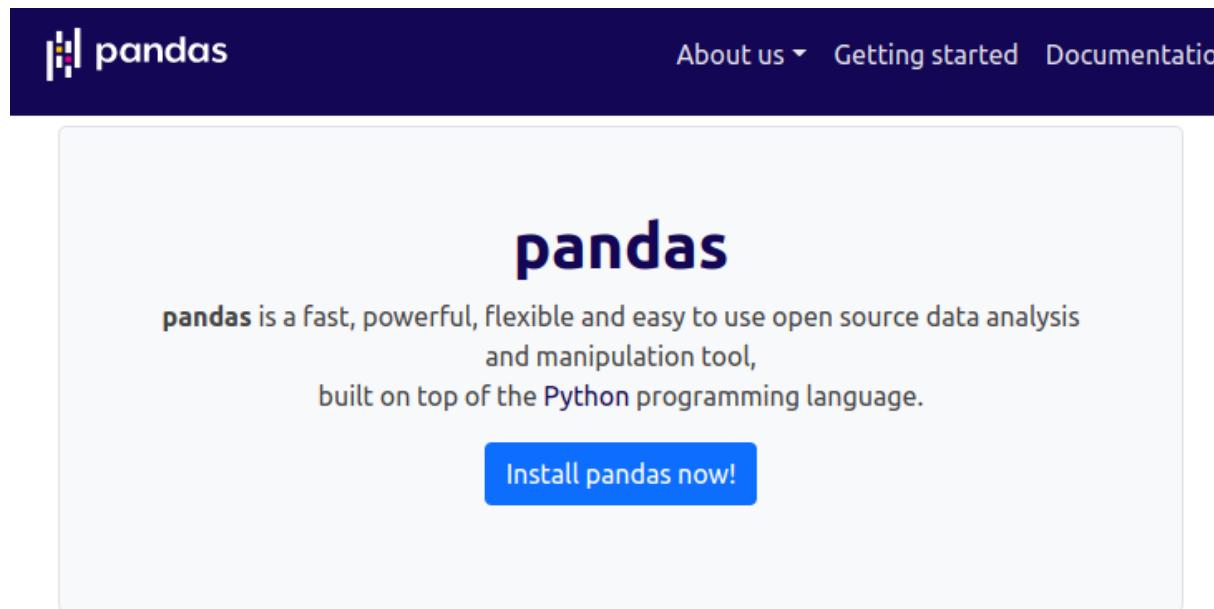
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.



Matplotlib makes easy things easy and hard things possible.

## Pandas

<https://pandas.pydata.org/>



The screenshot shows the official pandas website. At the top, there's a dark blue header bar with the pandas logo on the left and navigation links for "About us", "Getting started", and "Documentation" on the right. Below the header is a large white main content area. In the center of this area is a large blue button with the text "Install pandas now!". Above the button, there's a brief introduction to pandas: "pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language." To the left of the main content area, there's a sidebar with sections for "Getting started" and "Documentation". To the right, there's a section for "Community".

### Getting started

- [Install pandas](#)
- [Getting started](#)

### Documentation

- [User guide](#)
- [API reference](#)
- [Contributing to pandas](#)
- [Release notes](#)

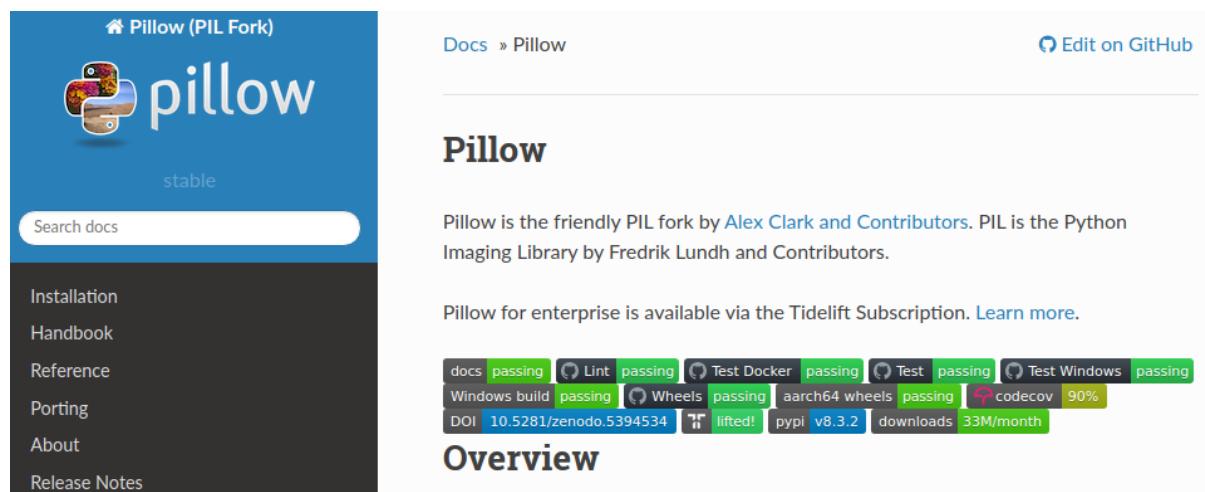
### Community

- [About pandas](#)
- [Ask a question](#)
- [Ecosystem](#)

Utilizada para fazer leitura de bases de dados, como arquivos .CSV, por exemplo.

## PIL

<https://pillow.readthedocs.io/en/stable/>



The screenshot shows the Pillow documentation website. At the top, there's a header with the Pillow logo (a colorful cross icon) and the word "pillow". Below the header, there's a search bar and a navigation menu with links to "Installation", "Handbook", "Reference", "Porting", "About", and "Release Notes". The main content area has a title "Pillow" and a paragraph about the project: "Pillow is the friendly PIL fork by [Alex Clark and Contributors](#). PIL is the Python Imaging Library by Fredrik Lundh and Contributors." It also mentions "Pillow for enterprise is available via the Tidelift Subscription. [Learn more](#)". Below this, there's a row of green status badges for various checks like "docs passing", "Windows build passing", etc. At the bottom, there's a section titled "Overview".

