OpenCV Read and Save Image

OpenCV Reading Images

OpenCV allows us to perform multiple operations on the image, but to do that it is necessary to read an image file as input, and then we can perform the various operations on it. OpenCV provides following functions which are used to read and write the images.

OpenCV imread function

The imread() function loads image from the specified file and returns it. The syntax is:

1. cv2.imread(filename[,flag])

Parameters:

filename: Name of the file to be loaded

flag: The flag specifies the color type of a loaded image:

- **CV_LOAD_IMAGE_ANYDEPTH -** If we set it as flag, it will return 16-bits/32-bits image when the input has the corresponding depth, otherwise convert it to 8-BIT.
- **CV_LOAD_IMAGE_COLOR -** If we set it as flag, it always return the converted image to the color one.
- **C V_LOAD_IMAGE_GRAYSCALE -** If we set it as flag, it always convert image into the grayscale.

The **imread()** function returns a matrix, if the image cannot be read because of unsupported file format, missing file, unsupported or invalid format. Currently, the following file formats are supported.

Windowbitmaps - *.bmp, *.dib
JPEG files - *.jpeg, *.jpg, *.jpe
Portable Network Graphics - *.png
Portable image format- *.pbm, *.pgm, *.ppm
TIFF files - *.tiff, *.tif

Note: The color images, the decoded images will have the channels stored in the BGR order.

Let's consider the following example:

```
#importing the opencv module
import cv2

# using imread('path') and 0 denotes read as  grayscale image
img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg',1)

#This is using for display the image
cv2.imshow('image',img)

cv2.waitKey(3) # This is necessary to be required so that the image doesn't close immediately.
#It will run continuously until the key press.
cv2.destroyAllWindows()
```

Output: it will display the following image.

OpenCV Save Images

OpenCV **imwrite()** function is used to save an image to a specified file. The file extension defines the image format. The syntax is the following:

1. cv2.imwrite(filename, img[,params])

Parameters:

filename- Name of the file to be loaded

image- Image to be saved.

params- The following parameters are currently supported:

- For JPEG, quality can be from 0 to 100. The default value is 95.
- For PNG, quality can be the compress level from 0 to 9. The default value is 1.
- For PPM, PGM, or PBM, it can be a binary format flag 0 or 1. The default value is 1.

Let's consider the following example:

```
import cv2

# read image as grey scale
img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg', 1)

# save image
status = cv2.imwrite(r'C:\Users\DEVANSH SHARMA\cat.jpeg', 0, img)
print("Image written to file-system : ", status)
```

Output:

```
Image written to file-system : True
```

If the imwrite() function returns the True, which means the file is successfully written in the specified file.

OpenCV Basic Operation on Images

In this tutorial, we will learn the essential operations that are related to the images. We are going to discuss the following topics.

- Access pixel values and modify them
- Access Image Properties
- Setting Region of Image
- Splitting and merging images
- Change the image color

Accessing and Modifying pixel values

We can retrieve a pixel value by its row and column coordinates. It returns an array of blue, green, red values of the BGR image. It returns the corresponding intensity for the grayscale image. First, we need to load the BGR image.

```
1. import numpy as np
2. import cv2
3. img = cv2.imread("C:\Users\DEVANSH SHARMA\cat.jpeg",1)
4. pixel = img[100,100]
5. print(pixel)
```

Output:

```
[190 166 250]
```

Accessing Image Properties

It is better to know the size of the image to work with the image processing application. In OpenCV, images are generally stored in the Numpy ndarray. To get the image shape or size, use ndarray.shape to find the dimension of the image. Then, we can use the index position to get the height, width, and number of channels.

Consider the following example:

```
1. import cv2
2. # read image
3. img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg',1)
4.
```

```
# height, width, number of channels in image
height = img.shape[0]
width = img.shape[1]
channels = img.shape[2]
size1 = img.size

print('Image Dimension    : ',dimensions)
print('Image Height       : ',height)
print('Image Width        : ',width)
print('Number of Channels : ',channels)
print('Image Size  :', size1)
```

Output:

```
Image Dimension    :  (4, 1, 3)
Image Height       :  4
Image Width        :  1
Number of Channels :  3
Image Size  : 12
```

Image ROI (Region of Interest)

Sometimes, we need to work with some areas of the image. As we discuss in the previous tutorial face detection is over the entire picture. When a face is obtained, we select only the face region and search for eyes inside it instead of searching the whole image. It enhances accuracy and performance because eyes are always on the face and don't need to search the entire image.

In the above image, if we need to select the ball. We only require selecting the ball region.

Splitting and Merging Image channels

An image's BGR channels can be split into their planes when needed. Then, the individual channels can be merged back together from the BGR image again. This can be done by following way:

1. b,g,r = cv2.split(img)
2. img = cv2.merge((b,g,r))

or

1. b = img[:,:,0]

> Note: The cv2.split() function is a slow function. Numpy indexing is quit efficient and it should be used if possible.

Making Borders for Images

OpenCV provides the **cv2.copyMakeBorder()** function to create a border around the image, something like a photo frame. The syntax of the function is given below.

1. cv2.copyMakeBorder(src,top,bottom,left,right,border type)

Parameters:

src - It denotes input image.

top, bottom, left, right - It defines the border width in the number of pixels in the corresponding direction.

borderType - It defines what kind of border to be added. The border can be the following types.

value - Color of border if border type is cv.BORDER_CONSTANT

Consider the following example:

```
1. import cv2 as cv
2. import numpy as np
3. from matplotlib import pyplot as plt
4. BLUE = [255,0,0]
5. img1 = cv.imread(r'C:\User\DEVANSH SHARMA\flower.jpg',1)
6. replicate = cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_REPLICATE)
7. reflect = cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_REFLECT)
8. reflect101 = cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_REFLECT_101)
9. wrap = cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_WRAP)
10. constant= cv.copyMakeBorder(img1,10,10,10,10,cv.BORDER_CONSTANT,value=B
    LUE)
11. plt.subplot(231),plt.imshow(img1,'gray'),plt.title('ORIGINAL')
12. plt.subplot(232),plt.imshow(replicate,'gray'),plt.title('REPLICATE')
13. plt.subplot(233),plt.imshow(reflect,'gray'),plt.title('REFLECT')
14. plt.subplot(234),plt.imshow(reflect101,'gray'),plt.title('REFLECT_101')
15. plt.subplot(235),plt.imshow(wrap,'gray'),plt.title('WRAP')
16. plt.subplot(236),plt.imshow(constant,'gray'),plt.title('CONSTANT')
17. plt.show()
```

Change in Image color

OpenCV cvtColor

The **cvtColor** is used to convert an image from one color space to another. The syntax is following:

1. cv2.cvtColor(src, dst, code)

Parameters:

src - It is used to input an image: 8-bit unsigned.

dst - It is used to display an image as output. The output image will be same size and depth as input image.

code - color space conversion code.

Consider the following example:

```
# importing cv2
import cv2

# path of the input image
path = (r'C:\Users\DEVANSH SHARMA\cat.jpeg')

# Reading an image in default mode
src = cv2.imread(path)

# Window name in which image is displayed
window_name = 'Image'
# Using cv2.cvtColor() method
# Using cv2.COLOR_BGR2GRAY color space for convert BGR image to grayscale
# conversion code
image = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY )
# Displaying the image
cv2.imshow(window_name, image)
```

Output:

OpenCV Resize the image

Sometimes, it is necessary to transform the loaded image. In the image processing, we need to resize the image to perform the particular operation. Images are generally stored in Numpy ndarray(array). The **ndarray.shape** is used to obtain the dimension of the image. We can get the width, height, and numbers of the channels for each pixel by using the index of the dimension variable.

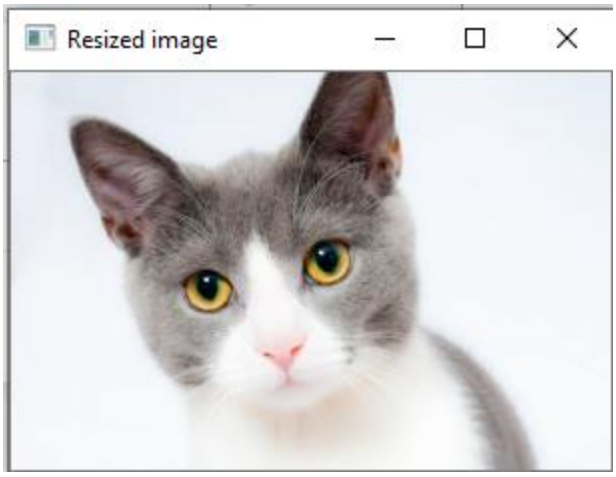
Example: 1

```
1. import cv2
2.
3. img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg', 1)
4. scale = 60
5. width = int(img.shape[1] * scale / 100)
6. height = int(img.shape[0] * scale / 100)
7. dim = (width, height)
8. # resize image
9. resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
10.
```

```
11. print('Resized Dimensions : ', resized.shape)
12.
13. cv2.imshow("Resized image", resized)
14. cv2.waitKey(0)
15. cv2.destroyAllWindows()
```

Output:

```
Resized Dimensions : (199, 300, 3)
```



The resizing of image means changing the dimension of the image, its width or height as well as both. Also the aspect ratio of the original image could be retained by resizing an image. OpenCV provides **cv2.resize()** function to resize the image. The syntax is given as:

```
1. cv2.resize(src, dsize[, dst[, fx[,fy[,interpolation]]]])
```

Parameters:

- **src** - source/input image (required).
- **dsize** - desired size for the output image(required)
- **fx** - Scale factor along the horizontal axis.(optional)
- **fy** - Scale factor along the vertical axis.
- **Interpolation(optional)** - This flag uses following methods:
    - INTER_NEAREST - A nearest-interpolation INTER_AREA - resampling using pixel area relation. When we attempt to do image zoom, it is similar to the INTER_NEAREST method.

- INTER_CUBIC - A bicubic interpolation over 4×4 pixel neighborhood.
- INTER_LANCOZS4 - Lanczos interpolation over 8×8 pixel neighborhood.

Example of resizing the images

There are several ways to resize the image. Below are some examples to perform resize operation:

1. Retain Aspect Ratio (height to width ratio of the image is retained)
    - Downscale(Decrement in the size of the image)
    - Upscale(Increment in the size of image)
2. Do not preserve Aspect Ratio
    - Resize only the width
    - Resize only the height
3. Resize the specified width and height

Retain the aspect ratio

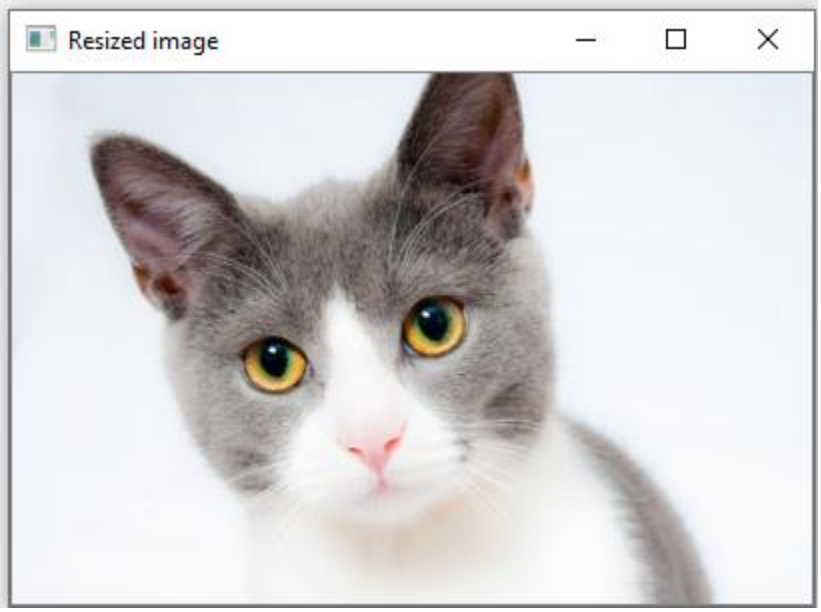
- **Downscale with resize()**

```
import cv2

img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg', 1)

print('Original Dimensions : ', img.shape)

scale = 60  # percent of original size
width = int(img.shape[1] * scale / 100)
height = int(img.shape[0] * scale / 100)
dim = (width, height)
# resize image
resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)

print('Resized Dimensions : ', resized.shape)

```

```
cv2.imshow("Resized image", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:

```
Original Dimensions :  (332, 500, 3)
Resized Dimensions :  (199, 300, 3)
```



In the above example, the scale_per variable holds the percentage of the image which needs to be scaled. The **value<100** is used to downscale the provided image. We will use this **scale_per** value along with the original image's dimension to calculate the width and height of the output image.

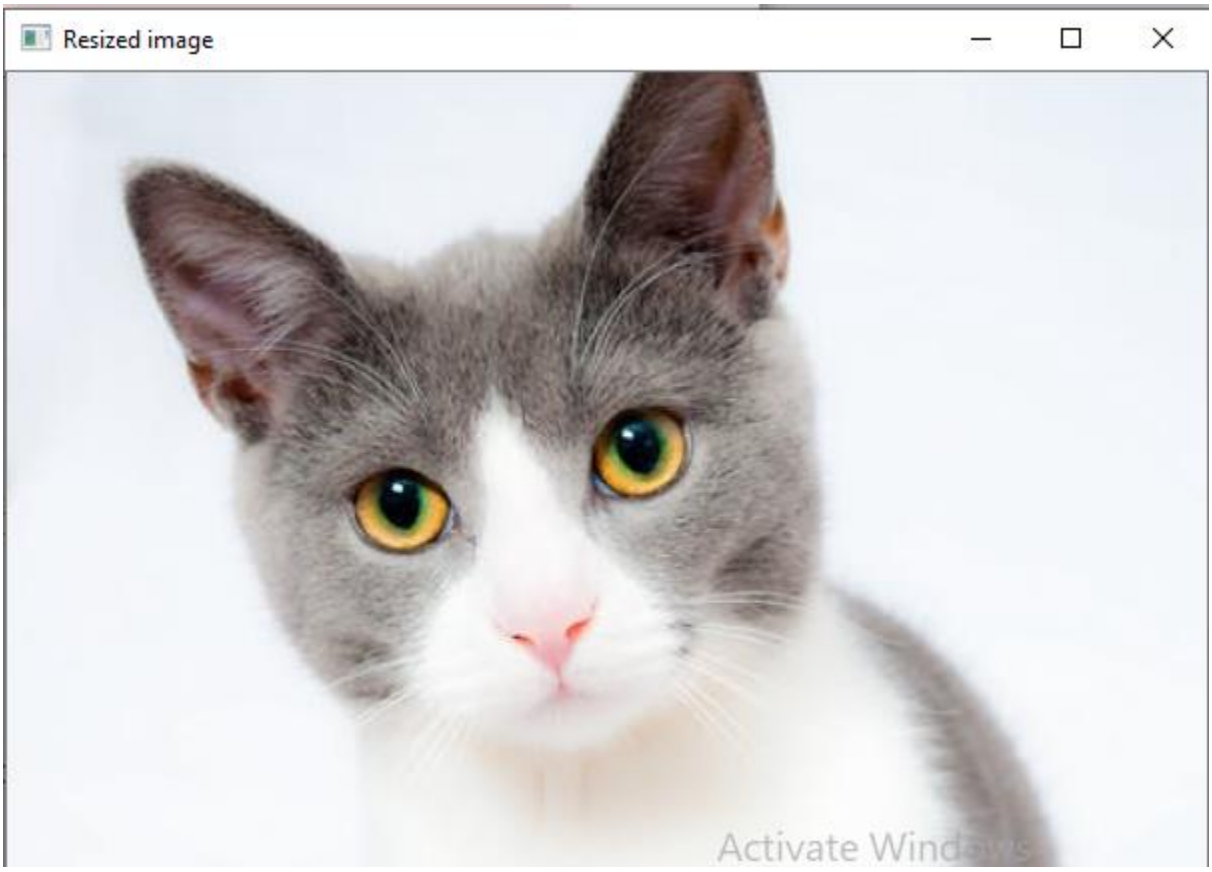
Upscale with resize()

```
import cv2

img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg', 1)

print('Original Dimensions : ', img.shape)

scale = 150  # percent of original size
width = int(img.shape[1] * scale / 100)
```

```
height = int(img.shape[0] * scale / 100)
dim = (width, height)
# resize image
resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)

print('Resized Dimensions : ', resized.shape)

cv2.imshow("Resized image", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:

```
Original Dimensions :  (332, 500, 3)
Resized Dimensions :  (398, 600, 3)
```



Not retaining the aspect ratio

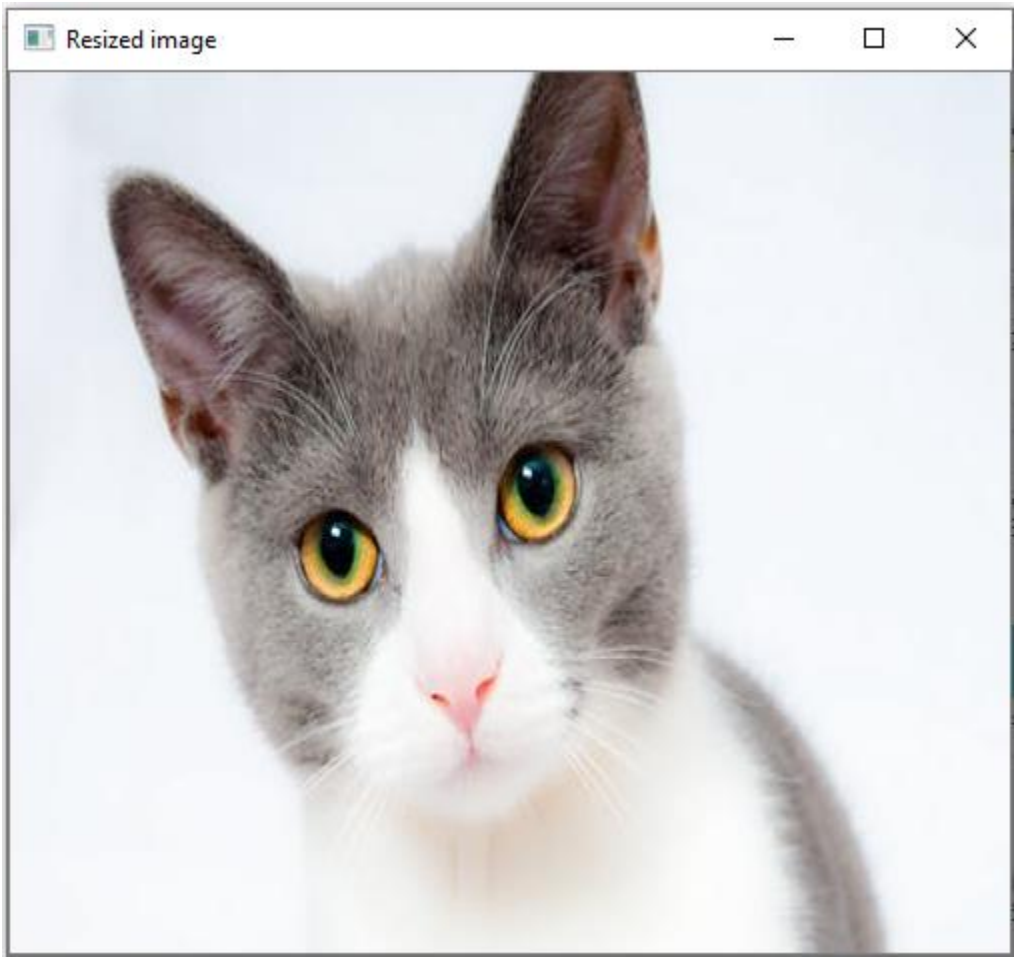
- **Resize only the width**

In the below example, we have provided a specific value in pixel for width and the height will remain unaffected.

```
import cv2

img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg', cv2.IMREAD_UNCHANGED)
print('Original Dimensions : ', img.shape)

width = img.shape[1]  # keep original width
height = 440
dim = (width, height)

# resize image
resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)

print('Resized Dimensions : ', resized.shape)

cv2.imshow("Resized image", resized)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:

```
Original Dimensions :  (332, 500, 3)
Resized Dimensions :  (440, 500, 3)
```



- **Resize the height**

In the below example, the **scale_per** value holds the percentage by which height has to be scaled or we can provide the specific value in pixels.

```
import cv2

img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg', 1)
print('Original Dimensions : ', img.shape)
width = img.shape[1]  # keep original width
height = 200
dim = (width, height)

# resize image
resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
```

```
11.
12. print('Resized Dimensions : ', resized.shape)
13.
14. cv2.imshow("Resized image", resized)
15. cv2.waitKey(0)
16. cv2.destroyAllWindows()
```

Output:

```
Original Dimensions :  (332, 500, 3)
Resized Dimensions :  (200, 500, 3)
```

Resize the specific width and height

- We can specify both width and height.

```
1. import cv2
2.
3. img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat.jpeg', 1)
4. print('Original Dimensions : ', img.shape)
5.
6. width = 350
7. height = 450
8. dim = (width, height)
9. # resize image
10. resized = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)
```

```
11.
12. print('Resized Dimensions : ', resized.shape)
13. cv2.imshow("Resized image", resized)
14. cv2.waitKey(0)
15. cv2.destroyAllWindows()
```

Output:

OpenCV Drawing Functions

We can draw the various shapes on an image such as **circle, rectangle, ellipse, polylines, convex,** etc. It is used when we want to highlight any object in the input image. The OpenCV provides functions for each shape. Here we will learn about the drawing functions.

Drawing Circle

We can draw the circle on the image by using the **cv2.circle()** function. The syntax is the following:

1. cv2.circle(img, center, radius, color[,thickness [, lineType[,shift]]])

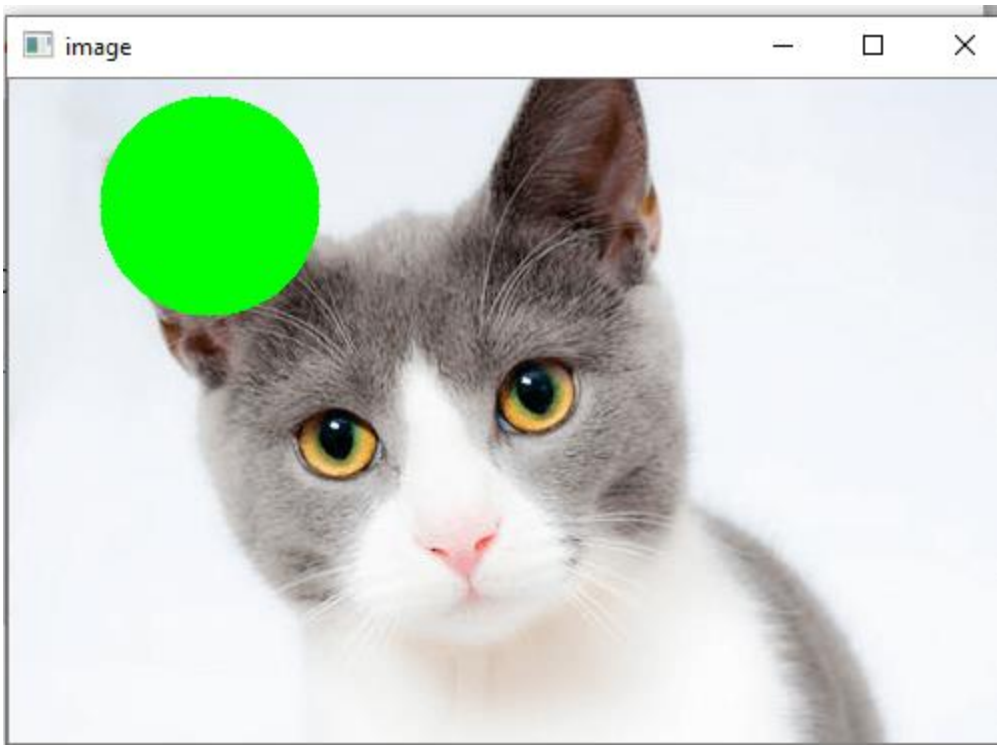
Parameters:

- **img-** It represents the given image.
- **center-** Center of the circle
- **radius-** Radius of the circle
- **color-** Circle color
- **thickness-** It denotes the thickness of the circle outline, if it is positive. And negative thickness means that a filled circle is to be drawn.
- **lineType-** Defines the type of the circle boundary.
- **shift-** It represents the number of fractional bits in the coordinate of the center and the radius value.

Consider the following example:

```
1. import numpy as np
2. import cv2
3. img = cv2.imread(r"C:\Users\DEVANSH SHARMA\cat.jpeg",1)
4. cv2.circle(img,(80,80), 55, (0,255,0), -1)
5. cv2.imshow('image',img)
6. cv2.waitKey(0)
7. cv2.destroyAllWindows()
```

Output:



Drawing Rectangle

The OpenCV provides a function to draw a simple, thick or filled up-right rectangle. The syntax is following:

1. cv2.rectangle(img, pt1, pt2, color[, thickness[,lineType[,shift]]])

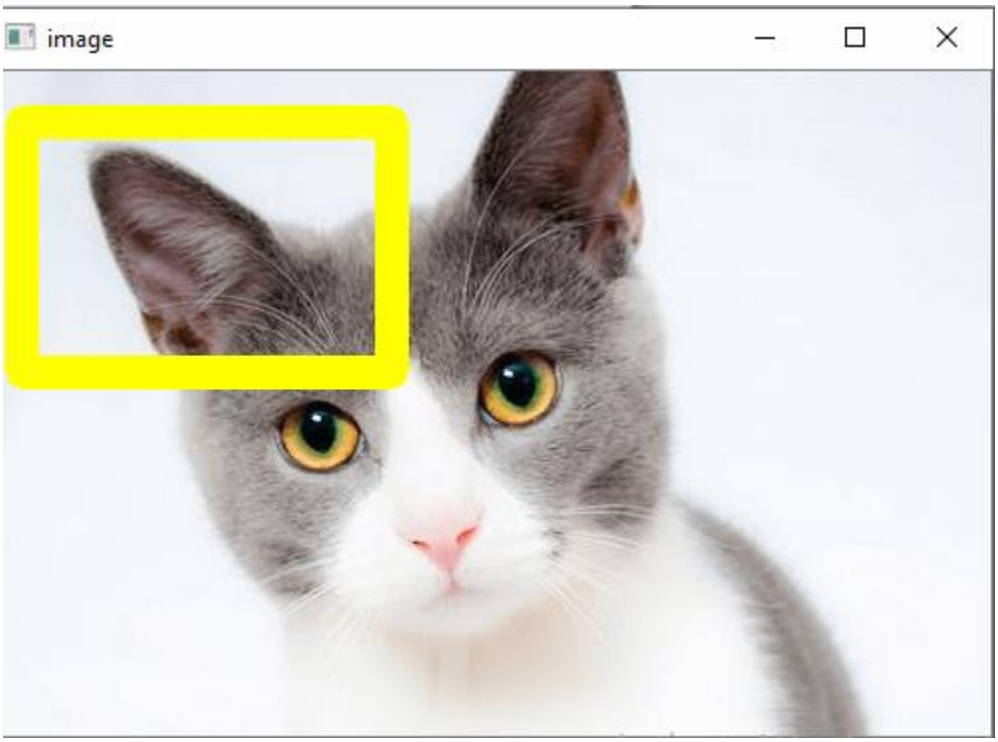
Parameters:

- **img-** It represents an image.
- **pt1-** It denotes **vertex** of the rectangle.
- **pt2-** It denotes the vertex of the rectangle opposite to pt1.
- **color-** It denotes the rectangle color of brightness (grayscale image).
- **thickness-** It represents the thickness of the lines that makes up the rectangle. Negative values (CV_FILLED) mean that the function has to draw a filled rectangle.
- **linetype-** It represents the types of the line.
- **shift-** It represents the number of fractional bits in the point coordinates.

Consider the following example:

```
1. import numpy as np
2. import cv2
3. img = cv2.imread(r"C:\Users\DEVANSH SHARMA\cat.jpeg",1)
4. cv2.rectangle(img,(15,25),(200,150),(0,255,255),15)
5. cv2.imshow('image',img)
6. cv2.waitKey(0)
7. cv2.destroyAllWindows()
```

Output:



Drawing Ellipse

We can draw an ellipse on an image by using the **cv2.ellipse()** function. It can draw a simple or thick elliptic arc or can fill an ellipse sector.

1. cv2.ellipse(img, center, axes, angle, startAngle, endAngle, color[, thickness[, lineType[, shift]]])
2. cv2.ellipse(img, box, color[, thickness[, lineType]])

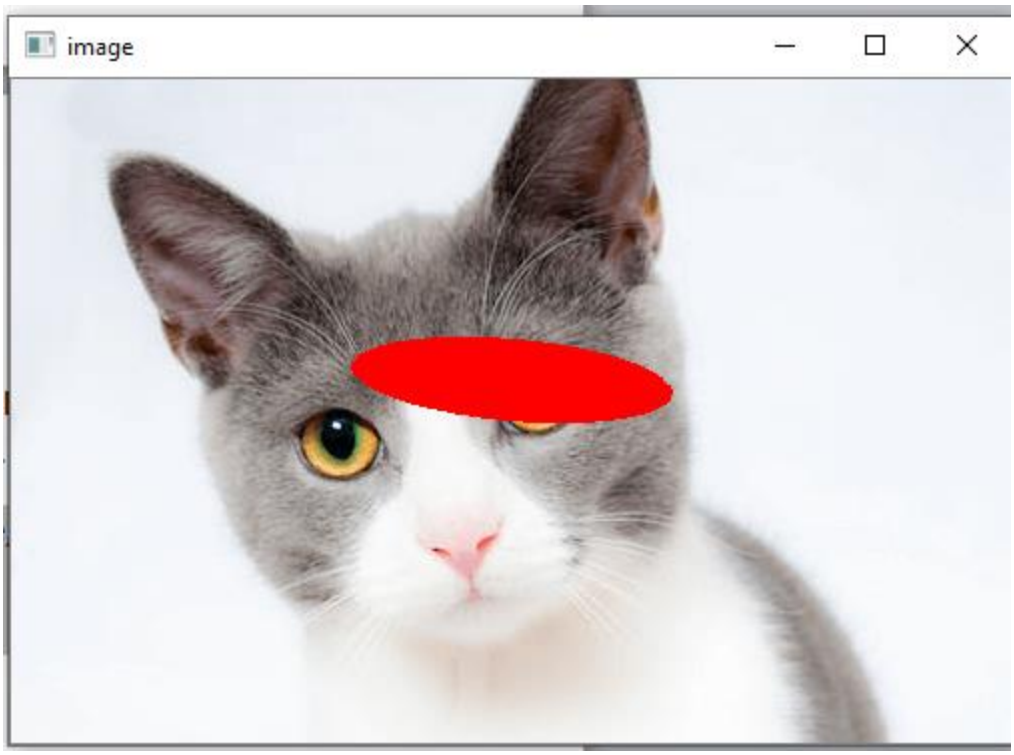
Parameters:

- **img -** It represents an image.
- **box -** It represents alternative ellipse representation via RotatedRect or CvBox2D. It means that the function is used to draws an ellipse in a curved rectangle.
- **color -** It denotes the ellipse color.
- **angle-** It denotes the angle of rotation.
- **startAngle -** It denotes the initial angle of the elliptic arc in degrees.
- **endAngle -** It denotes the ending angle of the elliptic arc in degrees.
- **thickness -** It is used to draw thickness of the ellipse arc outline if the value is positive. Otherwise, this specifies that a filled ellipse is to be drawn.
- **lineType -** It denotes the type of the ellipse boundary.
- **shift -** It represents the number of fractional bits in the coordinates of the center and values of axes.

Consider the following example:

```
1. import numpy as np
2. import cv2
3. img = cv2.imread(r"C:\Users\DEVANSH SHARMA\cat.jpeg",1)
4. 
5. cv2.ellipse(img, (250, 150), (80, 20), 5, 0, 360, (0,0,255), -1)
6. cv2.imshow('image',img)
7. cv2.waitKey(0)
8. cv2.destroyAllWindows()
```

Output:



There are two functions to draw the ellipse. The first function is used to draw the whole ellipse, not an arc bypassing **startAngle=0** and **endAngle = 360**. The second function of an ellipse is used to draw an ellipse outline, a filled ellipse, an elliptic arc, or a filled ellipse sector.

Drawing lines

OpenCV provides the **line()** function to draw the line on the image. It draws a line segment between ptr1 and ptr2 points in the image. The image boundary clips the line.

1. cv2.line(img, pt1, pt2, color[, thickness[, lineType[, shift]]])

Parameters:

- **img-** It represents an image.
- **pt1-** It denotes the first point of the line segments.
- **pt2-** It denotes the second point of the line segment.
- **color -** Represents the Line-color
- **thickness-** Represents the Line thickness
- **lineType-** There are various types of line:

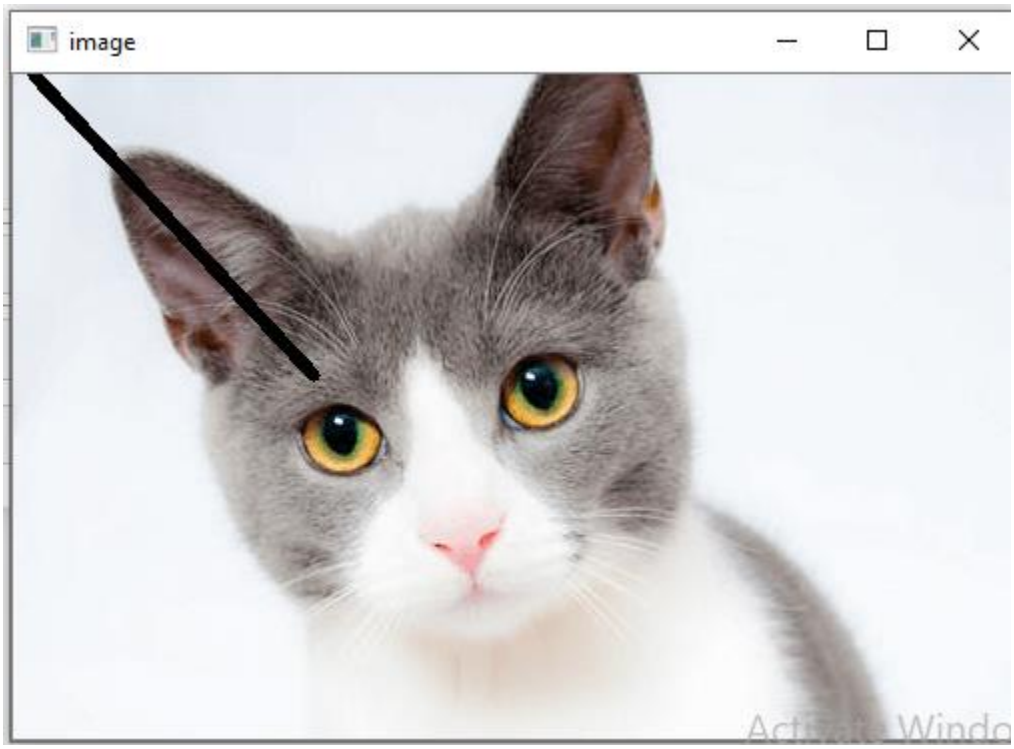
- 8 (or omitted) - 8 connected lines.
- 4 - 4-connected line.
- CV__AA- antialiased line

- **shift-** It represents the number of fractional bits in the point coordinates.

Consider the following example:

```
1. import numpy as np
2. import cv2
3. img = cv2.imread(r"C:\Users\DEVANSH SHARMA\cat.jpeg",1)
4. cv2.line(img,(10,0),(150,150),(0,0,0),15)
5. cv2.imshow('image',img)
6. cv2.waitKey(0)
7. cv2.destroyAllWindows()
```

Output:



Write Text on Image

We can write text on the image by using the **putText()** function. The syntax is given below.

1. cv2.putText(img, text, org, font, color)

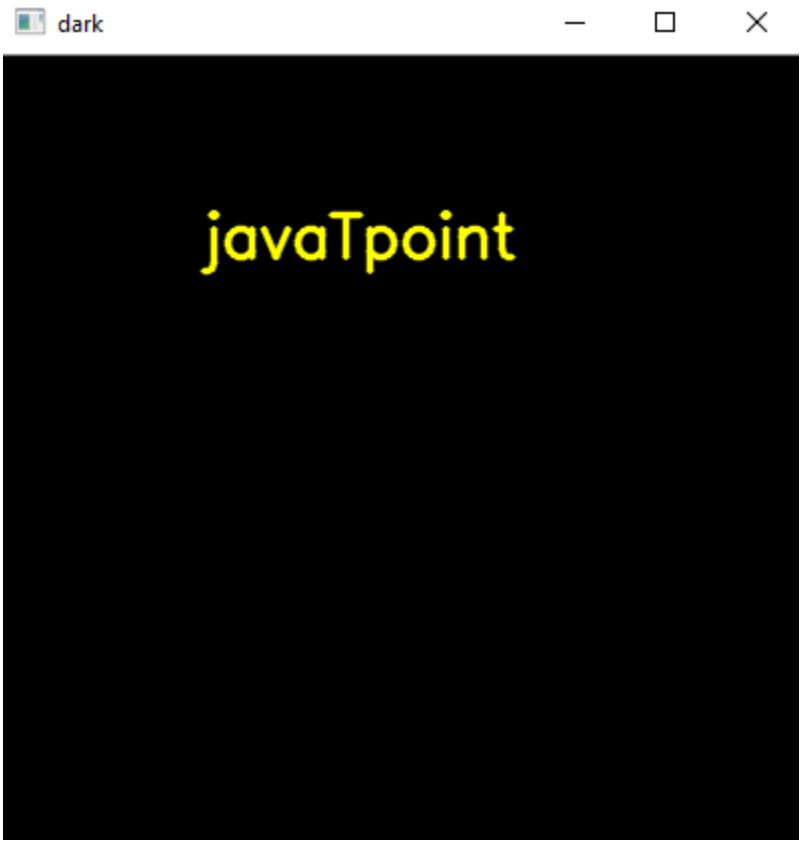
Parameters:

- **img:** It represents an image
- **text:** It represents a text which we want to write on the image.
- **org:** It denotes the Bottom-left corner of the text string in the image.
- **font:** CvFont structure is initialized using InitFont().
- **color:** Represents the Text color.

Consider the following example.

```
import numpy as np
import cv2
font = cv2.FONT_HERSHEY_SIMPLEX
# Create a black image.
img = cv2.imread(r"C:\Users\DEVANSH SHARMA\cat.jpeg",1)
cv2.putText(img,'Hack Projects',(10,500), font, 1,(255,255,255),2)
#Display the image
cv2.imshow("image",img)
cv2.waitKey(0)
```

Output:



Drawing Polylines

We can draw the polylines on the image. OpenCV provides the **polylines()** function, that is used to draw polygonal curves on the image. The syntax is given below:

1. cv2.polyLine(img, polys, is_closed, color, thickness=1, lineType=8, shift=0)

Parameters:

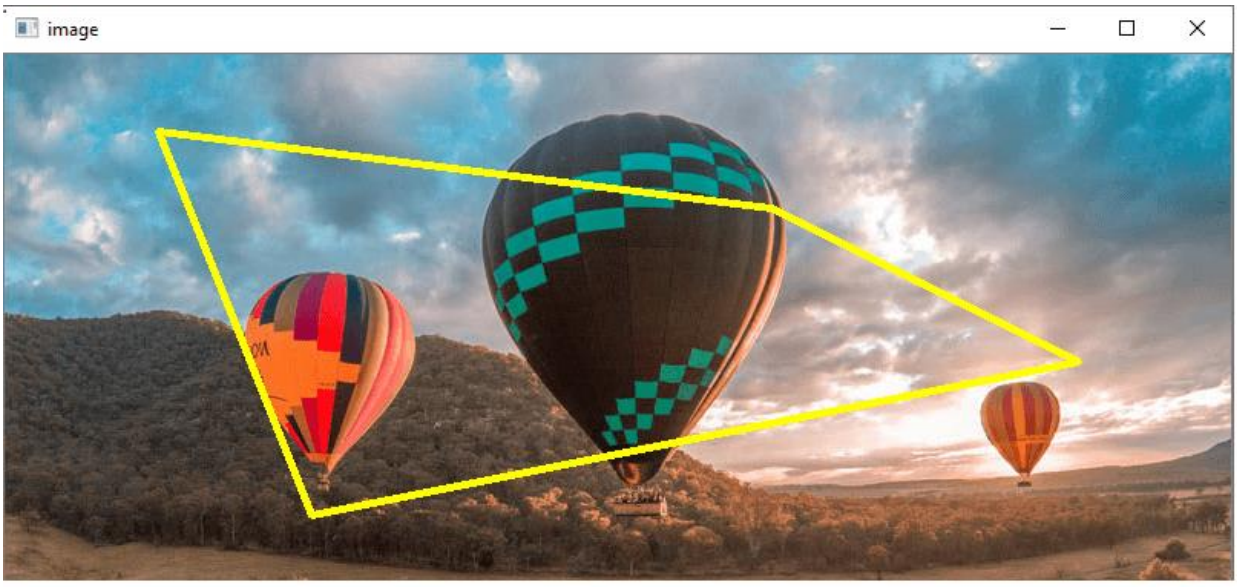
- **img -** It represents an image.
- **pts -** It denotes the array of polygon curves.
- **npts -** It denotes an array of polygon vertex counters.
- **ncounters -** It represents the number of curves.
- **is_Closed -** It is a flag that indicates whether the drawn polylines are closed or not.
- **color -** Color of polylines.
- **thickness -** It represents the Thickness of the polylines edges.

- **lineType -** Type of the line segment.
- **shift-** It represents the number of fractional bits in the point coordinates.

Consider the following program to draw polylines in image:

```
import numpy as np
import cv2
img = cv2.imread(r'C:\Users\DEVANSH SHARMA\forest.jpg',cv2.IMREAD_COLOR)

#defining points for polylines
pts = np.array([[100,50],[200,300],[700,200],[500,100]], np.int32)
# pts = pts.reshape((-1,1,2))
cv2.polylines(img, [pts], True, (0,255,255), 3)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:



OpenCV Blob Detection

Blob stands for **Binary Large Object** and refers to the connected pixel in the binary image. The term "Large" focuses on the object of a specific size, and that other "small" binary objects are usually noise. There are three processes regarding BLOB analysis.

