K J Somaiya Institute of Management

Vidyavihar, Mumbai – 400 077

Masters in Computer Applications
Semester IV (2020 – 21)

This is to certify that Mr. /~~Ms~~. Mrutunjay Biswal Roll No. 06 of MCA has completed his/~~her~~ Image Processing & Computer Vision Journal as per the Syllabus for the Academic year 2020 – 2021.The Journal has also been evaluated by the concerned faculty of K J Somaiya Institute of Management.

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Signature of the Faculty-Incharge

_ _ _ _ _ _ _ _ _ _ _ _ _ _

Signature of the Course Coordinator – MCA

Date: _____

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Signature of the External Examiner

INDEX

Practical 1

- To load any media in Google Colab, we need to use below code –

```
from google.colab import files
uploaded = files.upload()
```

- To list all files that are uploaded in Google Colab, we need to use below code –

```
import os
!ls
os.getcwd()
```

- In order to write any programs in OPENCV, it requires opencv-python module or cv2 module. We need to import it in every program and this is how it is done –

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
```

Note :- We also require numpy module as it has maximum usage in upcoming programs.
We require this module because, Colab is cloud service.
"from google.colab.patches import cv2_imshow"

- To read and show image, we need imread and imshow functions. Below is sample code of it.

```
img1=cv2.imread("leo.jpg")
cv2_imshow(img1)
```

Explaination –

1. imread function takes one parameter, which is source image. It reads the image. We can convert the image to grayscale at the time of reading the image. This can be done by giving second parameter as "0". This will make grayscale image. In the end we are assigning the input image to a variable.
2. Cv2_imshow function shows the image which was stored in the variable.

Note :- Parameters of above functions differ according to different platforms.

- To draw shapes, there are some functions available and they are as follows.

cv2.line	cv2.rectangle	cv2.circle	cv2.ellipse	cv2.polylines	cv2.putText

Explanation of these functions -

1. cv2.line – `this` method is used to draw a line on any image

   Syntax: cv2.line(image, start_point, end_point, color, thickness)
   Parameters:
   image: It is the image on which line is to be drawn.
   start_point: It is the starting coordinates of line. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
   end_point: It is the ending coordinates of line. The coordinates are represented as

tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
color: It is the color of line to be drawn. For **BGR**, we pass a tuple. eg: (255, 0, 0) for blue color.
thickness: It is the thickness of the line in **px**.
Return Value: It returns an image

2. cv2.rectangle - `cv2.rectangle()` method is used to draw a rectangle on any image.

Syntax: cv2.rectangle(image, start_point, end_point, color, thickness)
Parameters:
image: It is the image on which rectangle is to be drawn.
start_point: It is the starting coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
end_point: It is the ending coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
color: It is the color of border line of rectangle to be drawn. For **BGR**, we pass a tuple. eg: (255, 0, 0) for blue color.
thickness: It is the thickness of the rectangle border line in **px**. Thickness of **-1 px** will fill the rectangle shape by the specified color.
Return Value: It returns an image.

3. cv2.circle - cv2.circle() method is used to draw a circle on any image.

Syntax: cv2.circle(image, center_coordinates, radius, color, thickness)
Parameters:
image: It is the image on which circle is to be drawn.
center_coordinates: It is the center coordinates of circle. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
radius: It is the radius of circle.
color: It is the color of border line of circle to be drawn. For **BGR**, we pass a tuple. eg: (255, 0, 0) for blue color.
thickness: It is the thickness of the circle border line in **px**. Thickness of **-1 px** will fill the circle shape by the specified color.
Return Value: It returns an image.

4. cv2.ellipse - `cv2.ellipse()` method is used to draw a ellipse on any image.

Syntax: cv2.ellipse(image, centerCoordinates, axesLength, angle, startAngle, endAngle, color [, thickness[, lineType[, shift]]])
Parameters:
image: It is the image on which ellipse is to be drawn.
centerCoordinates: It is the center coordinates of ellipse. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
axesLength: It contains tuple of two variables containing major and minor axis of ellipse (major axis length, minor axis length).
angle: Ellipse rotation angle in degrees.
startAngle: Starting angle of the elliptic arc in degrees.
endAngle: Ending angle of the elliptic arc in degrees.
color: It is the color of border line of shape to be drawn. For **BGR**, we pass a tuple. eg: (255, 0, 0) for blue color.
thickness: It is the thickness of the shape border line in **px**. Thickness of **-1 px** will fill the shape by the specified color.
lineType: This is an optional parameter.It gives the type of the ellipse boundary.
shift: This is an optional parameter. It denotes the number of fractional bits in the coordinates of the center and values of axes.
Return Value: It returns an image.

5. cv2.polylines - **cv2.polylines()** method is used to draw a polygon on any image.

Syntax: cv2.polylines(image, [pts], isClosed, color, thickness)
Parameters:
image: It is the image on which circle is to be drawn.
pts: Array of polygonal curves.
npts: Array of polygon vertex counters.
ncontours: Number of curves.
isClosed: Flag indicating whether the drawn polylines are closed or not. If they are closed, the function draws a line from the last vertex of each curve to its first vertex.
color: It is the color of polyline to be drawn. For BGR, we pass a tuple.
thickness: It is thickness of the polyline edges.
Return Value: It returns an image.

6. cv2.putText - `cv2.putText()` method is used to draw a text string on any image.

Syntax: cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])
Parameters:
image: It is the image on which text is to be drawn.
text: Text string to be drawn.
org: It is the coordinates of the bottom-left corner of the text string in the image. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).
font: It denotes the font type. Some of font types are **FONT_HERSHEY_SIMPLEX, FONT_HERSHEY_PLAIN,** , etc.
fontScale: Font scale factor that is multiplied by the font-specific base size.
color: It is the color of text string to be drawn. For **BGR**, we pass a tuple. eg: (255, 0, 0) for blue color.
thickness: It is the thickness of the line in **px**.
lineType: This is an optional parameter.It gives the type of the line to be used.
bottomLeftOrigin: This is an optional parameter. When it is true, the image data origin is at the bottom-left corner. Otherwise, it is at the top-left corner.
Return Value: It returns an image.

- Showing Image using Matplotlib.

About Matplotlib - Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.
One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.
To import it, we use this code –

```
from matplotlib import pyplot as plt
```

Following are some functions in it –

`plt.imread()`	`plt.xticks([])`	`plt.yticks([])`	`plt.show()`

Explanation of these functions –

1. Plt.imread() - The **imread() function** in pyplot module of matplotlib library is used to read an image from a file into an array.

Syntax: matplotlib.pyplot.imread(fname, format=None)

Parameters: This method accepts the following parameters.
fname : This parameter is the image file to read.
format: This parameter is the image file format assumed for reading the data.
Returns: This method return the following.
imagedata : This returns the image data

2. Plt.xticks([]) & Plt.yticks([]) - is used to get and set the current tick locations and labels of the x-axis and y-axis.

Syntax:
matplotlib.pyplot.xticks(ticks=None, labels=None, **kwargs)

Parameters: This method accept the following parameters that are described below:
ticks: This parameter is the list of xtick locations. and an optional parameter. If an empty list is passed as an argument then it will removes all xticks
labels: This parameter contains labels to place at the given ticks locations. And it is an optional parameter.
****kwargs:** This parameter is Text properties that is used to control the appearance of the labels.
Returns: This returns the following:
locs : This returns the list of ytick locations.
labels : This returns the list of ylabel Text objects.

3. Plt.show() - The **show() function** in pyplot module of matplotlib library is used to display all figures.

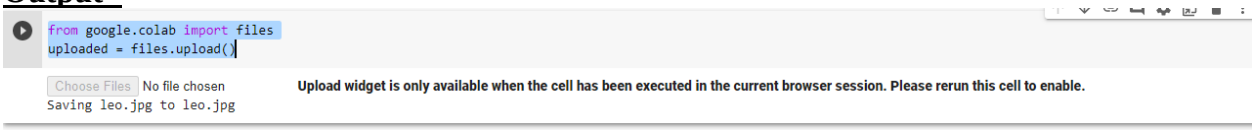
Syntax:
matplotlib.pyplot.show(*args, **kw)

Parameters: This method accepts only one parameter which is discussed below:
block : This parameter is used to override the blocking behavior described above.
Returns: This method does not return any value.

Practical 1a) Uploading Media.

```
from google.colab import files
uploaded = files.upload()
```

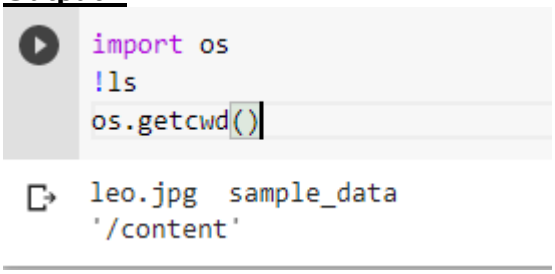
Output -



Practical 1b) Listing file contents.

```
import os
!ls
os.getcwd()
```

Output –



Practical 1c) imread and imshow functions.

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

img1=cv2.imread("leo.jpg")
cv2_imshow(img1)
```

