

OpenCV-Python - Guia rápido

OpenCV Python - Visão geral

OpenCV significa **Open Source Computer Vision** e é uma biblioteca de funções úteis na programação de aplicativos de visão computacional em tempo real. O termo Visão Computacional é utilizado para a realização da análise de imagens e vídeos digitais por meio de um programa de computador. A visão computacional é um constituinte importante de disciplinas modernas, como inteligência artificial e aprendizado de máquina.

Originalmente desenvolvido pela Intel, OpenCV é uma biblioteca multiplataforma escrita em C++, mas também possui C Interface Wrappers para OpenCV que foram desenvolvidos para muitas outras linguagens de programação, como Java e Python. Neste tutorial, será descrita a funcionalidade da biblioteca Python do OpenCV.

OpenCV-Python

OpenCV-Python é um wrapper Python em torno da implementação C++ da biblioteca OpenCV. Faz uso da biblioteca NumPy para operações numéricas e é uma ferramenta de prototipagem rápida para problemas de visão computacional.

OpenCV-Python é uma biblioteca multiplataforma, disponível para uso em todas as plataformas de sistema operacional (SO), incluindo Windows, Linux, MacOS e Android. OpenCV também oferece suporte à aceleração da Unidade de Processamento Gráfico (GPU).

Este tutorial foi desenvolvido para estudantes e profissionais de ciência da computação que desejam adquirir experiência na área de aplicações de visão computacional. O conhecimento prévio da biblioteca Python e NumPy é essencial para compreender a funcionalidade do OpenCV-Python.

OpenCV Python - Configuração do ambiente

Na maioria dos casos, usar pip deve ser suficiente para instalar OpenCV-Python em seu computador.

O comando usado para instalar o pip é o seguinte -

```
pip install opencv-python
```

É recomendável realizar esta instalação em um novo ambiente virtual. A versão atual do OpenCV-Python é 4.5.1.48 e pode ser verificada seguindo o comando -



```
>>> import cv2  
>>> cv2.__version__  
'4.5.1'
```

Como o OpenCV-Python depende do NumPy, ele também é instalado automaticamente. Opcionalmente, você pode instalar o Matplotlib para renderizar determinadas saídas gráficas.

No Fedora, você pode instalar o OpenCV-Python pelo comando mencionado abaixo -

```
$ yum install numpy opencv*
```

OpenCV-Python também pode ser instalado compilando a partir de sua fonte disponível em <http://sourceforge.net> Siga as instruções de instalação fornecidas para o mesmo.

OpenCV Python – Lendo uma imagem

O pacote **CV2** (nome da biblioteca OpenCV-Python) fornece a função **imread()** para ler uma imagem.

O comando para ler uma imagem é o seguinte -

```
img=cv2.imread(filename, flags)
```

Os parâmetros dos sinalizadores são a enumeração das seguintes constantes -

- cv2.IMREAD_COLOR (1) - Carrega uma imagem colorida.
- cv2.IMREAD_GRAYSCALE (0) - Carrega imagem em modo tons de cinza
- cv2.IMREAD_UNCHANGED (-1) - Carrega a imagem como tal, incluindo canal alfa

A função retornará um objeto de imagem, que pode ser renderizado usando a função **imshow()**. O comando para usar a função **imshow()** é fornecido abaixo -

```
cv2.imshow(window-name, image)
```

A imagem é exibida em uma janela nomeada. Uma nova janela é criada com o sinalizador **AUTOSIZE** definido. O **WaitKey()** é uma função de ligação do teclado. Seu argumento é o tempo em milissegundos.

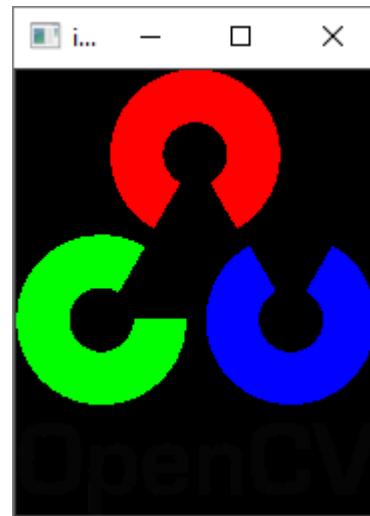
A função espera por milissegundos especificados e mantém a janela em exibição até que uma tecla seja pressionada. Finalmente, podemos destruir todas as janelas assim criadas.

A função espera por milissegundos especificados e mantém a janela em exibição até que uma tecla seja pressionada. Finalmente, podemos destruir todas as janelas assim criadas.

O programa para exibir o logotipo OpenCV é o seguinte -

```
import numpy as np
import cv2
# Load a color image in grayscale
img = cv2.imread('OpenCV_Logo.png',1)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

O programa acima exibe o logotipo OpenCV da seguinte forma -



OpenCV Python - Escreva uma imagem

O pacote CV2 possui a função **imwrite()** que salva um objeto de imagem em um arquivo especificado.

O comando para salvar uma imagem com a ajuda da função imwrite() é o seguinte -

```
cv2.imwrite(filename, img)
```

O formato da imagem é decidido automaticamente pelo OpenCV a partir da extensão do arquivo. OpenCV suporta tipos de arquivo de imagem *.bmp, *.dib, *.jpeg, *.jpg, *.png, *.webp, *.sr, *.tiff, *.tif etc.

Exemplo

O programa a seguir carrega a imagem do logotipo OpenCV e salva sua versão em escala de cinza quando a tecla 's' é pressionada -

```
import numpy as np
import cv2
# Load an color image in grayscale
```

```
img = cv2.imread('OpenCV_Logo.png',0)
cv2.imshow('image',img)
key=cv2.waitKey(0)
if key==ord('s'):
    cv2.imwrite("opencv_logo_GS.png", img)
cv2.destroyAllWindows()
```

Saída



OpenCV Python - Usando Matplotlib

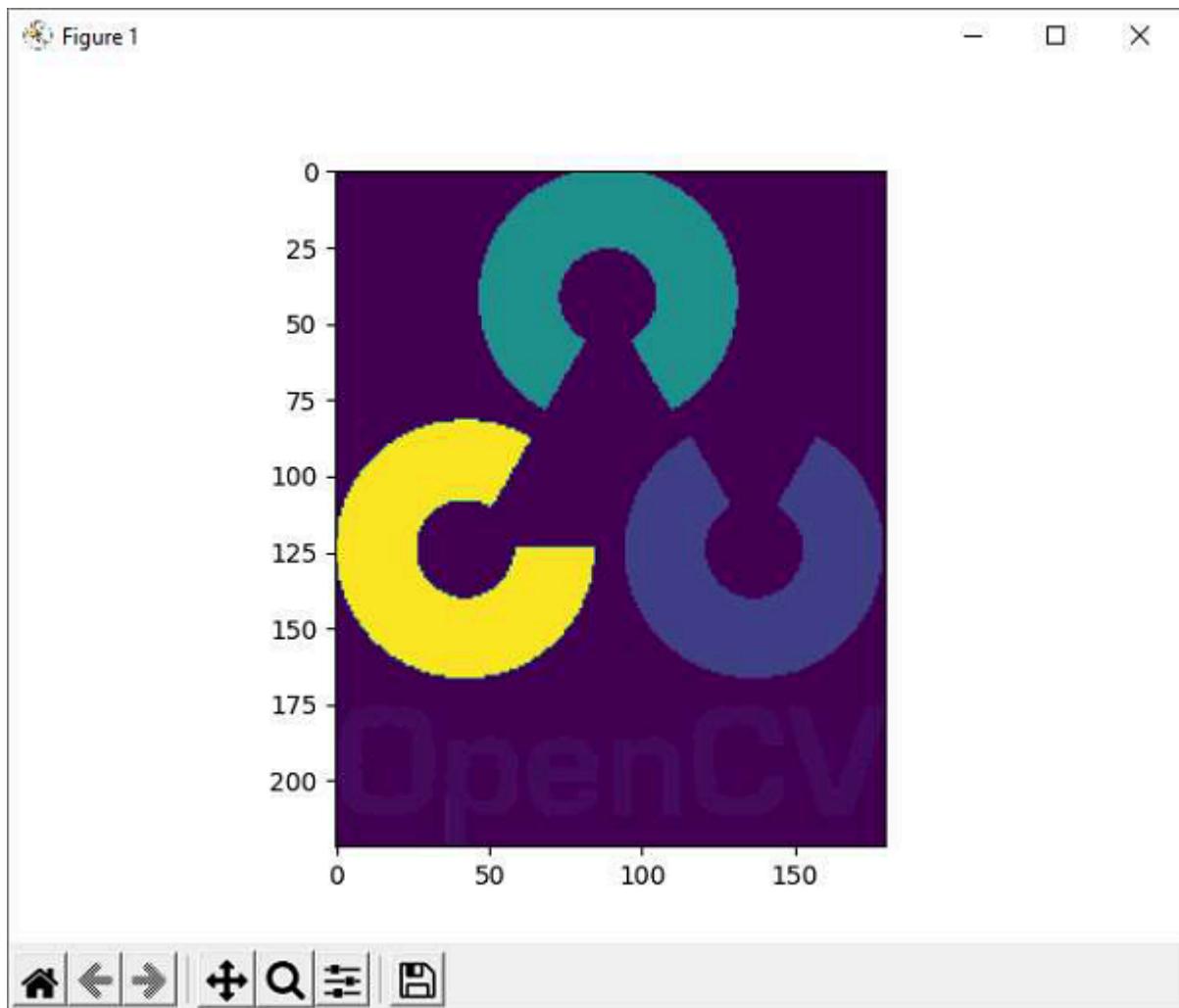
O Matplotlib do Python é uma biblioteca de plotagem poderosa com uma enorme coleção de funções de plotagem para uma variedade de tipos de plotagem. Ele também possui a função imshow() para renderizar uma imagem. Oferece recursos adicionais, como zoom, salvamento, etc.

Exemplo

Certifique-se de que o Matplotlib esteja instalado no ambiente de trabalho atual antes de executar o programa a seguir.

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
# Load an color image in grayscale
img = cv2.imread('OpenCV_Logo.png',0)
plt.imshow(img)
plt.show()
```

Saída



OpenCV Python - Propriedades da imagem

OpenCV lê os dados da imagem em um array NumPy. O método **shape()** deste objeto ndarray revela propriedades da imagem, como dimensões e canais.

O comando para usar o método `shape()` é o seguinte -

```
>>> img = cv.imread("OpenCV_Logo.png", 1)
>>> img.shape
(222, 180, 3)
```

No comando acima -

- Os dois primeiros itens `shape[0]` e `shape[1]` representam a largura e a altura da imagem.
- `Shape[2]` representa vários canais.
- 3 indica que a imagem possui canais Vermelho Verde Azul (RGB).

Da mesma forma, a propriedade size retorna o tamanho da imagem. O comando para o tamanho de uma imagem é o seguinte -

```
>>> img.size  
119880
```

Cada elemento no ndarray representa um pixel da imagem.

Podemos acessar e manipular o valor de qualquer pixel, com a ajuda do comando mencionado abaixo.

```
>>> p=img[50,50]  
>>> p  
array([ 1, 1, 255], dtype=uint8)
```

DE ANÚNCIOS

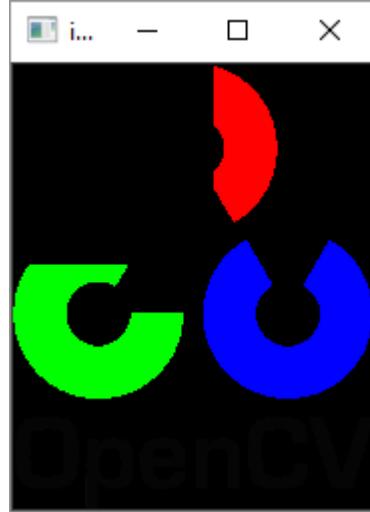
Exemplo

O código a seguir altera o valor da cor dos primeiros 100X100 pixels para preto. A função **imshow()** pode verificar o resultado.

```
>>> for i in range(100):  
    for j in range(100):  
        img[i,j]=[0,0,0]
```

DE ANÚNCIOS

Saída



Os canais de imagem podem ser divididos em planos individuais usando a função **split()** . Os canais podem ser mesclados usando a função **merge()** .

A função **split()** retorna um array multicanal.

Podemos usar o seguinte comando para dividir os canais de imagem -

```
>>> img = cv.imread("OpenCV_Logo.png", 1)
>>> b,g,r = cv.split(img)
```

Agora você pode realizar manipulações em cada plano.

Suponha que definimos todos os pixels no canal azul como 0, o código será o seguinte -

```
>>> img[:, :, 0]=0
>>> cv.imshow("image", img)
```

A imagem resultante será mostrada abaixo -



OpenCV Python - operações bit a bit

As operações bit a bit são usadas na manipulação de imagens e para extrair as partes essenciais da imagem.

Os seguintes operadores são implementados no OpenCV -

- bit a bit_e
- bit a bit_ou
- bit a bit_xor
- bit a bit_não

Exemplo 1

Para demonstrar o uso desses operadores, são tiradas duas imagens com círculos preenchidos e vazios.

O programa a seguir demonstra o uso de operadores bit a bit em OpenCV-Python -

```
import cv2
import numpy as np

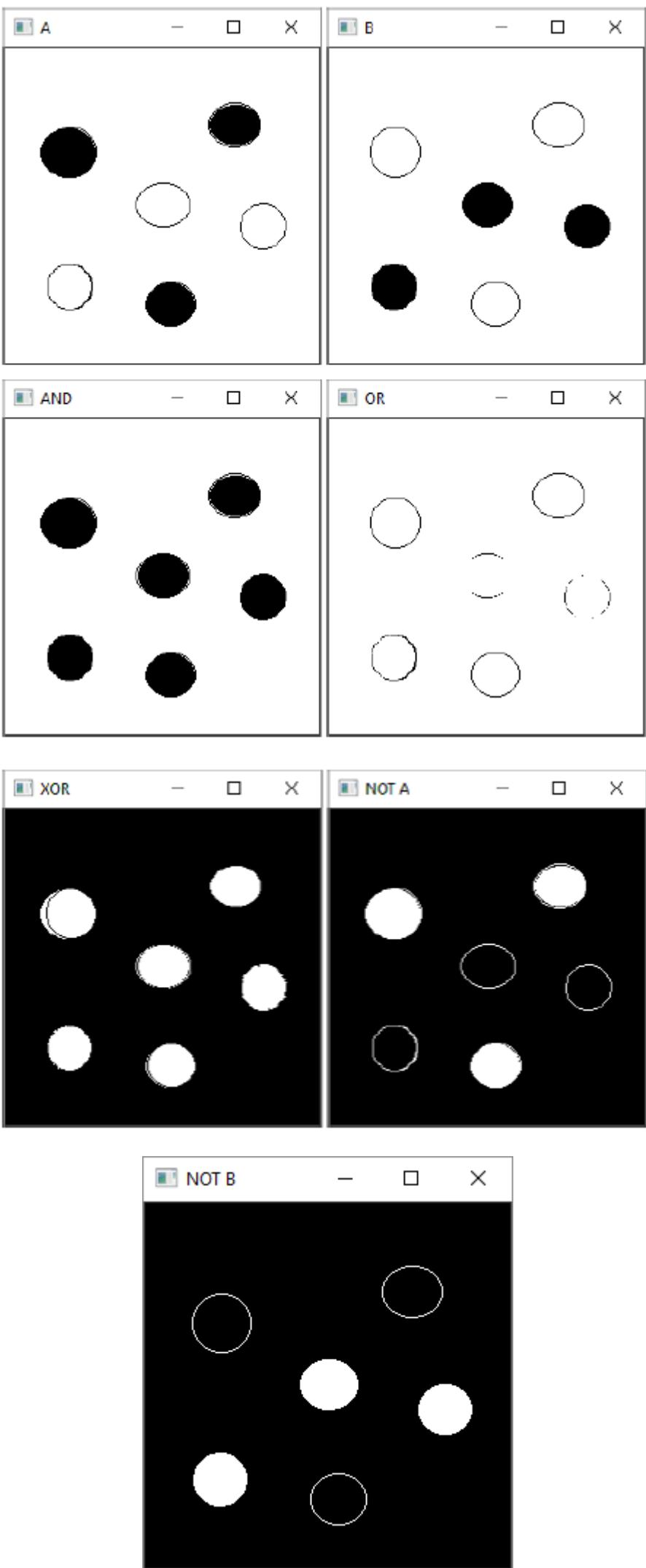
img1 = cv2.imread('a.png')
img2 = cv2.imread('b.png')

dest1 = cv2.bitwise_and(img2, img1, mask = None)
dest2 = cv2.bitwise_or(img2, img1, mask = None)
dest3 = cv2.bitwise_xor(img1, img2, mask = None)

cv2.imshow('A', img1)
cv2.imshow('B', img2)
cv2.imshow('AND', dest1)
cv2.imshow('OR', dest2)
cv2.imshow('XOR', dest3)
cv2.imshow('NOT A', cv2.bitwise_not(img1))
cv2.imshow('NOT B', cv2.bitwise_not(img2))

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

Saída



Exemplo 2

Em outro exemplo envolvendo operações bit a bit, o logotipo opencv é sobreposto a outra imagem. Aqui, obtemos um array de máscaras que chama a função limiar () no logotipo e realizamos a operação AND entre eles.

Da mesma forma, pela operação NOT, obtemos uma máscara inversa. Além disso, obtemos AND com a imagem de fundo.

A seguir está o programa que determina o uso de operações bit a bit -

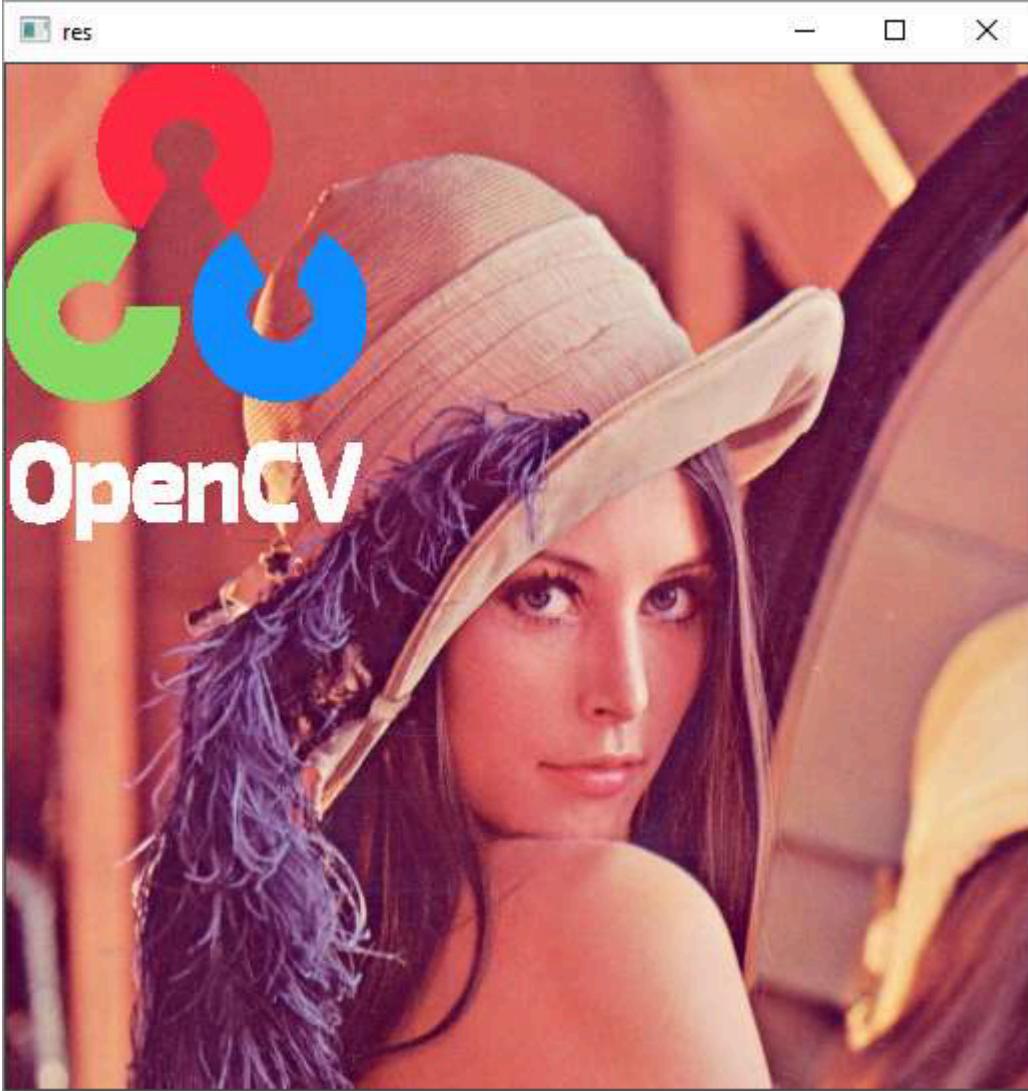
```
import cv2 as cv
import numpy as np

img1 = cv.imread('lena.jpg')
img2 = cv.imread('whitelogo.png')
rows,cols,channels = img2.shape
roi = img1[0:rows, 0:cols]
img2gray = cv.cvtColor(img2, cv.COLOR_BGR2GRAY)
ret, mask = cv.threshold(img2gray, 10, 255, cv.THRESH_BINARY)
mask_inv = cv.bitwise_not(mask)
# Now black-out the area of Logo
img1_bg = cv.bitwise_and(roi,roi,mask = mask_inv)