BLOB extraction

Blob extraction means to separate the BLOBs (objects) in a binary image. A BLOB contains a group of connected pixels. We can determine whether two pixels are connected or not by the connectivity, i.e., which pixels is neighbor of another pixel. There are two types of connectivity. The **8-connectivity** and the **4-connectivity**. The 8-connectivity is far better than 4-connectivity.

BLOB representation

BLOB representation is simply means that convert the BLOB into a few representative numbers. After the BLOB extraction, the next step is to classify the several BLOBs. There are two steps in the BLOB representation process. In the first step, each BLOB is denoted by several characteristics, and the second step is to apply some matching methods that compare the features of each BLOB.

BLOB classification

Here we determine the type of BLOB, for example, given BLOB is a circle or not. Here the question is how to define which BLOBs are circle and which are not based on their features that we described earlier. For this purpose, generally we need to make a prototype model of the object we are looking for.

```
1. import cv2
2. import numpy as np;
3.
4. img = cv2.imread(r"filename", cv2.IMREAD_GRAYSCALE)
5. # Set up the detector with default parameters.
6. detector = cv2.SimpleBlobDetector()
7.
8. # Detecting blobs.
9. keypoints = detector.detect(img)
```

```
# Draw detected blobs as red circles.
# cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of the circle corresponds to the size of blob
im_with_keypoints = cv2.drawKeypoints(img, keypoints, np.array([]), (0, 0, 255),
                                      cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
# Show keypoints
cv2.imshow("Keypoints", im_with_keypoints)
cv2.waitKey(0)
```

How to perform Background Subtraction?

Background subtraction is widely used to generating a foreground mask. The binary images contain the pixels which belong to moving objects in the scene. Background subtraction calculates the foreground mask and performs the subtraction between the current frame and background model.

There are two main steps in Background modeling

- **Background Initialization-** In this step, an initial model of the background is computed.
- **Background Update-** In this step, that model is updated that adapt the possible change in the scene.

Manual subtraction from the first frame

First, we import the libraries and load the video. Next, we take the first frame of the video, convert it into grayscale, and apply the Gaussian Blur to remove some noise. We use the while loop, so we load frame one by one. After doing this, we get the core part of the background of the subtraction where we calculate the absolute difference between the first frame and the current frame.

Example-1

```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)

```

```
first_frame = cap.read()
first_gray = cv2.cvtColor(first_frame,)
first_gray_col = cv2.GaussianBlur(first_gray, (5, 5), 0)

while True:
    frame = cap.read()
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    gray_frame = cv2.GaussianBlur(gray_frame, (5, 5), 0)

difference = cv2.absdiff(first_gray, gray_frame)
difference = cv2.threshold(difference, 25, 255, cv2.THRESH_BINARY)
```

Subtraction using Subtractor MOG2

OpenCV provides the subtractor MOG2 which is effective than the manual mode. The Subtractor MOG2 has the benefit of working with the frame history. The syntax is as follows:

```
cv2.createBackgroundSubtractorMOG2(history, varTheshold, detectShadow)
```

The first argument, history is the number of the last frame(by default 120).

The second argument, a **varThreshold** is the value that used when evaluating the difference to extract the background. A lower threshold will find more variation with the advantage of a noisier image.

The third argument, **detectShadows** is the functions of the algorithm which can remove the shadow if enabled.

Example-2:

```
import cv2
import numpy as np
cap = cv2.VideoCapture("filename")

subtractor = cv2.createBackgroundSubtractorMOG2(history=100, varThreshold=50, detectShadows=True)

```

```
while True:
    _, frame = cap.read()
    mask = subtractor.apply(frame)
    cv2.imshow("Frame", frame)
    cv2.imshow("mask", mask)
    key = cv2.waitKey(30)
    if key == 27:
        break
cap.release()
cv2.destroyWindowKey()
```

In the above code, The **cv2.VideoCapture("filename")** accepts the full path included the file where the **cv2.createBackgroundSubtractorMOG2()** will exclude the background from the video file.

OpenCV Canny Edge Detection

Edge detection is term where identify the boundary of object in image. We will learn about the edge detection using the canny edge detection technique. The syntax is canny edge detection function is given as:

```
edges = cv2.Canny('/path/to/img', minVal, maxVal, apertureSize, L2gradient)
```

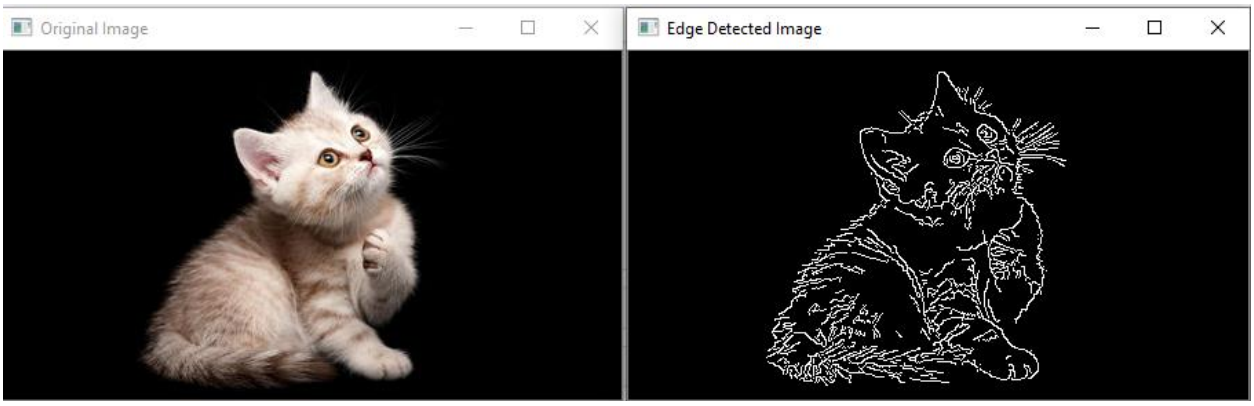
Parameters-

- **/path/to/img:** file path of the image (required)
- **minVal:** Minimum intensity gradient (required)
- **maxVal:** Maximum intensity gradient (required)
- **aperture:** It is optional argument.
- **L2gradient:** Its default value is false, if value is true, Canny () uses a more computationally expensive equation to detect edges, which provides more accuracy at the cost of resources.

Example: 1

```
import cv2
img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat_16x9.jpg')
edges = cv2.Canny(img, 100, 200)

cv2.imshow("Edge Detected Image", edges)
cv2.imshow("Original Image", img)
cv2.waitKey(0)  # waits until a key is pressed
cv2.destroyAllWindows()  # destroys the window showing image
```

Output:



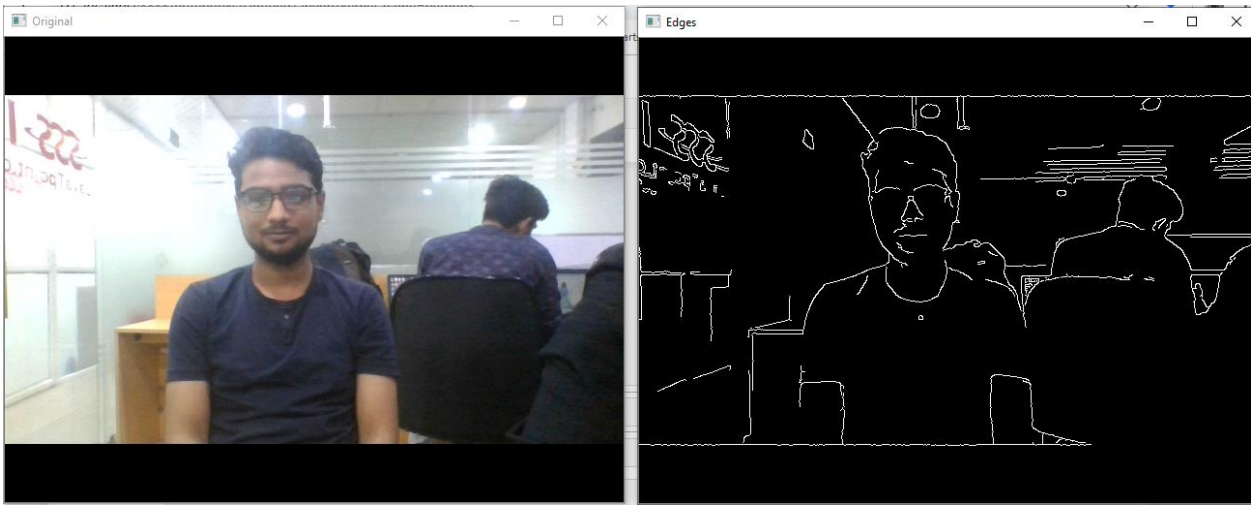
Example: Real Time Edge detection

```
# import libraries of python OpenCV
import cv2

# import Numpy by alias name np
import numpy as np

# capture frames from a camera
cap = cv2.VideoCapture(0)

# loop runs if capturing has been initialized
while (1):

    # reads frames from a camera
```

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

    # converting BGR to HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # define range of red color in HSV
    lower_red = np.array([30, 150, 50])
    upper_red = np.array([255, 255, 180])

    # create a red HSV colour boundary and
    # threshold HSV image
    mask = cv2.inRange(hsv, lower_red, upper_red)

    # Bitwise-AND mask and original image
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Display an original image
    cv2.imshow('Original', frame)

    # discovers edges in the input image image and
    # marks them in the output map edges
    edges = cv2.Canny(frame, 100, 200)

    # Display edges in a frame
    cv2.imshow('Edges', edges)

    # Wait for Esc key to stop
    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        break

# Close the window
cap.release()

# De-allocate any associated memory usage
```

48. cv2.destroyAllWindows()

Output:



OpenCV Blur (Image Smoothing)

Blurring is the commonly used technique for image processing to removing the noise. It is generally used to eliminate the high-frequency content such as noise, edges in the image. The edges are being blurred when we apply blur to the image. The advantages of blurring are the following:

Advantages of Blurring

The benefits of blurring are the following:

- It removes low-intensity edges.
- It helps in smoothing the image.
- It is beneficial in hiding the details; for example, blurring is required in many cases, such as police intentionally want to hide the victim's face.

OpenCV provides mainly the following type of blurring techniques.

OpenCV Averaging

In this technique, the image is convolved with a box filter (normalize). It calculates the average of all the pixels which are under the kernel area and replaces the central element with the calculated average. OpenCV provides the **cv2.blur()** or **cv2.boxFilter()** to perform this operation. We should define the width and height of the kernel. The syntax of **cv2.blur()** function is following.

1. cv2.blur(src, dst, ksize, anchor, borderType)

Parameters:

src - It represents the source (input) image.

dst - It represents the destination (output) image.

ksize - It represents the size of the kernel.

anchor - It denotes the anchor points.

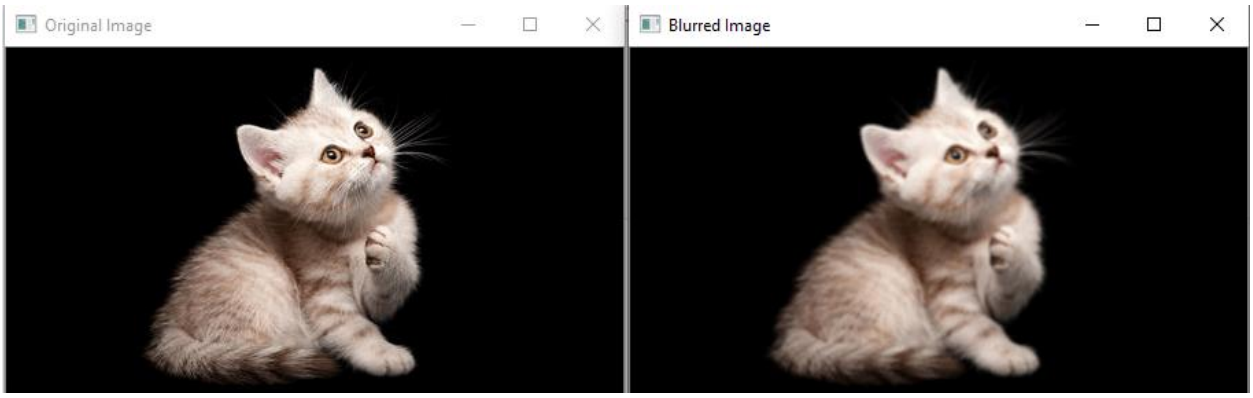
borderType - It represents the type of border to be used to the output.

Consider the following example:

```
1. im = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat_16x9.jpg')
2. cv2.imshow('Original Image',im)
3. cv2.imshow('Blurred Image', cv2.blur(im, (3,3)))
4. cv2.waitKey(0)
```

5. cv2.destroyAllWindows()

Output



OpenCV Median Blur

The median blur operation is quite similar to the Gaussian blur. OpenCV provides the **medianblur()** function to perform the blur operation. It takes the median of all the pixels under the kernel area, and the central element is replaced with this median value. It is extremely effective for the salt-and-paper noise in the image. The kernel size should be a positive odd integer. Following is the syntax of this method.

1. cv2.medianBlur(src, dst, ksize)

Parameters:

src- It represents the source (input image).

dst - It represents the destination (output image).

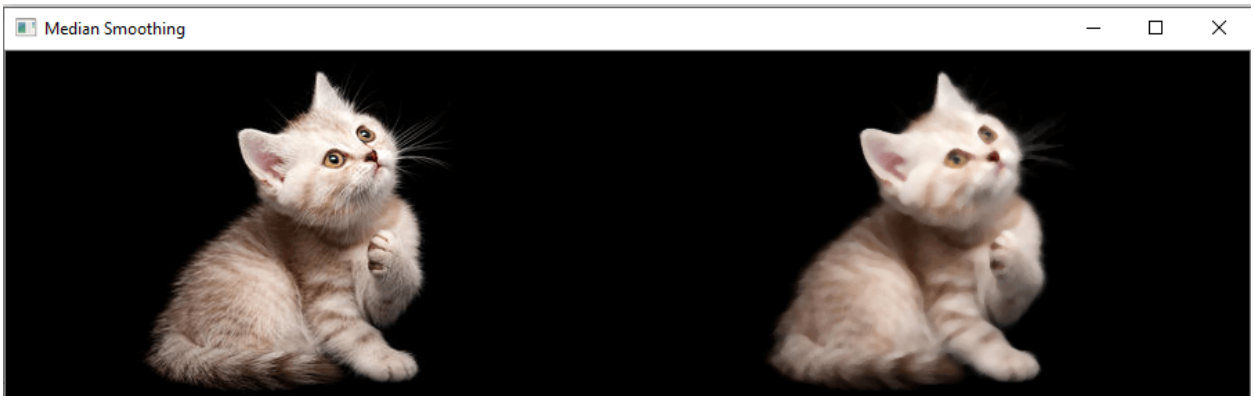
ksize - It represents the size of the kernel.

Consider the following example:

```
1. import cv2
2. import numpy
3. # read image
4. img = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat_16x9.jpg', 1)
5. # apply gaussian blur on src image
```

```
6. dst = median = cv2.medianBlur(img,5)
7. # display input and output image
8. cv2.imshow("Gaussian Smoothing", numpy.hstack((src, dst)))
9. cv2.waitKey(0)  # waits until a key is pressed
10. cv2.destroyAllWindows()  # destroys the window showing image
```

Output



OpenCV Gaussian Blur

Image smoothing is a technique which helps in reducing the noise in the images. Image may contain various type of noise because of camera sensor. It basically eliminates the high frequency (noise, edge) content from the image so edges are slightly blurred in this operation. OpenCV provide **gaussianblur()** function to apply smoothing on the images. The syntax is following:

```
1. dst=cv2.GuassiasBlur(src, ksize, sigmaX[,dst[,sigmaY[, borderType=BORDER_DEFAULT]]]
```

Parameters:

- **src** -It is used to input an Image.
- **dst** -It is a variable which stores an output Image.
- **ksize** -It defines the Gaussian Kernel Size[height width]. Height and width must be odd (1,3,5,..) and can have different values. If ksize is set to [0,0], then ksize is computed from sigma value.
- **sigmaX** - Kernel standard derivation along X-axis.(horizontal direction).

- **sigmaY** - Kernel standard derivation along Y-axis (vertical direction). If sigmaY = 0 then sigmaX value is taken for sigmaY.

borderType - These are the specified image boundaries while the kernel is applied on the image borders. Possible border type is:

- cv.BORDER_CONSTANT
- cv.BORDER_REPLICATE
- cv.BORDER_REFLECT
- cv.BORDER_WRAP
- cv.BORDER_REFLECT_101
- cv.BORDER_TRANSPARENT
- cv.BORDER_REFLECT101
- cv.BORDER_DEFAULT
- cv.BORDER_ISOLATED

Consider the following example:

```
import cv2
import numpy

# read image
src = cv2.imread(r'C:\Users\DEVANSH SHARMA\cat_16x9.jpg', 1)
# apply gaussian blur on src image
dst = cv2.GaussianBlur(src, (5, 5), cv2.BORDER_DEFAULT)

# display input and output image
cv2.imshow("Gaussian Smoothing", numpy.hstack((src, dst)))
cv2.waitKey(0)  # waits until a key is pressed
cv2.destroyAllWindows()  # destroys the window showing image
```

Output:

OpenCV Image Filters

Image filtering is the process of modifying an image by changing its shades or color of the pixel. It is also used to increase brightness and contrast. In this tutorial, we will learn about several types of filters.

Bilateral Filter

OpenCV provides the **bilateralFilter()** function to apply the bilateral filter on the image. The bilateral filter can reduce unwanted noise very well while keeping edges sharp. The syntax of the function is given below:

1. cv2.bilateralFilter(src, dst, d, sigmaSpace, borderType)

Parameters:

- **src-** It denotes the source of the image. It can be an 8-bit or floating-point, 1-channel image.
- **dst-** It denotes the destination image of the same size. Its type will be the same as the src image.
- **d -** It denotes the diameter of the pixel neighborhood (integer type) that is used during filtering. If its value is negative, then it is computed from sigmaSpace.
- **sigmaColor -** It denotes the filter sigma in the color space.
- **sigmaSpace -** It denotes the filter sigma in the coordinate space.

Consider the following example:

```
1. import cv2
2. import numpy as np
3. from matplotlib import pyplot as plt
4. img = cv2.imread(r'C:\Users\DEVANSH SHARMA\baloon.jpg',1)
5. 
6. kernel = np.ones((5,5),np.float32)/25
7. blur = cv2.bilateralFilter(img,9,75,75)
8. plt.subplot(121),plt.imshow(img),plt.title('Original')
9. plt.xticks([]), plt.yticks([])
10. plt.subplot(122),plt.imshow(blur),plt.title('Bilateral Filter')
11. plt.xticks([]), plt.yticks([])
12. cv2.imshow("Image",blur)
```

Output

Original Image Bilateral Filter

Box Filter

We can perform this filter using the **boxfilter()** function. It is similar to the averaging blur operation. The syntax of the function is given below:

```
1. cv2. boxfilter(src, dst, ddepth, ksize, anchor, normalize, bordertype)
```

Parameters:

- **src -** It denotes the source of the image. It can be an 8-bit or floating-point, 1-channel image.

- **dst-** It denotes the destination image of the same size. Its type will be the same as the src image.
- **ddepth -** It denotes the output image depth.
- **ksize -** It blurs the kernel size.
- **anchor -** It denotes the anchor points. By default, its value Point to coordinates (-1,1), which means that the anchor is at kernel center.
- **normalize -** It is the flag, specifying whether the kernel should be normalized or not.
- **borderType -** An integer object represents the type of the border used.

Consider the following example:

```
import cv2
import numpy as np
# using imread('path') and 0 denotes read as  grayscale image
img = cv2.imread(r'C:\Users\DEVANSH SHARMA\baloon.jpg',1)
img_1 = cv2.boxFilter(img, 0, (7,7), img, (-1,-1), False, cv2.BORDER_DEFAULT)
#This is using for display the image
cv2.imshow('Image',img_1)
cv2.waitKey(3) # This is necessary to be required so that the image doesn't close immediately.
#It will run continuously until the key press.
cv2.destroyAllWindows()
```

Output

Filter2D

It combines an image with the kernel. We can perform this operation on an image using the **Filter2D()** method. The syntax of the function is given below:

```
cv2.Filter2D(src, dst, kernel, anchor = (-1,-1))
```

Parameters:

- **src -** It represents the input image.
- **dst-** It denotes the destination image of the same size. Its type will be the same as the src image.
- **kernel -** It is a convolution kernel, a single-channel floating-point matrix. If you want to apply different kernels to different channels, split the image into a separate color plane using the split () process them individually.
- **anchor -** It denotes the anchor points, by default its value Point(-1,1), which means that the anchor is at kernel center.
- **borderType -** An integer object represents the type of the border used.

Consider the following example:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread(r'C:\Users\DEVANSH SHARMA\baloon.jpg',1)

kernel = np.ones((5,5),np.float32)/25
dst = cv2.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('Filter2D')
plt.xticks([]), plt.yticks([])
plt.show()
```

Output

OpenCV Image Threshold

The basic concept of the threshold is that more simplify the visual data for analysis. When we convert the image into gray-scale, we have to remember that grayscale still has at least 255 values. The threshold is converted everything to white or black, based on the threshold value. Let's assume we want the threshold to be 125(out of 255), then everything that was under the 125 would be converted to 0 or black, and everything above the 125 would be converted to 255, or white. The syntax is as follows:

1. retval,threshold = cv2.threshold(src, thresh, maxValue, cv2.THRESH_BINARY_INV)

Parameters-

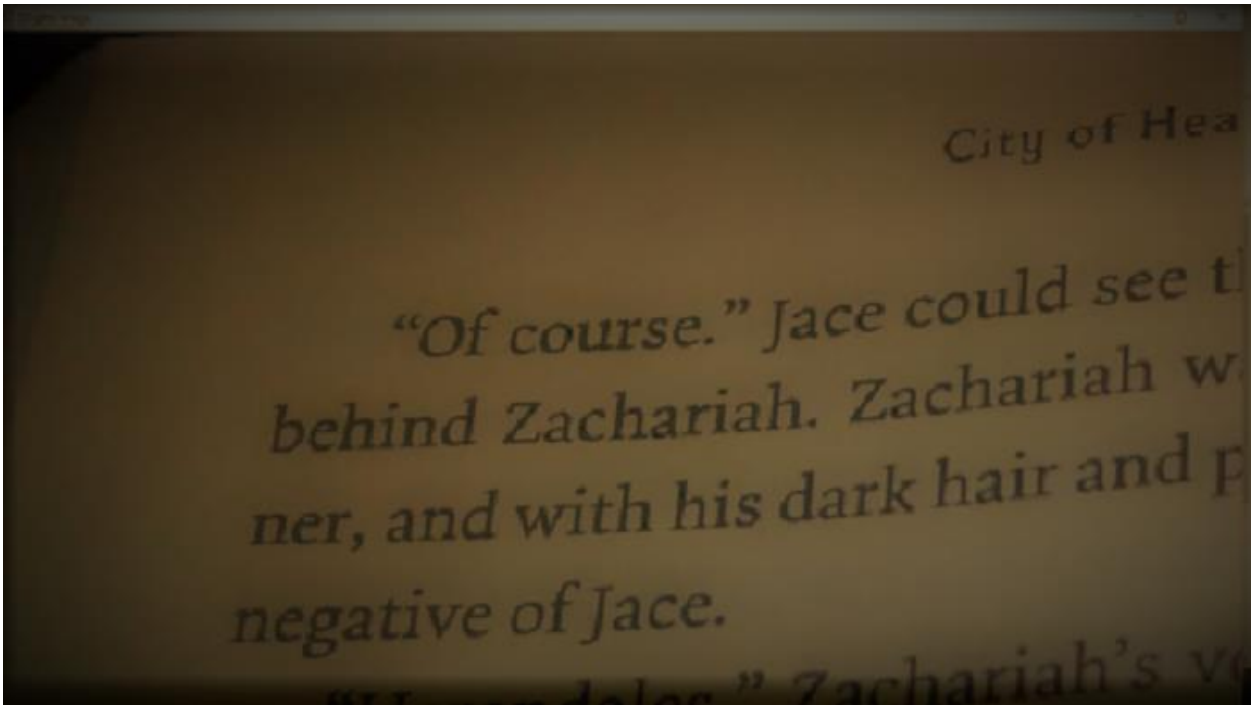
src: Source image, it should be a grayscale image.

thresh: It is used to classify the pixel value.

maxVal: It represents the value to be given if the pixel threshold value.OpenCV provides different styles of threshold that is used as fourth parameter of the function. These are the following:

- cv2.THRESH_BINARY
- cv2.THRESH_BINARY_INV
- cv2.THRESH_TRUNC
- cv2.THRESH_TOZERO
- cv2.THRESH_TOZERO_INV

Let's take a sample input image

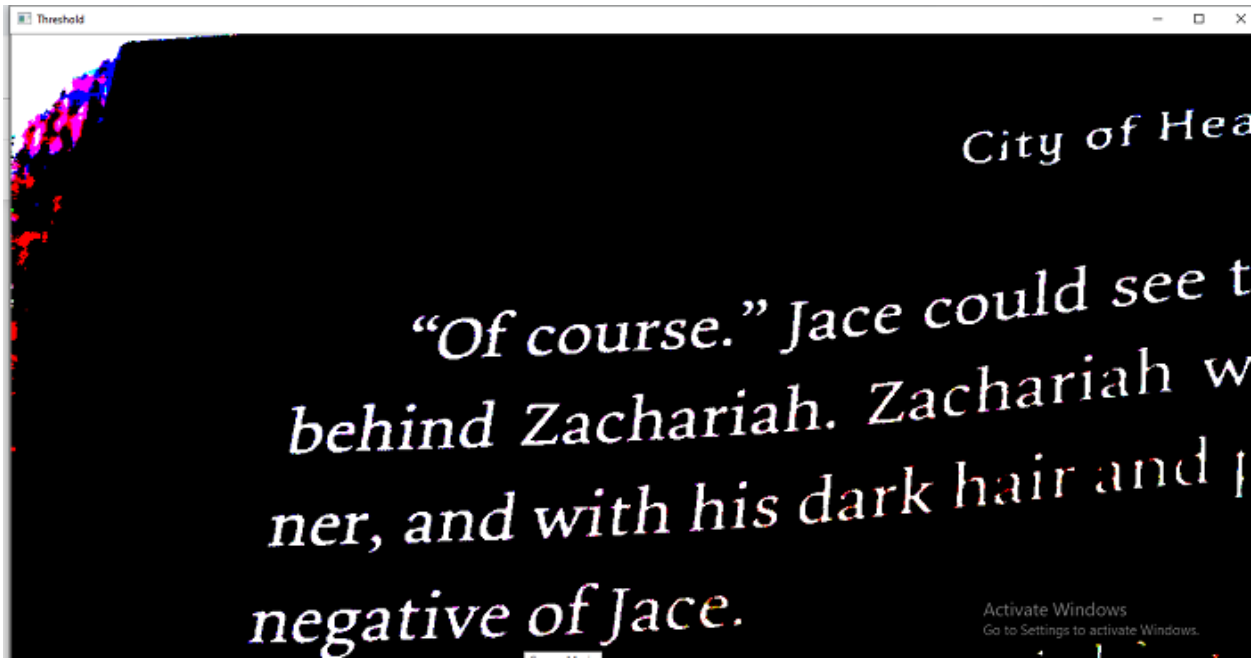


We have taken above image as an input. We describe how threshold actually works. The above image is slightly dim and little bit hard to read. Some parts are light enough to read, while other part is required more focus to read properly.

Let's consider the following example:

```
1. import cv2
2. img  = cv2.imread(r'C:\Users\DEVANSH SHARMA\book1.jpg',1)
3. retval, threshold = cv2.threshold(img, 62, 255, cv2.THRESH_BINARY)
4. cv2.imshow("Original Image", img)
5. cv2.imshow("Threshold",threshold)
6. cv2.waitKey(0)
```

Output:



The above program highlighted the image. Now we can easily read the content of the image.