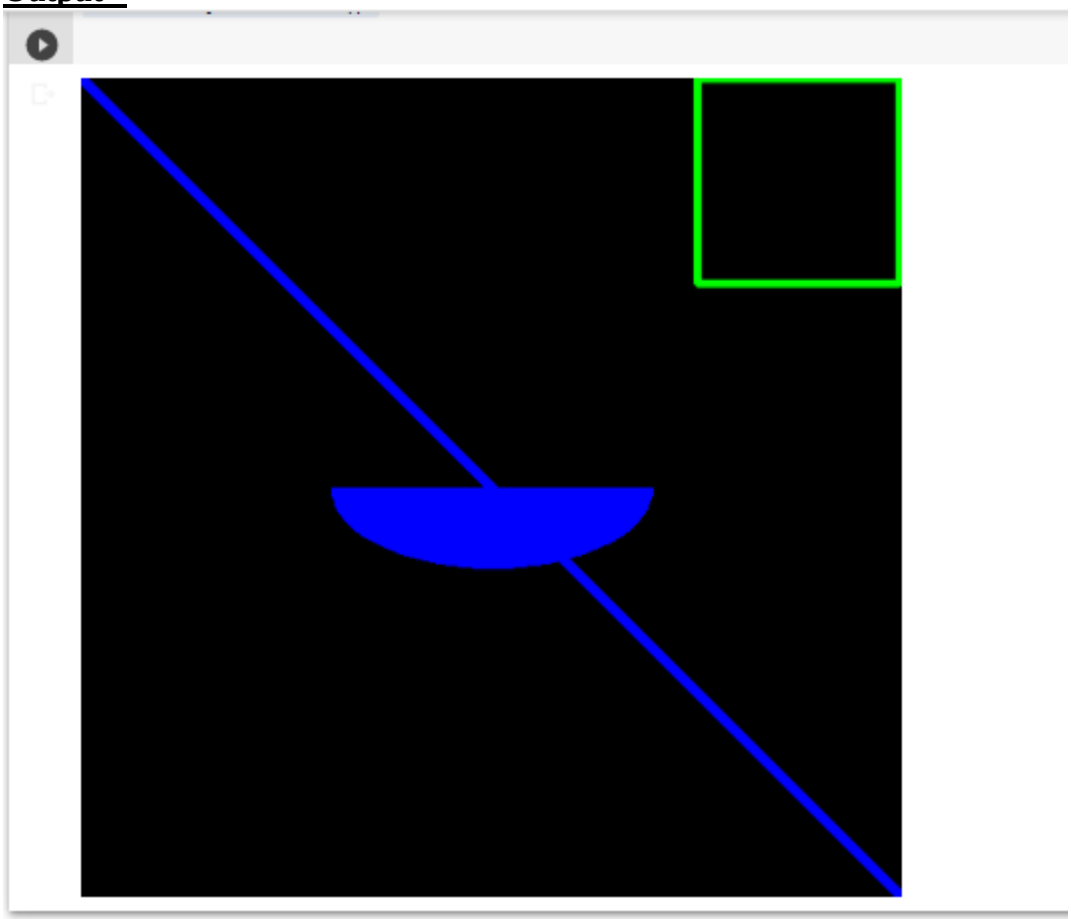
Output –

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

img1=cv2.imread("leo.jpg")
cv2_imshow(img1)
```

Practical 1d) Drawing Shapes.

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

#create a black image
img=np.zeros((512,512,3),np.uint8)
#draw a diagonal blue line with thickness of 5px
img=cv2.line(img,(0,0),(511,511),(255,0,0),5)
img=cv2.rectangle(img,(384,0),(510,128),(0,255,0),3)
#img=cv2.circle(img,(447,63),25,(0,0,255),-1)
img=cv2.ellipse(img,(256,256),(100,50),0,0,180,255,-1)
cv2_imshow(img)
cv2.waitKey(1)
cv2.destroyAllWindows()
```

Output –



Practical 1e) Polylines.

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

img=np.zeros((512,512,3),np.uint8)
pts=np.array([[10,5],[20,30],[70,20],[50,10]],np.int32)
pts=pts.reshape((-1,1,2))
img=cv2.polylines(img,[pts],True,(0,255,255))
font=cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,500),font,4,(255,255,255),2,cv2.LINE_AA)
cv2_imshow(img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output –

Practical 1f) Using Matplotlib module and showing image.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
%matplotlib inline
#img=cv2.imread("leo.jpg")
img=plt.imread("leo.jpg")
plt.imshow(img)
#plt.xticks([]),plt.yticks([])
#plt.show()
```

Output –

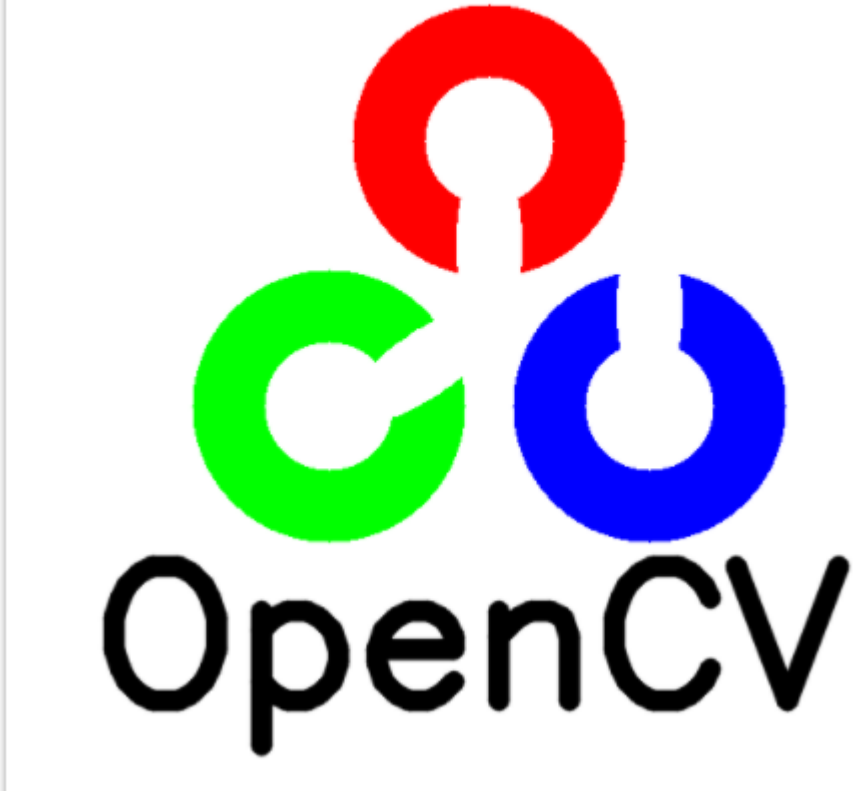
Assignment Exercise 1

1. OPEN CV Logo.

```
#import numpy as np
import cv2
from google.colab.patches import cv2_imshow

#img=np.zeros((512,512,3),np.uint8)
img[:]=(255,255,255)
#Selecting Font Name and Inserting Text
font=cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img,'OpenCV',(10,440),font,4,(0,0,0),12,cv2.LINE_AA)
#Red Circle
img=cv2.circle(img,(256,90),85,(0,0,255),-1)
#White Circle inside Red Circle
img=cv2.circle(img,(256,90),40,(255,255,255),-1)
#White Ellipse to cut the Red Circle
img=cv2.ellipse(img,(256,155),(20,60),0,0,360,(255,255,255),-1)
#Green Circle
img=cv2.circle(img,(156,256),85,(0,255,0),-1)
#White Circle inside Green circle
img=cv2.circle(img,(156,256),40,(255,255,255),-1)
#White Ellipse to cut Green Circle
img=cv2.ellipse(img,(215,230),(20,55),55,0,360,(255,255,255),-1)
#Blue Circle
img=cv2.circle(img,(356,256),85,(255,0,0),-1)
#White Circle inside Blue Circle
img=cv2.circle(img,(356,256),40,(255,255,255),-1)
#White Ellipse to cut Blue Circle
img=cv2.ellipse(img,(356,195),(20,55),0,0,360,(255,255,255),-1)
cv2_imshow(img)
```

Output –

```
#White Ellipse to cut Blue Circle
img=cv2.ellipse(img,(356,195),(20,55),0,0,360,(255,255,255),-1)
cv2_imshow(img)
```



2. Snowman

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

#create an image and filling with some color.
img=np.zeros((512,512,3),np.uint8)
img.fill(150)
#Head of Snowman
img=cv2.circle(img,(256,115),70,(255,255,255),-1)
#Body of Snowman (3-Circles)
img=cv2.circle(img,(256,230),85,(255,255,255),-1)
img=cv2.circle(img,(256,290),85,(255,255,255),-1)
img=cv2.circle(img,(256,360),85,(255,255,255),-1)
#Eyes of Snowman (2-Circles)
img=cv2.circle(img,(225,90),10,(0,0,0),-1)
img=cv2.circle(img,(285,90),10,(0,0,0),-1)
#Button of Snowman (5-Circles)
img=cv2.circle(img,(256,200),5,(0,0,0),-1)
img=cv2.circle(img,(256,220),5,(0,0,0),-1)
img=cv2.circle(img,(256,240),5,(0,0,0),-1)
img=cv2.circle(img,(256,260),5,(0,0,0),-1)
img=cv2.circle(img,(256,280),5,(0,0,0),-1)
#Nose of Snowman (1-Ellipse)
img=cv2.ellipse(img,(256,115),(10,15),0,0,360,(0,0,255),-1)
#Hand of Snowman (2-Rectangles)
cv2.rectangle(img, (70, 250) , (210,256) , (42,42,165) , -1)
cv2.rectangle(img, (450, 250) , (310,256) , (42,42,165) , -1)

#Grass on which snowman is standing (1-Rectangle)
cv2.rectangle(img, (0, 420) , (512,512) , (34,139,34) , -1)
#Ice Falling (Ellipse)
img=cv2.ellipse(img,(50,115),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(80,130),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(120,145),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(160,145),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(120,145),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(60,145),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(20,145),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(20,200),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(20,300),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(100,220),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(60,260),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(40,20),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(100,50),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(390,50),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(320,60),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(320,60),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(400,60),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(400,80),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(450,80),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(490,40),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(490,300),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(450,330),(2,7),15,0,360,(255,255,255),-1)
```

```
img=cv2.ellipse(img,(500,350),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(390,260),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(500,260),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(430,260),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(390,280),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(400,120),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(420,100),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(300,100),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(450,120),(2,7),15,0,360,(255,255,255),-1)
img=cv2.ellipse(img,(480,150),(2,7),15,0,360,(255,255,255),-1)
#Ice on grass (Circles)
img=cv2.circle(img,(20,450),3,(255,255,255),-1)
img=cv2.circle(img,(40,480),3,(255,255,255),-1)
img=cv2.circle(img,(100,450),3,(255,255,255),-1)
img=cv2.circle(img,(150,420),3,(255,255,255),-1)
img=cv2.circle(img,(180,440),3,(255,255,255),-1)
img=cv2.circle(img,(256,300),3,(255,255,255),-1)
img=cv2.circle(img,(280,495),3,(255,255,255),-1)
img=cv2.circle(img,(360,485),3,(255,255,255),-1)
img=cv2.circle(img,(400,510),3,(255,255,255),-1)
img=cv2.circle(img,(500,500),3,(255,255,255),-1)
img=cv2.circle(img,(510,490),3,(255,255,255),-1)
img=cv2.circle(img,(485,463),3,(255,255,255),-1)
#Hat of Snowman (Rectangle & Ellipse)
cv2.rectangle(img, (200, 10) , (310,50) , (0,0,0) , -1)
img=cv2.ellipse(img,(256,50),(70,10),0,0,360,(0,0,0),-1)
#Mouth of Snowman (Ellipse)
img=cv2.ellipse(img,(256,140),(40,10),0,0,180,(0,0,0),-1)
cv2_imshow(img)
cv2.waitKey(1)
cv2.destroyAllWindows()
```

Output –

```
cv2.rectangle(img, (200, 10) , (310,50) , (0,0,0) , -1)
img=cv2.ellipse(img,(256,50),(70,10),0,0,360,(0,0,0),-1)
#Mouth of Snowman (Ellipse)
img=cv2.ellipse(img,(256,140),(40,10),0,0,180,(0,0,0),-1)
cv2_imshow(img)
cv2.waitKey(1)
cv2.destroyAllWindows()
```

Practical 2

- Thresholding

cv2.threshold - the function **cv2.threshold** is used for thresholding.
Syntax: cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)
Parameters:
-> **source**: Input Image array (must be in Grayscale).
-> **thresholdValue**: Value of Threshold below and above which pixel values will change accordingly.
-> **maxVal**: Maximum value that can be assigned to a pixel.
-> **thresholdingTechnique**: The type of thresholding to be applied.