# Take only region of logo from logo image.
img2_fg = cv.bitwise_and(img2,img2,mask = mask)
# Put logo in ROI
dst = cv.add(img2_fg, img1_bg)
img1[0:rows, 0:cols ] = dst
cv.imshow(Result,img1)
cv.waitKey(0)
cv.destroyAllWindows()
```

Saída

As imagens mascaradas fornecem o seguinte resultado -



OpenCV Python - Desenhar formas e texto

Neste capítulo, aprenderemos como desenhar formas e texto em imagens com a ajuda do OpenCV-Python. Vamos começar entendendo como desenhar formas em imagens.

Desenhe formas em imagens

Precisamos entender as funções necessárias no OpenCV-Python, que nos ajudam a desenhar formas nas imagens.

Funções

O pacote OpenCV-Python (referido como cv2) contém as seguintes funções para desenhar as respectivas formas.

Função	Descrição	Comando
cv2.line()	Desenha um segmento de linha conectando dois pontos.	cv2.line(img, pt1, pt2, cor, espessura)
cv2.círculo()	Desenha um círculo de determinado raio em determinado ponto como	cv2.circle(img, centro, raio, cor, espessura)

	centro	
cv2.retângulo	Desenha um retângulo com pontos dados como canto superior esquerdo e canto inferior direito.	cv2.rectangle(img, pt1, pt2, cor, espessura)
cv2.ellipse()	Desenha um arco elíptico simples ou grosso ou preenche um setor de elipse.	cv2.ellipse(img, centro, eixos, ângulo, startAngle, endAngle, cor, espessura)

Parâmetros

Os parâmetros comuns para as funções acima são os seguintes -

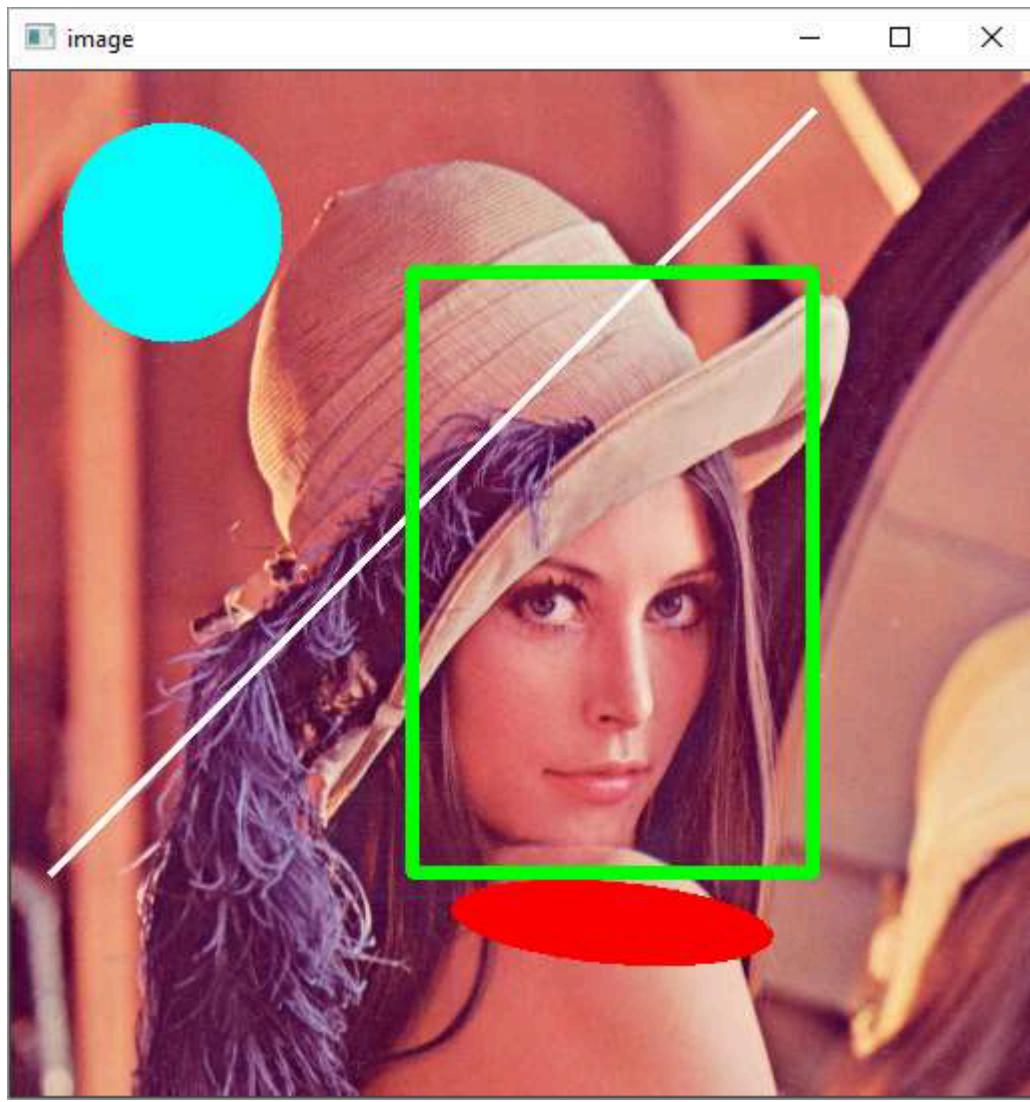
Sr. Não.	Descrição da função
1	imagem A imagem onde você deseja desenhar as formas
2	cor Cor da forma. para BGR, passe-o como uma tupla. Para tons de cinza, basta passar o valor escalar.
3	grossura Espessura da linha ou círculo, etc. Se -1 for passado para figuras fechadas como círculos, ele preencherá a forma.
4	tipo de linha Tipo de linha, seja linha de 8 conexões, linha suavizada, etc.

Exemplo

O exemplo a seguir mostra como as formas são desenhadas no topo de uma imagem. O programa para o mesmo é fornecido abaixo -

```
import numpy as np
import cv2
img = cv2.imread('LENA.JPG',1)
cv2.line(img,(20,400),(400,20),(255,255,255),3)
cv2.rectangle(img,(200,100),(400,400),(0,255,0),5)
cv2.circle(img,(80,80), 55, (255,255,0), -1)
cv2.ellipse(img, (300,425), (80, 20), 5, 0, 360, (0,0,255), -1)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Saída



Desenhar texto

A função **cv2.putText()** é fornecida para escrever um texto na imagem. O comando para o mesmo é o seguinte -

```
img, text, org, fontFace, fontScale, color, thickness)
```

Fontes

OpenCV suporta as seguintes fontes -

Nome da fonte	Tamanho da fonte
FONT_HERSHEY_SIMPLEX	0
FONT_HERSHEY_PLAIN	1
FONT_HERSHEY_DUPLEX	2

FONT_HERSHEY_COMPLEX

3

FONT_HERSHEY_TRIPLEX

4

FONT_HERSHEY_COMPLEX_SMALL

5

FONT_HERSHEY_SCRIPT_SIMPLEX

6

FONT_HERSHEY_SCRIPT_COMPLEX

7

FONT_ITALIC

16

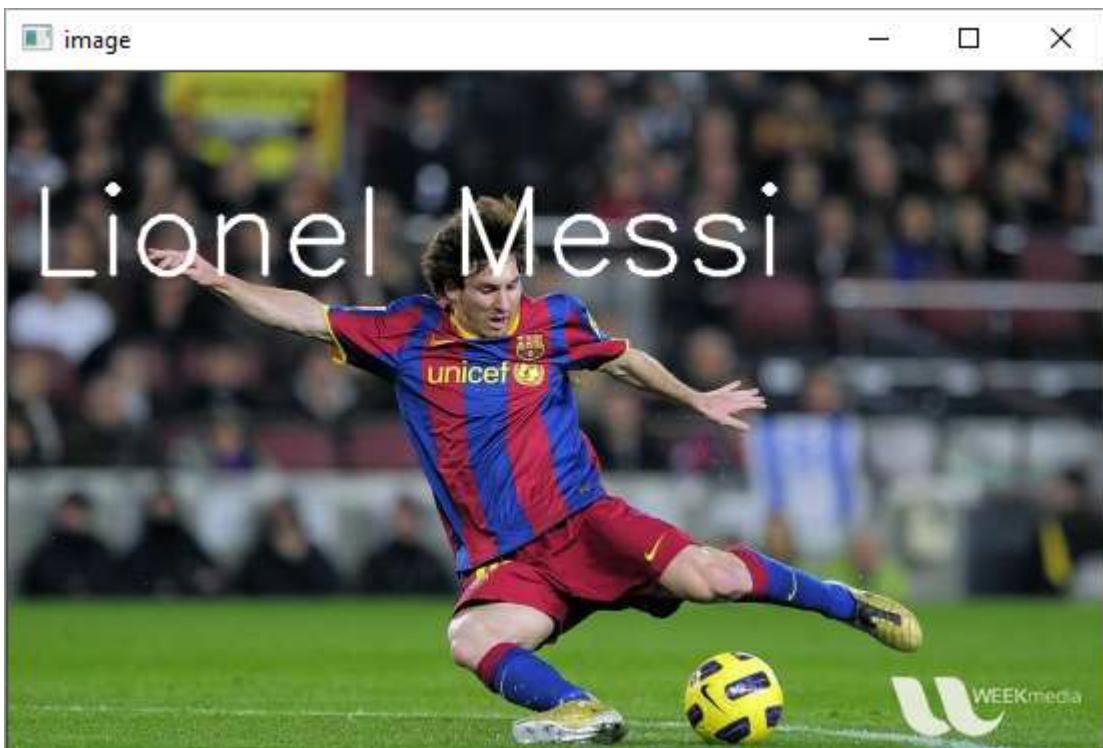
Exemplo

O programa a seguir adiciona uma legenda a uma fotografia que mostra Lionel Messi, o famoso jogador de futebol.

```
import numpy as np
import cv2
img = cv2.imread('messi.JPG',1)
txt="Lionel Messi"
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,txt,(10,100), font, 2,(255,255,255),2,cv2.LINE_AA)

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

Saída



OpenCV Python - Manipulação de eventos de mouse

OpenCV é capaz de registrar vários eventos relacionados ao mouse com uma função de retorno de chamada. Isso é feito para iniciar uma determinada ação definida pelo usuário, dependendo do tipo de evento do mouse.

Sr. Nó	Evento e descrição do mouse
1	cv.EVENT_MOUSEMOVE Quando o ponteiro do mouse passar sobre a janela.
2	cv.EVENT_LBUTTONDOWN Indica que o botão esquerdo do mouse está pressionado.
3	cv.EVENT_RBUTTONDOWN Caso o botão direito do mouse seja pressionado.
4	cv.EVENT_MBUTTONDOWN Indica que o botão do meio do mouse está pressionado.
5	cv.EVENT_LBUTTONUP Quando o botão esquerdo do mouse é liberado.
6	cv.EVENT_RBUTTONUP Quando o botão direito do mouse é liberado.
7	cv.EVENT_MBUTTONUP Indica que o botão do meio do mouse está liberado.
8	cv.EVENT_LBUTTONDOWNDBLCLK Este evento ocorre quando o botão esquerdo do mouse é clicado duas vezes.
9	cv.EVENT_RBUTTONDOWNDBLCLK Indica que o botão direito do mouse foi clicado duas vezes.
10	cv.EVENT_MBUTTONDOWNDBLCLK Indica que o botão do meio do mouse foi clicado duas vezes.
11	cv.EVENT_MOUSEWHEEL Positivo para avançar e negativo para rolar para trás.

Para disparar uma função em um evento de mouse, ela deve ser registrada com a ajuda da função **setMouseCallback()**. O comando para o mesmo é o seguinte -

```
cv2.setMouseCallback(window, callbak_function)
```

Esta função passa o tipo e a localização do evento para a função de retorno de chamada para processamento posterior.

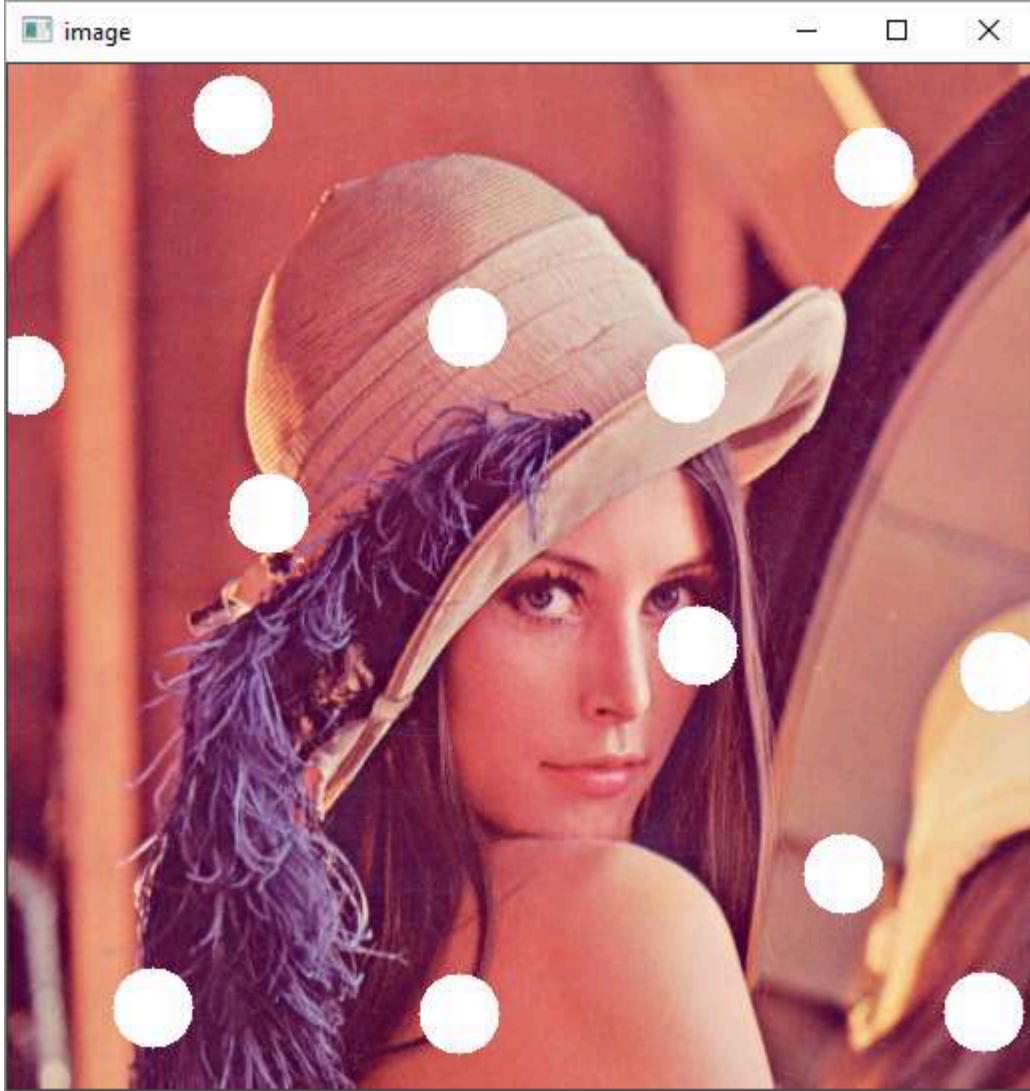
Exemplo 1

O código a seguir desenha um círculo sempre que ocorre um evento de clique duplo com o botão esquerdo na janela mostrando uma imagem como plano de fundo -

```
import numpy as np
import cv2 as cv
# mouse callback function
def drawfunction(event,x,y,flags,param):
    if event == cv.EVENT_LBUTTONDOWN:
        cv.circle(img,(x,y),20,(255,255,255),-1)
img = cv.imread('lena.jpg')
cv.namedWindow('image')
cv.setMouseCallback('image',drawfunction)
while(1):
    cv.imshow('image',img)
    key=cv.waitKey(1)
    if key == 27:
        break
cv.destroyAllWindows()
```

Saída

Execute o programa acima e clique duas vezes em locais aleatórios. A saída semelhante aparecerá -



Exemplo 2

O programa a seguir desenha interativamente retângulo, linha ou círculo, dependendo da entrada do usuário (1,2 ou 3) -

```
import numpy as np
import cv2 as cv
# mouse callback function

drawing=True
shape='r'

def draw_circle(event,x,y,flags,param):
    global x1,x2
    if event == cv.EVENT_LBUTTONDOWN:
        drawing = True
        x1,x2 = x,y
    elif event == cv.EVENT_LBUTTONUP:
        drawing = False
    if shape == 'r':
        cv.rectangle(img,(x1,x2),(x,y),(0,255,0),-1)
    if shape == 'l':
```

```

cv.line(img,(x1,x2),(x,y),(255,255,255),3)
if shape=='c':
    cv.circle(img,(x,y), 10, (255,255,0), -1)
img = cv.imread('lena.jpg')
cv.namedWindow('image')
cv.setMouseCallback('image',draw_circle)
while(1):
    cv.imshow('image',img)
    key=cv.waitKey(1)
    if key==ord('1'):
        shape='r'
    if key==ord('2'):
        shape='l'
    if key==ord('3'):
        shape='c'

#print (shape)
if key == 27:
    break
cv.destroyAllWindows()

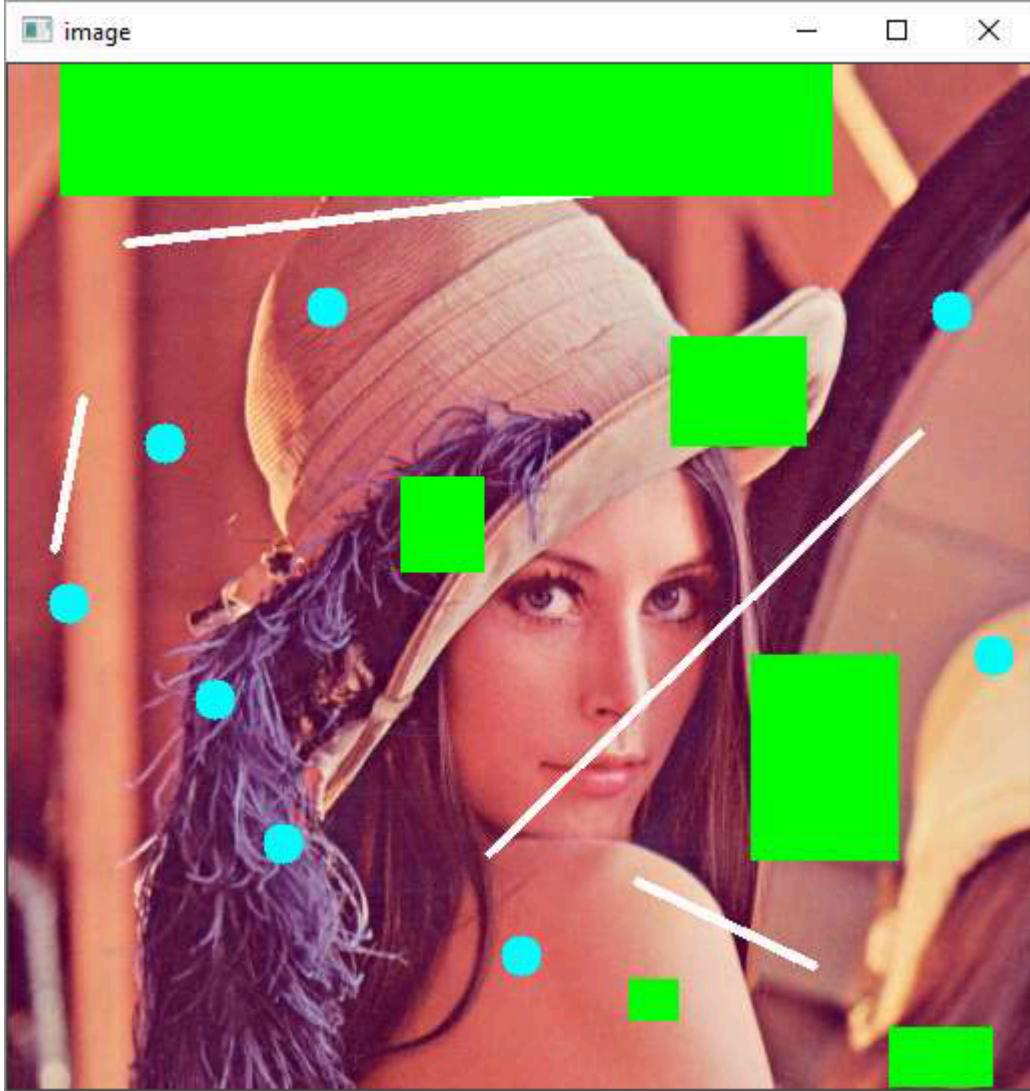
```

Na superfície da janela, um retângulo é desenhado entre as coordenadas do botão esquerdo do mouse para baixo e para cima se '1' for pressionado.

Se a escolha do usuário for 2, uma linha será desenhada usando coordenadas como pontos finais.

Ao escolher 3 para o círculo, ele é desenhado nas coordenadas do evento mouse up.

A imagem a seguir será a saída após a execução bem-sucedida do programa mencionado acima -



OpenCV Python - Adicionar Trackbar

Trackbar no OpenCV é um controle deslizante que ajuda a escolher um valor para a variável em um intervalo contínuo deslizando manualmente a guia sobre a barra. A posição da guia é sincronizada com um valor.

A função `createTrackbar()` cria um objeto Trackbar com o seguinte comando -