## Resize com PIL

//OS and Glob Modules for the showing absolute path of our Python file so that we can use shorter file names.

```
from PIL import Image
import os
import PIL
import glob

image = Image.open('1.jpeg')
print(image.size)
OUTPUT>>>
(2400, 1500)

resized_image = image.resize((500,500))
print(resized_image.size)
OUTPUT>>>
(500, 500)
```

## Detecção de borda com Pillow

A detecção de bordas é uma disciplina de processamento de imagem que incorpora métodos matemáticos para encontrar bordas em uma imagem digital. A detecção de bordas funciona internamente executando um filtro / kernel sobre uma imagem digital, que detecta descontinuidades nas regiões da imagem, como mudanças bruscas no brilho / valor de intensidade dos pixels.

Existem duas formas de detecção de borda:

- Detecção de borda baseada em pesquisa (derivada de primeira ordem)
- Detecção de borda baseada em cruzamento zero (derivada de segunda ordem)

Alguns dos métodos de detecção de borda comumente conhecidos são:

- Operador Laplaciano ou detecção de Borda Laplaciana (derivada de segunda ordem)
- Detector de borda Canny (derivada de primeira ordem)
- Operador Prewitt (derivada de primeira ordem)

- Operador de Sobel (derivada de primeira ordem)

Estaríamos implementando um Operador Laplaciano para incorporar a detecção de Borda em um de nossos exemplos posteriores. Para isso, estaremos utilizando a `pillow` biblioteca. Para instalar a biblioteca, execute o seguinte comando na linha de comando:

```
pip instalar pillow
```

Existem duas maneiras de implementar a detecção de Borda em nossas imagens. No primeiro método, estaríamos usando um método embutido fornecido na biblioteca de travesseiros (`ImageFilter.FIND_EDGES`) para detecção de borda. No segundo, estaríamos criando um filtro Laplaciano usando `PIL.ImageFilter.Kernel()`, e então usariammos esse filtro para detecção de bordas.

LAPLACIAN KERNEL: –

-1	-1	-1
-1	8	-1
-1	-1	-1

*Laplacian Operator*

AMOSTRA DE IMAGEM: –



Método 1:

```
from PIL import Image, ImageFilter

image = Image.open(r"Sample.png")
image = image.convert("L")
image = image.filter(ImageFilter.FIND_EDGES)
image.save(r"Edge_Sample.png")
```

Resultado (Edge\_Sample.png):



Explicação:-

Em primeiro lugar, criamos um objeto de imagem da nossa imagem usando `Image.open()`. Em seguida, convertemos o modo de cor da imagem para escala de cinza, pois a entrada para o operador Laplaciano está no modo de escala de cinza (em geral). Em seguida, passamos a imagem para `Image.filter()` função especificando o `ImageFilter.FIND_EDGES` argumento, que por sua vez executa um kernel de detecção de borda no topo da imagem. A saída da função acima resulta em uma imagem com alterações de alta intensidade (bordas) em tons de branco e o restante da imagem em preto.

Método 2:

```
from PIL import Image, ImageFilter

img = Image.open(r"sample.png")
img = img.convert("L")
final = img.filter(ImageFilter.Kernel((3, 3), (-1, -1, -1, -1, 8,
                                              -1, -1, -1, -1), 1, 0))

final.save("EDGE_sample.png")
```

Saída (EDGE\_sample.png):



Explicação:-

Em primeiro lugar, criamos um objeto de imagem da nossa imagem usando `Image.open()`. Em seguida, convertemos o modo de cor da imagem para escala de cinza, já que a entrada para o operador Laplaciano está no modo de escala de cinza (em geral). Em seguida, passamos a imagem para a `Image.filter()` função especificando nosso operador / Kernel dentro da função como um argumento. O Kernel é especificado usando o `ImageFilter.Kernel((3, 3), (-1, -1, -1, -1, 8, -1, -1, -1, -1), 1, 0)` que cria um Kernel 3X3 (3 pixels de largura e 3 pixels de comprimento) com os valores (-1, -1, -1, -1, 8, -1, -1, -1, -1) (conforme indicado na imagem do kernel Laplaciano). O 1 argumento (após o kernel) representa o valor da escala, que divide o valor final após cada operação do kernel, portanto, definimos esse valor como 1, pois não queremos nenhuma divisão em nosso valor final. O argumento (após o valor de escala) é o deslocamento que é adicionado após a divisão pelo valor de escala. Definimos esse valor como 0, pois não queremos nenhum incremento no valor de intensidade final após a convolução do kernel. A saída da função acima resulta em uma imagem com alterações de alta intensidade (bordas) em tons de branco e o restante da imagem em preto.

Termo aditivo –

Ambos os programas produziram o mesmo resultado. A razão para isso é o fato de que a função embutida `ImageFilter.FIND_EDGE` usa um núcleo / operador Laplaciano de tamanho 3 X 3 internamente. Devido a isso, acabamos com resultados idênticos. O benefício de usar um Kernel em vez de depender de funções embutidas é que podemos definir kernels de acordo com nossas necessidades, que podem / não estar na biblioteca. Por exemplo, podemos criar um kernel para desfoque, nitidez, detecção de borda (usando outros Kernels) etc. Além disso, eu escolhi intencionalmente o Laplaciano para que possamos manter a consistência nos resultados.

Benefícios de usar Laplacian: –

Resultados rápidos e decentes. Outros detectores de borda comuns como Sobel (derivada de primeira ordem) são mais caros na computação, pois exigem encontrar gradientes em duas direções e, em seguida, normalizar os resultados.

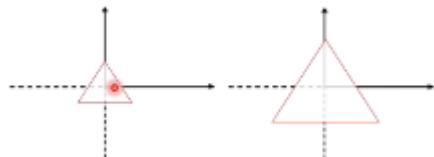
## Pré processamento de imagens

Para os exemplos de pré processamento, seguir este colab:

[https://colab.research.google.com/drive/1b6Hrud8cSHi\\_hmmQILMjcaqFx3b\\_CLjR?usp=sharing](https://colab.research.google.com/drive/1b6Hrud8cSHi_hmmQILMjcaqFx3b_CLjR?usp=sharing)

## Redimensionamento

Fator de escala para ampliar ou reduzir imagem. Uma multiplicação com matrizes.