The basic Thresholding technique is Binary Thresholding. For every pixel, the same threshold value is applied. If the pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to a maximum value.

The different Simple Thresholding Techniques are:

1. **cv2.THRESH_BINARY**: If pixel intensity is greater than the set threshold, value set to 255, else set to 0 (black).
2. **cv2.THRESH_BINARY_INV**: Inverted or Opposite case of cv2.THRESH_BINARY.
3. **cv.THRESH_TRUNC**: If pixel intensity value is greater than threshold, it is truncated to the threshold. The pixel values are set to be the same as the threshold. All other values remain the same.
4. **cv.THRESH_TOZERO**: Pixel intensity is set to 0, for all the pixels intensity, less than the threshold value.
5. **cv.THRESH_TOZERO_INV**: Inverted or Opposite case of cv2.THRESH_TOZERO.

- Log transformation

Logarithmic transformation of an image is one of the gray level image transformations. Log transformation of an image means replacing all pixel values, present in the image, with its logarithmic values. Log transformation is used for image enhancement as it expands dark pixels of the image as compared to higher pixel values.
The formula for applying log transformation in an image is,
$S = c * \log (1 + r)$
where,
R = input pixel value,
C = scaling constant and
S = output pixel value
The value of ‘c’ is chosen such that we get the maximum output value corresponding to the bit size used. So, the formula for calculating ‘c’ is as follows:
$c = 255 / (\log (1 + max\_input\_pixel\_value))$
When we apply log transformation in an image and any pixel value is ‘0’ then its log value will become infinite. That’s why we are adding ‘1’ to each pixel value at the time of log transformation so that if any pixel value is ‘0’, it will become ‘1’ and its log value will be ‘0’.

- Power-Law (Gamma) Transformation –

Power-law (gamma) transformations can be mathematically expressed as s = cr^{\gamma}. Gamma correction is important for displaying images on a screen correctly, to prevent bleaching or darkening of images when viewed from different types of monitors with different display settings. This is done because our eyes perceive images in a gamma-shaped curve, whereas cameras capture images in a linear fashion. Below is the Python code to apply gamma correction.

- Image Negative -

There 2 different ways to transform an image to negative using the OpenCV module. The first method explains negative transformation step by step and the second method explains negative transformation of an image in single line.

First method: Steps for negative transformation

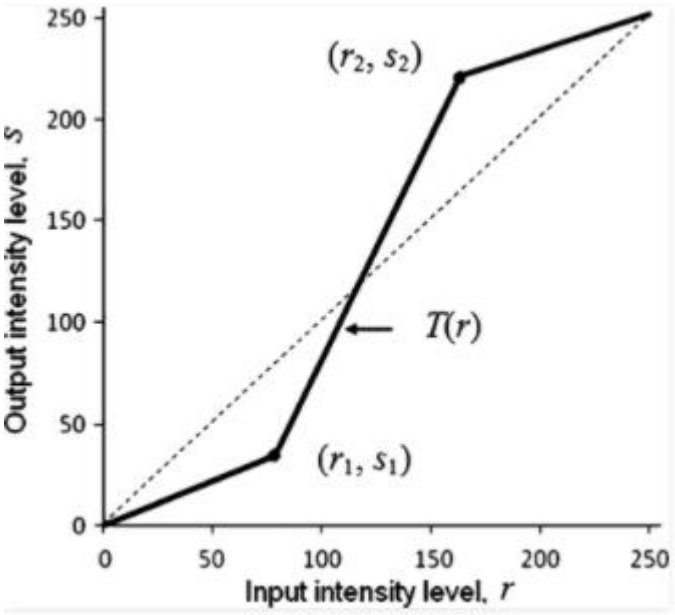
1. Read an image
2. Get height and width of the image
3. Each pixel contains 3 channels. So, take a pixel value and collect 3 channels in 3 different variables.
4. Negate 3 pixels values from 255 and store them again in pixel used before.
5. Do it for all pixel values present in image.

2nd method: Steps for negative transformation

1. Read an image and store it in a variable.
2. Subtract the variable from 1 and store the value in another variable.
3. All done. You successfully done the negative transformation.

- Contrast Stretching-

Contrast can be defined as:
Contrast = (I_max - I_min)/(I_max + I_min)
This process expands the range of intensity levels in an image so that it spans the full intensity of the camera/display. The figure below shows the graph corresponding to the contrast stretching.



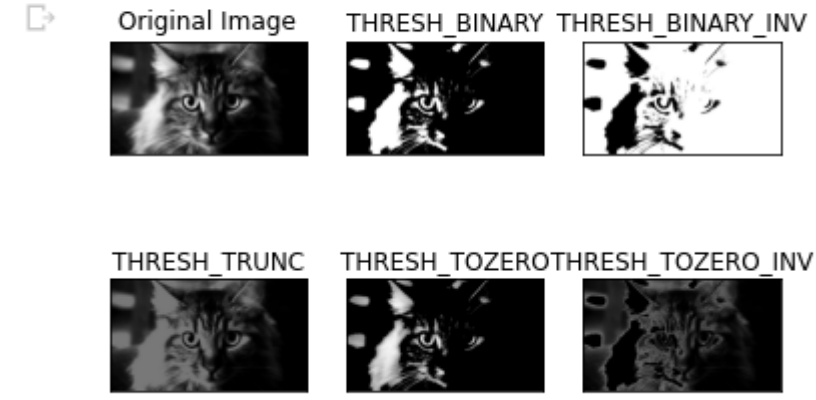
With (r1, s1), (r2, s2) as parameters, the function stretches the intensity levels by essentially decreasing the intensity of the dark pixels and increasing the intensity of the light pixels. If `r1 = s1 = 0` and `r2 = s2 = L-1`, the function becomes a straight dotted line in the graph (which gives no effect). The function is monotonically increasing so that the order of intensity levels between pixels is preserved.

Practical 2a) Thresholding

```
#Thresholding
import cv2
from matplotlib import pyplot as plt
img = cv2.imread("bw.jpg")
ret,thresh1=cv2.threshold(img,120,255,cv2.THRESH_BINARY)
ret,thresh2=cv2.threshold(img,120,255,cv2.THRESH_BINARY_INV)
ret,thresh3=cv2.threshold(img,120,255,cv2.THRESH_TRUNC)
ret,thresh4=cv2.threshold(img,120,255,cv2.THRESH_TOZERO)
ret,thresh5=cv2.threshold(img,120,255,cv2.THRESH_TOZERO_INV)
titles=['Original
Image','THRESH_BINARY','THRESH_BINARY_INV','THRESH_TRUNC','THRESH_TO
ZERO','THRESH_TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]
for i in range(6):
  plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
  plt.title(titles[i])
  plt.xticks([]),plt.yticks([])
plt.show()
```

Output –

```
  plt.title(titles[i])
  plt.xticks([]),plt.yticks([])
plt.show()
```



Practical 2b) Log Transformation

```
#log transformation
import cv2
import numpy as np
#Load the image
img=cv2.imread("bw.jpg")
#Apply log transform
img_log=(np.log(img+1)/(np.log(1+np.max(img))))*255
#Specify the data type
img_log=np.array(img_log,dtype=np.uint8)
#Display the image
cv2_imshow(img_log)
cv2_imshow(img)
```

Output –

Input Image	Output Image
	

Practical 2c) Power Law Transformation

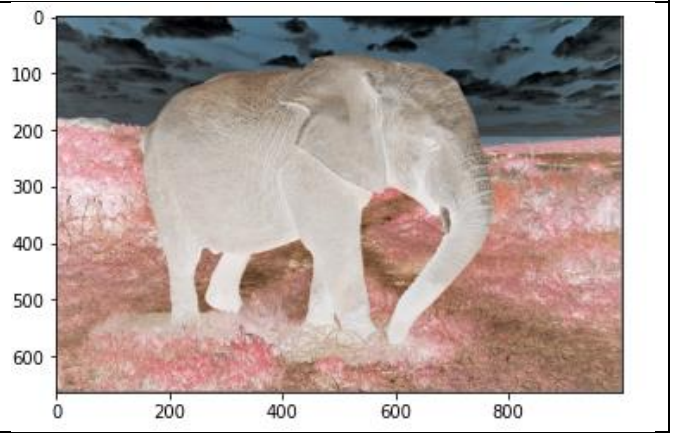
```
#Power Law Transformation - (Working)
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
img1= cv2.imread("elephnant.jpg")
cv2_imshow(img1)
#img2= img1/255.0
im1=np.array(255*(img1/255)**0.6, dtype="uint8")
cv2_imshow(im1)
```

Input Image	Output Image
	

Practical 2d) Image Negative Method-1

```
#Image Negative
import numpy as np
import cv2
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
img = cv2.imread("elephnant.jpg")
cv2_imshow(img)
#plt.imshow(img)
#plt.show()

img.shape
height, width, _ = img.shape
for i in range(0, height-1):
  for j in range(0,width-1):
    #Get the pixel value
    pixel = img[i,j]

    #Negate each channel by
    #Subtracting it from 255

    # 1st index contains red pixel
    pixel[0]= 255 - pixel[0]

    #2nd index contains green pixel
    pixel[1]= 255 - pixel[1]

    #3rd index contains blue pixel
    pixel[2]= 255 - pixel[2]

    #Store new values in the pixel
    img[i,j]=pixel

#Display the negative transformed image
plt.imshow(img)
plt.show()

#img_not=1-img
#cv2_imshow(img_not)
```