```
cv2.createTrackbar(trackbarname, winname, value, count, TrackbarCallback)
```

No exemplo a seguir, três trackbars são fornecidos para o usuário definir valores de R, G e B na faixa de tons de cinza de 0 a 255.

Usando os valores de posição da barra de controle, um retângulo é desenhado com a cor de preenchimento correspondente ao valor da cor RGB.

Exemplo

O programa a seguir é para adicionar uma trackbar -

```
import numpy as np  
import cv2 as cv
```

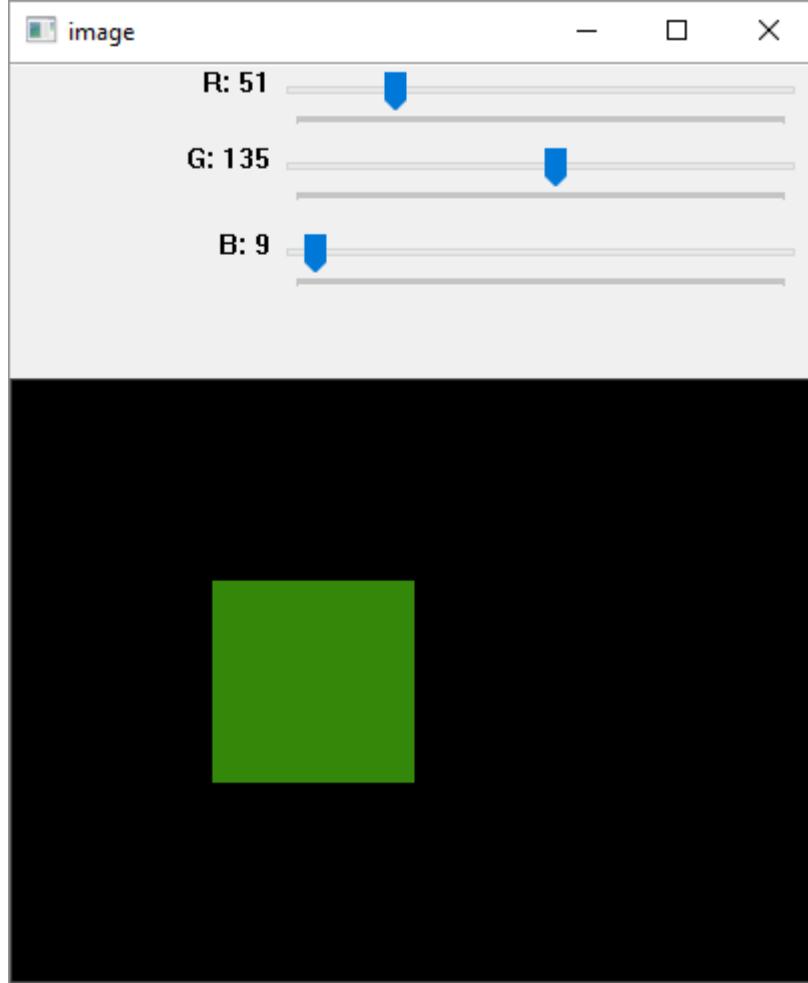
```
img = np.zeros((300,400,3), np.uint8)
cv.namedWindow('image')
def nothing(x):
    pass

# create trackbars for color change
cv.createTrackbar('R','image',0,255,nothing)
cv.createTrackbar('G','image',0,255,nothing)
cv.createTrackbar('B','image',0,255,nothing)

while(1):
    cv.imshow('image',img)
    k = cv.waitKey(1) & 0xFF
    if k == 27:
        break
# get current positions of four trackbars
r = cv.getTrackbarPos('R','image')
g = cv.getTrackbarPos('G','image')
b = cv.getTrackbarPos('B','image')

#s = cv.getTrackbarPos(switch,'image')
#img[:] = [b,g,r]
cv.rectangle(img, (100,100),(200,200), (b,g,r),-1)
cv.destroyAllWindows()
```

Saída



OpenCV Python - redimensionar e girar uma imagem

Neste capítulo, aprenderemos como redimensionar e girar uma imagem com a ajuda do OpenCVPython.

Redimensionar uma imagem

É possível aumentar ou diminuir uma imagem com o uso da função `cv2.resize()`.

A função **resize()** é usada da seguinte forma -

```
resize(src, dsize, dst, fx, fy, interpolation)
```

Em geral, a interpolação é um processo de estimativa de valores entre pontos de dados conhecidos.

Quando os dados gráficos contêm uma lacuna, mas os dados estão disponíveis em ambos os lados da lacuna ou em alguns pontos específicos dentro da lacuna. A interpolação nos permite estimar os valores dentro da lacuna.

Na função `resize()` acima, os sinalizadores de interpolação determinam o tipo de interpolação usada para calcular o tamanho da imagem de destino.

Tipos de interpolação

Os tipos de interpolação são os seguintes -

- **INTER_NEAREST** - Uma interpolação do vizinho mais próximo.
- **INTER_LINEAR** - Uma interpolação bilinear (usada por padrão)
- **INTER_AREA** - Reamostragem usando relação de área de pixel. É um método preferido para dizimação de imagem, mas quando a imagem é ampliada, é semelhante ao método INTER_NEAREST.
- **INTER_CUBIC** - Uma interpolação bicúbica em uma vizinhança de 4x4 pixels
- **INTER_LANCZOS4** - Uma interpolação Lanczos sobre uma vizinhança de 8x8 pixels

Os métodos de interpolação preferíveis são cv2.INTER_AREA para redução e cv2.INTER_CUBIC (lento) e cv2.INTER_LINEAR para zoom.

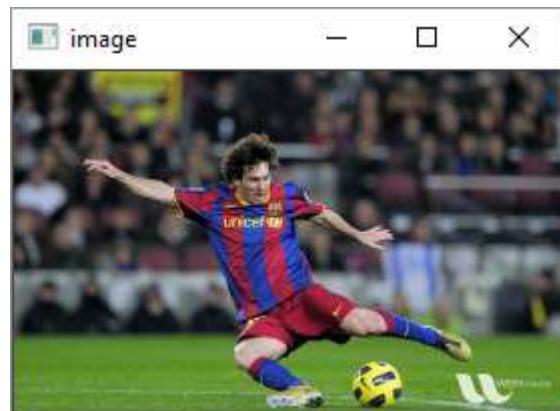
Exemplo

O código a seguir redimensiona a imagem 'messi.jpg' para metade de sua altura e largura originais.

```
import numpy as np
import cv2
img = cv2.imread('messi.JPG',1)
height, width = img.shape[:2]
res = cv2.resize(img,(int(width/2), int(height/2)), interpolation =
cv2.INTER_AREA)

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

Saída



Girar uma imagem

OpenCV usa funções de transformação afins para operações em imagens, como tradução e rotação. A transformação afim é uma transformação que pode ser expressa na forma de uma multiplicação de matrizes (transformação linear) seguida de uma adição vetorial (tradução).

O módulo cv2 fornece duas funções **cv2.warpAffine** e **cv2.warpPerspective**, com as quais você pode ter todos os tipos de transformações. cv2.warpAffine usa uma matriz de transformação 2x3 enquanto cv2.warpPerspective usa uma matriz de transformação 3x3 como entrada.

Para encontrar esta matriz de transformação para rotação, OpenCV fornece uma função, **cv2.getRotationMatrix2D**, que é a seguinte -

```
getRotationMatrix2D(center, angle, scale)
```

Em seguida, aplicamos a função **warpAffine** à matriz retornada pela função getRotationMatrix2D() para obter a imagem girada.

O programa a seguir gira a imagem original em 90 graus sem alterar as dimensões -

Exemplo

```
import numpy as np
import cv2
img = cv2.imread('OpenCV_Logo.png',1)
h, w = img.shape[:2]

center = (w / 2, h / 2)
mat = cv2.getRotationMatrix2D(center, 90, 1)
rotimg = cv2.warpAffine(img, mat, (h, w))
cv2.imshow('original',img)
cv2.imshow('rotated', rotimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Saída

Imagen original

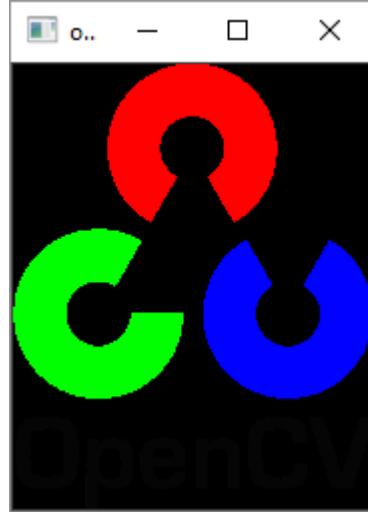
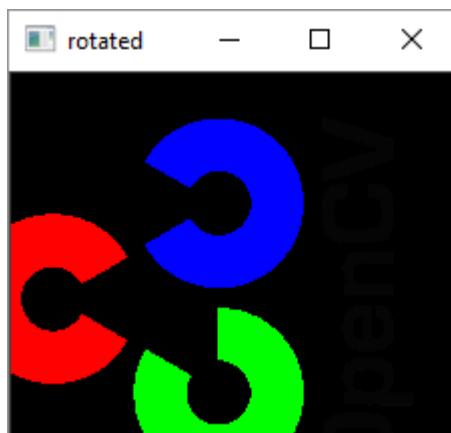


Imagen girada



OpenCV Python - Limite de imagem

No processamento digital de imagens, o limiar é um processo de criação de uma imagem binária com base em um valor limite de intensidade de pixel. O processo de limite separa os pixels do primeiro plano dos pixels do fundo.

OpenCV fornece funções para realizar limiares **simples, adaptativos e de Otsu** .

No limite simples, todos os pixels com valor menor que o limite são definidos como zero e permanecem no valor máximo do pixel. Esta é a forma mais simples de limiarização.

A função **cv2.threshold()** tem a seguinte definição.

```
cv2.threshold((src, thresh, maxval, type, dst)
```

Parâmetros

Os parâmetros para o limite da imagem são os seguintes -

- Src: Matriz de entrada.
- Dst: matriz de saída do mesmo tamanho.
- Limite: Valor limite.

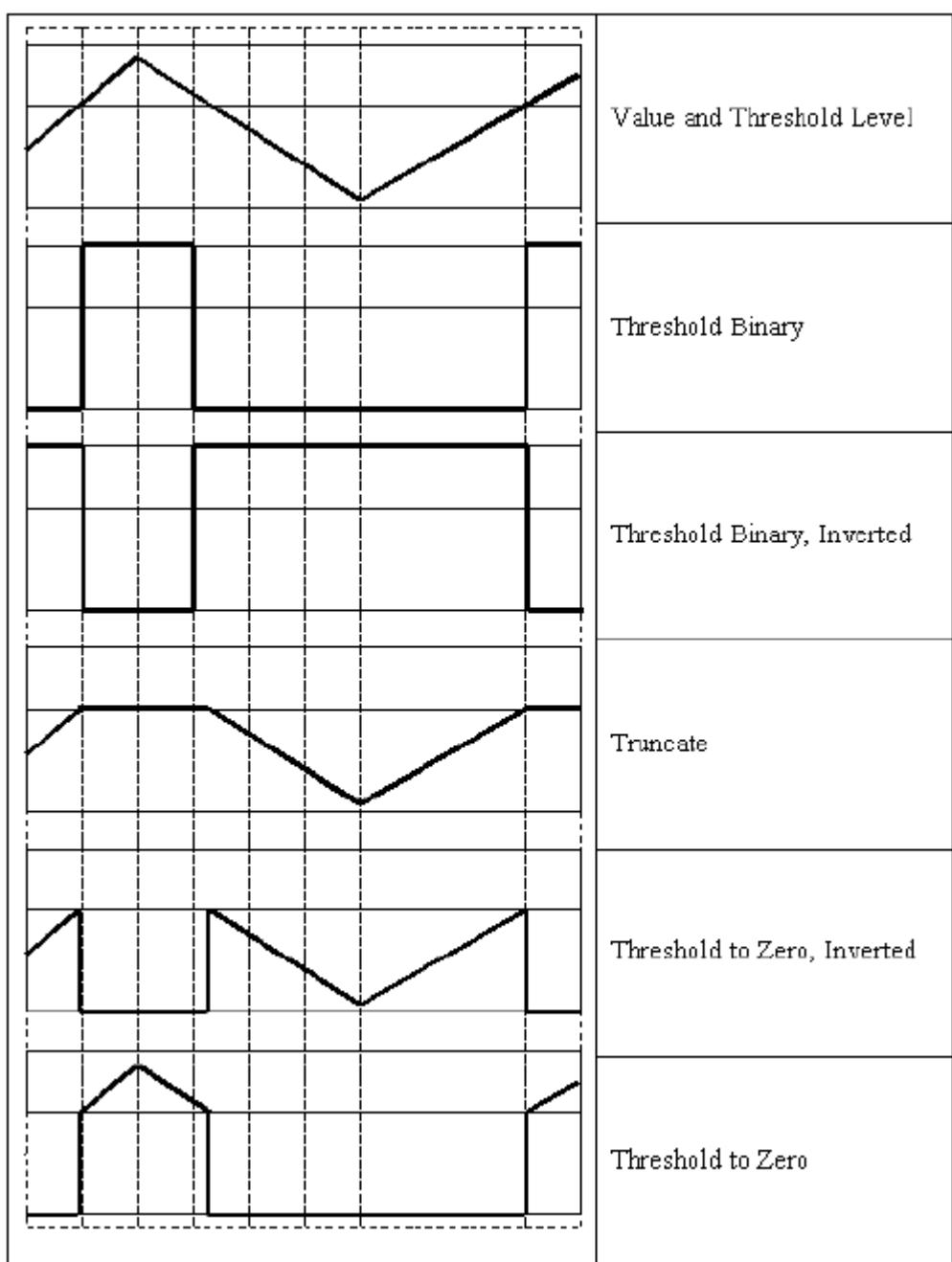
- Maxval: Valor máximo.
- Tipo: Tipo de limite.

Tipos de limite

Outros tipos de limite são enumerados abaixo -

Sr. Não	Tipo e função
1	cv.THRESH_BINARY dst(x,y) = maxval if src(x,y)>thresh 0 caso contrário
2	cv.THRESH_BINARY_INV dst(x,y)=0 if src(x,y)>thresh maxval caso contrário
3	cv.THRESH_TRUNC dst(x,y)=limite if src(x,y)>thresh src(x,y) caso contrário
4	cv.THRESH_TOZERO dst(x,y)=src(x,y) if src(x,y)>thresh 0 caso contrário
5	cv.THRESH_TOZERO_INV dst(x,y)=0 se src(x,y)>thresh src(x,y)caso contrário

Esses tipos de limite resultam na operação na imagem de entrada de acordo com o diagrama a seguir -



A função limite() retorna o limite usado e a imagem do limite.

O programa a seguir produz uma imagem binária do original com um gradiente de valores de cinza de 255 a 0, definindo um limite para 127.

Exemplo

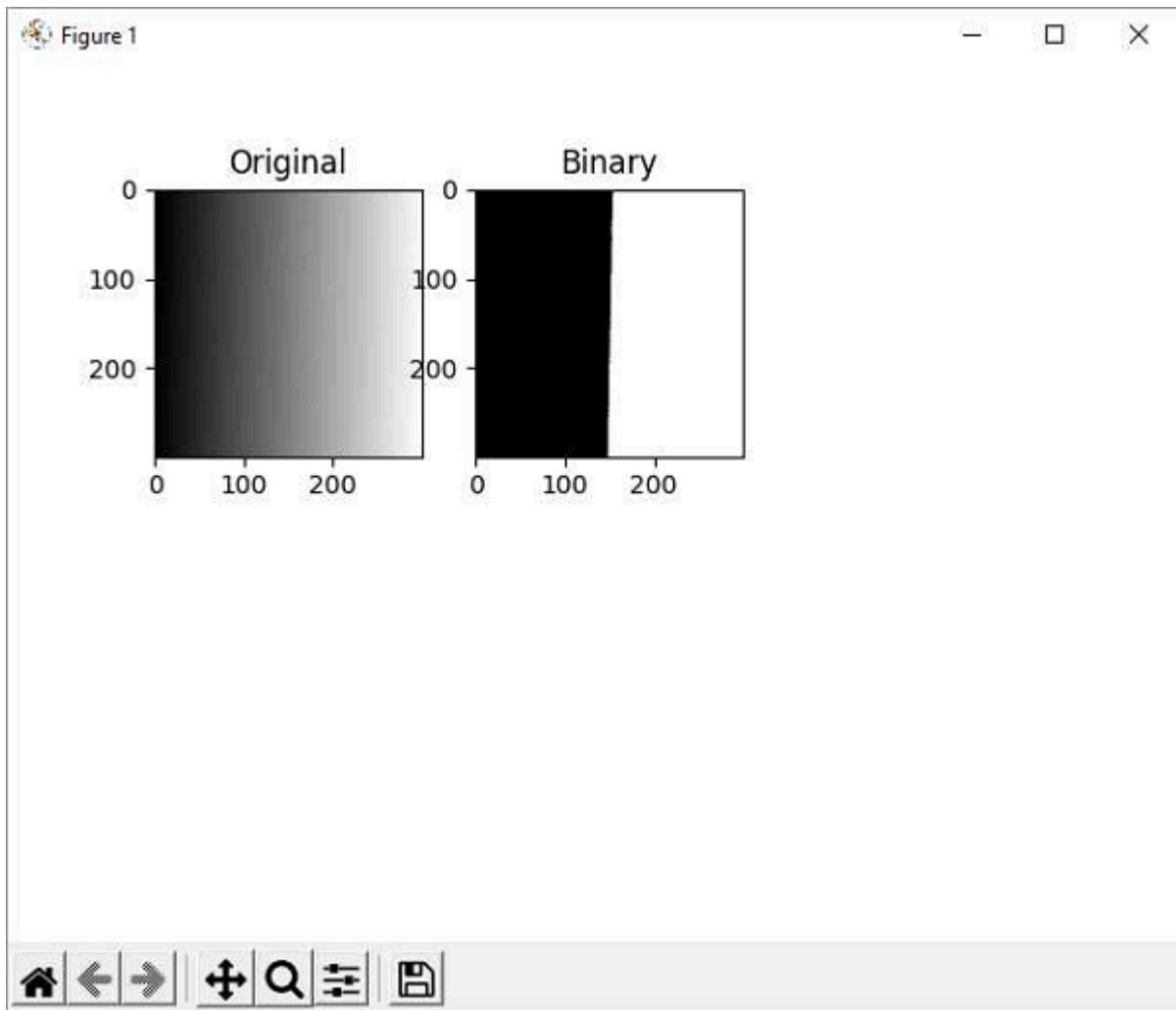
As imagens binárias de limite originais e resultantes são plotadas lado a lado usando a biblioteca Matplotlib.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('gradient.png',0)
ret,img1 = cv.threshold(img,127,255,cv.THRESH_BINARY)

plt.subplot(2,3,1),plt.imshow(img,'gray',vmin=0,vmax=255)
plt.title('Original')
```

```
plt.subplot(2,3,2),plt.imshow(img1,'gray',vmin=0,vmax=255)
plt.title('Binary')
plt.show()
```

Saída



The adaptive thresholding determines the threshold for a pixel based on a small region around it. So, different thresholds for different regions of the same image are obtained. This gives better results for images with varying illumination.

The cv2.adaptiveThreshold() method takes following input arguments –

```
cv.adaptiveThreshold( src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst] )
```

The adaptiveMethod has following enumerated values –

- **cv.ADAPTIVE_THRESH_MEAN_C** – The threshold value is the mean of the neighbourhood area minus the constant C.
- **cv.ADAPTIVE_THRESH_GAUSSIAN_C** – The threshold value is a Gaussianweighted sum of the neighbourhood values minus the constant C.