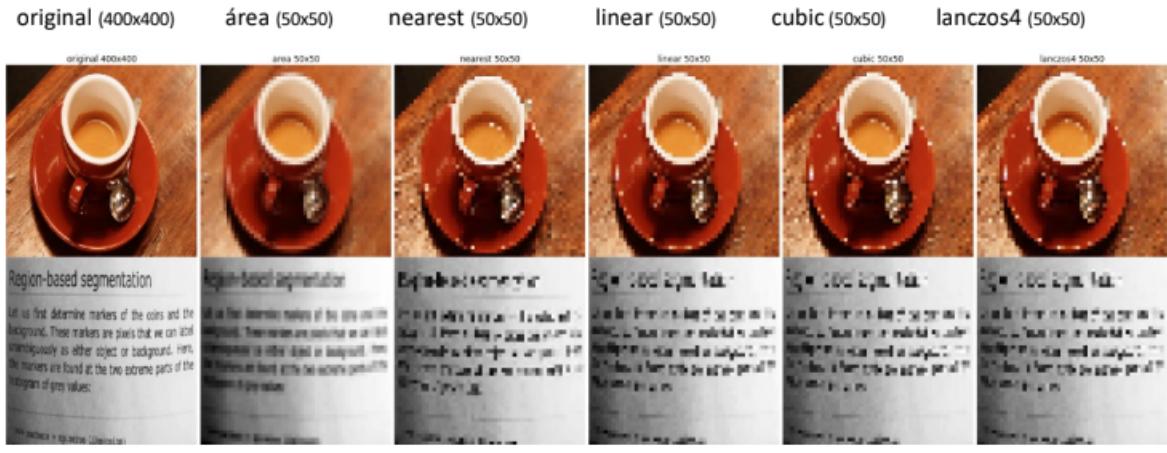
## Redimensionamento no openCV

### Opções OpenCV

- INTER\_NEAREST - interpolação de vizinho mais próximo. É muito usado por ser o mais rápido
- INTER\_LINEAR - interpolação bilinear (é usada por padrão), para aumentar e também pra diminuir imagens.
- INTER\_AREA - relação de área de pixel. Pode ser um método preferido para a redução de imagens porque fornece resultados sem moiré (efeito geralmente indesejado na imagem). Mas quando a imagem é ampliada, é semelhante ao método INTER\_NEAREST
- INTER\_CUBIC - bicúbica (4x4 pixel vizinhos). Possui resultados melhores
- INTER\_LANCZOS4 - interpolação Lanczos (8x8 pixel vizinhos). Dentre esses algoritmos, é o que apresenta resultados com a melhor qualidade



<http://tanbakuchi.com/posts/comparison-of-opencv-interpolation-algorithms/>



<http://tanbakuchi.com/posts/comparison-of-openv-interpolation-algorithms/>

```
maior = cv2.resize(gray, None, fx=1.5, fy=1.5,
interpolation=cv2.INTER_CUBIC)
cv2_imshow(maior)
```

## Segmentação

Separar a imagem, por exemplo, o fundo do objeto. A técnica mais simples para isto é a limiarização

## Limiarização

A limiarização, também chamada de binarização ou THRESHOLDING, é o método mais simples de segmentação de imagens.

- Consiste em separar uma imagem em regiões de interesse e não interesse através da escolha de um ponto de corte (chamado de limiar, ou threshold)

## Limiarização Simples

Escolher manualmente um ponto de corte (limiar) e a partir da cor deste ponto, separar a imagem pelo cálculo da cor deste ponto.

Escolha feita por tentativa e erro.

```
val, thresh = cv2.threshold(gray, 110, 255, cv2.THRESH_BINARY)
cv2_imshow(thresh)
```



### Teste Limiar Simples

**Limiar** é uma técnica em OpenCV, que é a atribuição de valores de pixel em relação ao valor de limite fornecido. No limiar, cada valor de pixel é comparado com o valor do limiar. Se o valor do pixel for menor que o limite, ele é definido como 0, caso contrário, ele é definido como um valor máximo (geralmente 255). Limiar é uma técnica de segmentação muito popular, usada para separar um objeto considerado como primeiro plano de seu plano de fundo. Um limite é um valor que tem duas regiões em cada lado, ou seja, abaixo ou acima do limite.

Na Visão por Computador, essa técnica de limiar é feita em imagens em tons de cinza. Portanto, inicialmente, a imagem deve ser convertida em um espaço de cores em tons de cinza.

Se  $f(x, y) > T$   
então  $f(x, y) = 0$   
outro  
 $f(x, y) = 255$

Onde

$f(x, y)$  = Valor do pixel da coordenada

T = valor limite.

No OpenCV com Python, a função `cv2.threshold` é usada para limite.

**Sintaxe:** `cv2.threshold (source, thresholdValue, maxVal, thresholdingTechnique)`

### **Parâmetros:**

-> **fonte** : array de imagem de entrada (deve estar em escala de cinza).

-> **thresholdValue** : Valor do Limiar abaixo e acima do qual os valores dos pixels serão alterados de acordo.

-> **maxVal** : Valor máximo que pode ser atribuído a um pixel.

-> **thresholdingTechnique** : O tipo de thresholding a ser aplicado.

### Limiar Simples

A técnica básica de Limiar é a Limiar Binária. Para cada pixel, o mesmo valor de limite é aplicado. Se o valor do pixel for menor que o limite, ele é definido como 0, caso contrário, ele é definido como um valor máximo.

As diferentes técnicas de limitação simples são:

- **cv2.THRESH\_BINARY**: Se a intensidade do pixel for maior que o limite definido, valor definido como 255, caso contrário, definido como 0 (preto).
- **cv2.THRESH\_BINARY\_INV**: Caso invertido ou oposto de `cv2.THRESH_BINARY`.
- **cv.THRESH\_TRUNC**: Se o valor da intensidade do pixel for maior do que o limite, ele será truncado até o limite. Os valores de pixel são definidos para serem iguais ao limite. Todos os outros valores permanecem os mesmos.
- **cv.THRESH\_TOZERO**: A intensidade do pixel é definida como 0, para todas as intensidades dos pixels, menor que o valor limite.

- **cv.THRESH\_TOZERO\_INV**: Caso invertido ou oposto de cv2.THRESH\_TOZERO.