Output –

Input Image	Output Image
	

Practical 2d) Image Negative Method-2

```
#Black & White Image
import cv2
from google.colab.patches import cv2_imshow
img = cv2.imread("bw.jpg")
cv2_imshow(img)

img_not=1-img
cv2_imshow(img_not)
```

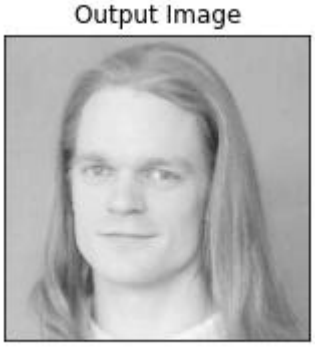
Output –

Input Image	Output Image
	

Practical 2e) Contrast Stretching

```
#Contrast Stretching
import cv2
from matplotlib import pyplot as plt
img = cv2.imread("lc.png")
nmax = 255
nmin = 0
out = cv2.normalize(img,None,alpha = nmin, beta = nmax, norm_type=cv2.NORM_MINMA
X)
plt.subplot(1,2,1), plt.imshow(img,cmap='gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2), plt.imshow(out,cmap='gray')
plt.title('Output Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

Output –

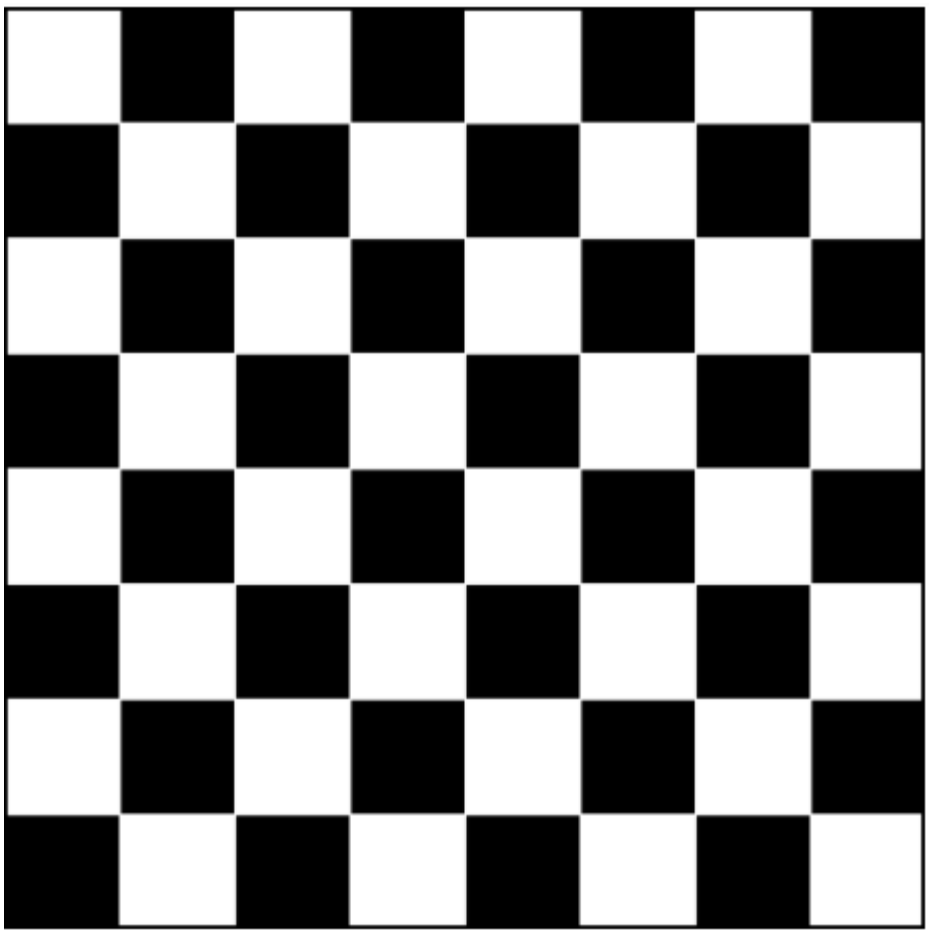
Input Image	Output Image
	

Assignment Exercise 2

1. Chessboard

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

#create a black image
img=np.zeros((512,512,3),np.uint8)

cv2.rectangle(img, (2, 2) , (64,64) , (255,255,255) , -1)
cv2.rectangle(img, (128,2) , (192,64) , (255,255,255) , -1)
cv2.rectangle(img, (256,2) , (320,64) , (255,255,255) , -1)
cv2.rectangle(img, (384,2) , (448,64) , (255,255,255) , -1)

cv2.rectangle(img, (64,64) , (128,128) , (255,255,255) , -1)
cv2.rectangle(img, (192,64) , (256,128) , (255,255,255) , -1)
cv2.rectangle(img, (320,64) , (384,128) , (255,255,255) , -1)
cv2.rectangle(img, (448,64) , (509,128) , (255,255,255) , -1)

cv2.rectangle(img, (2,128) , (64,192) , (255,255,255) , -1)
cv2.rectangle(img, (128,128) , (192,192) , (255,255,255) , -1)
cv2.rectangle(img, (256,128) , (320,192) , (255,255,255) , -1)
cv2.rectangle(img, (384,128) , (448,192) , (255,255,255) , -1)

cv2.rectangle(img, (64,192) , (128,256) , (255,255,255) , -1)
cv2.rectangle(img, (192,192) , (256,256) , (255,255,255) , -1)
cv2.rectangle(img, (320,192) , (384,256) , (255,255,255) , -1)
cv2.rectangle(img, (448,192) , (509,256) , (255,255,255) , -1)

cv2.rectangle(img, (2,256) , (64,320) , (255,255,255) , -1)
cv2.rectangle(img, (128,256) , (192,320) , (255,255,255) , -1)
cv2.rectangle(img, (256,256) , (320,320) , (255,255,255) , -1)
cv2.rectangle(img, (384,256) , (448,320) , (255,255,255) , -1)

cv2.rectangle(img, (64,320) , (128,384) , (255,255,255) , -1)
cv2.rectangle(img, (192,320) , (256,384) , (255,255,255) , -1)
cv2.rectangle(img, (320,320) , (384,384) , (255,255,255) , -1)
cv2.rectangle(img, (448,320) , (509,384) , (255,255,255) , -1)

cv2.rectangle(img, (2,384) , (64,448) , (255,255,255) , -1)
cv2.rectangle(img, (128,384) , (192,448) , (255,255,255) , -1)
cv2.rectangle(img, (256,384) , (320,448) , (255,255,255) , -1)
cv2.rectangle(img, (384,384) , (448,448) , (255,255,255) , -1)

cv2.rectangle(img, (64,448) , (128,509) , (255,255,255) , -1)
cv2.rectangle(img, (192,448) , (256,509) , (255,255,255) , -1)
cv2.rectangle(img, (320,448) , (384,509) , (255,255,255) , -1)
cv2.rectangle(img, (448,448) , (509,509) , (255,255,255) , -1)

cv2_imshow(img)
```

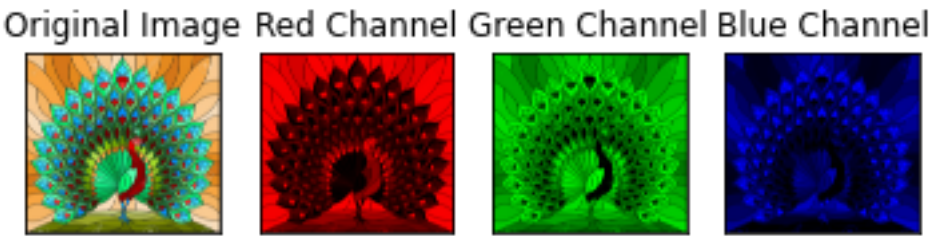
Output –



2. Isolation of Color Channels

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

img = cv2.imread("peacock.jpg")

plt.subplot(2,4,1), plt.imshow(img)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])

# WAY -1 (WORKING)
img_1=img.copy()
img_1[:,:,1]=0
img_1[:,:,2]=0
plt.subplot(2,4,2), plt.imshow(img_1)
plt.title('Red Channel'), plt.xticks([]), plt.yticks([])

img_2=img.copy()
img_2[:,:,0]=0
img_2[:,:,2]=0
plt.subplot(2,4,3), plt.imshow(img_2)
plt.title('Green Channel'), plt.xticks([]), plt.yticks([])