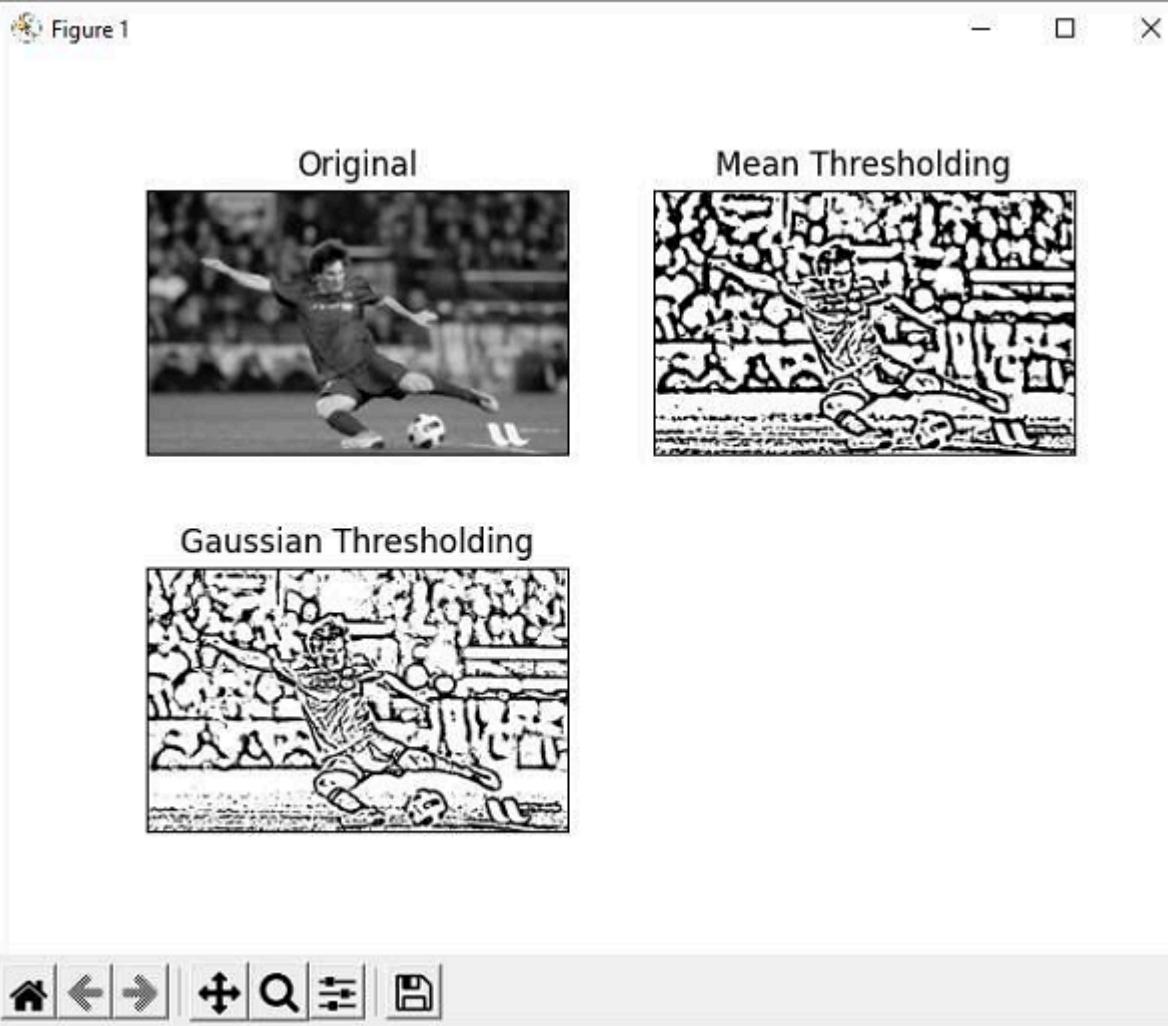
Example

In the example below, the original image (messi.jpg is applied with mean and Gaussian adaptive thresholding.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('messi.jpg',0)
img = cv.medianBlur(img,5)
th1 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_MEAN_C,\n    cv.THRESH_BINARY,11,2)
th2 = cv.adaptiveThreshold(img,255,cv.ADAPTIVE_THRESH_GAUSSIAN_C,\n    cv.THRESH_BINARY,11,2)
titles = ['Original', 'Mean Thresholding', 'Gaussian Thresholding']
images = [img, th1, th2]
for i in range(3):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

Output

Original as well as adaptive threshold binary images are plotted by using matplotlib as shown below –



Example

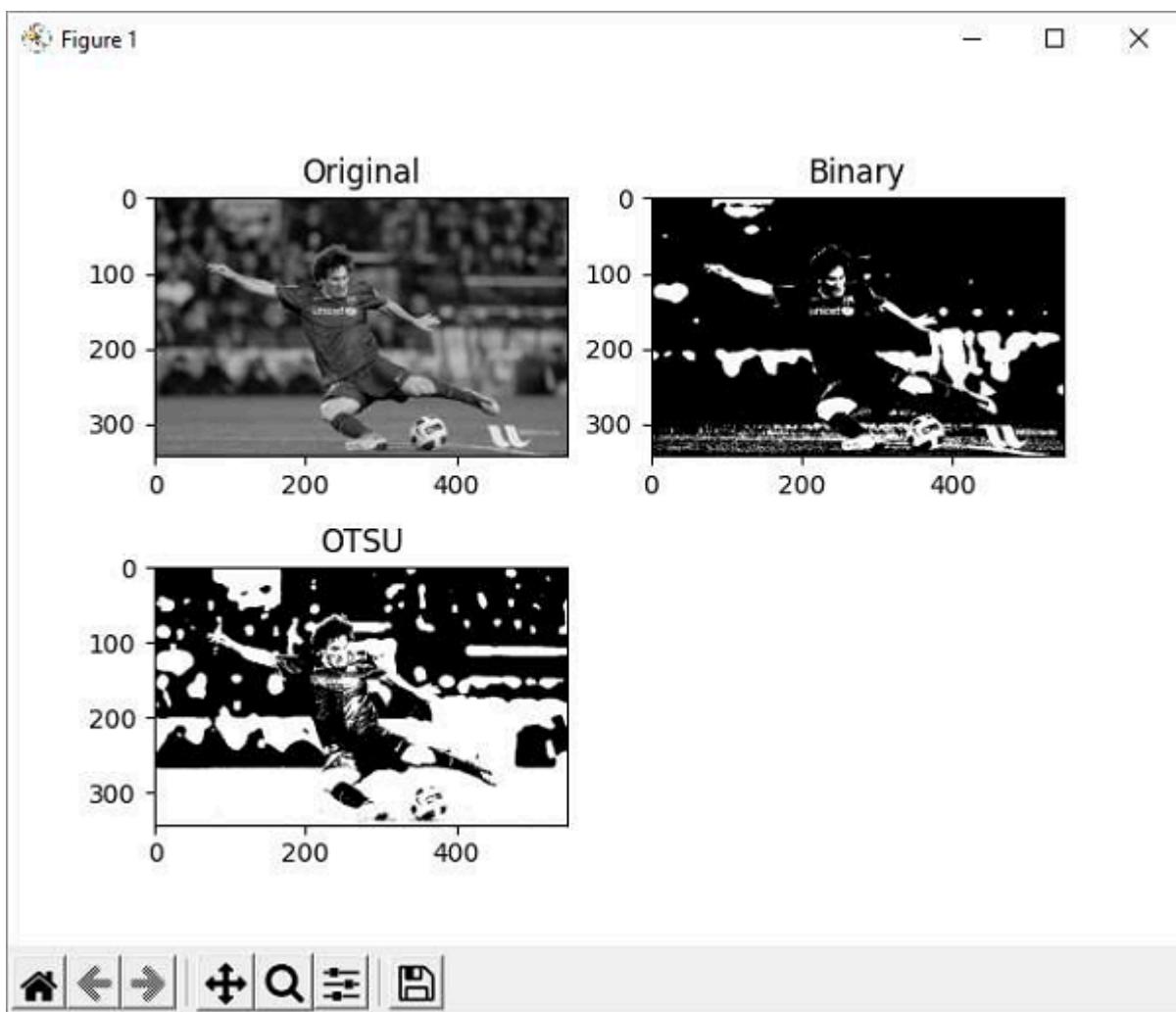
OTSU algorithm determines the threshold value automatically from the image histogram. We need to pass the cv.THRES_OTSU flag in addition to the THRESH-BINARY flag.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('messi.jpg',0)
# global thresholding
ret1,img1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
# Otsu's thresholding
ret2,img2 = cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
plt.subplot(2,2,1),plt.imshow(img,'gray',vmin=0,vmax=255)
plt.title('Original')
plt.subplot(2,2,2),plt.imshow(img1,'gray')

plt.title('Binary')
plt.subplot(2,2,3),plt.imshow(img2,'gray')
plt.title('OTSU')
plt.show()
```

Output

The matplotlib's plot result is as follows –



OpenCV Python - Image Filtering

An image is basically a matrix of pixels represented by binary values between 0 to 255 corresponding to gray values. A color image will be a three dimensional matrix with a number of channels corresponding to RGB.

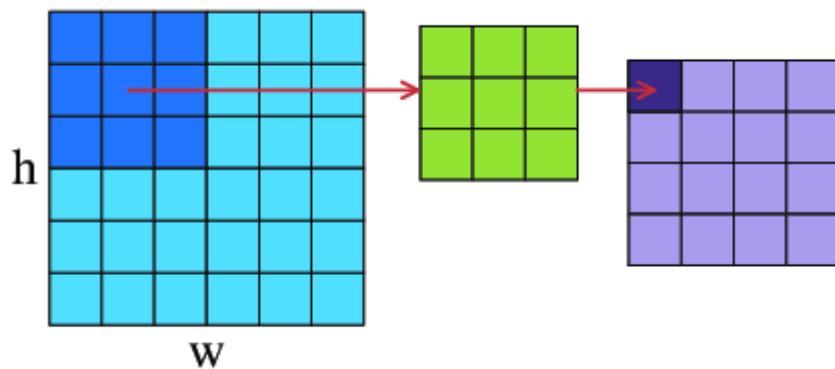
Image filtering is a process of averaging the pixel values so as to alter the shade, brightness, contrast etc. of the original image.

By applying a low pass filter, we can remove any noise in the image. High pass filters help in detecting the edges.

The OpenCV library provides **cv2.filter2D()** function. It performs convolution of the original image by a kernel of a square matrix of size 3X3 or 5X5 etc.

Convolution slides a kernel matrix across the image matrix horizontally and vertically. For each placement, add all pixels under the kernel, take the average of pixels under the kernel and replace the central pixel with the average value.

Perform this operation for all pixels to obtain the output image pixel matrix. Refer the diagram given below –



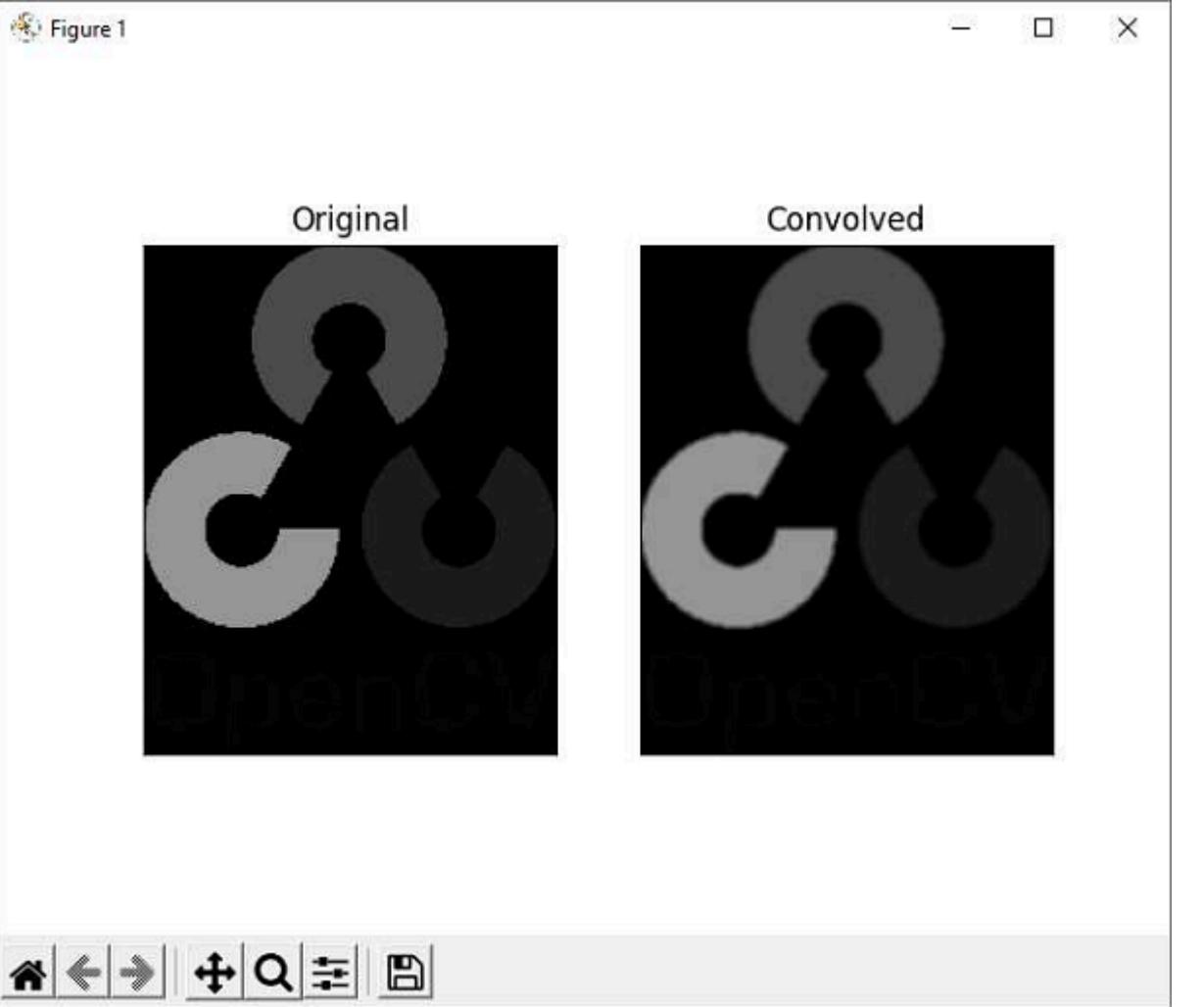
The cv2.filter2D() function requires input array, kernel matrix and output array parameters.

Example

Following figure uses this function to obtain an averaged image as a result of 2D convolution. The program for the same is as follows –

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('opencv_logo_gs.png')
kernel = np.ones((3,3),np.float32)/9
dst = cv.filter2D(img,-1,kernel)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Convolved')
plt.xticks([]), plt.yticks([])
plt.show()
```

Output



Types of Filtering Function

Other types of filtering function in OpenCV includes –

- **BilateralFilter** – Reduces unwanted noise keeping edges intact.
- **BoxFilter** – This is an average blurring operation.
- **GaussianBlur** – Eliminates high frequency content such as noise and edges.
- **MedianBlur** – Instead of average, it takes the median of all pixels under the kernel and replaces the central value.

OpenCV Python - Edge Detection

Uma aresta aqui significa o limite de um objeto na imagem. OpenCV possui uma função **cv2.Canny()** que identifica as bordas de vários objetos em uma imagem implementando o algoritmo de Canny.

O algoritmo de detecção de bordas Canny foi desenvolvido por John Canny. Segundo ele, as bordas do objeto são determinadas executando as seguintes etapas -

O primeiro passo é reduzir os pixels ruidosos na imagem. Isso é feito aplicando o Filtro Gaussiano 5X5.

A segunda etapa envolve encontrar o gradiente de intensidade da imagem. A imagem suave do primeiro estágio é filtrada aplicando-se o operador Sobel para obter derivadas de primeira ordem nas direções horizontal e vertical (G_x e G_y).

O valor da raiz quadrada média fornece o gradiente da borda e a razão inversa tan das derivadas fornece a direção da borda.

$$\text{Edge gradient } G = \sqrt{G_x^2 + G_y^2}$$

$$\text{Angle } \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Depois de obter a magnitude e a direção do gradiente, é feita uma varredura completa da imagem para remover quaisquer pixels indesejados que possam não constituir a borda.

A próxima etapa é realizar o limite de histerese usando limites mínimo e máximo. Gradientes de intensidade menores que minval e maxval não são arestas, portanto descartados. Aqueles intermediários são tratados como pontos de borda ou não-arestas com base em sua conectividade.

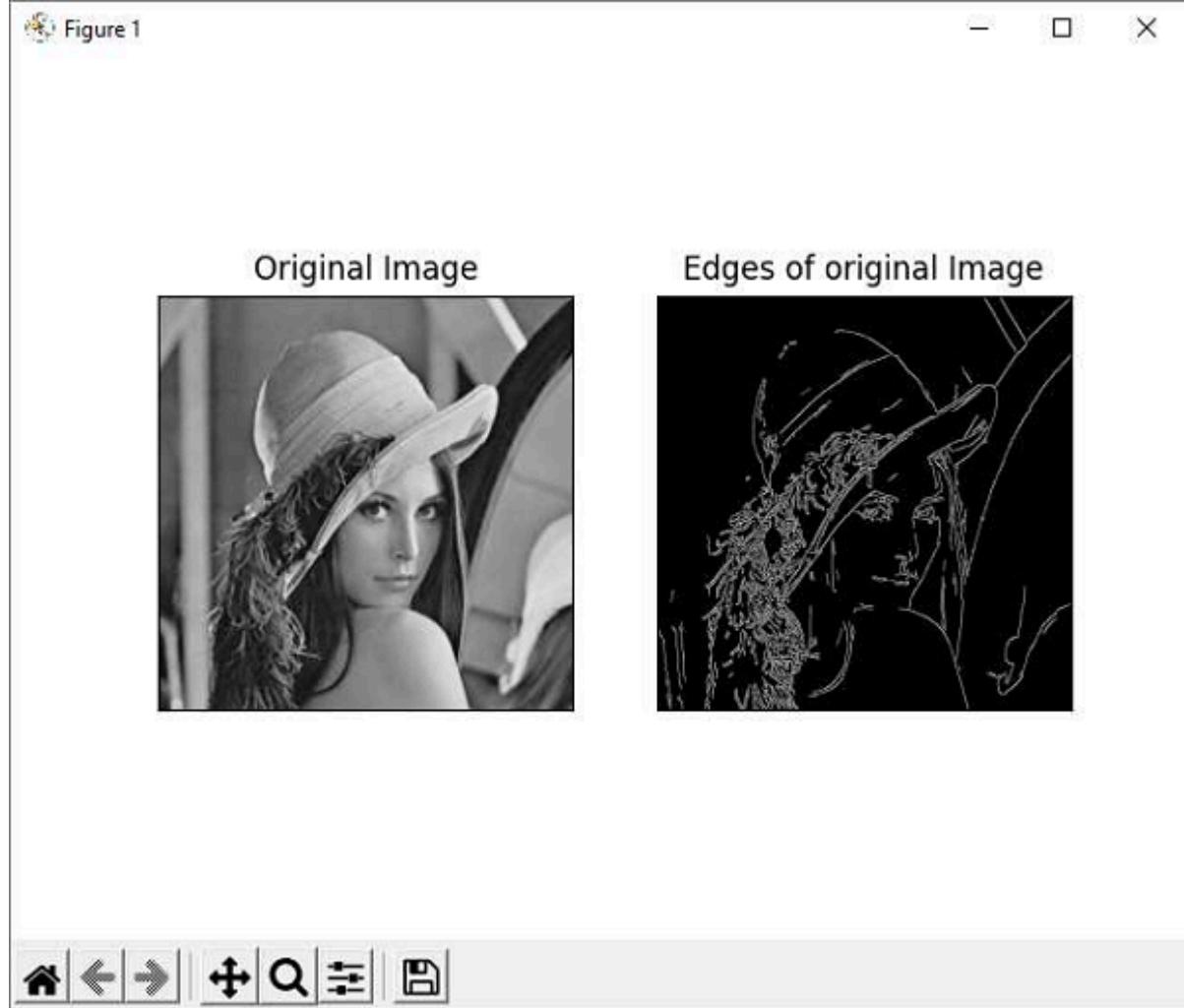
Todas essas etapas são executadas pela função cv2.Canny() do OpenCV que precisa da matriz de imagem de entrada e dos parâmetros minval e maxval.

Exemplo

Aqui está o exemplo de detecção inteligente de bordas. O programa para o mesmo é o seguinte -

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('lena.jpg', 0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edges of original Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

Saída



OpenCV Python - Histograma

O histograma mostra a distribuição de intensidade em uma imagem. Ele plota os valores de pixel (0 a 255) no eixo X e o número de pixels no eixo Y.

Ao usar o histograma, pode-se entender a distribuição de contraste, brilho e intensidade da imagem especificada. Os comportamentos em um histograma representam partes incrementais dos valores no eixo X.

No nosso caso, é o valor do pixel e o tamanho do comportamento padrão é um.

Na biblioteca OpenCV, a função **cv2.calcHist()** função que calcula o histograma a partir da imagem de entrada. O comando para a função é o seguinte -

```
cv.calcHist(images, channels, mask, histSize, ranges)
```

Parâmetros

Os parâmetros da função **cv2.calcHist()** são os seguintes -

- **images** - É a imagem fonte do tipo uint8 ou float32, entre colchetes, ou seja, "[img]".
- **canais** - É o índice do canal para o qual calculamos o histograma. Para uma imagem em tons de cinza, seu valor é [0]. Para imagens BGR, pode-se passar [0], [1] ou [2]

para calcular o histograma de cada canal.

- **mask** - A imagem da máscara é fornecida como "Nenhum" para imagem completa. Para uma determinada região da imagem, você deve criar uma imagem de máscara para isso e fornecê-la como máscara.
- **histSize** - Isso representa nossa contagem de BIN.
- **ranges** - Normalmente é [0,256].