Binary

$$dst(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$


---

Inverted Binary

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$$


---

Truncated

$$dst(x, y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$


---

To Zero

$$dst(x, y) = \begin{cases} \text{src}(x, y) & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$


---

To Zero Inverted

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$


---

Abaixo está o código Python explicando diferentes técnicas de limitação simples –

```
import cv2
import numpy as np
image1 = cv2.imread('input1.jpg')

img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO_INV)

cv2.imshow('Binary Threshold', thresh1)
cv2.imshow('Binary Threshold Inverted', thresh2)
cv2.imshow('Truncated Threshold', thresh3)
cv2.imshow('Set to 0', thresh4)
cv2.imshow('Set to 0 Inverted', thresh5)

if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Entrada:

# GeeksforGeeks

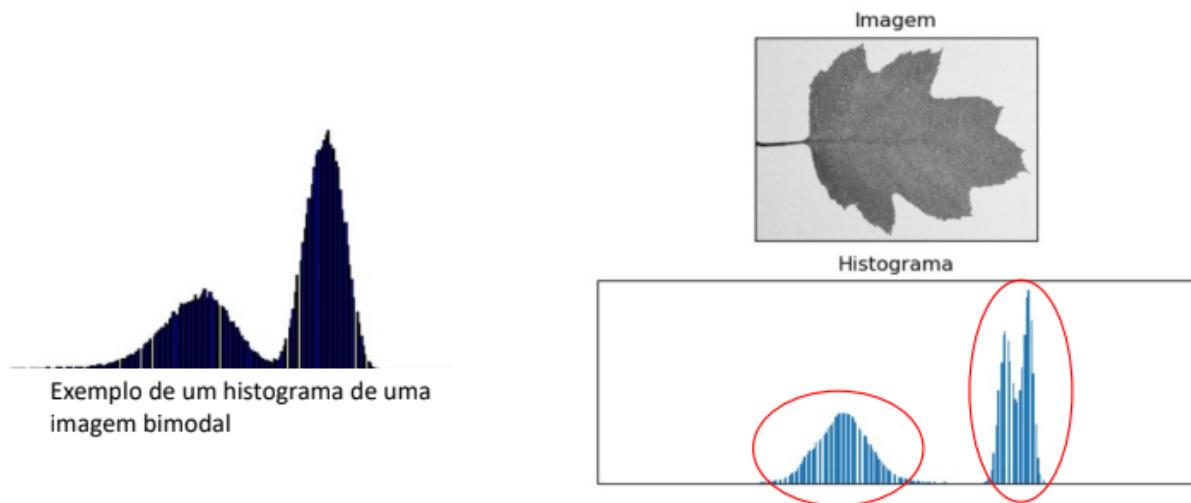
A computer science portal for geeks

Saída:



Método de Otsu

Limiar de forma automática.



```
val, otsu = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY |
cv2.THRESH_OTSU)
cv2_imshow(otsu)
print(val)
```

```
val, otsu = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
cv2_imshow(otsu)
print(val)
```



### Limiarização Adaptativa

Usada para quando existe variação de luz na imagem. Por exemplo, uma foto de um livro.

## Limiarização Adaptativa

```
▶ img = cv2.imread('/content/trecho-livro.jpg')
cv2_imshow(img)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
val, otsu = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
cv2_imshow(otsu)
print(val)
```



A inteligência humana é caracterizada pela capacidade de se adaptar a mudanças. É resultado de gerações de seleção natural sobre indivíduos com capacidade para lidar com as novas circunstâncias. Não devemos, portanto, temer as transformações. Precisamos apenas fazer com que elas operem em nosso benefício.

138.0

Primeira atividade de Limiarização Adaptativa

**Equalização de histograma adaptativo limitada por contraste (CLAHE)** para equalizar imagens. CLAHE é uma variante da *equalização adaptativa do histograma (AHE)* que cuida da super amplificação do contraste. O CLAHE opera em pequenas regiões da imagem, chamadas de blocos, em vez da imagem inteira. Os ladrilhos vizinhos são então combinados usando interpolação bilinear para remover os limites artificiais.

Este algoritmo pode ser aplicado para melhorar o contraste das imagens.

Também podemos aplicar CLAHE a imagens coloridas, onde normalmente é aplicado no canal de luminância e os resultados após equalizar apenas o canal de luminância de uma imagem HSV são muito melhores do que equalizar todos os canais da imagem BGR.

Neste tutorial, vamos aprender como aplicar CLAHE e processar uma determinada imagem de entrada para equalização de histograma.

Parâmetros:

Ao aplicar CLAHE, existem dois parâmetros a serem lembrados:

**clipLimit** – Este parâmetro define o limite para limitação de contraste. O valor padrão é 40.

**tileGridSize** – Isso define o número de ladrilhos na linha e coluna. Por padrão, é  $8 \times 8$ . É usado enquanto a imagem é dividida em blocos para a aplicação de CLAHE.

Código para acima:

```
import cv2
import numpy as np
image = cv2.imread("image.jpg")
image = cv2.resize(image, (500, 600))
image_bw = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
clahe = cv2.createCLAHE(clipLimit = 5)
final_img = clahe.apply(image_bw) + 30
_, ordinary_img = cv2.threshold(image_bw, 155, 255, cv2.THRESH_BINARY)
cv2.imshow("ordinary threshold", ordinary_img)
cv2.imshow("CLAHE image", final_img)
```

Imagen de entrada:

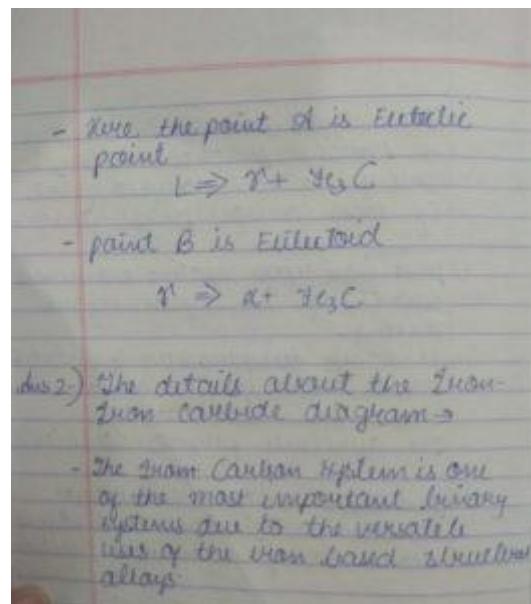
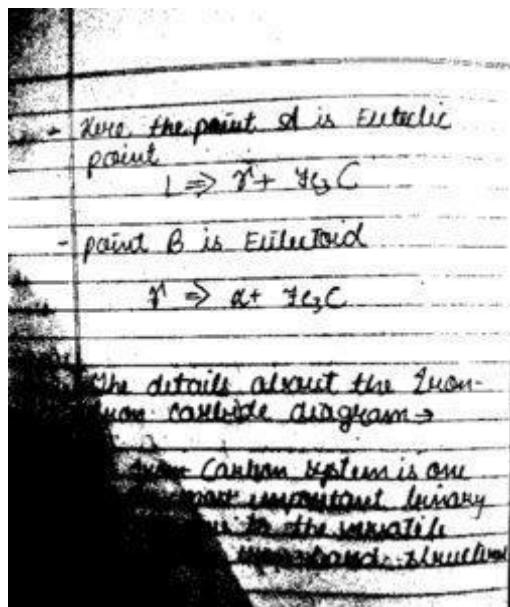
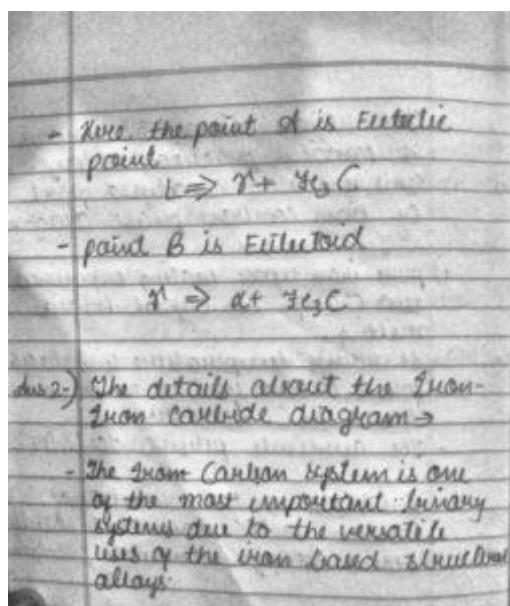


imagem de entrada

Resultado:



### Limiar Ordinário



### CLAHE Aplicado

### Limiarização Adaptativa Gaussiana

O limiar é calculado para pequenas regiões da imagem, sendo obtidos diferentes limiares para pequenas regiões da imagem (utilizada a média)

- Gaussiana: também utiliza o desvio padrão (considerada a variação nos tons da imagem)
  - utiliza convolução

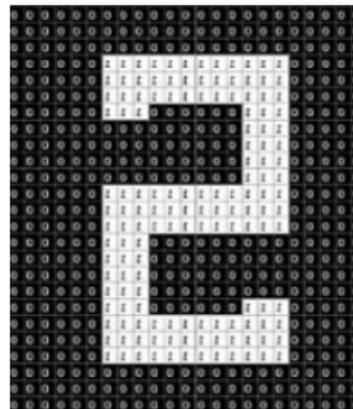
```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
adapt_media_gauss = cv2.adaptiveThreshold(gray, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 9)
cv2_imshow(adapt_media_gauss)
```

Inversão de cor, pois o reconhecimento quando o fundo estiver escuro é melhor.

```
img = cv2.imread('/content/img-process.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
invert = 255 - gray
cv2_imshow(invert)
```

## Operações Morfológicas

### Dilatação e Erosão



0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0
0	1	1	1	1	0	0	0
0	1	1	1	1	0	0	0
0	1	1	1	1	0	0	0
0	0	0	0	0	0	0	0

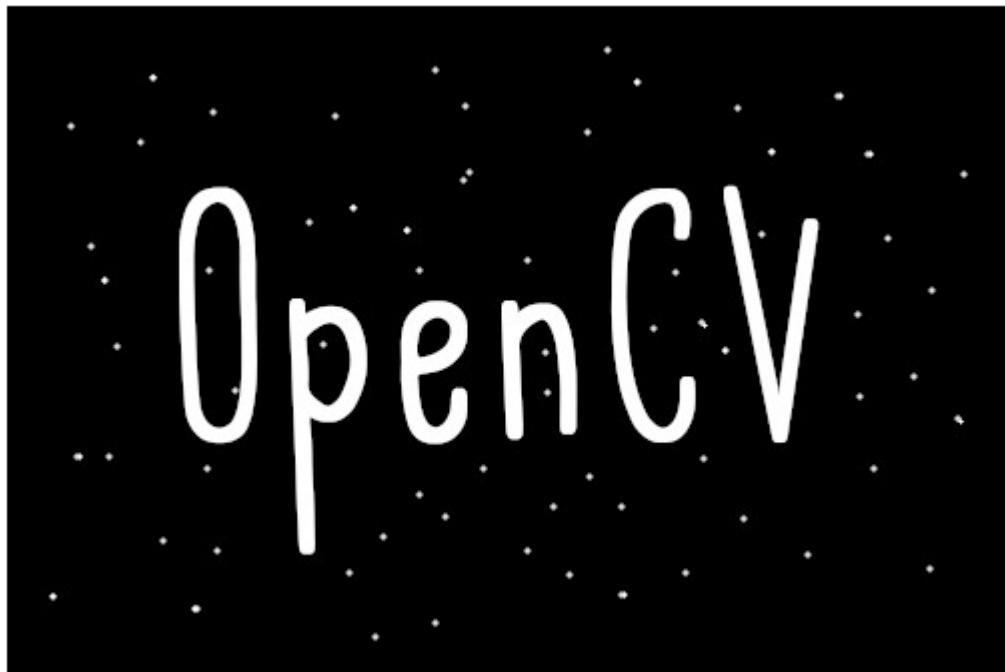
EROSÃO  
DILATAÇÃO

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Erosão

```
img = cv2.imread('/content/texto-opencv.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```



```
erosao = cv2.erode(gray, np.ones((3, 3), np.uint8))
cv2_imshow(erosao)
```