img_3=img.copy()
img_3[:,:,0]=0
img_3[:,:,1]=0
plt.subplot(2,4,4), plt.imshow(img_3)
plt.title('Blue Channel'), plt.xticks([]), plt.yticks([])
```

Output –



Practical 3

- Hist() – The **hist() function** in pyplot module of matplotlib library is used to plot a histogram.

Syntax: matplotlib.pyplot.hist(x, bins=None, range=None, density=False, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, *, data=None, **kwargs)

Parameters: This method accept the following parameters that are described below:

x : This parameter are the sequence of data.
bins : This parameter is an optional parameter and it contains the integer or sequence or string.
range : This parameter is an optional parameter and it the lower and upper range of the bins.
density : This parameter is an optional parameter and it contains the boolean values.
weights : This parameter is an optional parameter and it is an array of weights, of the same shape as x.
bottom : This parameter is the location of the bottom baseline of each bin.
histtype : This parameter is an optional parameter and it is used to draw type of histogram. {'bar', 'barstacked', 'step', 'stepfilled'}
align : This parameter is an optional parameter and it controls how the histogram is plotted. {'left', 'mid', 'right'}
rwidth : This parameter is an optional parameter and it is a relative width of the bars as a fraction of the bin width
log : This parameter is an optional parameter and it is used to set histogram axis to a log scale
color : This parameter is an optional parameter and it is a color spec or sequence of color specs, one per dataset.
label : This parameter is an optional parameter and it is a string, or sequence of strings to match multiple datasets.
normed : This parameter is an optional parameter and it contains the boolean values.It uses the density keyword argument instead.
Returns: This returns the following:
n :This returns the values of the histogram bins.
bins :This returns the edges of the bins.
patches :This returns the list of individual patches used to create the histogram.

- Ravel() - The **ravel**() **function** is used to create a contiguous flattened array. A 1-D array, containing the elements of the input, is returned.

- calcHist() –

Syntax - cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])

images : it is the source image of type uint8 or float32 represented as "[img]".
channels : it is the index of channel for which we calculate histogram. For grayscale image, its value is [0] and
color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.
mask : mask image. To find histogram of full image, it is given as "None".
histSize : this represents our BIN count. For full scale, we pass [256].
ranges : this is our RANGE. Normally, it is [0,256].

- Arithmetic Operations on Image -

1. cv2.bitwise_and :- Bit-wise conjunction of input array elements.

Syntax: cv2.bitwise_and(source1, source2, destination, mask)
Parameters:
source1: First Input Image array(Single-channel, 8-bit or floating-point)
source2: Second Input Image array(Single-channel, 8-bit or floating-point)
dest: Output array (Similar to the dimensions and type of Input image array)
mask: Operation mask, Input / output 8-bit single-channel mask

2. cv2.bitwise_or :- Bit-wise disjunction of input array elements.

Syntax: cv2.bitwise_or(source1, source2, destination, mask)
Parameters:
source1: First Input Image array(Single-channel, 8-bit or floating-point)
source2: Second Input Image array(Single-channel, 8-bit or floating-point)
dest: Output array (Similar to the dimensions and type of Input image array)
mask: Operation mask, Input / output 8-bit single-channel mask

3. cv2.bitwise_not :- Inversion of input array elements.

Syntax: cv2.bitwise_not(source, destination, mask)
Parameters:
source: Input Image array(Single-channel, 8-bit or floating-point)
dest: Output array (Similar to the dimensions and type of Input image array)
mask: Operation mask, Input / output 8-bit single-channel mask

4. cv2.bitwise_xor :- Bit-wise exclusive-OR operation on input array elements.

Syntax: cv2.bitwise_xor(source1, source2, destination, mask)
Parameters:
source1: First Input Image array(Single-channel, 8-bit or floating-point)
source2: Second Input Image array(Single-channel, 8-bit or floating-point)
dest: Output array (Similar to the dimensions and type of Input image array)
mask: Operation mask, Input / output 8-bit single-channel mask

- cv2.blur - `cv2.blur()` method is used to blur an image using the normalized box filter.

$$K = \frac{1}{\texttt{ksize.width*ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \cdots & & & & & \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

Syntax: cv2.blur(src, ksize[, dst[, anchor[, borderType]]])
Parameters:
src: It is the image whose is to be blurred.
ksize: A tuple representing the blurring kernel size.
dst: It is the output image of the same size and type as src.
anchor: It is a variable of type integer representing anchor point and it's default value Point is (-1, -1) which means that the anchor is at the kernel center.
borderType: It depicts what kind of border to be added. It is defined by flags

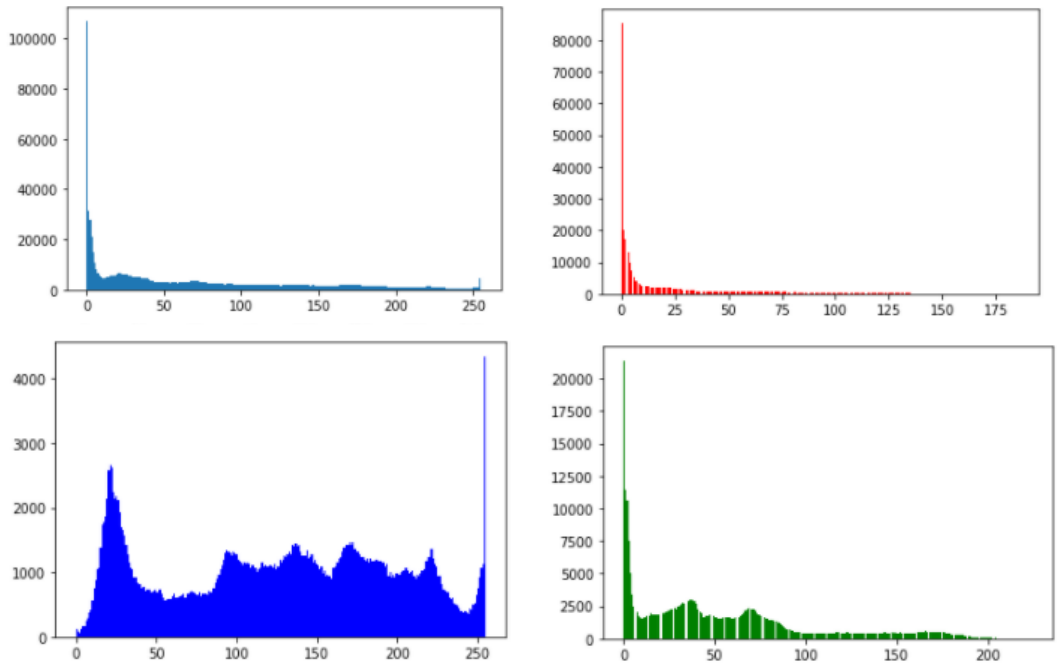
like **cv2.BORDER_CONSTANT**, **cv2.BORDER_REFLECT**, etc
Return Value: It returns an image.

Practical 3a) Finding Dominant Color in Image using histogram.

```
%matplotlib inline
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
img1 = cv2.imread("Fig0638(a)(lenna_RGB).tif")
cv2_imshow(img1)
plt.hist(img1.ravel(),bins=256)
plt.show()

red = img1[:,:,0]
plt.hist(red.ravel(),bins=256,color="red")
plt.show()

blue = img1[:,:,2]
plt.hist(blue.ravel(),bins=256,color="blue")
plt.show()

green = img1[:,:,1]
plt.hist(green.ravel(),bins=256,color="green")
plt.show()
```

Output –

```
plt.show()
```

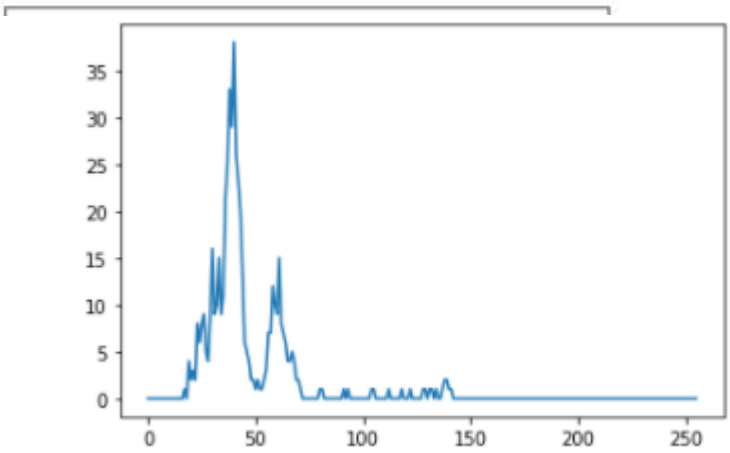

100000
80000
60000
40000
20000
0
0 50 100 150 200 250
80000
70000
60000
50000
40000
30000
20000
10000
0
0 25 50 75 100 125 150 175
4000
3000
2000
1000
0
0 50 100 150 200 250
20000
17500
15000
12500
10000
7500
5000
2500
0
0 50 100 150 200

Practical 3b) Histogram masking.

Code 1

```
%matplotlib inline
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
img1 = cv2.imread("Fig0638(a)(lenna_RGB).tif",0)
cv2_imshow(img1)
histo=cv2.calcHist(img1,[0],None,[256],[0,256])
plt.plot(histo)
plt.show()
```

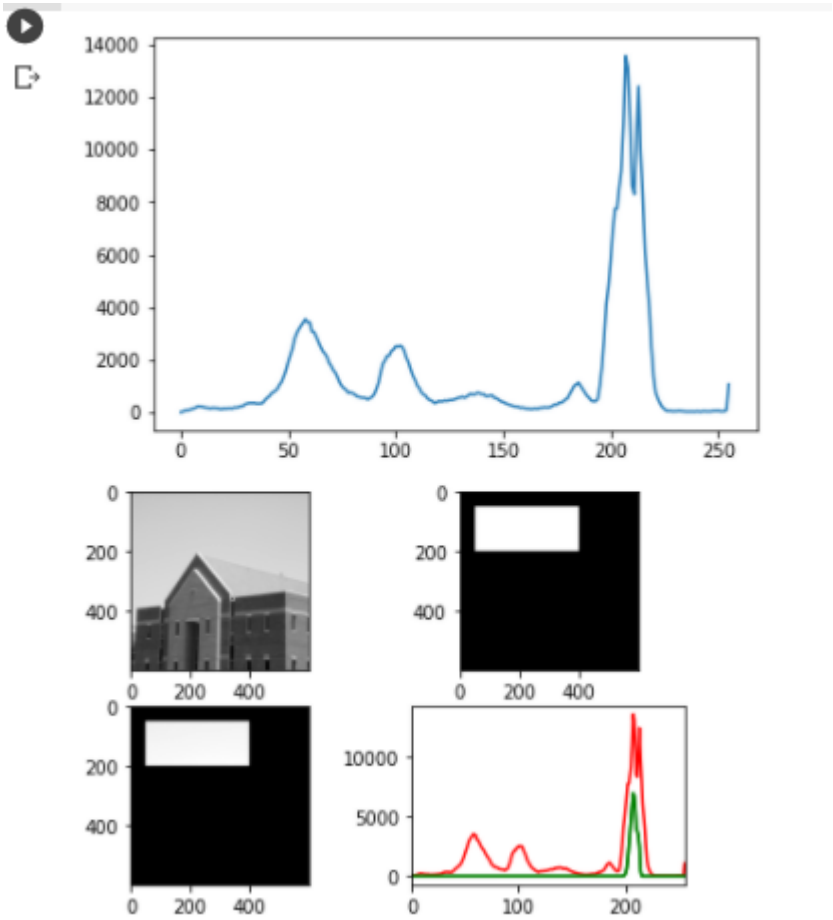
Output –



Code 2 –

```
%matplotlib inline
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
from matplotlib import pyplot as plt
img = cv2.imread("home.tif",0)
#cv2_imshow(img)

#Create a mask
mask=np.zeros(img.shape[:2],np.uint8)
mask[50:200,50:400]=255
masked_img = cv2.bitwise_and(img,img,mask=mask)

#Calculate histogram with mask and without mask
#Check third arguement for mask
hist_full = cv2.calcHist([img],[0],None,[256],[0,256])
plt.plot(hist_full)
plt.show()
hist_mask = cv2.calcHist([img],[0],mask,[256],[0,256])

plt.subplot(221), plt.imshow(img,"gray")
plt.subplot(222), plt.imshow(mask,"gray")
plt.subplot(223), plt.imshow(masked_img,"gray")
plt.subplot(224),plt.plot(hist_full,color="red"),plt.plot(hist_mask,color="green")
plt.plot(hist_mask,color="green")
plt.xlim([0,256])
plt.show()
```

Output –


14000
12000
10000
8000
6000
4000
2000
0
0 50 100 150 200 250
0 200 400
0 200 400
0 200 400
0 200 400
10000
5000
0
0 100 200

Practical 3c) Arithmetic Operations