Histograma usando Matplotlib

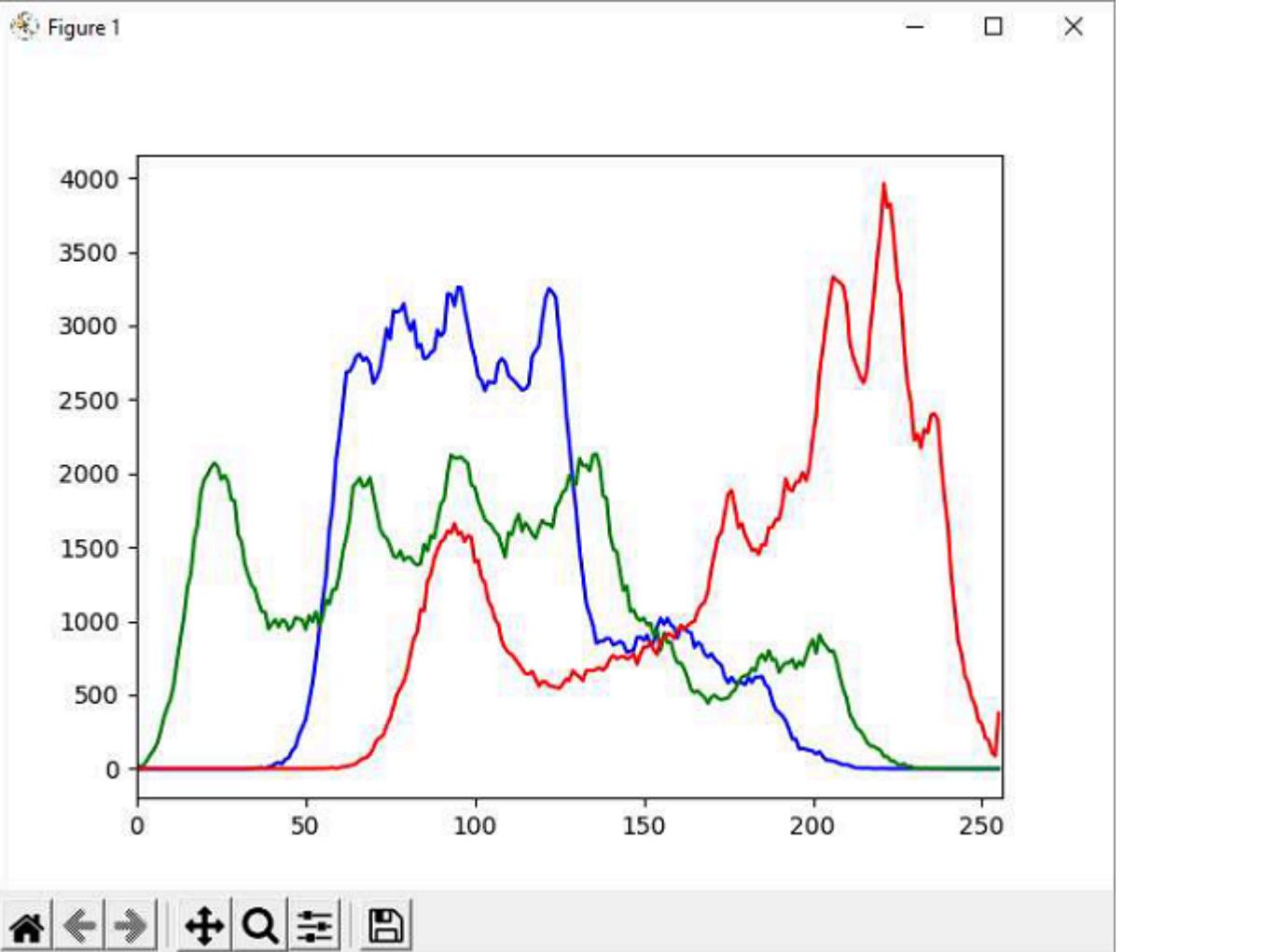
Um gráfico de histograma pode ser obtido com a ajuda da função **pyplot.plot()** do Matplotlib ou chamando a função **Polyline()** da biblioteca OpenCV.

Exemplo

O programa a seguir calcula o histograma para cada canal na imagem (lena.jpg) e traça a distribuição de intensidade para cada canal -

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('lena.jpg')
color = ('b','g','r')
for i,col in enumerate(color):
    hist = cv.calcHist([img],[i],None,[256],[0,256])
    plt.plot(hist, color = col)
    plt.xlim([0,256])
plt.show()
```

Saída



OpenCV Python - Espaços de cores

Um espaço de cores é um modelo matemático que descreve como as cores podem ser representadas. É descrito em uma faixa específica, mensurável e fixa de cores e valores de luminância possíveis.

OpenCV suporta espaços de cores bem conhecidos -

- **Espaço de cores RGB** - É um espaço de cores aditivo. Um valor de cor é obtido pela combinação dos valores das cores vermelho, verde e azul. Cada um é representado por um número que varia entre 0 e 255.
- **Espaço de cores HSV** - H, S e V significam Matiz, Saturação e Valor. Este é um modelo de cores alternativo ao RGB. Supõe-se que este modelo esteja mais próximo da forma como o olho humano percebe qualquer cor. O valor de matiz está entre 0 e 179, enquanto os números S e V estão entre 0 e 255.
- **Espaço de cores CMYK** - Em contraste com RGB, CMYK é um modelo de cores subtrativo. Os alfabetos representam Ciano, Magenta, Amarelo e Preto. A luz branca menos o vermelho deixa o ciano, o verde subtraído do branco deixa o magenta e o branco menos o azul retorna o amarelo. Todos os valores são representados na escala de 0 a 100%.
- **Espaço de cores CIELAB** - O espaço de cores LAB possui três componentes que são L para luminosidade, componentes de cores A variando de Verde a Magenta e B para componentes de Azul a Amarelo.

- **Espaço de cores YCrCb** - Aqui, Cr significa RY e Cb significa BY. Isso ajuda na separação da luminância da crominância em diferentes canais.

OpenCV suporta conversão de imagem entre espaços de cores com a ajuda da função **cv2.cvtColor()**.

O comando para a função cv2.cvtColor() é o seguinte -

```
cv.cvtColor(src, code, dst)
```

Códigos de conversão

A conversão é regida pelos seguintes códigos de conversão predefinidos.

Sr. Não.	Código e função de conversão
1	cv.COLOR_BGR2BGRA Adicione canal alfa à imagem RGB ou BGR.
2	cv.COLOR_BGRA2BGR Remova o canal alfa da imagem RGB ou BGR.
3	cv.COLOR_BGR2GRAY Converta entre RGB/BGR e tons de cinza.
4	cv.COLOR_BGR2YCrCb Converter RGB/BGR em luma-chroma
5	cv.COLOR_BGR2HSV Converter RGB/BGR em HSV
6	cv.COLOR_BGR2Lab Converter RGB/BGR em CIE Lab
7	cv.COLOR_HSV2BGR Conversões reversas de HSV para RGB/BGR

Exemplo

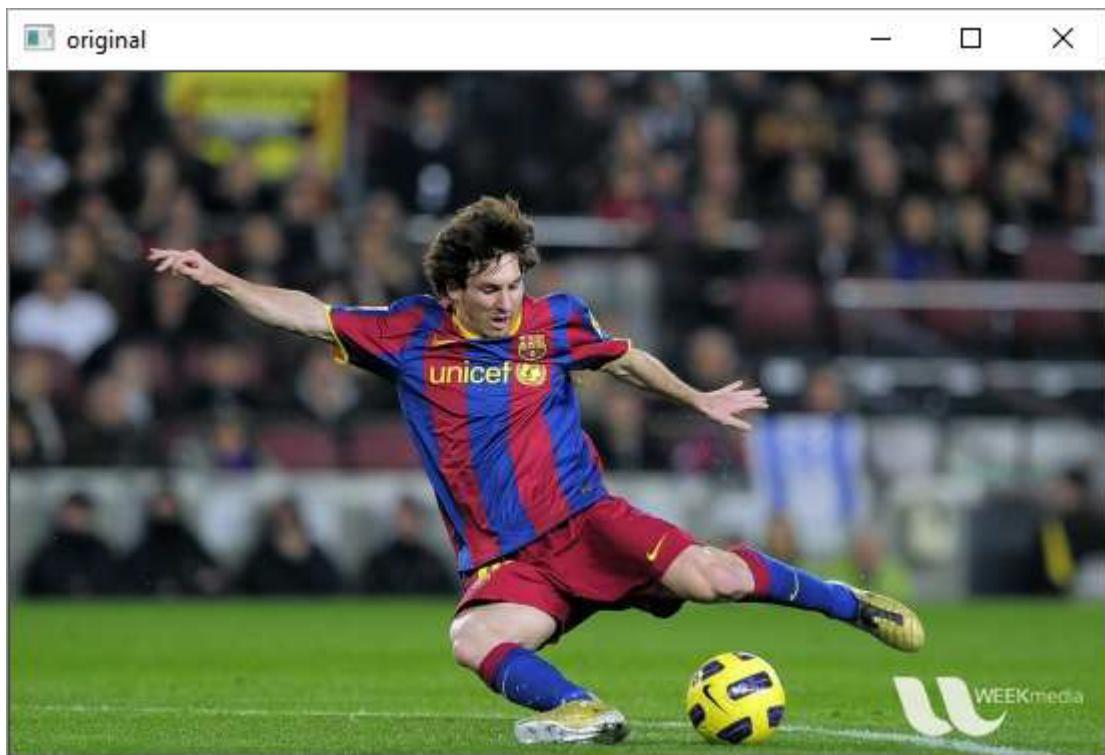
O programa a seguir mostra a conversão da imagem original com espaço de cores RGB para esquemas HSV e Gray -

```
import cv2
img = cv2.imread('messi.jpg')
img1 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY )
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

```
# Displaying the image
```

```
cv2.imshow('original', img)
cv2.imshow('Gray', img1)
cv2.imshow('HSV', img2)
```

Saída





OpenCV Python - Transformações Morfológicas

Operações simples em uma imagem com base em sua forma são denominadas transformações morfológicas. As duas transformações mais comuns são **erosão e dilatação**.

Erosão

A erosão elimina os limites do objeto em primeiro plano. Semelhante à convolução 2D, um kernel desliza pela imagem A. O pixel na imagem original é retido, se todos os pixels sob o kernel forem 1.

Caso contrário, é colocado em 0 e, portanto, causa erosão. Todos os pixels próximos ao limite são descartados. Este processo é útil para remover ruídos brancos.

O comando para a função **erode()** em OpenCV é o seguinte -

```
cv.erode(src, kernel, dst, anchor, iterations)
```

Parâmetros

A função **erode()** em OpenCV usa os seguintes parâmetros -

Os parâmetros src e dst são matrizes de imagens de entrada e saída do mesmo tamanho. Kernel é uma matriz de elementos estruturantes utilizados para erosão. Por exemplo, 3X3 ou 5X5.

O parâmetro âncora é -1 por padrão, o que significa que o elemento âncora está no centro. Iterações referem-se ao número de vezes que a erosão é aplicada.

Dilatação

É exatamente o oposto da erosão. Aqui, um elemento de pixel é 1, se pelo menos um pixel sob o kernel for 1. Como resultado, aumenta a região branca na imagem.

O comando para a função dilate() é o seguinte -

```
cv.dilate(src, kernel, dst, anchor, iterations)
```

Parâmetros

A função **dilate()** tem os mesmos parâmetros da função erode(). Ambas as funções podem ter parâmetros opcionais adicionais como BorderType e borderColor.

BorderType é um tipo enumerado de limites de imagem (CONSTANT, REPLICATE, TRANSPERANT etc.)

borderValue é usado no caso de uma borda constante. Por padrão, é 0.

Exemplo

Abaixo está um exemplo de programa mostrando as funções erode() e dilate() em uso -

```
import cv2 as cv
import numpy as np
img = cv.imread('LinuxLogo.jpg',0)
kernel = np.ones((5,5),np.uint8)
erosion = cv.erode(img,kernel,iterations = 1)
dilation = cv.dilate(img,kernel,iterations = 1)
cv.imshow('Original', img)
cv.imshow('Erosion', erosion)
cv.imshow('Dialation', dilation)
```

Saída

Imagen original



Erosão



Dilatação



OpenCV Python - Contornos de imagem

Contorno é uma curva que une todos os pontos contínuos ao longo do limite com a mesma cor ou intensidade. Os contornos são muito úteis para análise de formas e detecção de objetos.

Encontrar contorno

Antes de encontrar contornos, devemos aplicar detecção de limite ou de borda astuta. Então, usando o método **findContours()**, podemos encontrar os contornos na imagem binária.

O comando para usar a função **findContours()** é o seguinte -

```
cv.findContours(image, mode, method, contours)
```

Parâmetros

Os parâmetros da função **findContours()** são os seguintes -

- **image** - Fonte, uma imagem de canal único de 8 bits.
- **mode** - Modo de recuperação de contorno.
- **method** - Método de aproximação de contorno.

Os valores do parâmetro de modo são enumerados da seguinte forma -

- **cv.RETR_EXTERNAL** - Recupera apenas os contornos externos extremos.
- **cv.RETR_LIST** - Recupera todos os contornos sem estabelecer nenhuma relação hierárquica.
- **cv.RETR_CCOMP** - Recupera todos os contornos e os organiza em uma hierarquia de dois níveis.
- **cv.RETR_TREE** - Recupera todos os contornos e reconstrói uma hierarquia completa de contornos aninhados.

Por outro lado, o método de aproximação pode ser um dos seguintes -

- **cv.CHAIN_APPROX_NONE** - Armazena absolutamente todos os pontos do contorno.
- **cv.CHAIN_APPROX_SIMPLE** - Compacta segmentos horizontais, verticais e diagonais e deixa apenas seus pontos finais.

Desenhar contorno

Após detectar os vetores de contorno, os contornos são desenhados sobre a imagem original usando a função **cv.drawContours()**.

O comando para a função **cv.drawContours()** é o seguinte -

```
cv.drawContours(image, contours, contourIdx, color)
```

Parâmetros

Os parâmetros da função **drawContours()** são os seguintes -

- image - Imagem de destino.
- contornos - Todos os contornos de entrada. Cada contorno é armazenado como um vetor de ponto.
- contornoIdx - Parâmetro que indica um contorno a ser desenhado. Se for negativo, todos os contornos são desenhados.
- color - Cor dos contornos.

Exemplo

O código a seguir é um exemplo de desenho de contornos em uma imagem de entrada com três formas preenchidas com cores pretas.

Na primeira etapa, obtemos uma imagem cinza e em seguida realizamos a astuta detecção de bordas.

Na imagem resultante, chamamos então a função `findContours()`. Seu resultado é um vetor pontual. Em seguida, chamamos a função `drawContours()`.

O código completo é o seguinte -

```
import cv2
import numpy as np

img = cv2.imread('shapes.png')
cv2.imshow('Original', img)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

canny = cv2.Canny(gray, 30, 200)

contours, hierarchy = cv2.findContours(canny,
    cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
print("Number of Contours = " ,len(contours))
cv2.imshow('Canny Edges', canny)

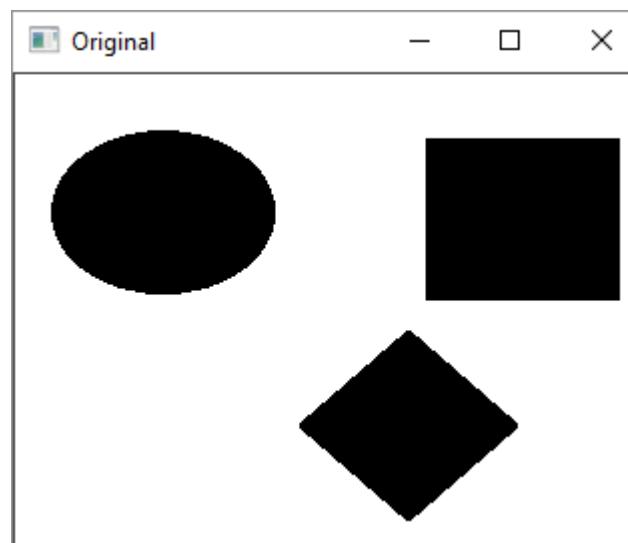
cv2.drawContours(img, contours, -1, (0, 255, 0), 3)

cv2.imshow('Contours', img)
```

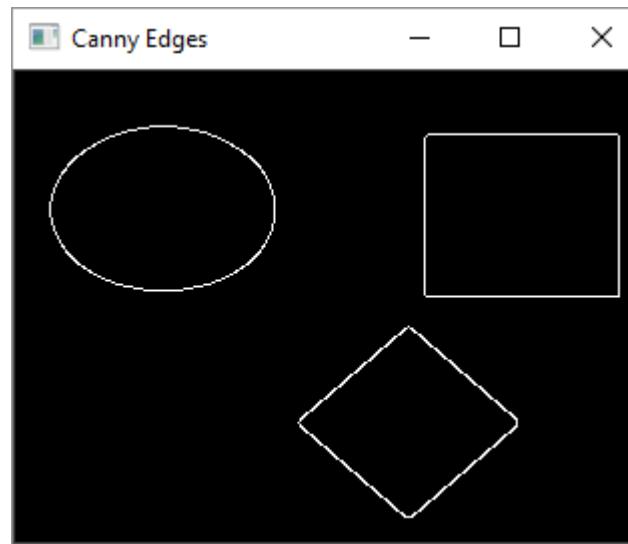
```
cv2.waitKey(0)  
cv2.destroyAllWindows()
```

Saída

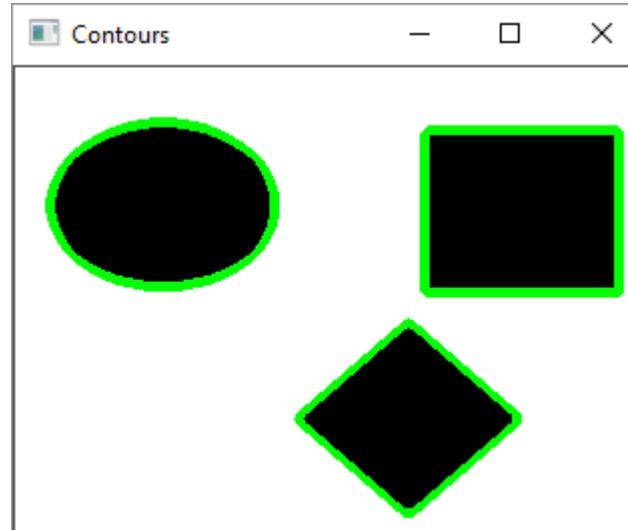
A imagem original, após a detecção inteligente das bordas e uma com contornos desenhados, será exibida em janelas separadas, conforme mostrado aqui -



Após a **detecção astuta da borda**, a imagem será a seguinte -



Depois que os **contornos forem desenhados**, a imagem será a seguinte -



OpenCV Python - correspondência de modelo

A técnica de correspondência de modelo é usada para detectar uma ou mais áreas em uma imagem que corresponde a uma amostra ou imagem de modelo.

A função **Cv.matchTemplate()** no OpenCV é definida para esse propósito e o comando para a mesma é o seguinte:

```
cv.matchTemplate(image, templ, method)
```

Onde **imagem** é a imagem de entrada na qual o padrão **templ** (modelo) deve ser localizado. O parâmetro do método assume um dos seguintes valores -

- cv.TM_CCOEFF,
- cv.TM_CCOEFF_NORMED, cv.TM_CCORR,
- cv.TM_CCORR_NORMED,
- cv.TM_SQDIFF,
- cv.TM_SQDIFF_NORMED

Este método desliza a imagem do modelo sobre a imagem de entrada. Este é um processo semelhante à convolução e compara o modelo e o patch da imagem de entrada na imagem do modelo.

Ele retorna uma imagem em tons de cinza, onde cada pixel indica o quanto ela corresponde ao modelo. Se a imagem de entrada tiver tamanho ($L \times A$) e a imagem do modelo tiver tamanho ($L \times A$), a imagem de saída terá um tamanho de $(L-w+1, H-h+1)$. Portanto, esse retângulo é a sua região do modelo.

Exemplo

No exemplo abaixo, uma imagem com o rosto do jogador de críquete indiano Virat Kohli é usada como modelo para ser combinada com outra imagem que retrata sua fotografia com outro jogador de críquete indiano MSDhoni.