OpenCV Contours

Contours are defined as a curve joining all the continuous points (along the boundary), having the same color or intensity. In the other, we find counter in a binary image, we focus to find the boundary in the binary image. The official definition is following:

> The Contours are the useful tool for shape analysis and object detection and recognition.

To maintain accuracy, we should use the binary images. First, we apply the threshold or canny edge detection.

In OpenCV, finding the contour in the binary image is the same as finding white object from a black background.

OpenCV provides **findContours()**, which is used to find the contour in the binary image. The syntax is following:

1. cv2. findContours (thes, cv2.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

The **findContours ()** accepts the three argument first argument is source image, second is contour retrieval mode, and the third is contours approximation.

Let's consider the following example:

```
import numpy as np
import cv2 as cv
im = cv.imread(r'C:\Users\DEVANSH SHARMA\binary.png')
imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imgray, 127, 255, 0)
contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
```

How to draw the Contours?

OpenCV provides the cv2.drawContours() function, which is used to draw the contours. It is also used to draw any shape by providing its boundary points. Syntax of cv2.drawContours() function is given below:

To draw all the contours in an image:

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

To draw an individual contour, suppose 3rd counter

```
cnt = contours[3]
cv2.drawCounter(img,[cnt],0,(0,255,0),3)
```

The first argument represents the image source, second argument represents the contours which should be passed as a Python list, the third argument is used as index of Contours, and other arguments are used for color thickness.

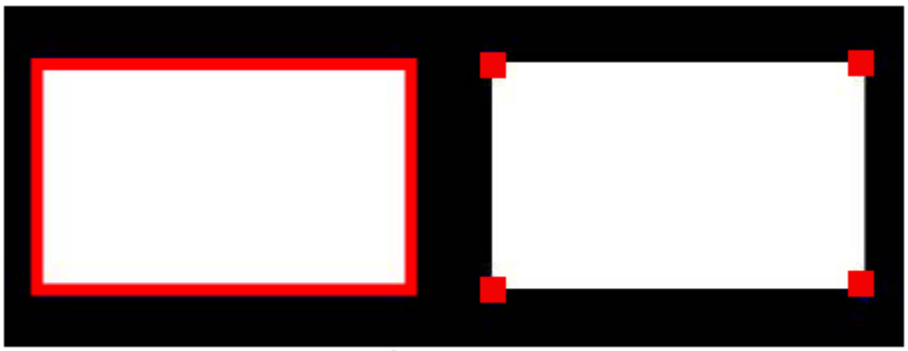
Contour Approximation Method

It is the third argument in the **cv2.findCounter()**. Above, we have described it to draw the boundary of the shape with same intensity. It stores the **(x,y)** coordinates of the boundary of a shape. But here the question arise does it store all the coordinates? That is specified by the contour approximation method.

If we pass the **cv.CHAIN_APPROX_NONE**, it will store all the boundary points. Sometimes it does not need to store all the points coordinate, suppose we found the contours of a straight line where it does not require to store all the contour points, it requires only two

endpoints to store. So for such case, we use **cv.CHAIN_APPROX_NONE**, it removes all redundant points and compresses the contours, thereby saving memory.

Example-1



images

In the above image of rectangle, the first image shows points using with **cv.CHAIN_APPROX_NONE(734)** and the second image shows the one with **cv2.CHAIN_APPROX_SIMPLE(only 4 points)**. We can see the difference between both the images.

OpenCV Contours

Contours are defined as a curve joining all the continuous points (along the boundary), having the same color or intensity. In the other, we find counter in a binary image, we focus to find the boundary in the binary image. The official definition is following:

> The Contours are the useful tool for shape analysis and object detection and recognition.

To maintain accuracy, we should use the binary images. First, we apply the threshold or canny edge detection.

In OpenCV, finding the contour in the binary image is the same as finding white object from a black background.

OpenCV provides **findContours()**, which is used to find the contour in the binary image. The syntax is following:

1. cv2. findContours (thes, cv2.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

The **findContours ()** accepts the three argument first argument is source image, second is contour retrieval mode, and the third is contours approximation.

Let's consider the following example:

```
1. import numpy as np
2. import cv2 as cv
3. im = cv.imread(r'C:\Users\DEVANSH SHARMA\binary.png')
4. imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
5. ret, thresh = cv.threshold(imgray, 127, 255, 0)
6. contours, hierarchy = cv.findContours(thresh, cv.RETR_TREE, cv.CHAIN_APPROX_S
   IMPLE)
```

How to draw the Contours?

OpenCV provides the cv2.drawContours() function, which is used to draw the contours. It is also used to draw any shape by providing its boundary points. Syntax of cv2.drawContours() function is given below:

To draw all the contours in an image:

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

To draw an individual contour, suppose 3rd counter

```
1. cnt = contours[3]
2. cv2.drawCounter(img,[cnt],0,(0,255,0),3)
```

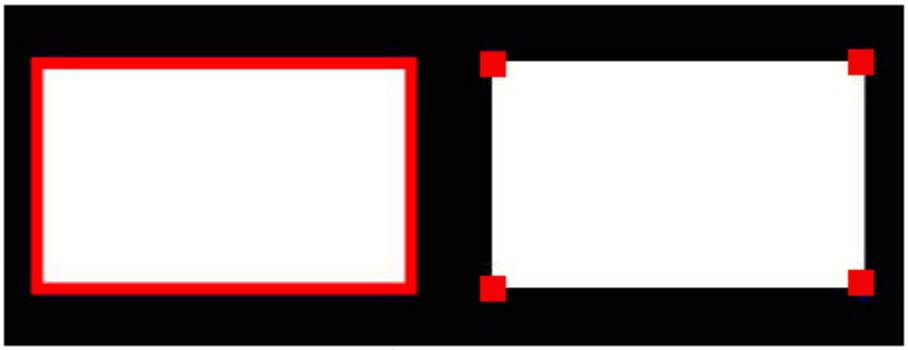
The first argument represents the image source, second argument represents the contours which should be passed as a Python list, the third argument is used as index of Contours, and other arguments are used for color thickness.

Contour Approximation Method

It is the third argument in the **cv2.findCounter()**. Above, we have described it to draw the boundary of the shape with same intensity. It stores the **(x,y)** coordinates of the boundary of a shape. But here the question arise does it store all the coordinates? That is specified by the contour approximation method.

If we pass the **cv.CHAIN_APPROX_NONE**, it will store all the boundary points. Sometimes it does not need to store all the points coordinate, suppose we found the contours of a straight line where it does not require to store all the contour points, it requires only two endpoints to store. So for such case, we use **cv.CHAIN_APPROX_NONE**, it removes all redundant points and compresses the contours, thereby saving memory.

Example-1



images

In the above image of rectangle, the first image shows points using with **cv.CHAIN_APPROX_NONE(734)** and the second image shows the one with **cv2.CHAIN_APPROX_SIMPLE(only 4 points)**. We can see the difference between both the images.

OpenCV Template Matching

Template matching is a technique that is used to find the location of template images in a larger image. OpenCV provides the **cv2.matchTemplates()** function for this purpose. It simply slides the template images over the input image and compares the templates and patch under the input image.

There are various methods available for the comparison; we will discuss a few popular methods in further topics.

It returns a grayscale image, where every pixel represents the number of the neighborhood of that pixel match with the input templates.

Template matching in OpenCV

The templates matching consist of the following step:

Step - 1: Take the actual image and convert it into a grayscale image.

Step - 2: Select the template as a grayscale image.

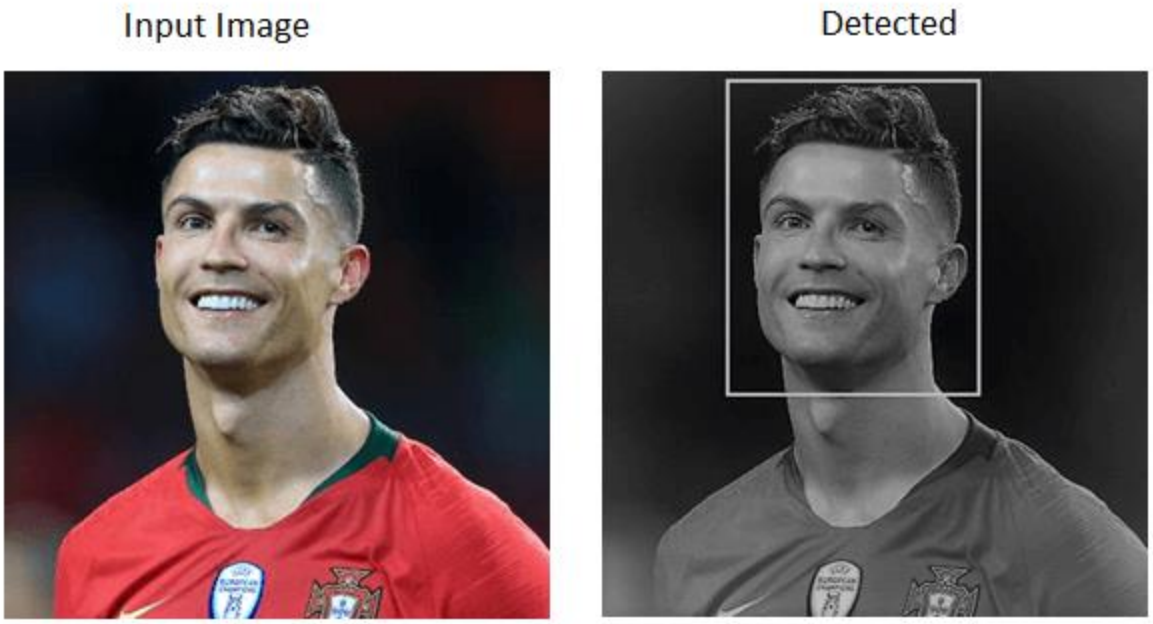
Step - 3: Find the location where the accuracy level matches. It is done by template image slide over the actual image.

Step - 4: When the result is greater than the accuracy level, mark that position as detected.

Consider the following example:

```
1. import cv2
2. import numpy as np
3. # Reading the main image
4. rgb_img = cv2.imread(r'C:\Users\DEVANSH SHARMA\rolando.jpg',1)
5. # It is need to be convert it to grayscale
6. gray_img = cv2.cvtColor(rgb_img, cv2.COLOR_BGR2GRAY)
7. # Reading the template image
8. template = cv2.imread(r'C:\Users\DEVANSH SHARMA\ronaldo_face.jpg',0)
9. # Store width in variable w and height in variable h of template
10. w, h = template.shape[:-1]
11. # Now we perform match operations.
12. res = cv2.matchTemplate(gray_img,template,cv2.TM_CCOEFF_NORMED)
13. # Declare a threshold
14. threshold = 0.8
15. # Store the coordinates of matched location in a numpy array
16. loc = np.where(res >= threshold)
17. # Draw the rectangle around the matched region.
18. for pt in zip(*loc[::-1]):
19.     cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,255,255), 2)
20. # Now display the final matched template image
21. cv2.imshow('Detected',img_rgb)
```

Output:



Template Matching with Multiple Objects

In the above example, we searched image for template image that occurred only once in the image. Suppose a particular object that occur multiple times in particular image. In this scenario, we will use the thresholding because **cv2.minMaxLoc()** won't give all location of template image. Consider the following example.

```
import cv2
import numpy as np
# Reading the main image
img_rgb = cv2.imread(r'C:\Users\DEVANSH SHARMA\mario.png',1)
# It is need to be convert it to grayscale
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
# Read the template
template = cv2.imread(r'C:\Users\DEVANSH SHARMA\coin1.png',0)
# Store width in variable w and height in variable h of template
w, h = template.shape[:-1]
# Now we perform match operations.
res = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
# Declare a threshold
threshold = 0.8
```

```
# Store the coordinates of matched region in a numpy array
loc = np.where( res >= threshold)
# Draw a rectangle around the matched region.
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0] + w, pt[1] + h), (0,255,255), 2)
# Now display the final matched template image
cv2.imshow('Detected',img_rgb)
```

Output:

In the above program, we took an image of popular super Mario game as main image and coin image as template image. The coins occur multiple times in main image. When it find the coin in the image it draw rectangle on the coin.

Limitation of Templates Matching

There are few limitations in template matching given as follows:

- It is a time-consuming process to calculate the pattern correlation image for medium to large images.
- Pattern occurrence has to preserve the orientation of the reference template image.a

- Template matching doesn't apply on the rotated or scaled version of the template as a change in shape/size/shear etc.

OpenCV Erosion and Dilation

Erosion and Dilation are **morphological image processing** operations. OpenCV morphological image processing is a procedure for modifying the geometric structure in the image. In morphism, we find the shape and size or structure of an object. Both operations are defined for binary images, but we can also use them on a grayscale image. These are widely used in the following way:

- Removing Noise
- Identify intensity bumps or holes in the picture.
- Isolation of individual elements and joining disparate elements in image.

In this tutorial, we will explain the erosion and dilation briefly.

Dilation

Dilation is a technique where we expand the image. It adds the number of pixels to the boundaries of objects in an image. The structuring element controls it. The structuring element is a matrix of 1's and 0's.