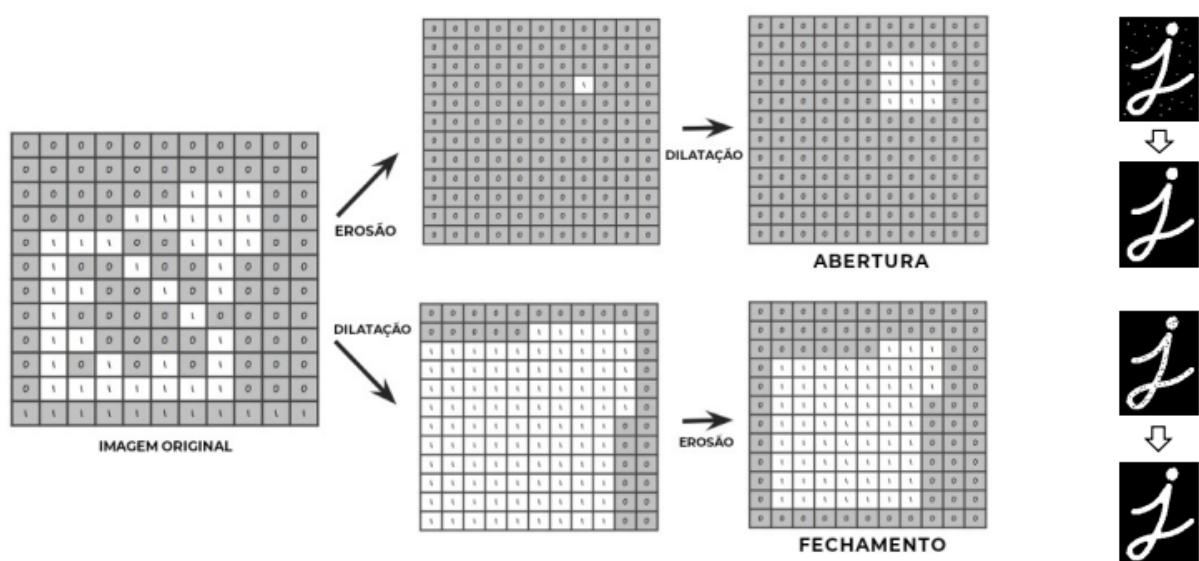


## Dilatação

```
dilatacao = cv2.dilate(gray, np.ones((3,3), np.uint8))  
cv2_imshow(dilatacao)
```

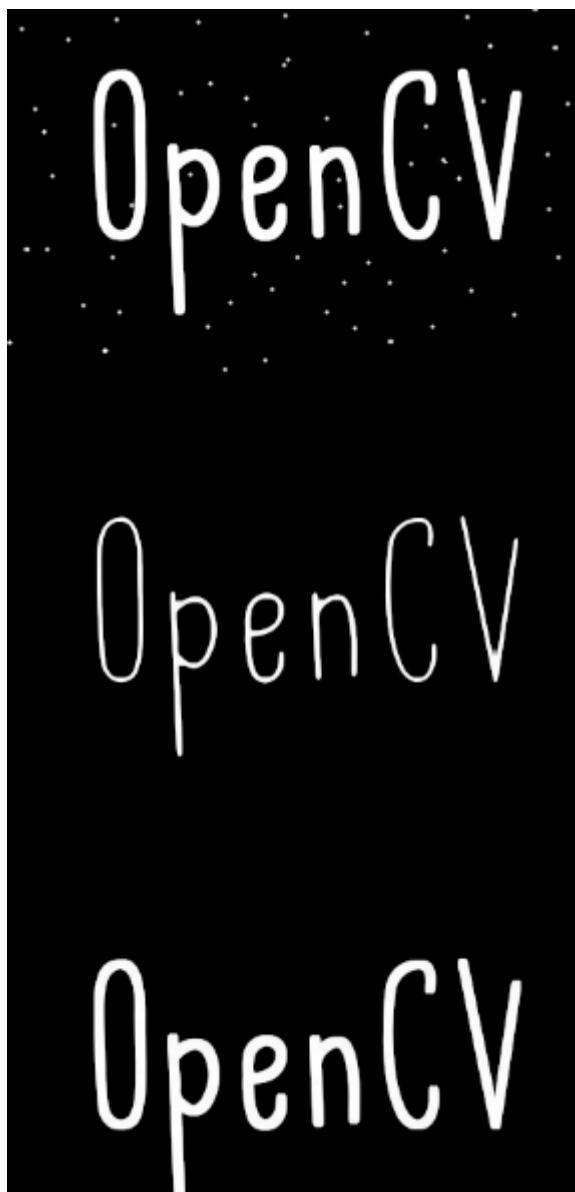


## Abertura e Fechamento



## Abertura

```
erosao = cv2.erode(gray, np.ones((5, 5), np.uint8))
abertura = cv2.dilate(erosao, np.ones((5,5), np.uint8))
cv2_imshow(gray)
cv2_imshow(erosao)
cv2_imshow(abertura)
```



## Fechamento

```
img = cv2.imread('texto-opencv2.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```



Para resolver:

```
dilatacao = cv2.dilate(gray, np.ones((5,5)))
fechamento = cv2.erode(dilatacao, np.ones((5,5)))
cv2_imshow(gray)
cv2_imshow(dilatacao)
cv2_imshow(fechamento)
```



OpenCV

OpenCV

OpenCV

## Operações morfológicas no processamento de imagens (gradiente)

É usado para gerar o contorno da imagem. Existem dois tipos de gradientes, gradientes internos e externos. O gradiente interno aprimora os limites internos dos objetos mais brilhantes do que o fundo e os limites externos dos objetos mais escuros do que o fundo. Para imagens binárias, o gradiente interno gera uma máscara dos limites internos dos objetos de imagem em primeiro plano.

**Sintaxe:** `cv2.morphologyEx (imagem, cv2.MORPH_GRADIENT, kernel)`

**Parâmetros:**

-> **imagem**: array de imagens de entrada.

-> **cv2.MORPH\_GRADIENT**: Aplicando a operação de Gradiente Morfológico.

-> **kernel**: elemento estruturante.

Abaixo está o código Python que explica a operação morfológica do gradiente -

```
import cv2
import numpy as np
screenRead = cv2.VideoCapture(0)
while(1):

    _, image = screenRead.read()

    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    blue1 = np.array([110, 50, 50])
    blue2 = np.array([130, 255, 255])

    mask = cv2.inRange(hsv, blue1, blue2)

    res = cv2.bitwise_and(image, image, mask = mask)