O programa a seguir usa um valor limite de 80% e desenha um retângulo ao redor da face correspondente -

```
import cv2
import numpy as np

img = cv2.imread('Dhoni-and-Virat.jpg',1)
cv2.imshow('Original',img)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```

template = cv2.imread('virat.jpg',0)
cv2.imshow('Template',template)
w,h = template.shape[0], template.shape[1]

matched = cv2.matchTemplate(gray,template, cv2.TM_CCOEFF_NORMED)
threshold = 0.8

loc = np.where( matched >= threshold)

for pt in zip(*loc[::-1]):
    cv2.rectangle(img, pt, (pt[0] + w, pt[1] + h), (0,255,255), 2)

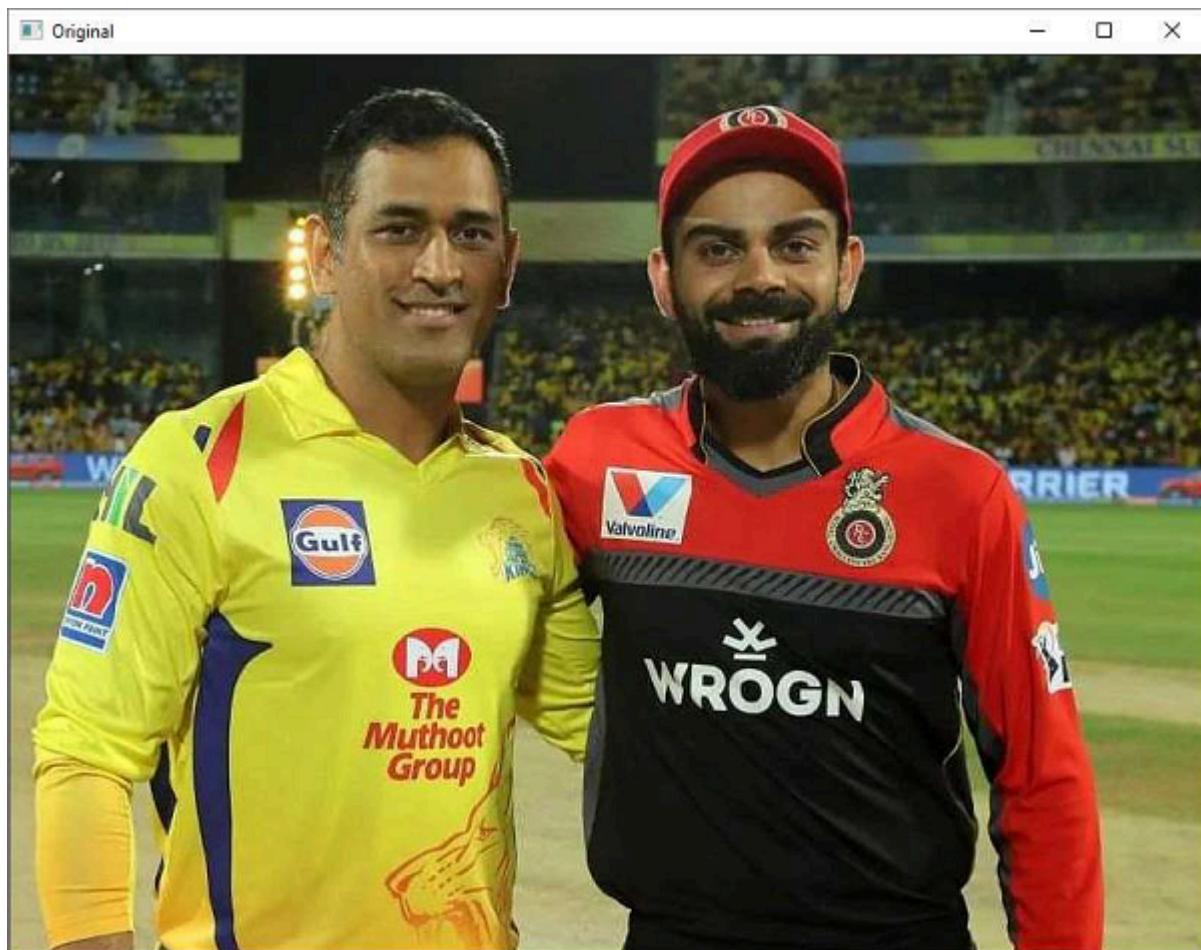
cv2.imshow('Matched with Template',img)

```

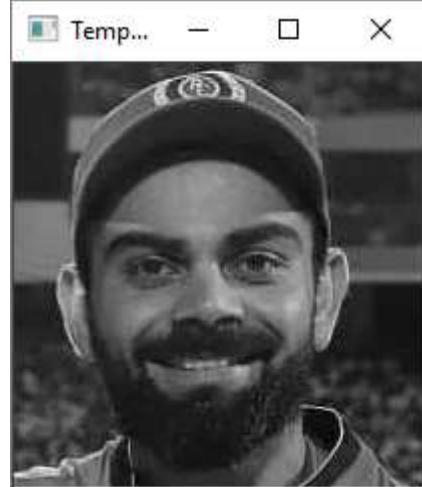
Saída

A imagem original, o modelo e a imagem correspondente do resultado são os seguintes -

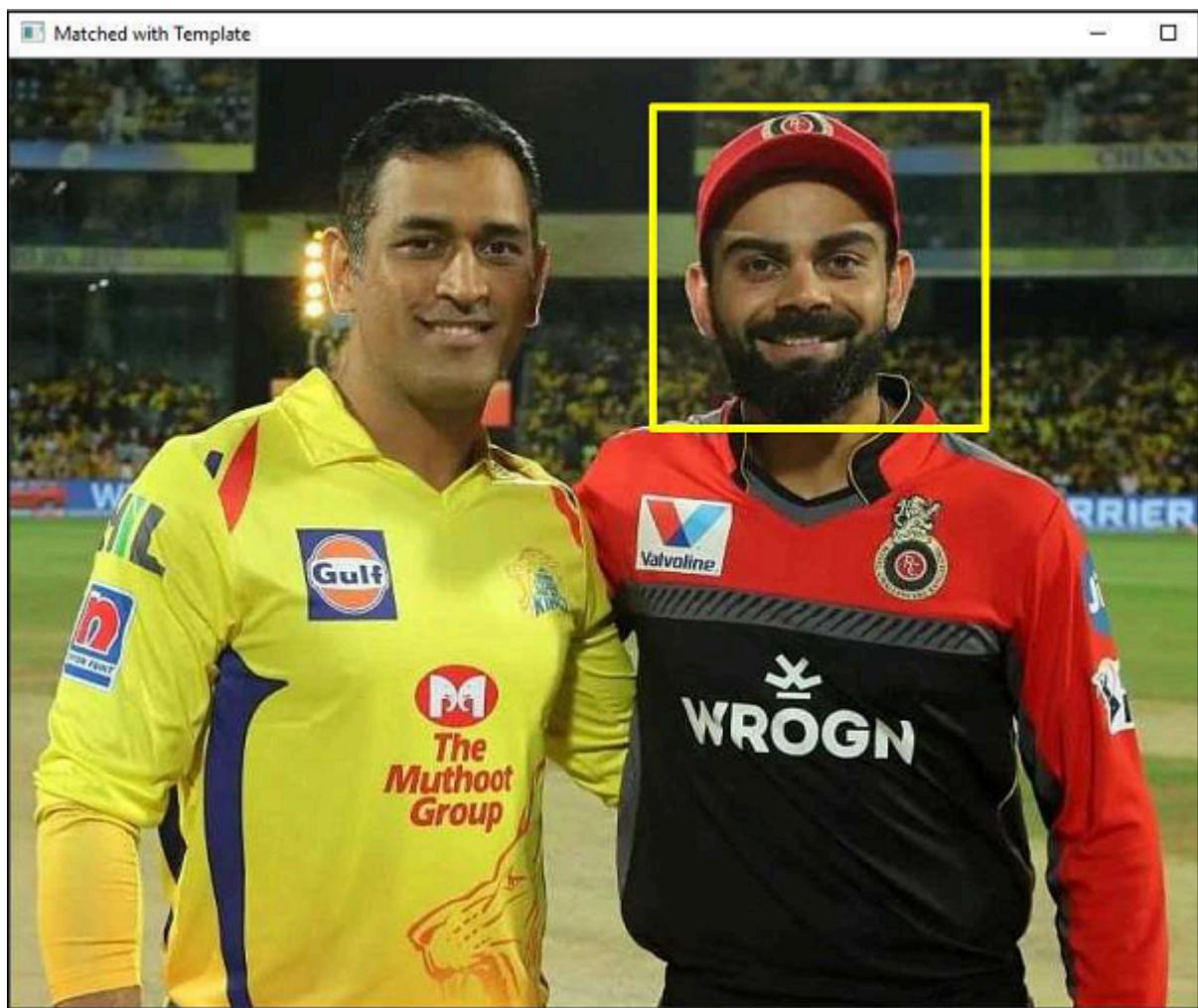
Imagen original



O **modelo** é o seguinte -



A imagem quando **combinada com o modelo** é a seguinte -



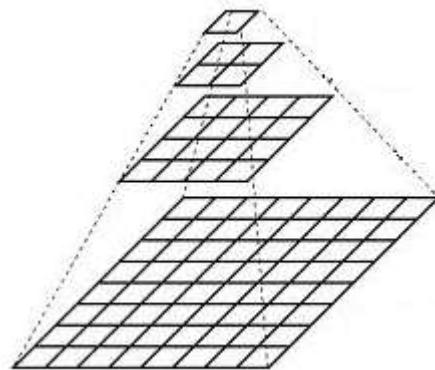
OpenCV Python - Pirâmides de Imagens

Ocasionalmente, podemos precisar converter uma imagem para um tamanho diferente do original. Para isso, você pode aumentar o tamanho da imagem (aumentar o zoom) ou reduzi-la (diminuir o zoom).

Uma pirâmide de imagens é uma coleção de imagens (construídas a partir de uma única imagem original) amostradas sucessivamente um determinado número de vezes.

A pirâmide gaussiana é usada para reduzir imagens de amostra, enquanto a pirâmide laplaciana reconstrói uma imagem amostrada a partir de uma imagem inferior na pirâmide com menos resolução.

Considerando a pirâmide como um conjunto de camadas. A imagem é mostrada abaixo -



A imagem na camada superior da pirâmide é menor em tamanho. Para produzir uma imagem na próxima camada da pirâmide gaussiana, envolvemos uma imagem de nível inferior com um kernel gaussiano.

$$\frac{1}{16} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Agora remova todas as linhas e colunas pares. A imagem resultante terá 1/4 da área de sua antecessora. A iteração desse processo na imagem original produz a pirâmide inteira.

Para aumentar as imagens, as colunas foram preenchidas com zeros. Primeiro, aumente o tamanho da imagem para dobrar o original em cada dimensão, com as novas linhas pares e depois execute uma convolução com o kernel para aproximar os valores dos pixels ausentes.

A função **cv.pyrUp()** dobra o tamanho original e a função **cv.pyrDown()** o diminui para metade.

Exemplo

O programa a seguir chama as funções pyrUp() e pyrDown() dependendo da entrada do usuário "I" ou "o", respectivamente.

Observe que quando reduzimos o tamanho de uma imagem, as informações da imagem são perdidas. Uma vez, diminuímos a escala e se redimensionarmos para o tamanho original, perdemos algumas informações e a resolução da nova imagem é muito menor que a original.

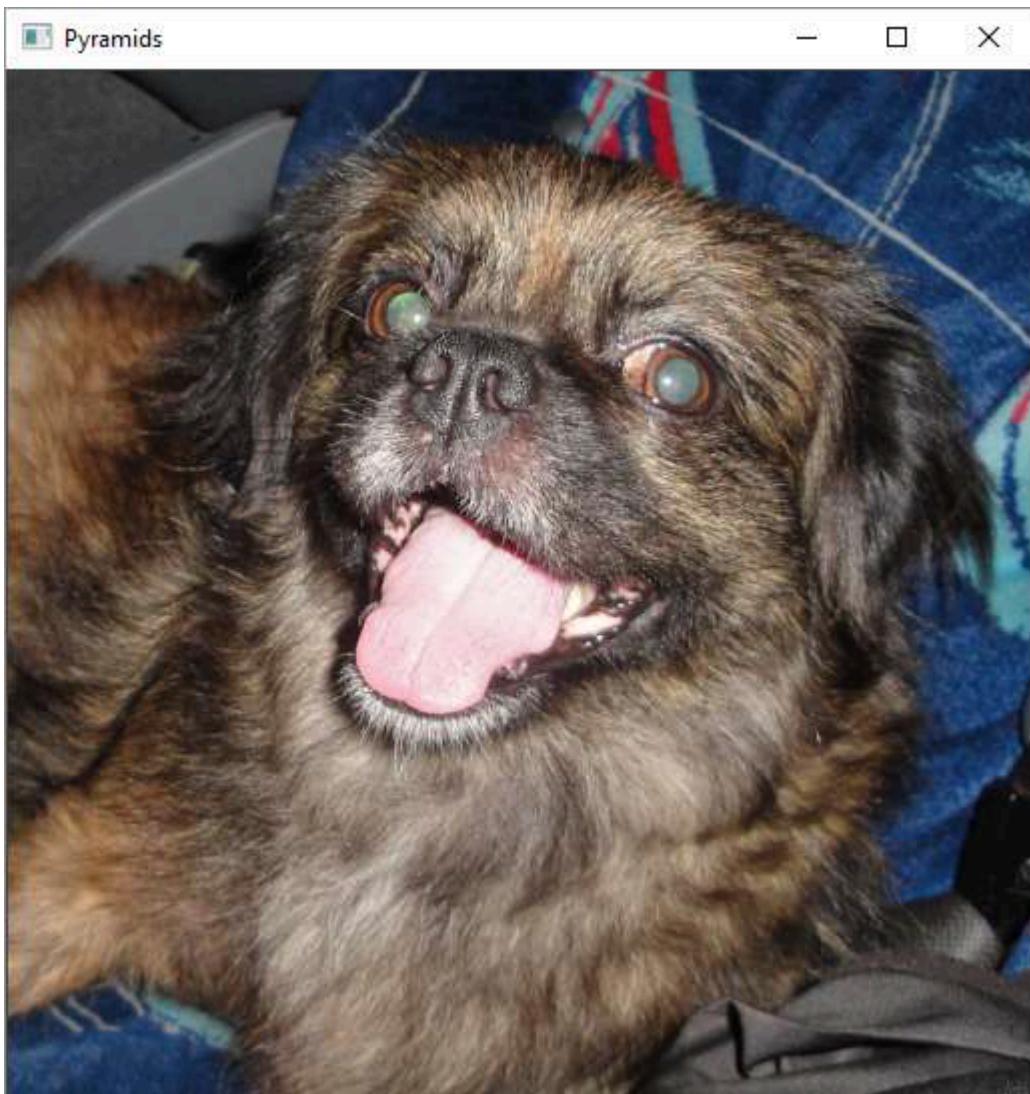
```
import sys
import cv2 as cv

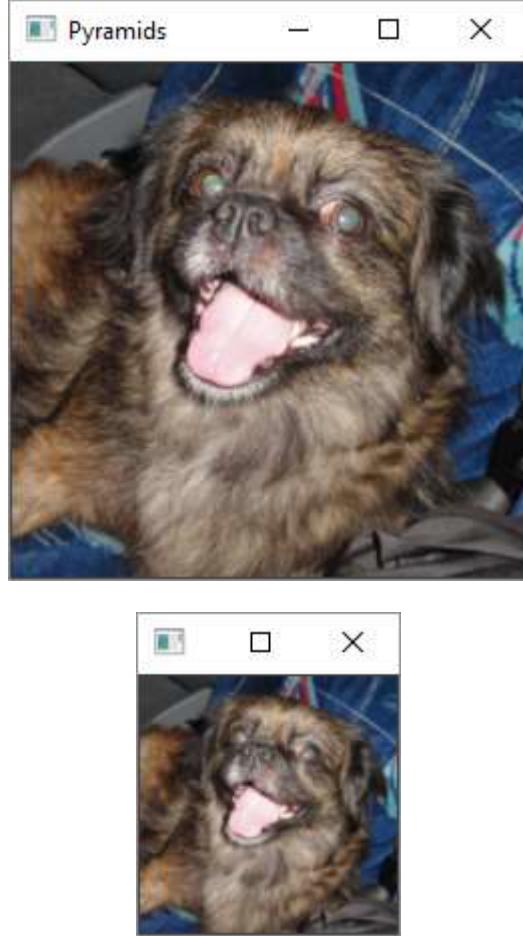
filename = 'chicky_512.png'

src = cv.imread(filename)
```

```
while 1:  
    print ("press 'i' for zoom in 'o' for zoom out esc to stop")  
    rows, cols, _channels = map(int, src.shape)  
    cv.imshow('Pyramids', src)  
    k = cv.waitKey(0)  
  
    if k == 27:  
        break  
  
    elif chr(k) == 'i':  
        src = cv.pyrUp(src, dstsize=(2 * cols, 2 * rows))  
  
    elif chr(k) == 'o':  
        src = cv.pyrDown(src, dstsize=(cols // 2, rows // 2))  
  
cv.destroyAllWindows()
```

Saída





OpenCV Python - adição de imagem

Quando uma imagem é lida pela função `imread()`, o objeto de imagem resultante é na verdade uma matriz bidimensional ou tridimensional, dependendo se a imagem é em escala de cinza ou RGB.

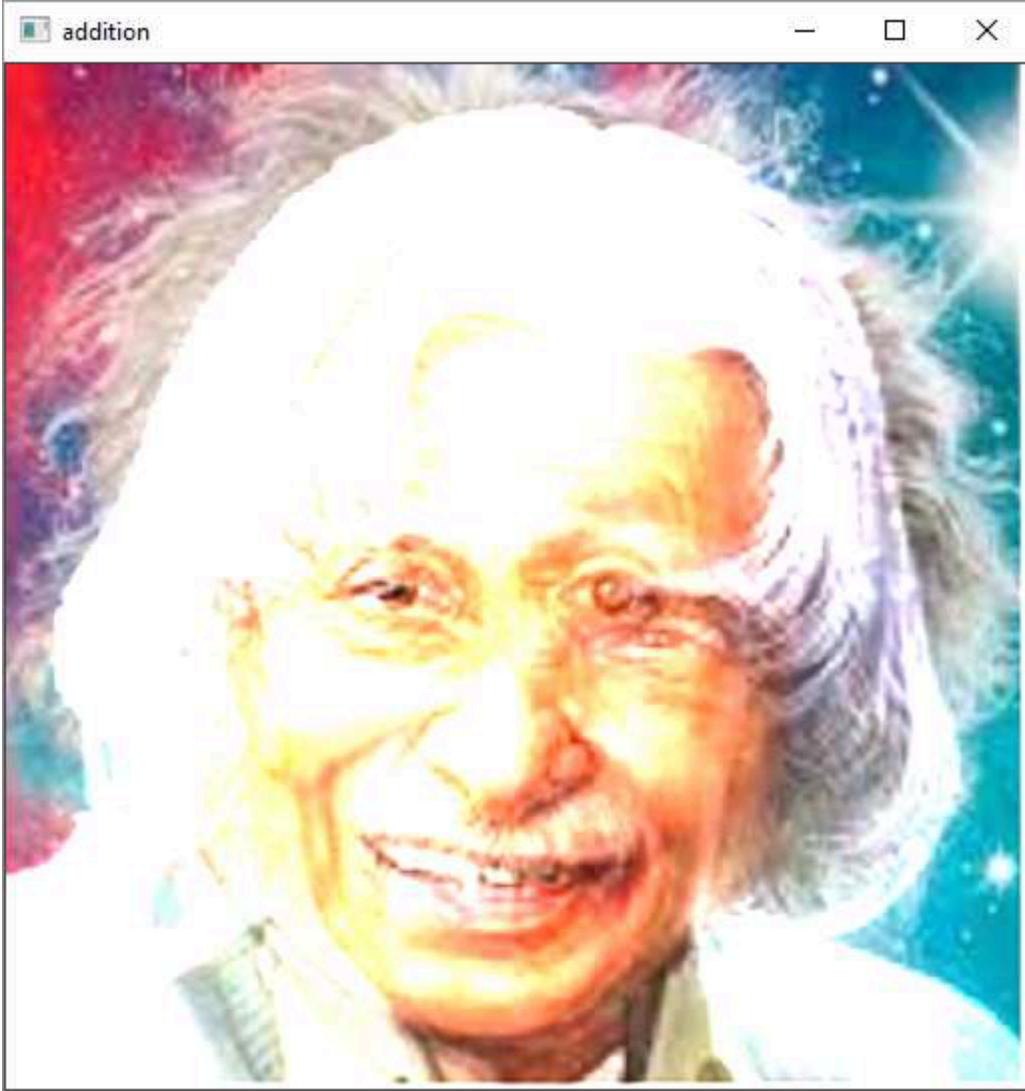
Conseqüentemente, as funções **`cv2.add()`** adicionam duas matrizes de imagem e retorna outra matriz de imagem.

Exemplo

O código a seguir lê duas imagens e realiza sua adição binária -

```
kalam = cv2.imread('kalam.jpg')
einst = cv2.imread('einstein.jpg')
img = cv2.add(kalam, einst)
cv2.imshow('addition', img)
```

Resultado



Em vez de uma adição binária linear, OpenCV possui uma função **addWeighted()** que executa a soma ponderada de dois arrays. O comando para o mesmo é o seguinte

```
Cv2.addWeighted(src1, alpha, src2, beta, gamma)
```

Parâmetros

Os parâmetros da função **addWeighted()** são os seguintes -

- src1 - Primeira matriz de entrada.
- alpha - Peso dos primeiros elementos do array.
- src2 - Segunda matriz de entrada do mesmo tamanho e número de canal da primeira
- beta - Peso dos elementos do segundo array.
- gama - Escalar adicionado a cada soma.

Esta função adiciona as imagens conforme a seguinte equação -

$$g(x) = (1 - \alpha)f_0(x) + \alpha f_1(x)$$

As matrizes de imagem obtidas no exemplo acima são utilizadas para realizar a soma ponderada.