```
%matplotlib inline
import numpy as np
import cv2
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

image1=np.zeros((400,400),dtype="uint8")
cv2.rectangle(image1,(100,100),(250,250),255,-1)
image2=np.zeros((400,400),dtype="uint8")
cv2.circle(image2,(150,150),90,255,-1)

#img1 = cv2.imread("home.tif",0)

bitand=cv2.bitwise_and(image1,image2)
#cv2_imshow(bitand)

bitor=cv2.bitwise_or(image1,image2)
#cv2_imshow(bitor)

bitnot=cv2.bitwise_not(image1)
#bitnot=cv2.bitwise_not(img1)
cv2_imshow(bitnot)

bitxor=cv2.bitwise_xor(image1,image2)
#cv2_imshow(bitxor)
```

Output –

Practical 3d) Image smoothening by Blur.

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

image = cv2.imread("elephnant.jpg")
cv2_imshow(image)
avg = cv2.blur(image,(15,15))
cv2_imshow(avg)
```

Output –

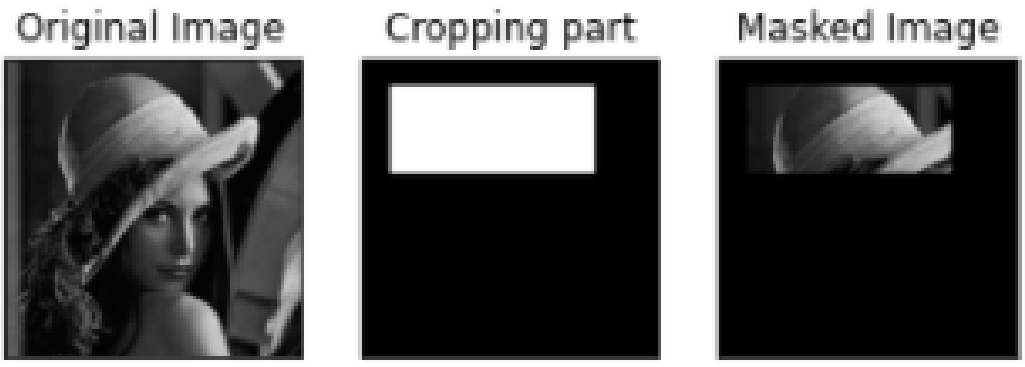
Input Image	Output Image
	

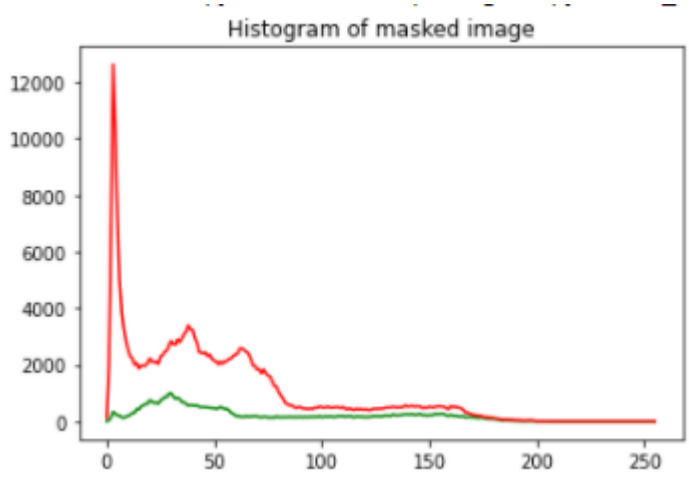
Assignment Exercise 3

1. Finding Histogram of Lena's Hat.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

image = cv2.imread("Fig0638(a)(lenna_RGB).tif",0)
#cv2_imshow(image)

#Create a mask
mask=np.zeros(image.shape[:2],np.uint8)
mask[50:200,50:400]=255
masked_img = cv2.bitwise_and(image,image,mask=mask)

#Calculate histogram with mask and without mask
#Check third arguement for mask
hist_full = cv2.calcHist([image],[0],None,[256],[0,256])
hist_mask = cv2.calcHist([image],[0],mask,[256],[0,256])

titles=['Original Image','Cropping part','Masked Image']
images = [image, mask, masked_img]

for i in range(3):
  plt.subplot(2,3,i+1),plt.imshow(images[i],"gray")
  plt.title(titles[i])
  plt.xticks([]),plt.yticks([])
plt.show()

plt.title("Histogram of masked image")
plt.subplot(1,1,1), plt.plot(hist_mask,color="green"), plt.plot(hist_full,color="red")
plt.show()
```

Output –




Histogram of masked image
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250

2. Masking & Thresholding on Hubble image.

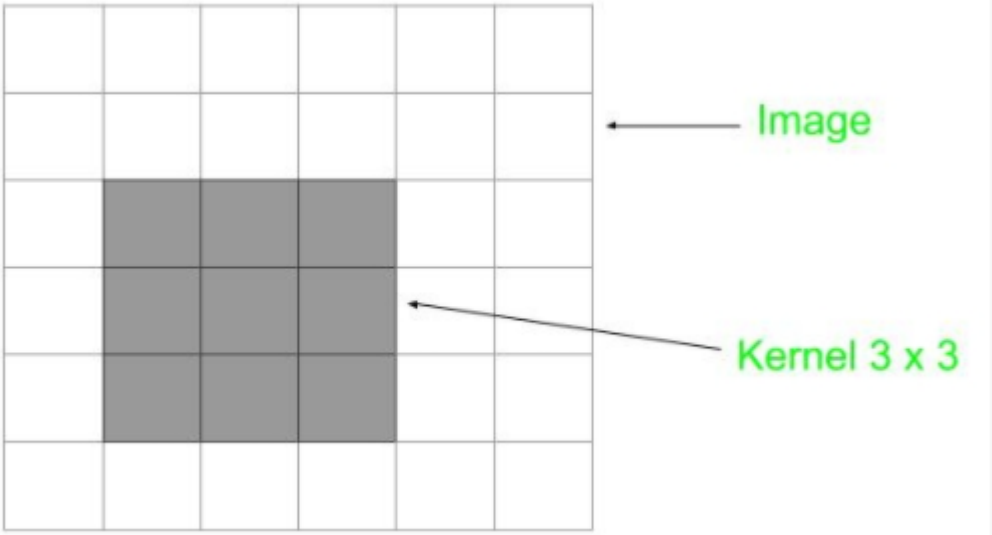
```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

image = cv2.imread("Fig0334(a)(hubble-original).tif")
#cv2_imshow(image)

avg = cv2.blur(image,(15,15))
#cv2_imshow(avg)

ret,thresh1=cv2.threshold(avg,65,255,cv2.THRESH_BINARY)

titles=['Original Image','15x15 Masked Image','Threshold Image']
images = [image, avg, thresh1]

for i in range(3):
  plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
  plt.title(titles[i])
  plt.xticks([]),plt.yticks([])
plt.show()
```

Output –

Practical 4

- Mean filter – cv2.filter2D()

Image is denoted as matrix inside computer. An image contains a lot of features like edge, contrast etc. In image processing features have to be extracted from the image for further study of image.
Convolution is a fundamental operation on images in which a mathematical operation is applied to each pixel to get the desired result.
For this purpose, another matrix called as kernel is used which is smaller in size of image. This is also called filter. This filter is applied on each pixel of the image and new value obtained is the value of that pixel. The image obtained is called filtered image.
In kernel each cell contain some value, that kernel is kept above the pixel and corresponding values are multiplied and then summed up this value obtained is new the value of pixel.



If a blurred image is observed carefully then a common thing to notice is that image is smooth meaning edges are not observed. A filter used for blurring is also called low pass filter, because it allows low frequency to enter and stop high frequency. Here frequency means the change of pixel value. Around edge pixel value changes rapidly as blur image is smooth so high frequency should be filtered out.
For blur purpose a filter with every call having value 1 is used because to blur image a pixel value should be close to neighbor value.
In filter it is divided by 9 for normalization otherwise value of a pixel will increase resulting in more contrast which is not the goal.

$$\text{Blur} = 1/9 \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

We can choose the size of the kernel depending on how much we want to smoothen the image. Choosing a bigger size will be averaging over a larger area. This tends to increase the smoothening effect.

- Median Filter – cv2.medianBlur()

The Median Filter is a non-linear digital filtering technique, often used to remove noise from an image or signal. Median filtering is very widely used in digital image processing because, under certain conditions, it preserves edges while removing noise. It is one of the best algorithms to remove Salt and pepper noise.

- Gaussian Blur – cv2.GaussianBlur()

Gaussian blur is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. It is also used as a preprocessing stage before applying our machine learning or deep learning models.
E.g. of a Gaussian kernel(3×3)

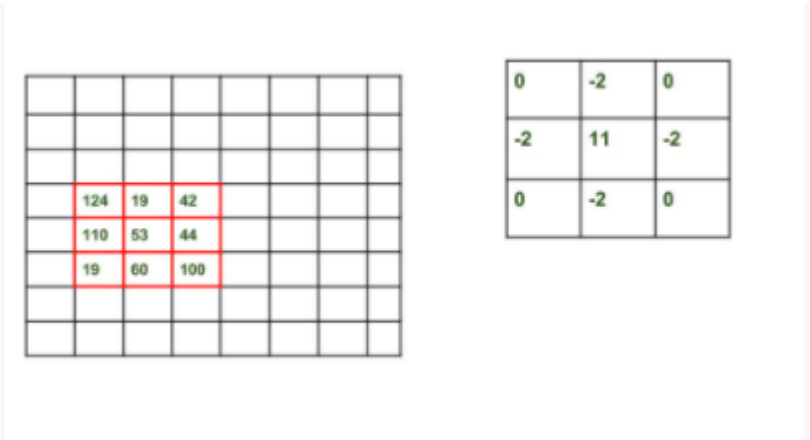
$$1/16 \quad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Laplacian Filter – cv2.Laplacian()

Syntax - cv2.Laplacian(frame,cv2.CV_64F)
the first parameter is the original image and the second parameter is the depth of the destination image.When depth=-1/CV_64F, the destination image will have the same depth as the source.

- Sobel Filter – cv2.Sobel()

A digital image is represented by a matrix that stores the RGB/BGR/HSV(whichever color space the image belongs to) value of each pixel in rows and columns.
The derivative of a matrix is calculated by an operator called the Laplacian. In order to calculate a Laplacian, you will need to calculate first two derivatives, called derivatives of Sobel, each of which takes into account the gradient variations in a certain direction: one horizontal, the other vertical.
Horizontal Sobel derivative (Sobel x): It is obtained through the convolution of the image with a matrix called kernel which has always odd size. The kernel with size 3 is the simplest case.
Vertical Sobel derivative (Sobel y): It is obtained through the convolution of the image with a matrix called kernel which has always odd size. The kernel with size 3 is the simplest case.
Convolution is calculated by the following method: Image represents the original image matrix and filter is the kernel matrix.



Factor = 11 – 2- 2- 2- 2 = 3
Offset = 0
Weighted Sum = 124*0 + 19*(-2) + 110*(-2) + 53*11 + 44*(-2) + 19*0 + 60*(-2) + 100*0 = 117
O[4,2] = (117/3) + 0 = 39
So in the end to get the Laplacian (approximation) we will need to combine the two previous results (Sobelx and Sobely) and store it in laplacian.

Parameters:
cv2.Sobel(): The function cv2.Sobel(frame,cv2.CV_64F,1,0,ksize=5) can be written as cv2.Sobel(original_image,ddepth,xorder,yorder,kernelsize)
where the first parameter is the original image, the second parameter is the depth of the destination image. When ddepth=-1/CV_64F, the destination image will have the same depth as the source. The third parameter is the order of the derivative x. The fourth parameter is the order of the derivative y. While calculating Sobelx we will set xorder as 1 and yorder as 0 whereas while calculating Sobely, the case will be reversed. The last parameter is the size of the extended Sobel kernel; it must be 1, 3, 5, or 7.

Practical 4a) – Mean, Median & Gaussian Filter (Code – 1)

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
image=cv2.imread('Fig0809(a).tif')

plt.subplot(141)
plt.title('Original Image')
plt.imshow(image)
plt.xticks([]),plt.yticks([])

kernel_6x6=np.ones((6,6),np.float32)/36
blurred=cv2.filter2D(image,-1,kernel_6x6)
plt.subplot(142)
plt.title('Mean')
plt.imshow(blurred)
plt.xticks([]),plt.yticks([])

medi=cv2.medianBlur(image,5)
plt.subplot(143)
plt.title('Median')
plt.imshow(medi)
plt.xticks([]),plt.yticks([])

gauss=cv2.GaussianBlur(image,(5,5),7)
plt.subplot(144)
plt.title('Gauss')
plt.imshow(gauss)
plt.xticks([]),plt.yticks([])

plt.show()
```