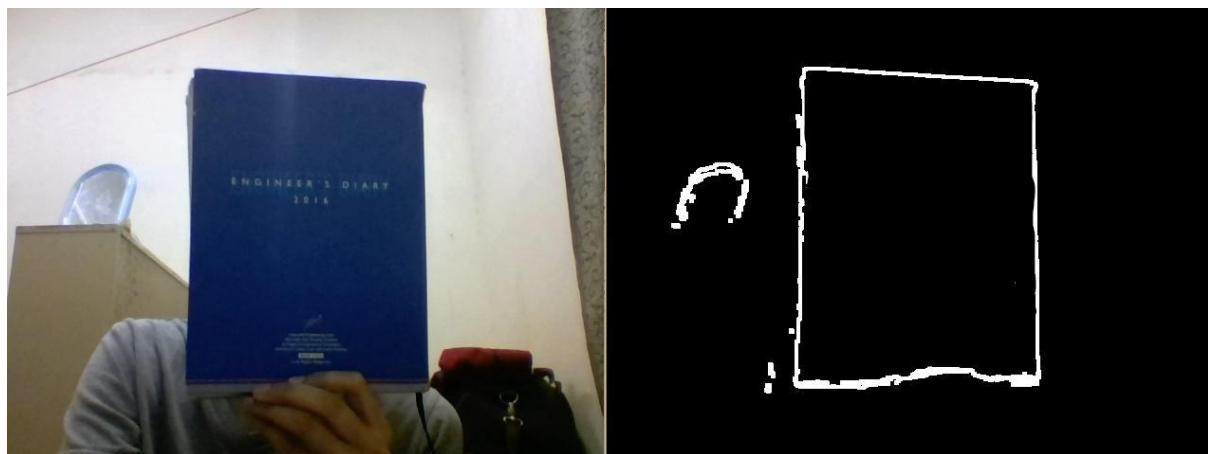
    kernel = np.ones((5, 5), np.uint8)

    gradient = cv2.morphologyEx(mask, cv2.MORPH_GRADIENT, kernel)

    cv2.imshow('Gradient', gradient)

    if cv2.waitKey(1) & 0xFF == ord('a'):
        break
cv2.destroyAllWindows()
screenRead.release()
```

**Resultado:** o quadro da imagem de saída mostra o contorno gerado sobre o livro azul e o objeto azul no canto superior esquerdo.



## Remoção de Ruído

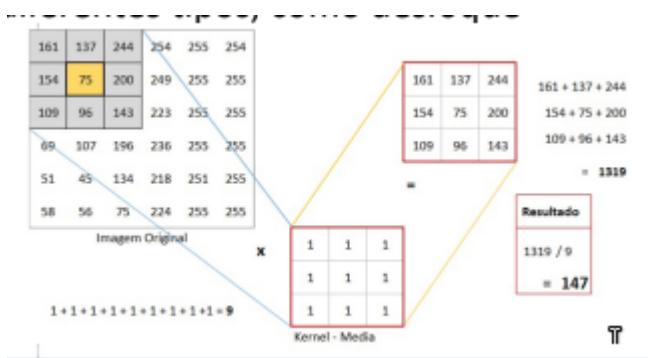
Remover informações que podem atrapalhar.

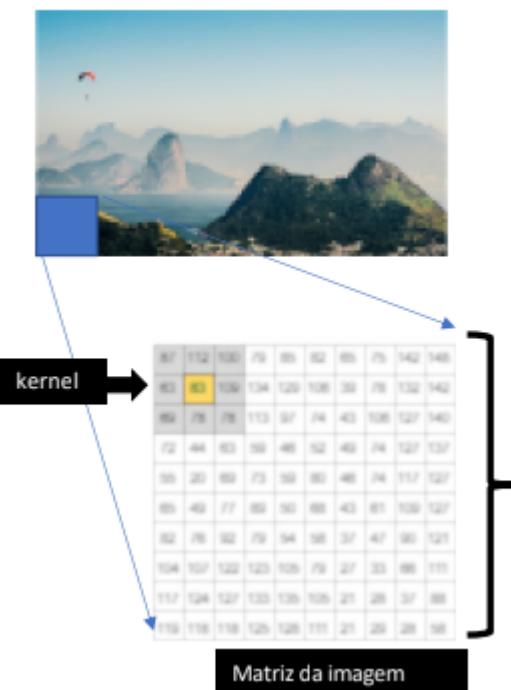
Um filtro utilizado com a convolução de um kernel e gerando uma nova matriz.

## Convolução

Convolução é uma operação matemática realizada em duas matrizes, que produz uma terceira matriz que é o resultado da operação

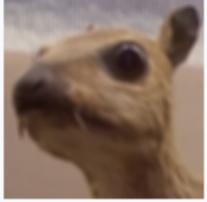
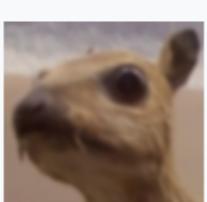
- A matriz primária é a imagem a ser tratada, e o tratamento dessa matriz da imagem original se dá por outra matriz chamada “núcleo” (kernel), ou máscara
- Dependendo dos valores do kernel é possível obter filtros de diferentes tipos, como desfoco e nitidez





Cada filtro pode ter um diferente objetivo  
[https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))

Operation	Kernel $\omega$	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

<b>Sharpen</b>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
<b>Box blur (normalized)</b>	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
<b>Gaussian blur 3 × 3 (approximation)</b>	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
<b>Gaussian blur 5 × 5 (approximation)</b>	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

## Remoção de Ruído com Desfoco

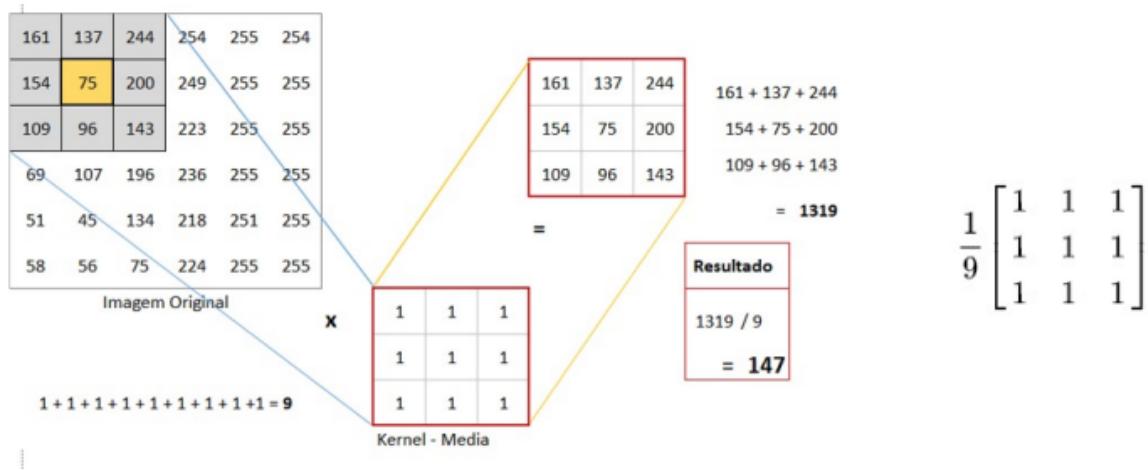
Os filtros espaciais são implementados através de kernels (máscaras/matrizes), com dimensões ímpares. Os filtros podem ser:

- Passa-baixa (Low-pass) – usados para desfoco
- Passa-alta (High-pass) – usados para aumentar nitidez

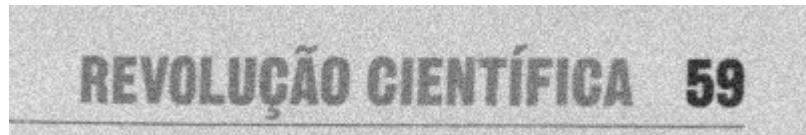


## Desfoque com Média

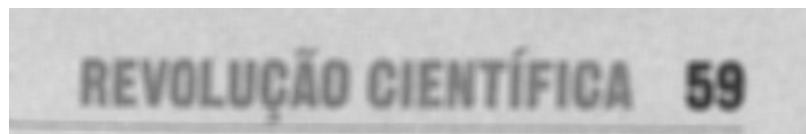
O desfoque de cada pixel é feito pela média dos seus vizinhos.