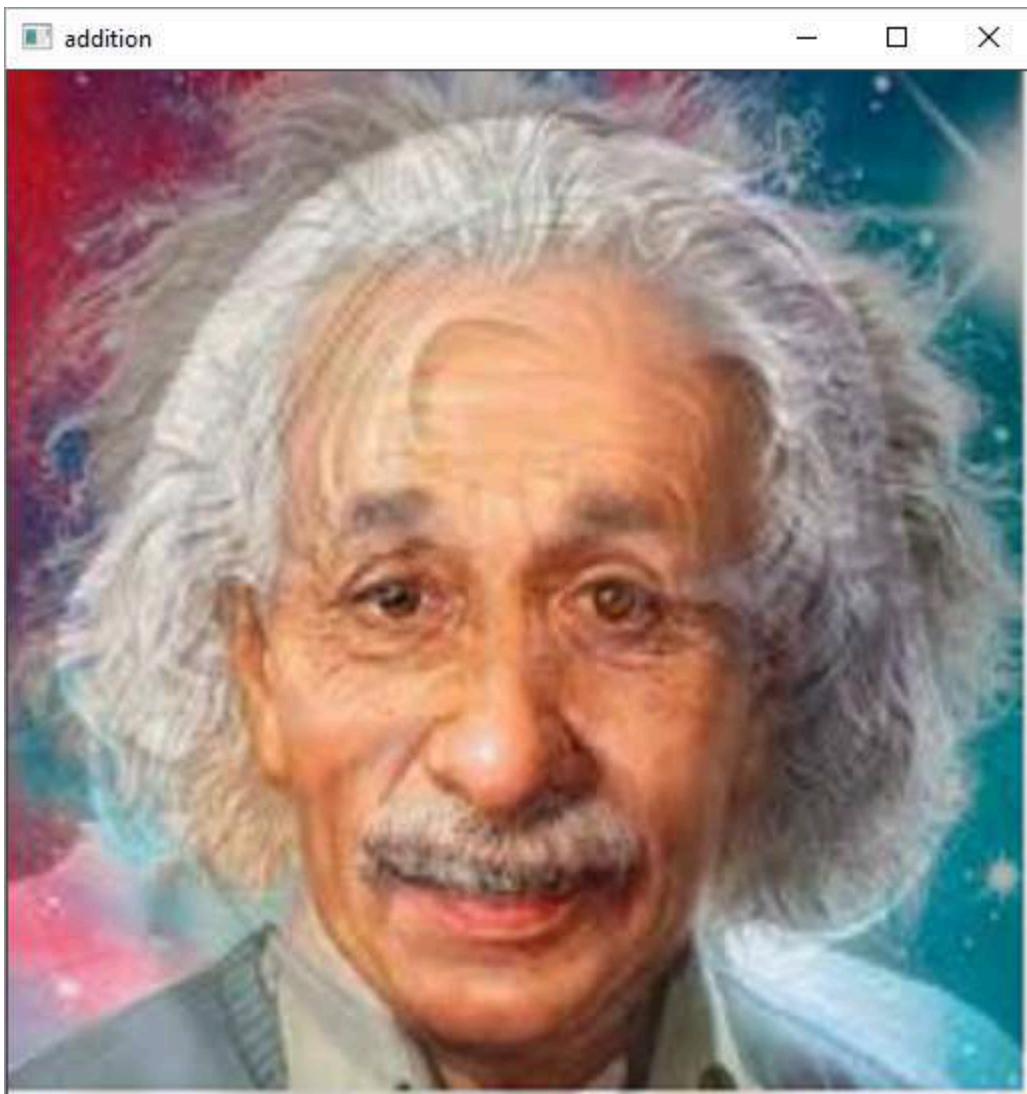
Variando a de 0 -> 1, ocorre uma transição suave de uma imagem para outra, para que elas se misturem.

A primeira imagem recebe um peso de 0,3 e a segunda imagem recebe 0,7. O fator gama é considerado 0.

O comando para a função **addWeighted()** é o seguinte -

```
img = cv2.addWeighted(kalam, 0.3, einst, 0.7, 0)
```

Pode-se observar que a adição de imagens é mais suave em comparação com a adição binária.



OpenCV Python - Combinação de imagens com pirâmides

A descontinuidade das imagens pode ser minimizada pelo uso de pirâmides de imagens. Isso resulta em uma imagem mesclada perfeita.

As seguintes etapas são executadas para alcançar o resultado final -

Primeiro carregue as imagens e encontre as pirâmides gaussianas para ambas. O programa para o mesmo é o seguinte -

```
import cv2
import numpy as np,sys

kalam = cv2.imread('kalam.jpg')
einst = cv2.imread('einstein.jpg')
### generate Gaussian pyramid for first
G = kalam.copy()
gpk = [G]
for i in range(6):
    G = cv2.pyrDown(G)
    gpk.append(G)
# generate Gaussian pyramid for second
G = einst.copy()
gpe = [G]
for i in range(6):
    G = cv2.pyrDown(G)
    gpe.append(G)
```

Das pirâmides gaussianas, obtenha as respectivas pirâmides laplacianas. O programa para o mesmo é o seguinte -

```
# generate Laplacian Pyramid for first
lpk = [gpk[5]]
for i in range(5,0,-1):
    GE = cv2.pyrUp(gpk[i])
    L = cv2.subtract(gpk[i-1],GE)
    lpk.append(L)

# generate Laplacian Pyramid for second
lpe = [gpe[5]]
for i in range(5,0,-1):
    GE = cv2.pyrUp(gpe[i])
    L = cv2.subtract(gpe[i-1],GE)
    lpe.append(L)
```

Em seguida, junte a metade esquerda da primeira imagem com a metade direita da segunda em cada nível das pirâmides. O programa para o mesmo é o seguinte -

```
# Now add Left and right halves of images in each Level
LS = []
```

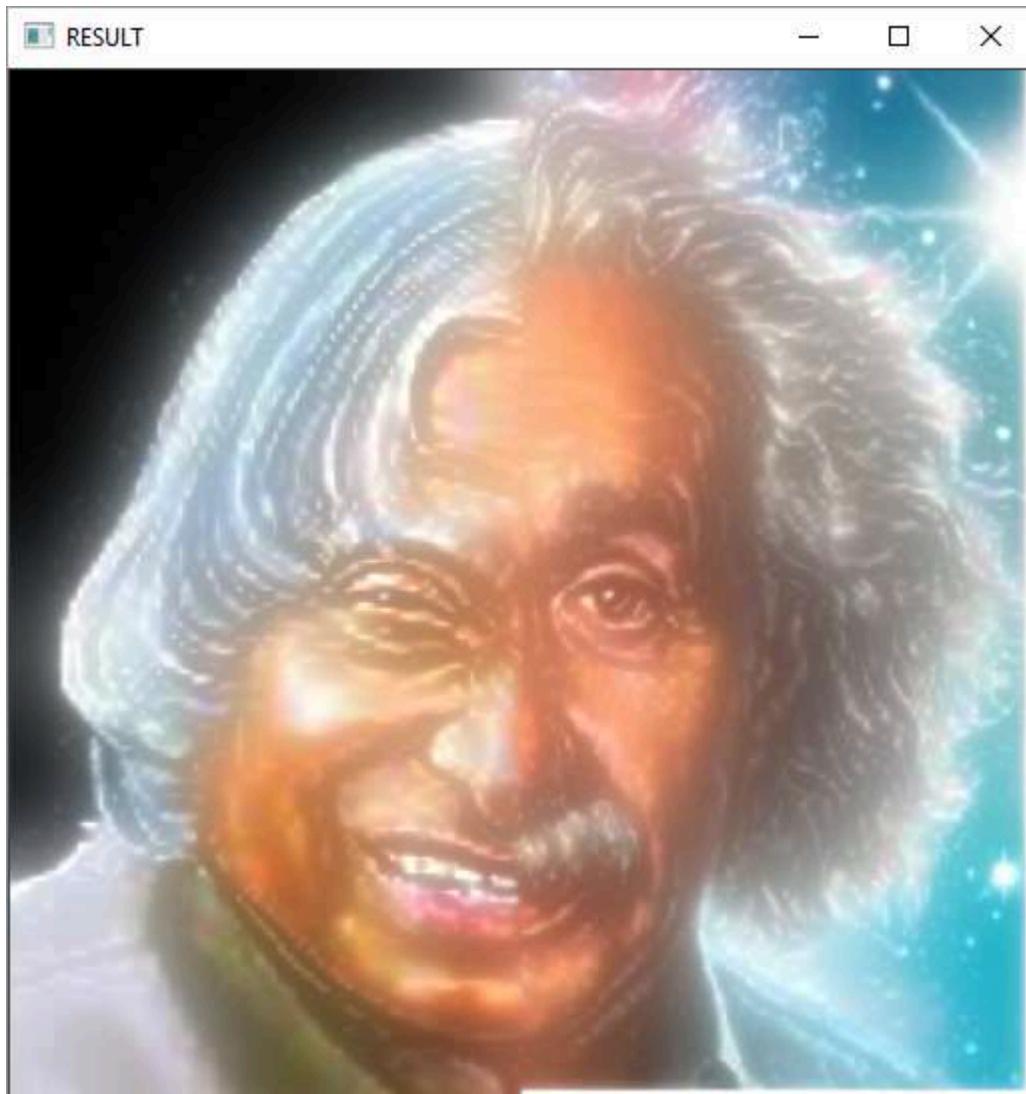
```
for la,lb in zip(lpk,lpe):
    rows,cols,dpt = la.shape
    ls = np.hstack((la[:,0:int(cols/2)], lb[:,int(cols/2):]))
    LS.append(ls)
```

Finalmente, reconstrua a imagem desta pirâmide conjunta. O programa para o mesmo é fornecido abaixo -

```
ls_ = LS[0]
for i in range(1,6):
    ls_ = cv2.pyrUp(ls_)
    ls_ = cv2.add(ls_, LS[i])
cv2.imshow('RESULT',ls_)
```

Saída

O resultado combinado deve ser o seguinte -



OpenCV Python - Transformada de Fourier

A Transformada de Fourier é usada para transformar uma imagem de seu domínio espacial para seu domínio de frequência, decompondo-a em seus componentes seno e cosseno.

No caso de imagens digitais, os valores básicos de uma imagem em escala de cinza geralmente estão entre zero e 255. Portanto, a Transformada de Fourier também precisa ser uma **Transformada Discreta de Fourier (DFT)**. É usado para encontrar o domínio da frequência.

Matematicamente, a Transformada de Fourier de uma imagem bidimensional é representada da seguinte forma -

$$F(k, l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i, j) e^{-i2\pi(\frac{ki}{N}, \frac{lj}{N})}$$

Se a amplitude variar tão rapidamente em um curto espaço de tempo, podemos dizer que é um sinal de alta frequência. Se variar lentamente, é um sinal de baixa frequência.

No caso de imagens, a amplitude varia drasticamente nos pontos de borda, ou ruídos. Portanto, bordas e ruídos são conteúdos de alta frequência em uma imagem. Se não houver muitas mudanças na amplitude, é um componente de baixa frequência.

OpenCV fornece as funções **cv.dft()** e **cv.idft()** para esta finalidade.

`cv.dft()` executa uma transformação discreta de Fourier de uma matriz de ponto flutuante 1D ou 2D. O comando para o mesmo é o seguinte -

```
cv.dft(src, dst, flags)
```

Aqui,

- `src` - Matriz de entrada que pode ser real ou complexa.
- `dst` - Matriz de saída cujo tamanho e tipo dependem dos sinalizadores.
- `flags` - Sinalizadores de transformação, representando uma combinação de DftFlags.

`cv.idft()` calcula a Transformada Discreta de Fourier inversa de um array 1D ou 2D. O comando para o mesmo é o seguinte -

```
cv.idft(src, dst, flags)
```

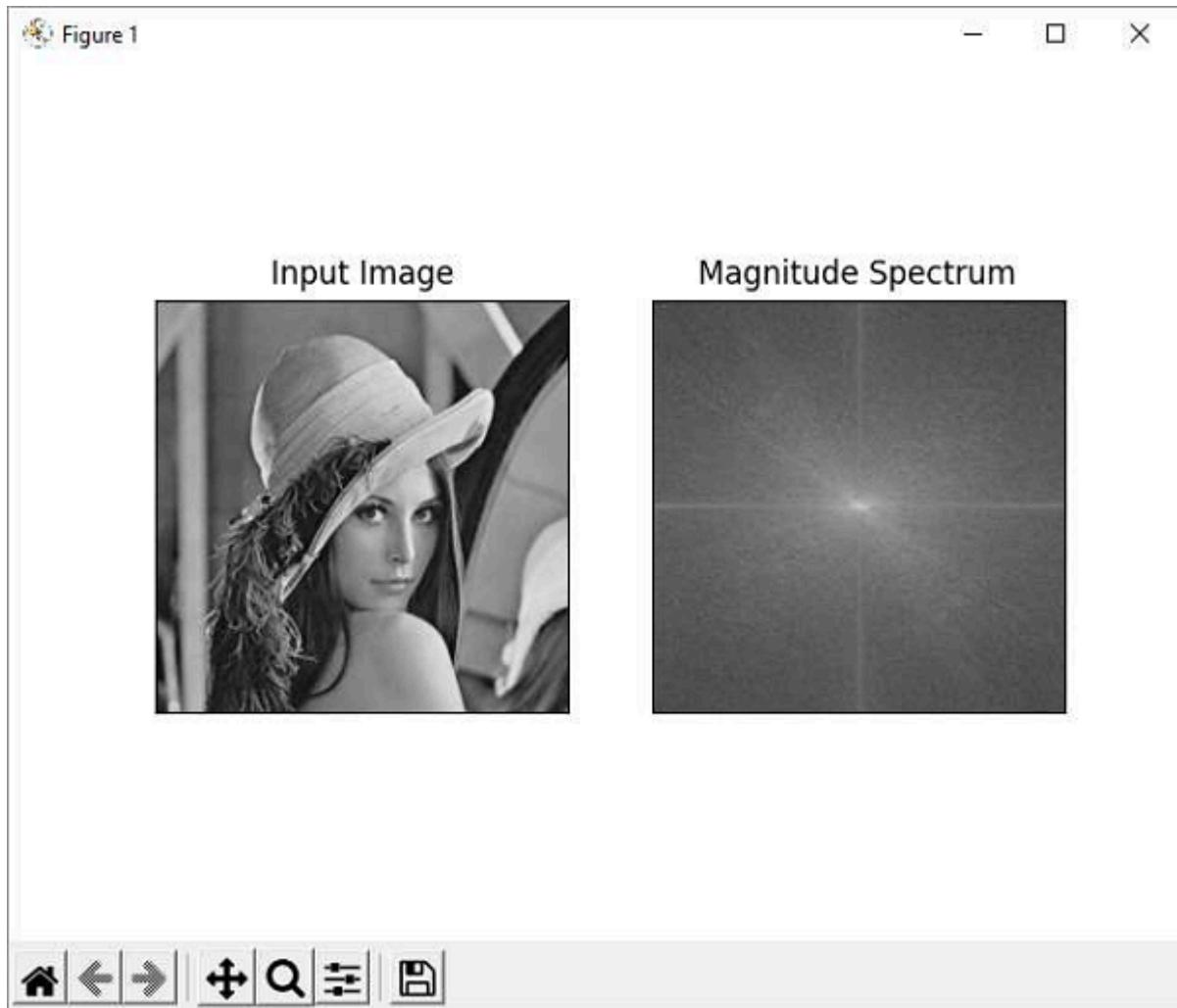
Para obter uma transformada discreta de Fourier, a imagem de entrada é convertida para o tipo de dados `np.float32`. A transformada obtida é então usada para deslocar o componente de frequência zero para o centro do espectro, a partir do qual o espectro de magnitude é calculado.

Exemplo

A seguir está o programa usando Matplotlib, plotamos a imagem original e o espectro de magnitude -

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('lena.jpg',0)
dft = cv.dft(np.float32(img),flags = cv.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20*np.log(cv.magnitude(dft_shift[:, :, 0], dft_shift[:, :, 1]))
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```

Saída



OpenCV Python - Capturar vídeo da câmera

Usando a função **VideoCapture()** na biblioteca OpenCV, é muito fácil capturar uma transmissão ao vivo de uma câmera na janela OpenCV.

Esta função precisa de um índice de dispositivo como parâmetro. Seu computador pode ter várias câmeras conectadas. Eles são enumerados por um índice começando em 0 para webcam integrada. A função retorna um objeto VideoCapture

```
cam = cv.VideoCapture(0)
```

Depois que a câmera é aberta, podemos ler quadros sucessivos dela com a ajuda da função **read()**

```
ret,frame = cam.read()
```

A função **read()** lê o próximo quadro disponível e um valor de retorno (True/False). Este quadro agora é renderizado no espaço de cores desejado com a função **cvtColor()** e exibido na janela OpenCV.

```
img = cv.cvtColor(frame, cv.COLOR_BGR2RGB)  
# Display the resulting frame  
cv.imshow('frame', img)
```

Para capturar o quadro atual em um arquivo de imagem, você pode usar a função **imwrite()**

```
cv2.imwrite("capture.png", img)
```

Para salvar a transmissão ao vivo da câmera em um arquivo de vídeo, OpenCV fornece uma função **VideoWriter()**.

```
cv.VideoWriter( filename, fourcc, fps, frameSize)
```

O parâmetro **fourcc** é um código padronizado para codecs de vídeo. OpenCV suporta vários codecs como DIVX, XVID, MJPG, X264 etc. Os parâmetros **fps** e **framesize** dependem do dispositivo de captura de vídeo.

A função **VideoWriter()** retorna um objeto de fluxo **VideoWrite**, no qual os quadros capturados são gravados sucessivamente em um loop. Por fim, libere os objetos **frame** e **VideoWriter** para finalizar a criação do vídeo.

Exemplo

O exemplo a seguir lê o feed ao vivo da webcam integrada e o salva no arquivo **output.avi**.

```
import cv2 as cv  
cam = cv.VideoCapture(0)
```

```

cc = cv.VideoWriter_fourcc(*'XVID')
file = cv.VideoWriter('output.avi', cc, 15.0, (640, 480))
if not cam.isOpened():
    print("error opening camera")
    exit()
while True:
    # Capture frame-by-frame
    ret, frame = cam.read()
    # if frame is read correctly ret is True
    if not ret:
        print("error in retrieving frame")
        break
    img = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
    cv.imshow('frame', img)
    file.write(img)

    if cv.waitKey(1) == ord('q'):
        break

cam.release()
file.release()
cv.destroyAllWindows()

```

OpenCV Python - Reproduzir vídeo do arquivo

A função **VideoCapture()** também pode recuperar quadros de um arquivo de vídeo em vez de uma câmera. Portanto, substituímos apenas o índice da câmera pelo nome do arquivo de vídeo a ser reproduzido na janela OpenCV.

```
video=cv2.VideoCapture(file)
```

Embora isso deva ser suficiente para iniciar a renderização de um arquivo de vídeo, se ele estiver acompanhado de som. O som não irá tocar junto. Para isso, você precisará instalar o módulo ffpypyplayer.

FFPyPlayer

FFPyPlayer é uma ligação python para a biblioteca FFmpeg para reproduzir e gravar arquivos de mídia. Para instalar, use o utilitário pip installer usando o seguinte comando.

```
pip3 install ffpypyplayer
```

O método **get_frame()** do objeto MediaPlayer neste módulo retorna o quadro de áudio que será reproduzido junto com cada quadro lido do arquivo de vídeo.

A seguir está o código completo para reproduzir um arquivo de vídeo junto com seu áudio -

```
import cv2

from ffpypyplayer.player import MediaPlayer
file="video.mp4"

video=cv2.VideoCapture(file)
player = MediaPlayer(file)
while True:
    ret, frame=video.read()
    audio_frame, val = player.get_frame()
    if not ret:
        print("End of video")
        break
    if cv2.waitKey(1) == ord("q"):
        break
    cv2.imshow("Video", frame)
    if val != 'eof' and audio_frame is not None:
        #audio
        img, t = audio_frame
video.release()
cv2.destroyAllWindows()
```

OpenCV Python - Extraia imagens de vídeo

Um vídeo nada mais é do que uma sequência de frames e cada frame é uma imagem. Ao usar OpenCV, todos os frames que compõem um arquivo de vídeo podem ser extraídos executando a função **imwrite()** até o final do vídeo.

A função **cv2.read()** retorna o próximo quadro disponível. A função também fornece um valor de retorno que continua verdadeiro até o final do fluxo. Aqui, um contador é incrementado dentro do loop e usado como nome de arquivo.

O programa a seguir demonstra como extrair imagens do vídeo -

```
import cv2
import os

cam = cv2.VideoCapture("video.avi")

frameno = 0
while(True):
```

```

ret, frame = cam.read()
if ret:
    # if video is still left continue creating images
    name = str(frameno) + '.jpg'
    print ('new frame captured...' + name)

    cv2.imwrite(name, frame)
    frameno += 1
else:
    break

cam.release()
cv2.destroyAllWindows()

```

OpenCV Python - Vídeo de imagens

No capítulo anterior, usamos a função VideoWriter() para salvar a transmissão ao vivo de uma câmera como um arquivo de vídeo. Para juntar várias imagens em um vídeo, usaremos a mesma função.

Primeiro, certifique-se de que todas as imagens necessárias estejam em uma pasta. A função glob() do Python no módulo glob integrado cria um array de imagens para que possamos iterá-lo.

Leia o objeto de imagem das imagens na pasta e anexe a uma matriz de imagens.

O programa a seguir explica como juntar várias imagens em um vídeo -

```

import cv2
import numpy as np
import glob

img_array = []
for filename in glob.glob('*.png'):
    img = cv2.imread(filename)
    height, width, layers = img.shape
    size = (width,height)
    img_array.append(img)

```

Crie um stream de vídeo usando a função VideoWriter() para gravar o conteúdo do array de imagens nele. Abaixo está o programa para o mesmo.

```

out = cv2.VideoWriter('video.avi',cv2.VideoWriter_fourcc(*'DIVX'), 15, size)

for i in range(len(img_array)):

```

```
out.write(img_array[i])
out.release()
```

Você deve encontrar o arquivo chamado '**video.avi**' na pasta atual.

OpenCV Python - Detecção de rosto

OpenCV usa classificadores em cascata baseados em recursos **Haar para a detecção de objetos**. É um algoritmo baseado em aprendizado de máquina, onde uma função em cascata é treinada a partir de muitas imagens positivas e negativas. Em seguida, é usado para detectar objetos em outras imagens. O algoritmo utiliza o conceito de Cascata de Classificadores.