Output –

Practical 4a) – Mean, Median & Gaussian Filter (Code – 2)

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
image = cv2.imread("elephnant.jpg")
cv2_imshow(image)
kernel_3x3 = np.ones((3,3),np.float32)/9
blurred = cv2.filter2D(image,-1,kernel_3x3)
cv2_imshow(blurred)
```

Output –

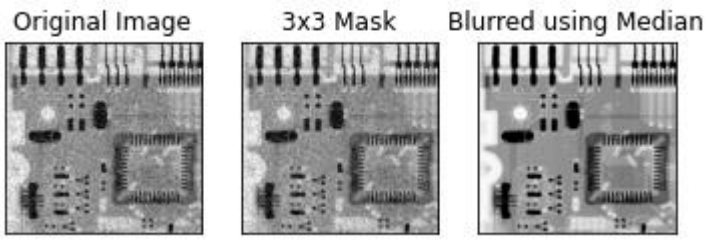
Output –

Input Image	Output Image
	

Practical 4a) – Mean, Median & Gaussian Filter (Code – 3)

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
image=cv2.imread('Fig0335(a)(ckt_board_saltpep_prob_pt05).tif')
plt.subplot(131)
plt.title('Original Image')
plt.xticks([]),plt.yticks([])
plt.imshow(image)
kernel_3x3=np.ones((3,3),np.float32)/9
blurred=cv2.filter2D(image,-1,kernel_3x3)
plt.subplot(132)
plt.title('3x3 Mask')
plt.xticks([]),plt.yticks([])
plt.imshow(blurred)
medi=cv2.medianBlur(image,3)
plt.subplot(133)
plt.title('Blurred using Median')
plt.imshow(medi)
plt.xticks([]),plt.yticks([])
plt.show()
```

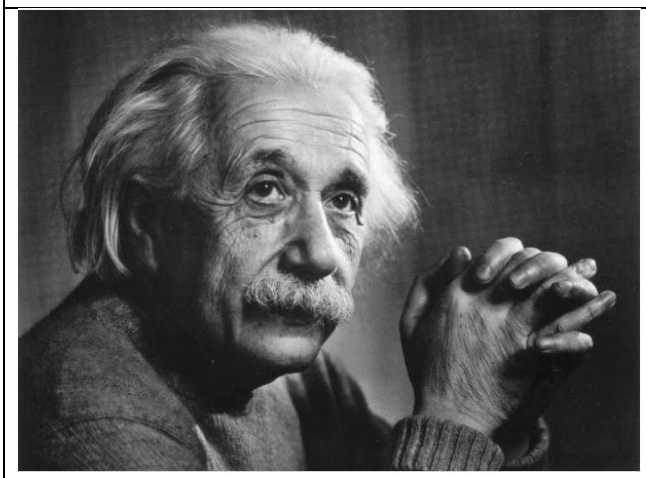
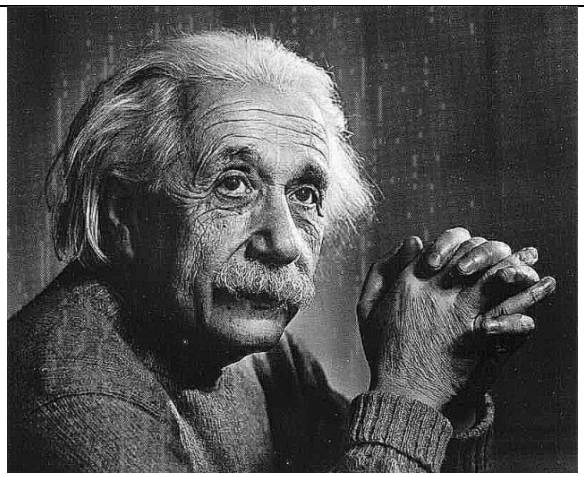
Output –



Practical 4b) - Sharpening Filter

```
#SHARPENING FILTER
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
#Reading in and displaing our image
image = cv2.imread("Fig0241(c)(einstein high contrast).tif")
cv2_imshow(image)
#Create our sharpening kernel, it must equal to one eventually
kernel_sharpening = np.array([[-1, -1, -1],
                              [-1, 9, -1],
                              [-1, -1, -1]])
#Applying te sharpening kernel to the input image & displaying it
sharpened = cv2.filter2D(image,-1, kernel_sharpening)
cv2_imshow(sharpened)
```

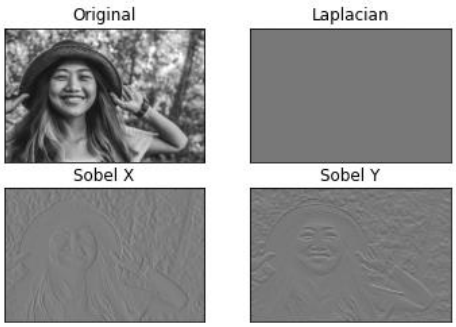
Output –

Input Image	Output Image
	

Practical 4c) – Laplacian and Sobel Filter. (Code 1)

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
img=cv2.imread('what-makes-people-happy.jpeg',0)

laplacian=cv2.Laplacian(img,cv2.CV_64F)
sobelx=cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely=cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

plt.subplot(2,2,1),plt.imshow(img,cmap='gray')
plt.title('Original'),plt.xticks([]),plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap='gray')
plt.title('Laplacian'),plt.xticks([]),plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap='gray')
plt.title('Sobel X'),plt.xticks([]),plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap='gray')
plt.title('Sobel Y'),plt.xticks([]),plt.yticks([])

plt.show()
```

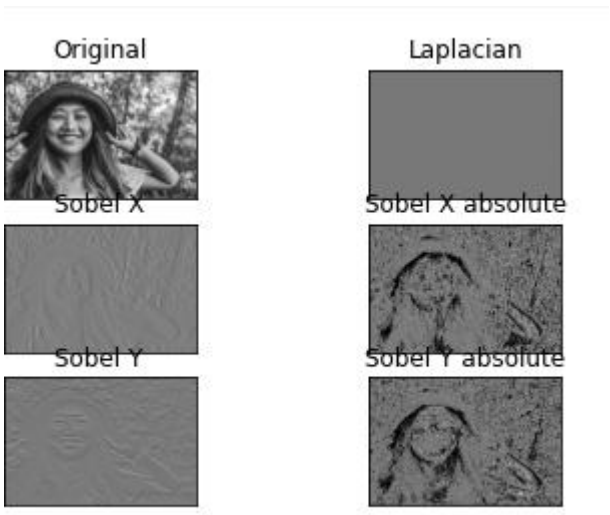
Output –



Practical 4c) – Laplacian and Sobel Filter. (Code 2)

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
img=cv2.imread('what-makes-people-happy.jpeg',0)
laplacian=cv2.Laplacian(img,cv2.CV_64F)
sobelx=cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
sobely=cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)

abs_sobelx64f=np.absolute(sobelx)
sobelx_8u=np.uint8(abs_sobelx64f)
abs_sobely64f=np.absolute(sobely)
sobely_8u=np.uint8(abs_sobely64f)

plt.subplot(3,2,1),plt.imshow(img,cmap='gray')
plt.title('Original'),plt.xticks([]),plt.yticks([])

plt.subplot(3,2,2),plt.imshow(laplacian,cmap='gray')
plt.title('Laplacian'),plt.xticks([]),plt.yticks([])

plt.subplot(3,2,3),plt.imshow(sobelx,cmap='gray')
plt.title('Sobel X'),plt.xticks([]),plt.yticks([])

plt.subplot(3,2,4),plt.imshow(sobelx_8u,cmap='gray')
plt.title('Sobel X absolute'),plt.xticks([]),plt.yticks([])

plt.subplot(3,2,5),plt.imshow(sobely,cmap='gray')
plt.title('Sobel Y'),plt.xticks([]),plt.yticks([])

plt.subplot(3,2,6),plt.imshow(sobely_8u,cmap='gray')
plt.title('Sobel Y absolute'),plt.xticks([]),plt.yticks([])

plt.show()
```