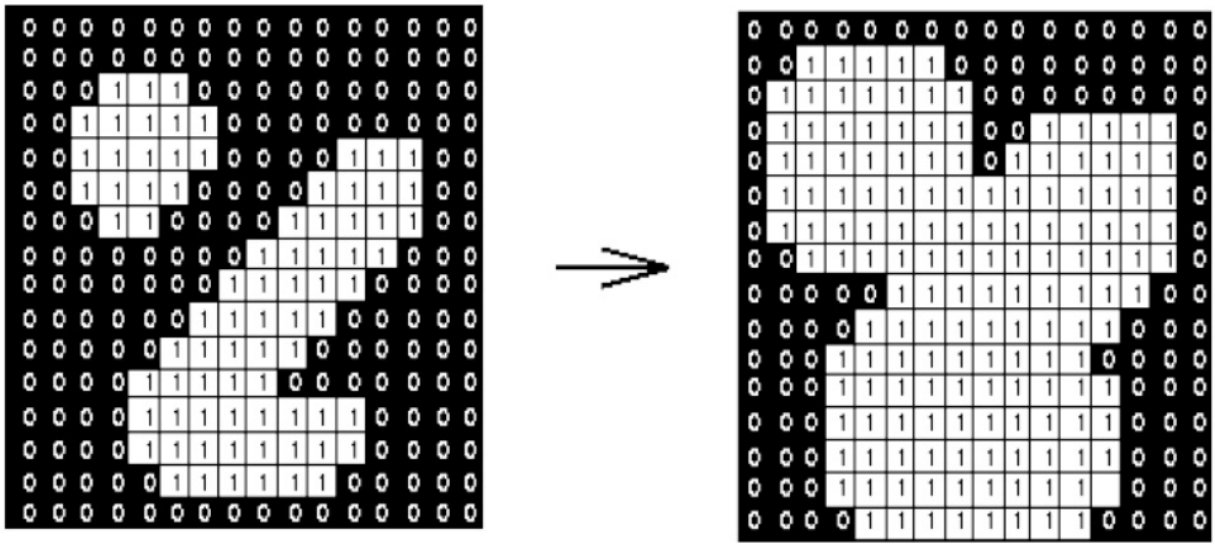
Structuring Element

The size and shape of the structuring element define how many numbers of the pixel should be added or removed from the objects in an image.

It is a matrix of 1's and 0's. The center pixel of the image is called the origin.

It contains an image A with some kernel (B), which can have any shape or size, generally a square or circle. Here the kernel B has a defined anchor point. It is the center of the kernel.

In the next step, the kernel is overlapped over the image to calculate maximum pixel values. When the computation is completed, the image is replaced with an anchor at the center. The brighter areas increase in size that made increment in image size.



Effect of dilation using a 3×3 square structuring element

For example, the size of the object increases in the white shade; on the other side, the size of an object in black shades automatically decreases.

The dilation operation is performed by using the **cv2.dilate()** method. The syntax is given below:

1. cv2.dilate(src, dst, kernel)

Parameters: The **dilate()** function accepts the following argument:

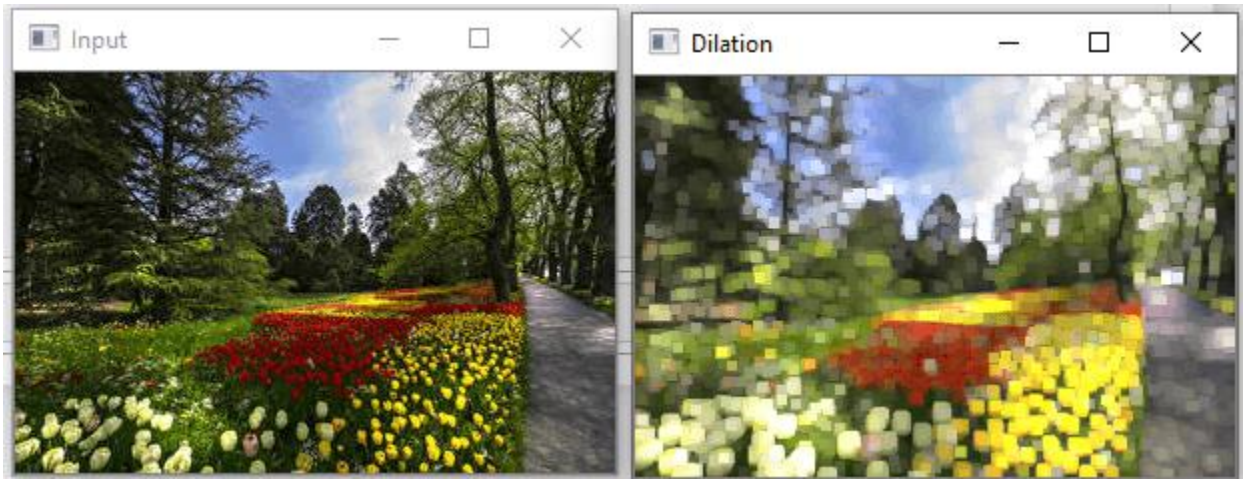
- **src** - It represents the input image.
- **dst** - It represents the output image.
- **kernel** - It represents the kernel.

Consider the following example:

```
import cv2
import numpy as np
img = cv2.imread(r'C:\Users\DEVANSH SHARMA\jtp_flower.jpg, 0)

kernel = np.ones((5,5), np.uint8)
img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)
```

```
8. cv2.imshow('Input', img)
9. cv2.imshow('Dilation', img_dilation)
10. cv2.waitKey(0)
```

Output



Erosion

Erosion is much similar to dilation. The difference is that the pixel value calculated minimum rather than the maximum in dilation. The image is replaced under the anchor point with that calculated minimum pixel. Unlikely dilation, the regions of darker shades increase. While it decreases in white shade or brighter side.

OpenCV provides **cv2.erode()** function to perform this operation. The syntax of the function is the following:

```
1. cv2.erode(src, dst, kernel)
```

Parameters:

- **src** - It represents the source(input) image.
- **dst** - It represents the destination (output) image.
- **kernel** - It represents the Kernel.

Consider the following example:

```
1. import cv2
```

```
2. import numpy as np
3. img = cv2.imread(r'C:\Users\DEVANSH SHARMA\baloon.jpg', 1)
4. kernel = np.ones((5,5), np.uint8)
5. img_erosion = cv2.erode(img, kernel, iterations=1)
6. img_dilation = cv2.dilate(img, kernel, iterations=1)
7. cv2.imshow('Input', img)
8. cv2.imshow('Erosion', img_erosion)
9. cv2.waitKey(0)
```

Output

The above program will give the following output. We can see the different between both images.

Erosion operation applied to the input image.

OpenCV VideoCapture

OpenCV provides the **VideoCature()** function which is used to work with the Camera. We can do the following task:

- Read video, display video, and save video.
- Capture from the camera and display it.

Capture Video from Camera

OpenCV allows a straightforward interface to capture live stream with the camera (webcam). It converts video into grayscale and display it.

We need to create a **VideoCapture** object to capture a video. It accepts either the device index or the name of a video file. A number which is specifying to the camera is called device index. We can select the camera by passing the O or 1 as an argument. After that we can capture the video frame-by-frame.

```
1. import cv2
```

```
import numpy as np

cap = cv2.VideoCapture(0)

while(True):
    # Capture image frame-by-frame
    ret, frame = cap.read()

    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

The **cap.read()** returns a boolean value(True/False).It will return True, if the frame is read correctly.

Playing Video from file

We can play the video from the file. It is similar to capturing from the camera by changing the camera index with the file name. The time must be appropriate for **cv2.waitKey()** function, if time is high, video will be slow. If time is too less, then the video will be very fast.

```
import numpy as np
import cv2

cap = cv2.VideoCapture('filename')

while(cap.isOpened()):
```

```
    ret, frame = cap.read()
#it will open the camera in the grayscale mode
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Saving a Video

The **cv2.imwrite()** function is used to save the video into the file. First, we need to create a VideoWriter object. Then we should specify the **FourCC** code and the number of frames per second (fps). The frame size should be passed within the function.

FourCC is a 4-byte code used to identify the video codec. The example is given below for saving the video.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        frame = cv2.flip(frame,0)