Classificadores pré-treinados para rosto, olhos, etc. podem ser baixados em <https://github.com>

Para o exemplo a seguir, baixe e **copie haarcascade_frontalface_default.xml** e **haarcascade_eye.xml** deste URL. Em seguida, carregue nossa imagem de entrada para ser usada para detecção de rosto no modo tons de cinza.

O método **DetectMultiScale()** da classe CascadeClassifier detecta objetos na imagem de entrada. Ele retorna as posições das faces detectadas na forma de retângulo e suas dimensões (x,y,w,h). Assim que obtivermos essas localizações, poderemos usá-lo para detecção de olhos, já que os olhos estão sempre no rosto!

Exemplo

O código completo para detecção de rosto é o seguinte -

```
import numpy as np
import cv2

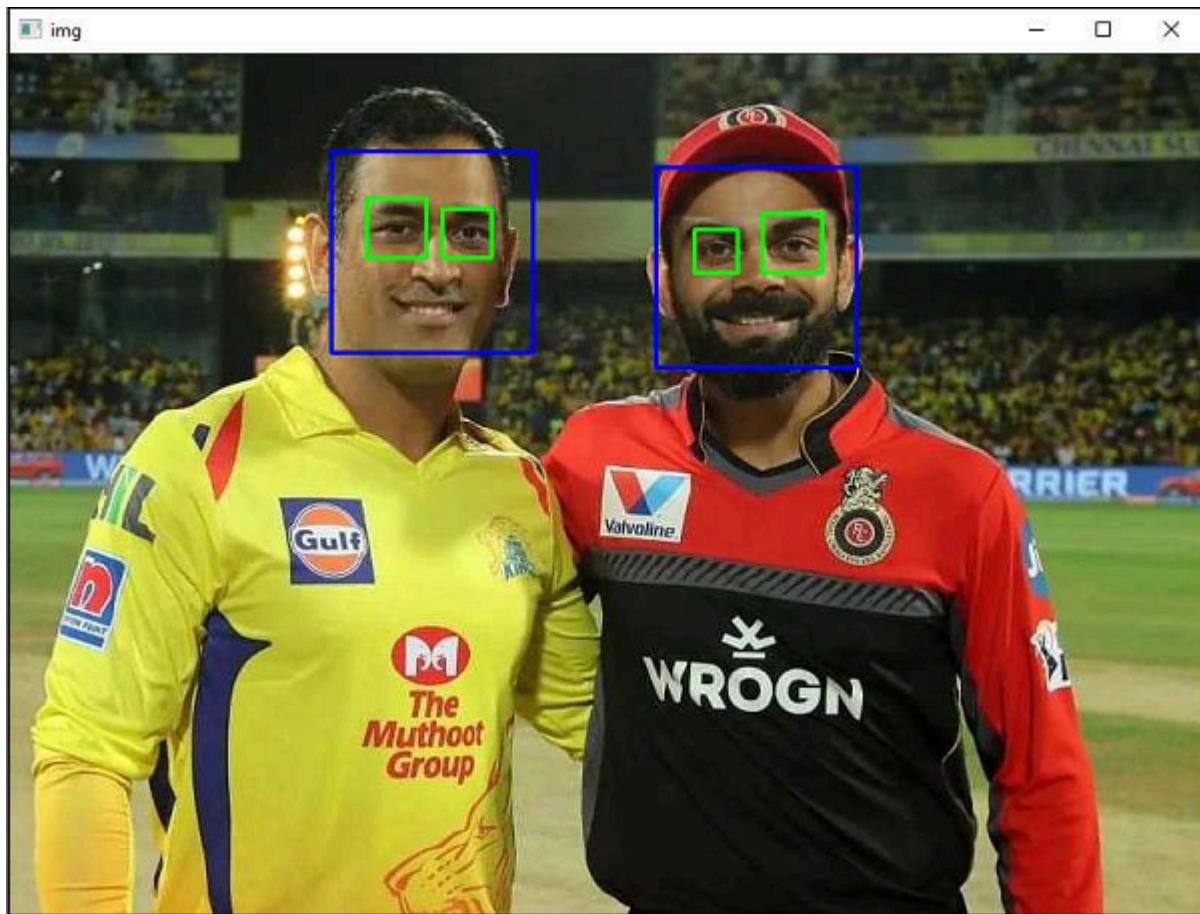
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

img = cv2.imread('Dhoni-and-virat.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
for (x,y,w,h) in faces:
    img = cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
```

```
cv2.imshow('img',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Saída

Você obterá retângulos desenhados ao redor dos rostos na imagem de entrada, conforme mostrado abaixo -



OpenCV Python - Meanshift e Camshift

Neste capítulo, vamos aprender sobre o desvio médio e o deslocamento cam no OpenCV-Python. Primeiro, vamos entender o que é mudança de média.

Mudança média

O algoritmo de deslocamento médio identifica locais no conjunto de dados com uma alta concentração de pontos de dados, ou clusters. O algoritmo coloca um kernel em cada ponto de dados e os soma para fazer uma **estimativa de densidade do kernel** (KDE).

O KDE terá locais com alta e baixa densidade de pontos de dados, respeitosamente. Meanshift é um método muito útil para rastrear um objeto específico dentro de um vídeo.

Cada instância do vídeo é verificada na forma de distribuição de pixels naquele quadro. Uma janela inicial como região de interesse (**ROI**) é geralmente um quadrado ou um círculo. Para

isso, as posições são especificadas por hardcoding e a área de distribuição máxima de pixels é identificada.

A janela ROI se move em direção à região de distribuição máxima de pixels à medida que o vídeo é executado. A direção do movimento depende da diferença entre o centro da nossa janela de rastreamento e o centróide de todos os k-pixels dentro dessa janela.

Para usar Meanshift no OpenCV, primeiro encontre o histograma (do qual apenas Hue é considerado) de nosso alvo e possa retroprojetar seu alvo em cada quadro para cálculo de Meanshift. Também precisamos fornecer uma localização inicial da janela ROI.

Calculamos repetidamente a retroprojeção do histograma e calculamos o Meanshift para obter a nova posição da janela da trilha. Posteriormente, desenhamos um retângulo usando suas dimensões na moldura.

Funções

As funções openCV usadas no programa são -

- **cv.calcBackProject()** - Calcula a retroprojeção de um histograma.
- **cv.meanShift()** - Retroprojeção do histograma do objeto usando janela de pesquisa inicial e critérios de parada para o algoritmo de pesquisa iterativo.

Exemplo

Aqui está o programa de exemplo Meanshift -

```
import numpy as np
import cv2 as cv

cap = cv.VideoCapture('traffic.mp4')

ret,frame = cap.read()

# dimensions of initial location of window
x, y, w, h = 300, 200, 100, 50
tracker = (x, y, w, h)

region = frame[y:y+h, x:x+w]
hsv_reg = cv.cvtColor(region, cv.COLOR_BGR2HSV)
mask = cv.inRange(hsv_reg, np.array((0., 60., 32.)), np.array((180., 255., 255.)))
reg_hist = cv.calcHist([hsv_reg], [0], mask, [180], [0,180])
cv.normalize(reg_hist, reg_hist, 0, 255, cv.NORM_MINMAX)

# Setup the termination criteria
criteria = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )
```

```

while(1):
    ret, frame = cap.read()

    if ret == True:
        hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        dst = cv.calcBackProject([hsv],[0],reg_hist,[0,180],1)

        # apply meanshift
        ret, tracker = cv.meanShift(dst, tracker, criteria)

        # Draw it on image
        x,y,w,h = tracker
        img = cv.rectangle(frame, (x,y), (x+w,y+h), 255,2)
        cv.imshow('img',img)

    k = cv.waitKey(30) & 0xff
    if k==115:
        cv.imwrite('capture.png', img)
    if k == 27:
        break

```

À medida que o programa é executado, o algoritmo Meanshift move nossa janela para o novo local com densidade máxima.

Saída

Aqui está um instantâneo da janela em movimento -



Camshift

Uma das desvantagens do algoritmo Meanshift é que o tamanho da janela de rastreamento permanece o mesmo, independentemente da distância do objeto à câmera. Além disso, a janela rastreará o objeto somente se ele estiver na região desse objeto. Portanto, devemos fazer a codificação manual da janela e isso deve ser feito com cuidado.

A solução para estes problemas é dada pelo CAMshift (significa **Continuously Adaptive Meanshift**). Depois que o desvio médio converge, o algoritmo Camshift atualiza o tamanho da janela de modo que a janela de rastreamento possa mudar de tamanho ou até mesmo girar para se correlacionar melhor com os movimentos do objeto rastreado.

No código a seguir, em vez da função meanshift(), a função camshift() é usada.

Primeiro, ele encontra o centro do objeto usando meanShift e depois ajusta o tamanho da janela e encontra a rotação ideal. A função retorna a posição, tamanho e orientação do objeto. A posição é desenhada no quadro usando a função de desenho polylines().

Exemplo

Em vez da função Meanshift() no programa anterior, use a função CamShift() conforme abaixo -

```
# apply camshift
ret, tracker = cv.CamShift(dst, tracker, criteria)
pts = cv.boxPoints(ret)
pts = np.int0(pts)
img = cv.polylines(frame,[pts],True, 255,2)
cv.imshow('img',img)
```

Saída

Um instantâneo do resultado do programa modificado mostrando o retângulo girado da janela de rastreamento é o seguinte -



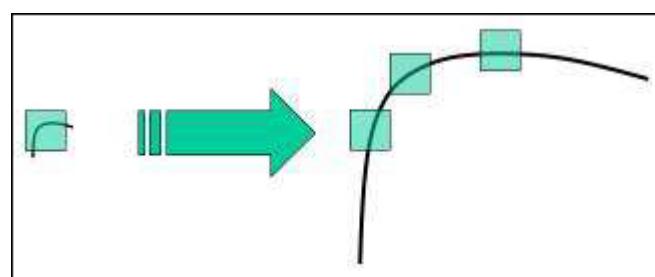
OpenCV Python - Detecção de recursos

No contexto do processamento de imagens, os recursos são representações matemáticas de áreas-chave de uma imagem. São as representações vetoriais do conteúdo visual de uma imagem.

Os recursos permitem realizar operações matemáticas sobre eles. Várias aplicações de visão computacional incluem detecção de objetos, estimativa de movimento, segmentação, alinhamento de imagens, etc.

As características proeminentes em qualquer imagem incluem bordas, cantos ou partes de uma imagem. OpenCV suporta algoritmos de detecção **de canto Harris** e **detecção de canto Shi-Tomasi**. A biblioteca OpenCV também fornece funcionalidade para implementar **SIFT** (Scale-Invariant Feature Transform), **SURF** (Speeded-Up Robust Features) e algoritmo FAST para detecção de cantos.

Os algoritmos de Harris e Shi-Tomasi são invariantes à rotação. Mesmo que a imagem seja girada, podemos encontrar os mesmos cantos. Mas quando uma imagem é ampliada, um canto pode não ser um canto da imagem. A figura abaixo mostra o mesmo.



O novo algoritmo de D.Lowe, **Scale Invariant Feature Transform** (SIFT) extrai os pontos-chave e calcula seus descritores.

Isso é conseguido seguindo as etapas -

- Detecção de extremos no espaço de escala.
- Localização de ponto-chave.
- Atribuição de orientação.
- Descritor de ponto-chave.
- Correspondência de pontos-chave.

No que diz respeito à implementação do SIFT no OpenCV, ele começa carregando uma imagem e convertendo-a em escala de cinza. A função **cv.SHIFT_create()** cria um objeto SIFT.

Exemplo

Chamar seu método **detect()** obtém pontos-chave que são desenhados no topo da imagem original. O código a seguir implementa este procedimento

```
import numpy as np
import cv2 as cv
img = cv.imread('home.jpg')
gray= cv.cvtColor(img, cv.COLOR_BGR2GRAY)
sift = cv.SIFT_create()
kp = sift.detect(gray,None)
img=cv.drawKeypoints(gray,kp,img)
cv.imwrite('keypoints.jpg',img)
```

Saída

A imagem original e aquela com pontos-chave desenhados são mostradas abaixo -

Esta é uma **imagem original** .



A imagem fornecida abaixo é aquela **com pontos-chave** -



OpenCV Python - correspondência de recursos

OpenCV fornece duas técnicas para correspondência de recursos. Combinação de força bruta e técnica de correspondência FLANN.

Exemplo

O exemplo a seguir usa método de força bruta

```
import numpy as np
import cv2

img1 = cv2.imread('lena.jpg')
img2 = cv2.imread('lena-test.jpg')

# Convert it to grayscale
img1_bw = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2_bw = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)

orb = cv2.ORB_create()

queryKeypoints, queryDescriptors = orb.detectAndCompute(img1_bw, None)
trainKeypoints, trainDescriptors = orb.detectAndCompute(img2_bw, None)

matcher = cv2.BFMatcher()
matches = matcher.match(queryDescriptors, trainDescriptors)

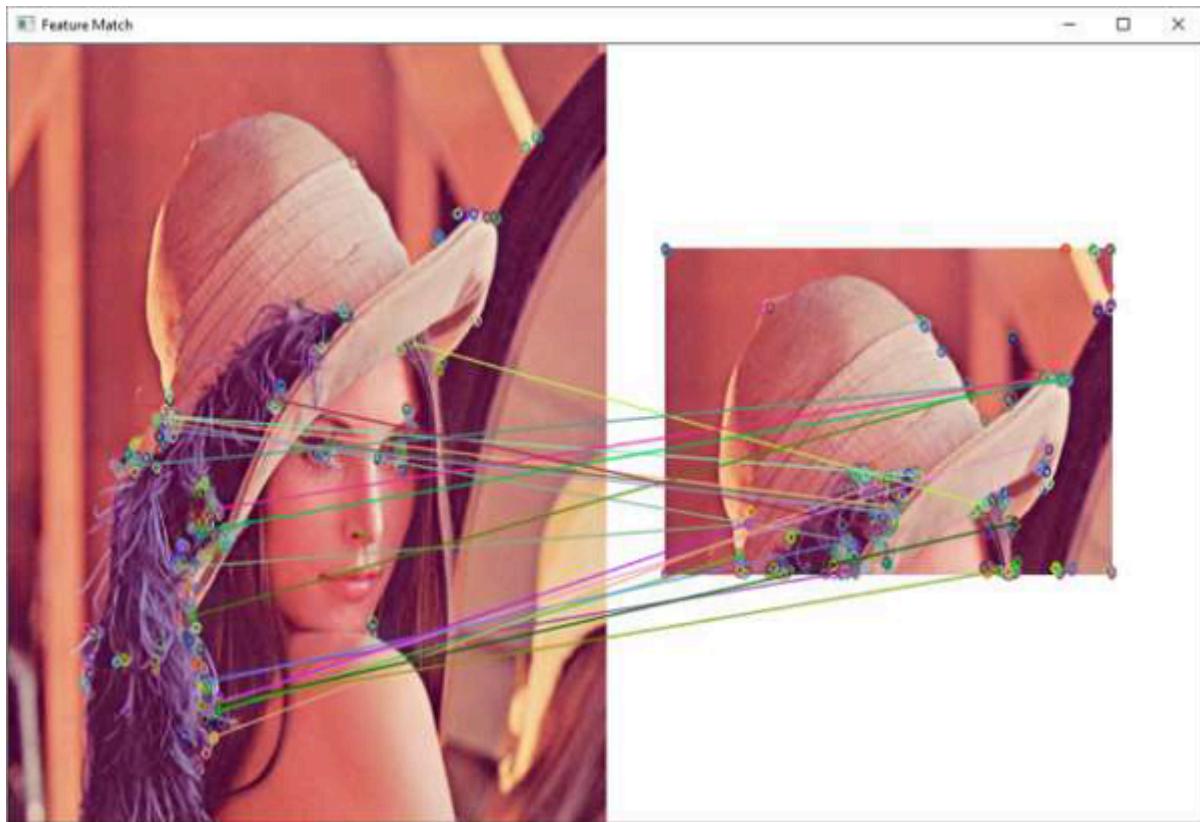
img = cv2.drawMatches(img1, queryKeypoints,
```

```
img2, trainKeypoints, matches[:20],None)
```

```
img = cv2.resize(img, (1000,650))
```

```
cv2.imshow("Feature Match", img)
```

Saída



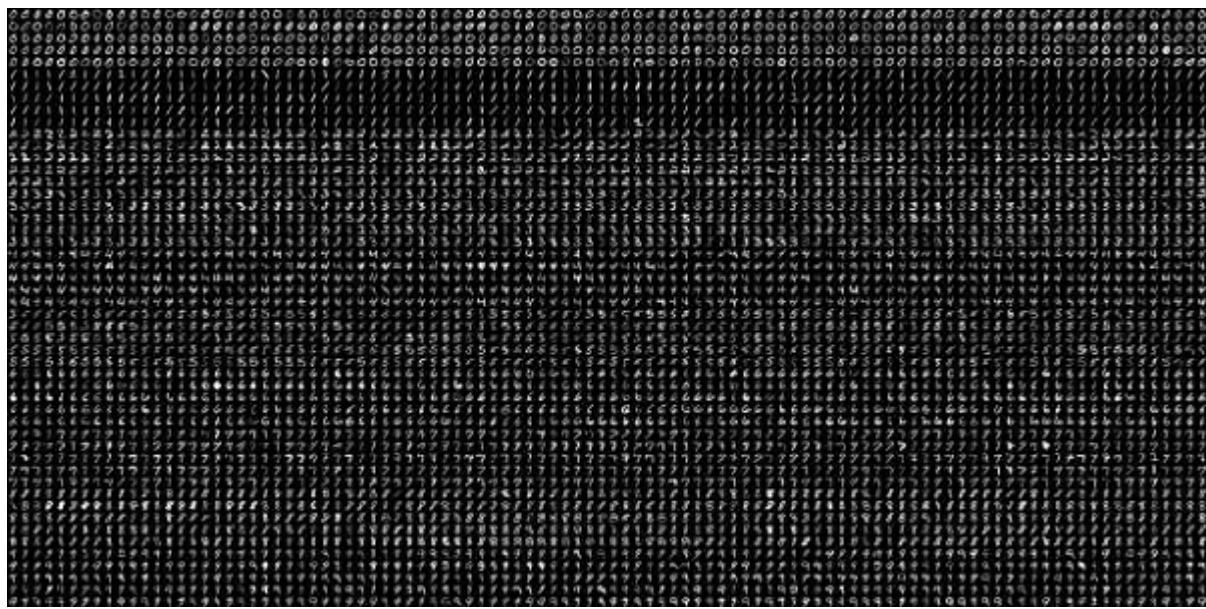
OpenCV Python - Reconhecimento de dígitos com KNN

KNN, que significa **K-Nearest Neighbour**, é um algoritmo de aprendizado de máquina baseado em aprendizado supervisionado. Ele tenta colocar um novo ponto de dados na categoria que seja mais semelhante às categorias disponíveis. Todos os dados disponíveis são classificados em categorias distintas e um novo ponto de dados é colocado em uma delas com base na similaridade.

O algoritmo KNN funciona segundo o seguinte princípio -

- Escolha preferencialmente um número ímpar como K para o número de vizinhos a serem verificados.
- Calcule sua distância euclidiana.
- Pegue os K vizinhos mais próximos de acordo com a distância euclidiana calculada.
- conte o número de pontos de dados em cada categoria.
- Categoria com máximo de pontos de dados é a categoria na qual o novo ponto de dados é classificado.

Como exemplo de implementação do algoritmo KNN usando OpenCV, usaremos a seguinte imagem digits.png composta por 5.000 imagens de dígitos manuscritos, cada uma com 20X20 pixels.



A primeira tarefa é dividir esta imagem em 5.000 dígitos. Este é o nosso conjunto de recursos. Converta-o em um array NumPy. O programa é fornecido abaixo -

```
import numpy as np
import cv2

image = cv2.imread('digits.png')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

fset=[]
for i in np.vsplit(gray,50):
    x=np.hsplit(i,100)
    fset.append(x)

NP_array = np.array(fset)
```

Agora dividimos esses dados em conjunto de treinamento e conjunto de teste, cada um de tamanho (2500,20x20) da seguinte forma -

```
trainset = NP_array[:, :50].reshape(-1,400).astype(np.float32)
testset = NP_array[:, 50:100].reshape(-1,400).astype(np.float32)
```

A seguir, temos que criar 10 rótulos diferentes para cada dígito, conforme mostrado abaixo -

```
k = np.arange(10)
train_labels = np.repeat(k,250)[:,np.newaxis]
test_labels = np.repeat(k,250)[:,np.newaxis]
```

Estamos agora em condições de iniciar a classificação KNN. Crie o objeto classificador e treine os dados.

```
knn = cv2.ml.KNearest_create()
knn.train(trainset, cv2.ml.ROW_SAMPLE, train_labels)
```

Escolhendo o valor de k como 3, obtenha a saída do classificador.

```
ret, output, neighbours, distance = knn.findNearest(testset, k = 3)
```

Compare a saída com rótulos de teste para verificar o desempenho e a precisão do classificador.

O programa mostra uma precisão de 91,64% na detecção precisa do dígito manuscrito.

```
result = output==test_labels
correct = np.count_nonzero(result)
accuracy = (correct*100.0)/(output.size)
print(accuracy)
```