```
img = cv2.imread('teste_ruido.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow(gray)
```

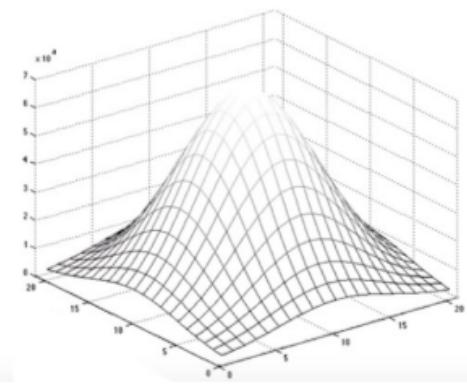


```
desfoco_media = cv2.blur(gray, (5,5))
cv2.imshow(desfoco_media)
```

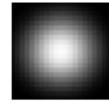


## Desfoque Gaussiano

O mesmo método anterior, porém utiliza uma janela Gaussiana, e como resultado, o desfoque no centro é maior do que nas bordas. Conforme se afasta do centro, menor é a distorção, preservando os cantos da imagem

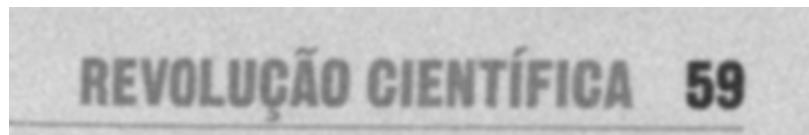


$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



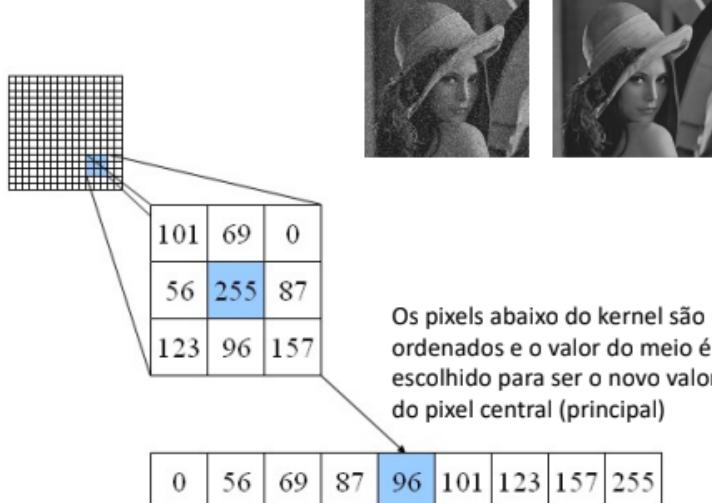
Representação gráfica de um filtro gaussiano 21x21

```
desfoque_gaussiano = cv2.GaussianBlur(gray, (5, 5), 0)
cv2_imshow(desfoque_gaussiano)
```



## Desfoco com Mediana

No desfoco por mediana, o valor dos pixels são ordenados e é escolhido o pixel do meio se for ímpar, ou se a quantidade for par, faz a média da soma dos dois do meio.



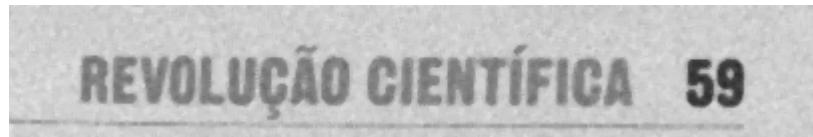
2, 2, 3, **7**, 8, 9, 9

Mediana = **7**

1, 4, 4, **5**, 6, 7, 7, 7

Mediana =  $(5+6) \div 2$   
= 5.5

```
desfoque_médiana = cv2.medianBlur(gray, 3)
cv2_imshow(desfoque_médiana)
```



## Filtro Bilateral

Desfoca a imagem, para preservar as bordas. Utiliza o filtro gaussiano mas utiliza apenas os pixels próximos ao pixel central do kernel.

```
desfoque_bilateral = cv2.bilateralFilter(gray, 15, 55, 45)
cv2_imshow(desfoque_bilateral)
```