        # write the flipped frame
        out.write(frame)
```

```
17.
18.         cv2.imshow('frame',frame)
19.         if cv2.waitKey(1) & 0xFF == ord('q'):
20.             break
21.     else:
22.         break
23.
24. # Release everything if job is finished
25. cap.release()
26. out.release()
27. cv2.destroyAllWindows()
```

It will save the video at the desired location. Run the above code and see the output.

Face recognition and Face detection using the OpenCV

The face recognition is a technique to identify or verify the face from the digital images or video frame. A human can quickly identify the faces without much effort. It is an effortless task for us, but it is a difficult task for a computer. There are various complexities, such as low resolution, occlusion, illumination variations, etc. These factors highly affect the accuracy of the computer to recognize the face more effectively. First, it is necessary to understand the difference between face detection and face recognition.

Face Detection: The face detection is generally considered as finding the faces (location and size) in an image and probably extract them to be used by the face detection algorithm.

Face Recognition: The face recognition algorithm is used in finding features that are uniquely described in the image. The facial image is already extracted, cropped, resized, and usually converted in the grayscale.

There are various algorithms of face detection and face recognition. Here we will learn about face detection using the HAAR cascade algorithm.

Basic Concept of HAAR Cascade Algorithm

The HAAR cascade is a machine learning approach where a cascade function is trained from a lot of positive and negative images. Positive images are those images that consist of faces, and negative images are without faces. In face detection, image features are treated as numerical information extracted from the pictures that can distinguish one image from another.

We apply every feature of the algorithm on all the training images. Every image is given equal weight at the starting. It founds the best threshold which will categorize the faces to positive and negative. There may be errors and misclassifications. We select the features with a minimum error rate, which means these are the features that best classifies the face and non-face images.

All possible sizes and locations of each kernel are used to calculate the plenty of features.

HAAR-Cascade Detection in OpenCV

OpenCV provides the trainer as well as the detector. We can train the classifier for any object like cars, planes, and buildings by using the OpenCV. There are two primary states of the cascade image classifier first one is training and the other is detection.

OpenCV provides two applications to train cascade classifier **opencv_haartraining** and **opencv_traincascade**. These two applications store the classifier in the different file format.

For training, we need a set of samples. There are two types of samples:

- **Negative sample:** It is related to non-object images.
- **Positive samples:** It is a related image with detect objects.

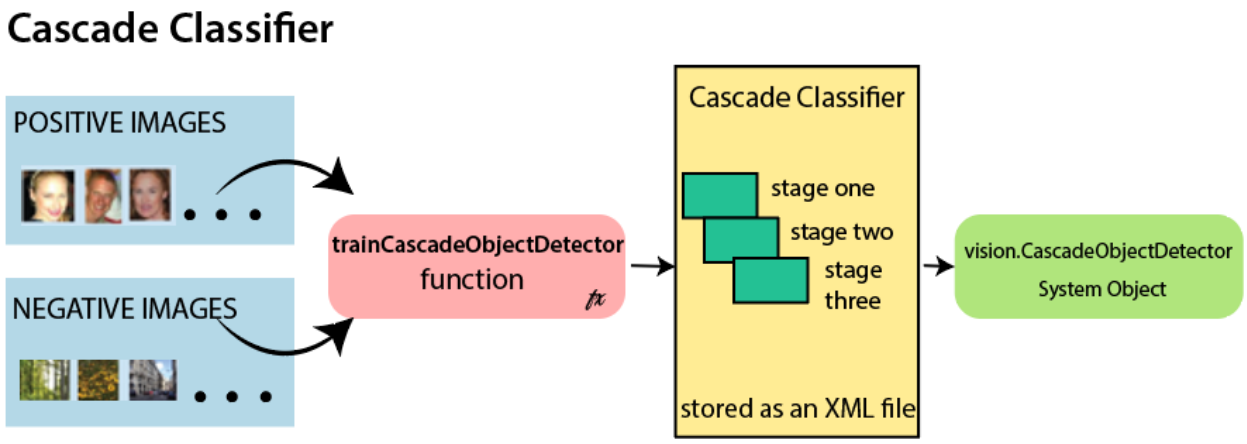
A set of negative samples must be prepared manually, whereas the collection of positive samples are created using the **opencv_createsamples** utility.

Negative Sample

Negative samples are taken from arbitrary images. Negative samples are added in a text file. Each line of the file contains an image filename (relative to the directory of the description file) of the negative sample. This file must be created manually. Defined images may be of different sizes.

Positive Sample

Positive samples are created by opencv_createsamples utility. These samples can be created from a single image with an object or from an earlier collection. It is important to remember that we require a large dataset of positive samples before you give it to the mentioned utility because it only applies the perspective transformation.



Here we will discuss detection. OpenCV already contains various pre-trained classifiers for face, eyes, smile, etc. Those XML files are stored in **opencv/data/haarcascades/** folder. Let's understand the following steps:

Step - 1

First, we need to load the necessary XML classifiers and load input images (or video) in grayscale mode.

Step -2

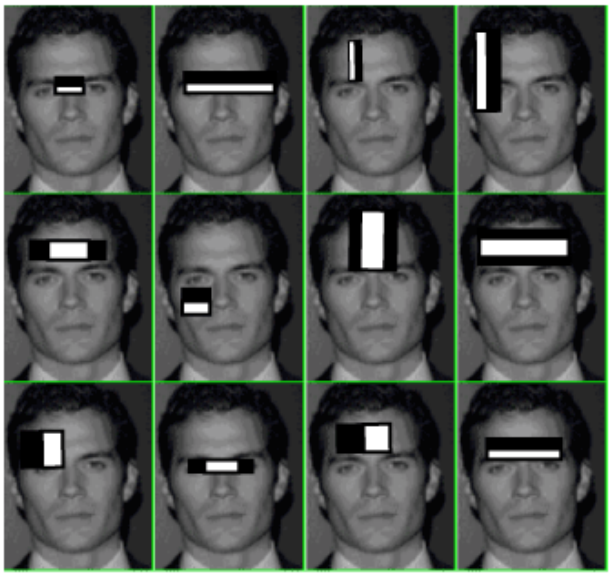
After converting the image into grayscale, we can do the image manipulation where the image can be resized, cropped, blurred, and sharpen if required. The next step is image segmentation; identify the multiple objects in the single image, so the classifier quickly detects the objects and faces in the picture.

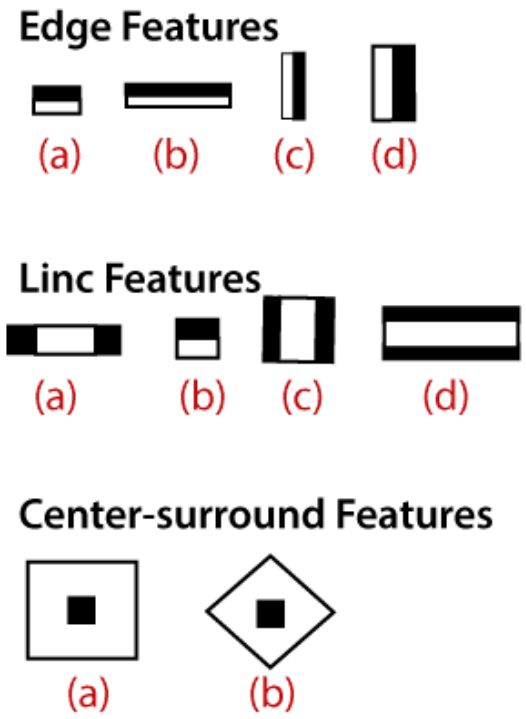
Step - 3

The haar-Like feature algorithm is used to find the location of the human faces in frame or image. All the Human faces have some common universal properties of faces like the eye region is darker than it's neighbor's pixels and nose region is more bright than the eye region.

Step -4

In this step, we extract the features from the image, with the help of edge detection, line detection, and center detection. Then provide the coordinate of x, y, w, h, which makes a rectangle box in the picture to show the location of the face. It can make a rectangle box in the desired area where it detects the face.





Face recognition using OpenCV

Face recognition is a simple task for humans. Successful face recognition tends to effective recognition of the inner features (eyes, nose, mouth) or outer features (head, face, hairline). Here the question is that how the human brain encode it?

David Hubel and **Torsten Wiesel** show that our brain has specialized nerve cells responding to unique local feature of the scene, such as lines, edges angle, or movement. Our brain combines the different sources of information into the useful patterns; we don't see the visual as scatters. If we define face recognition in the simple word, "Automatic face recognition is all about to take out those meaningful features from an image and putting them into a useful representation then perform some classification on them".

The basic idea of face recognition is based on the geometric features of a face. It is the feasible and most intuitive approach for face recognition. The first automated face recognition system was described in the position of eyes, ears, nose. These positioning points are called features vector (distance between the points).

The face recognition is achieved by calculating the **Euclidean** distance between feature vectors of a probe and reference image. This method is effective in illumination change by its nature, but it has a considerable drawback. The correct registration of the maker is very hard.

The face recognition system can operate basically in two modes:

- **Authentication or Verification of a facial image-**

It compares the input facial image with the facial image related to the user, which is required authentication. It is a 1x1 comparison.

- **Identification or facial recognition**

It basically compares the input facial images from a dataset to find the user that matches that input face. It is a 1xN comparison.

There are various types of face recognition algorithms, for example:

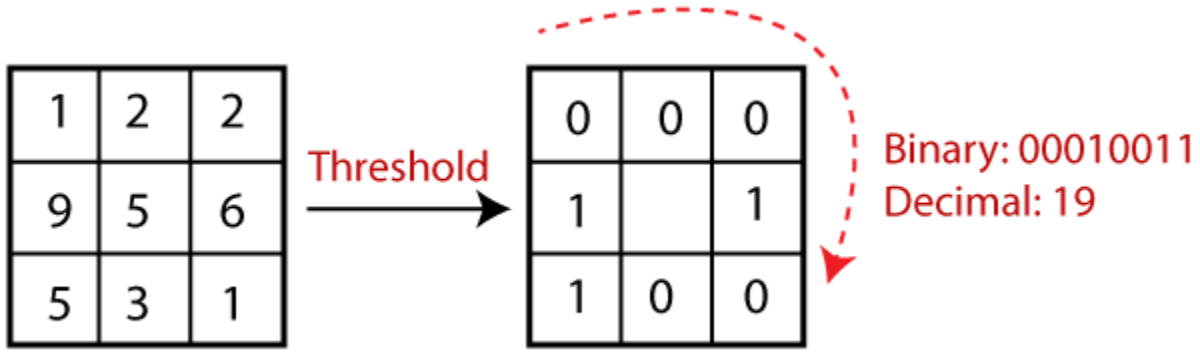
- **Eigenfaces (1991)**
- **Local Binary Patterns Histograms (LBPH) (1996)**
- **Fisherfaces (1997)**
- **Scale Invariant Feature Transform (SIFT) (1999)**
- **Speed Up Robust Features (SURF) (2006)**

Each algorithm follows the different approaches to extract the image information and perform the matching with the input image. Here we will discuss the Local Binary Patterns Histogram (LBPH) algorithm which is one of the oldest and popular algorithm.

Introduction of LBPH

Local Binary Pattern Histogram algorithm is a simple approach that labels the pixels of the image thresholding the neighborhood of each pixel. In other words, LBPH summarizes the local structure in an image by comparing each pixel with its neighbors and the result is converted into a binary number. It was first defined in 1994 (LBP) and since that time it has been found to be a powerful algorithm for texture classification.

This algorithm is generally focused on extracting local features from images. The basic idea is not to look at the whole image as a high-dimension vector; it only focuses on the local features of an object.



In the above image, take a pixel as center and threshold its neighbor against. If the intensity of the center pixel is greater-equal to its neighbor, then denote it with 1 and if not then denote it with 0.

Let's understand the steps of the algorithm:

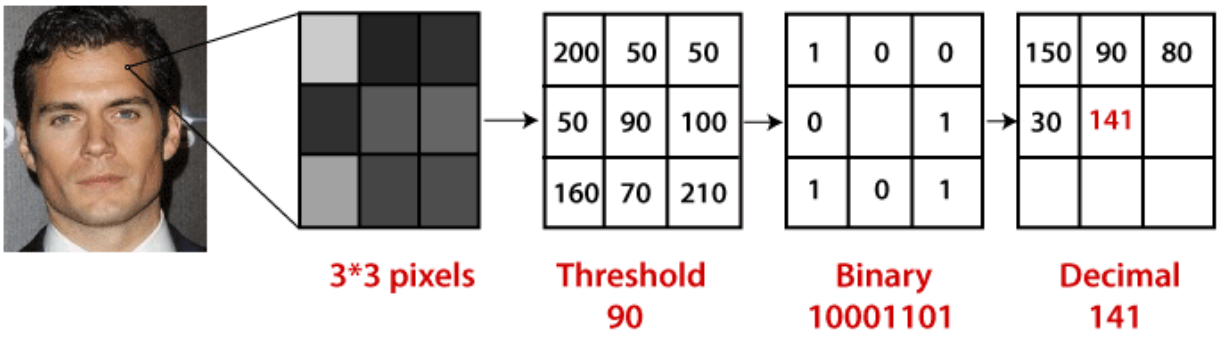
1. Selecting the Parameters: The LBPH accepts the four parameters:

- **Radius:** It represents the radius around the central pixel. It is usually set to 1. It is used to build the circular local binary pattern.
- **Neighbors:** The number of sample points to build the circular binary pattern.
- **Grid X:** The number of cells in the horizontal direction. The more cells and finer grid represents, the higher dimensionality of the resulting feature vector.
- **Grid Y:** The number of cells in the vertical direction. The more cells and finer grid represents, the higher dimensionality of the resulting feature vector.

Note: The above parameters are slightly confusing. It will be more clear in further steps.

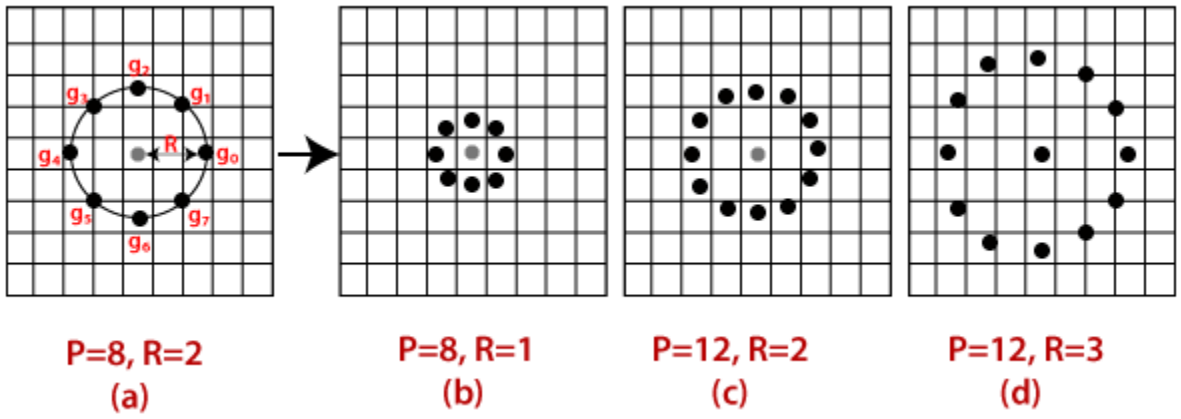
2. Training the Algorithm: The first step is to train the algorithm. It requires a dataset with the facial images of the person that we want to recognize. A unique ID (it may be a number or name of the person) should provide with each image. Then the algorithm uses this information to recognize an input image and give you the output. An Image of particular person must have the same ID. Let's understand the LBPH computational in the next step.

3. Using the LBP operation: In this step, LBP computation is used to create an intermediate image that describes the original image in a specific way through highlighting the facial characteristic. The parameters **radius** and **neighbors** are used in the concept of sliding window.

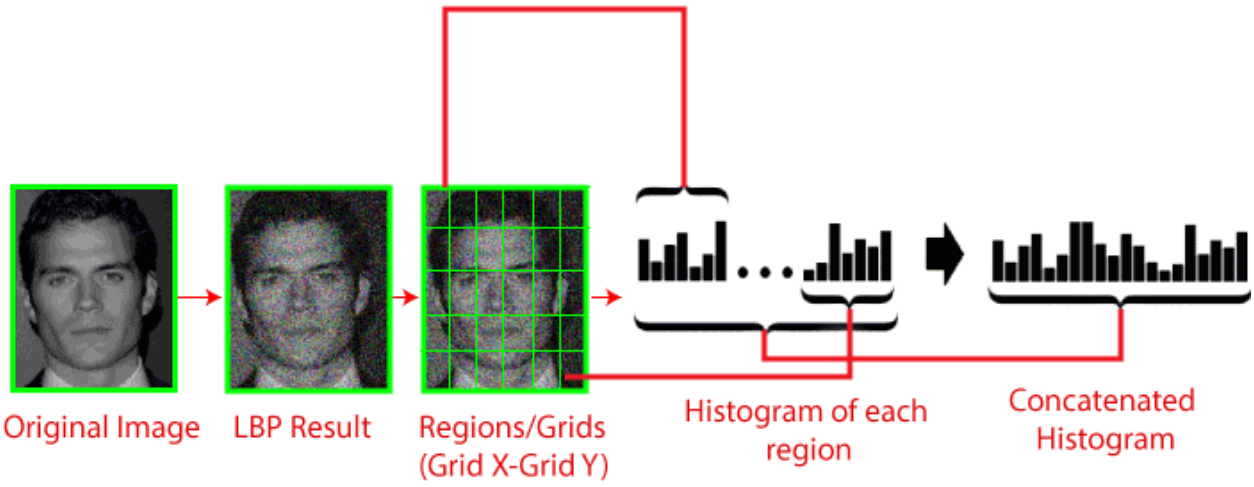


To understand in a more specific way, let's break it into several small steps:

- Suppose the input facial image is grayscale.
- We can get part of this image as a window of 3x3 pixels.
- We can use the 3x3 matrix containing the intensity of each pixel (0-255).
- Then, we need to take the central value of the matrix to be used as a threshold.
- This value will be used to define the new values from the 8 neighbors.
- For every neighbor of the central value (threshold), we set a new binary value. The value 1 is set for equal or higher than the threshold and 0 for values lower than the threshold.
- Now the matrix will consist of only binary values (skip the central value). We need to take care of each binary value from each position from the matrix line by line into new binary values (10001101). There are other approaches to concatenate the binary values (clockwise direction), but the final result will be the same.
- We convert this binary value to decimal value and set it to the central value of the matrix, which is a pixel from the original image.
- After completing the LBP procedure, we get the new image, which represents better characteristics of the original image.



4. Extracting the Histograms from the image: The image is generated in the last step, we can use the **Grid X** and **Grid Y** parameters to divide the image into multiple grids, let's consider the following image:



- We have an image in grayscale; each histogram (from each grid) will contain only 256 positions representing the occurrence of each pixel intensity.
- It is required to create a new bigger histogram by concatenating each histogram.

5. Performing face recognition: Now, the algorithm is well trained. The extracted histogram is used to represent each image from the training dataset. For the new image, we perform steps again and create a new histogram. To find the image that matches the given image, we just need to match two histograms and return the image with the closest histogram.

- There are various approaches to compare the histograms (calculate the distance between two histograms), for example: **Euclidean distance, chi-square, absolute value,** etc. We can use the Euclidean distance based on the following formula:

$$D = \sqrt{\sum_{i=1}^{n}(\text{hist}\,1_i - \text{hist}2_i)^2}$$

- The algorithm will return ID as an output from the image with the closest histogram. The algorithm should also return the calculated distance that can be called **confidence** measurement. If the confidence is lower than the threshold value, that means the algorithm has successfully recognized the face.

We have discussed the face detection and face recognition. The haar like cascade algorithm is used for face detection. There are various algorithms for face recognition, but LBPH is easy and popular algorithm among them. It generally focuses on the local features in the image.

Limitation in the Face Detection

The Facial Recognition System is essential nowadays, and it has come a long way. Its use is essential in quite some applications, for example - Photo retrieval, surveillance, authentication/access, control systems etc. But there are a few challenges that have continuously occurred during image or face recognition system.

These challenges need to be overcome to create more effective face recognition systems. The Following are the challenges which affect the ability of Facial Recognition System to go that extra mile.

- **Illumination**

The illumination plays an essential role during image recognition. If there is a slight change in lighting conditions, it will make major impact on its results. It is the lighting to vary, and then the result may be different for the same object cause of low or high illumination.

- **Background**

The background of the object also plays a significant role in Face detection. The result might not the same outdoor as compared to what is produces indoors because the factor - affecting its performance-change as soon as the locations change.

- **Pose**

The facial recognition system is highly sensitive to pose variations. The movement of head or different camera positions can cause changes of facial texture and it will generate the wrong result.

- **Occlusion**

Occlusion means the face as beard, mustache, accessories (goggles, caps, mask, etc.) also interfere with the estimate of a face recognition system.

- **Expressions**

Another important factor that should be kept in mind is the different expression of the same individual. Change in facial expressions may produce a different result for the same individual.

In this tutorial, we have learned about the OpenCV library and its basic concept. We have described all the basic operation of the image. In the next tutorial we will learn about the face recognition and face detection