Output –

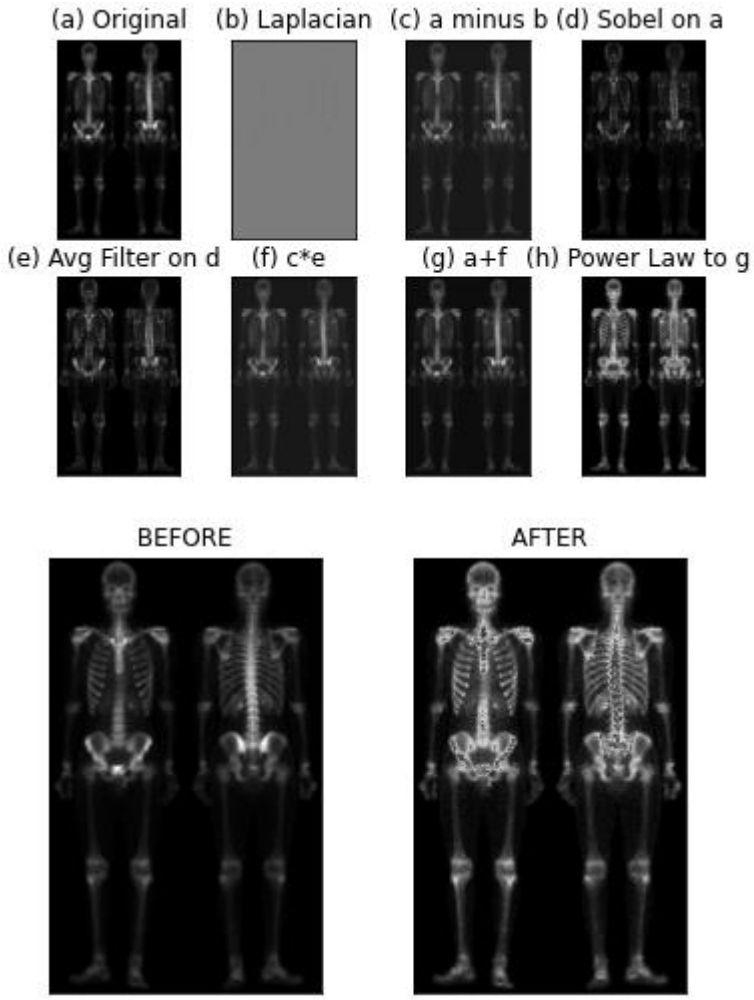


Assignment Exercise 4

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

# (a) Original Image
img_a=cv2.imread('Fig0106(a)(bone-scan-GE).tif',0)
plt.subplot(2,4,1),plt.imshow(img_a,cmap="gray")
plt.title('(a) Original'),plt.xticks([]),plt.yticks([])

# (b) Applying Laplacian filter to Original Image (a)
img_b=cv2.Laplacian(img_a,cv2.CV_64F)
plt.subplot(2,4,2),plt.imshow(img_b,cmap="gray")
plt.title('(b) Laplacian'),plt.xticks([]),plt.yticks([])

# (c) Subtracting (a) and (b)
img_c=img_a-img_b
plt.subplot(2,4,3),plt.imshow(img_c,cmap="gray")
plt.title('(c) a minus b'),plt.xticks([]),plt.yticks([])

# (d) Sobel Filter on (a)
img_d=cv2.Sobel(img_a,cv2.CV_64F,1,0,ksize=3)
abs_sobelx64f=np.absolute(img_d)
sobelx_8u=np.uint8(abs_sobelx64f)
plt.subplot(2,4,4),plt.imshow(sobelx_8u,cmap="gray")
plt.title('(d) Sobel on a '),plt.xticks([]),plt.yticks([])

# (e) Smoothed with 5x5 averaging filter.
#img_e = cv2.blur(sobelx_8u,(5,5))
kernel = np.ones((5,5),np.float32)/25
img_e = cv2.filter2D(sobelx_8u,-1,kernel)
plt.subplot(2,4,5),plt.imshow(img_e,cmap="gray")
plt.title('(e) Avg Filter on d '),plt.xticks([]),plt.yticks([])

# (f) The product of (c) and (e) which is used as mask.
#img_f= img_c*img_e
img_f=cv2.bitwise_and(img_c,img_c,mask=img_e)
#img_f=cv2.bitwise_and(img_c,img_e)
plt.subplot(2,4,6),plt.imshow(img_f,cmap="gray")
plt.title('(f) c*e '),plt.xticks([]),plt.yticks([])

# (g) Sharpened Image which Summation of (a) and (f)
img_g = img_a+img_f
plt.subplot(2,4,7),plt.imshow(img_g,cmap="gray")
plt.title('(g) a+f '),plt.xticks([]),plt.yticks([])

# (h) Applying Power Law Transformation to (g)
img_h=np.array(255*(img_g/255)**0.99, dtype="uint8")
plt.subplot(2,4,8),plt.imshow(img_h,cmap="gray")
plt.title('(h) Power Law to g '),plt.xticks([]),plt.yticks([])

plt.show()
```

```
# Image (a) and (h) to show BEFORE AND AFTER applying spatial filters.
plt.subplot(1,2,1), plt.xticks([]),plt.yticks([]), plt.title(' BEFORE '),
plt.imshow(img_a,cmap="gray")
plt.subplot(1,2,2), plt.xticks([]),plt.yticks([]), plt.title(' AFTER '),
plt.imshow(img_h,cmap="gray")
plt.show()
```

Output –



Practical 5

- Numpy.histogram() - Numpy has a built-in `numpy.histogram()` function which represents the frequency of data distribution in the graphical form. The rectangles having equal horizontal size corresponds to class interval called bin and variable height corresponding to the frequency.

Syntax - numpy.histogram(data, bins=10, range=None, normed=None, weights=None, density=None)

Attributes of the above function are listed below:

ATTRIBUTE	PARAMETER
DATA	array or sequence of array to be plotted
BINS	int or sequence of str defines number of equal width bins in a range, default is 10
RANGE	optional parameter sets lower and upper range of bins
NORMED	optional parameter same as density attribute, gives incorrect result for unequal bin width
WEIGHTS	optional parameter defines array of weights having same dimensions as data
DENSITY	optional parameter if False result contain number of sample in each bin, if True result contain probability density function at bin

- Numpy.cumsum() – numpy.cumsum() function is used when we want to compute the cumulative sum of array elements over a given axis.

Syntax : numpy.cumsum(arr, axis=None, dtype=None, out=None)

Parameters :
arr : [array_like] Array containing numbers whose cumulative sum is desired. If arr is not an array, a conversion is attempted.
axis : Axis along which the cumulative sum is computed. The default is to compute the sum of the flattened array.
dtype : Type of the returned array, as well as of the accumulator in which the elements are multiplied. If dtype is not specified, it defaults to the dtype of arr, unless arr has an integer dtype with a precision less than that of the default platform integer. In that case, the default platform integer is used instead.
out : [ndarray, optional] A location into which the result is stored.
-> If provided, it must have a shape that the inputs broadcast to.
-> If not provided or None, a freshly-allocated array is returned.

Return : A new array holding the result is returned unless out is specified, in which case it is returned.

- Cv2.cvtColor() - cv2.cvtColor() method is used to convert an image from one color space to another. There are more than 150 color-space conversion methods available in OpenCV.

Syntax: cv2.cvtColor(src, code[, dst[, dstCn]])
Parameters:
src: It is the image whose color space is to be changed.
code: It is the color space conversion code.
dst: It is the output image of the same size and depth as src image. It is an optional parameter.

dstCn: It is the number of channels in the destination image. If the parameter is 0 then the number of the channels is derived automatically from src and code. It is an optional parameter.
Return Value: It returns an image.

- Np.full –

numpy.full(shape, fill_value, dtype = None, order = 'C') : Return a new array with the same shape and type as a given array filled with a fill_value.
Parameters :
shape : Number of rows
order : C_contiguous or F_contiguous
dtype : [optional, float(by Default)] Data type of returned array.
fill_value : [bool, optional] Value to fill in the array.
Returns :
Ndarray

- matplotlib.colors.hsv_to_rgb() - The matplotlib.colors.hsv_to_rgb() function is used to convert hsv values to rgb.

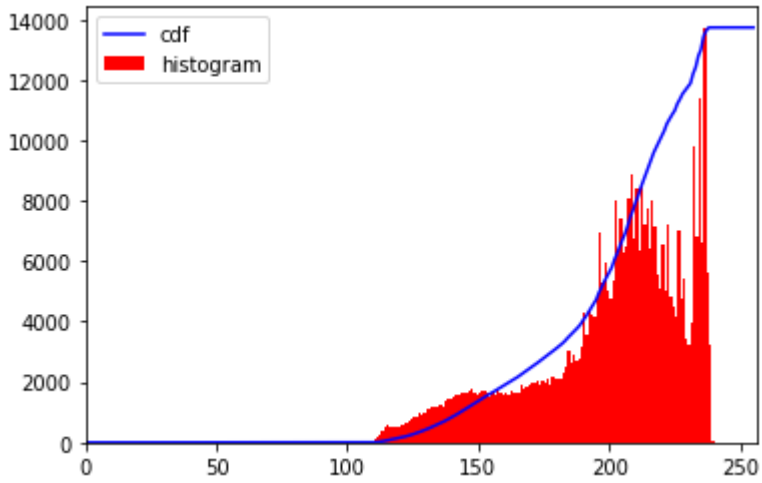
Syntax: matplotlib.colors.hsv_to_rgb(hsv)
Parameters:
hsv: It is an array-like argument in the form of (…, 3) where all values are assumed to be in the range of 0 to 1.
Returns:
rgb: It returns an ndarray in the form of (…, 3) that comprises of colors converted to RGB values within the range of 0 to 1.

- Cv2.inRange() - Here we are defining range of bluecolor in HSV.

Practical 5a) - Black and White Low Contrast.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

#Black and White Low Contrast Image
#img = cv2.imread("black-and-white-low-contrast-fog.jpg",0)

#Color Low Contrast Image
img = cv2.imread("p2219651326-5-800x533.jpg",0)
#Original Image
cv2_imshow(img)
hist,bins = np.histogram(img.flatten(),256,[0,256])
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max())/cdf.max()
plt.plot(cdf_normalized, color='b')
plt.hist(img.flatten(),256,[0,256],color='r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'),loc='upper left')
plt.show()

equ = cv2.equalizeHist(img)
cv2_imshow(equ)
```

Output –

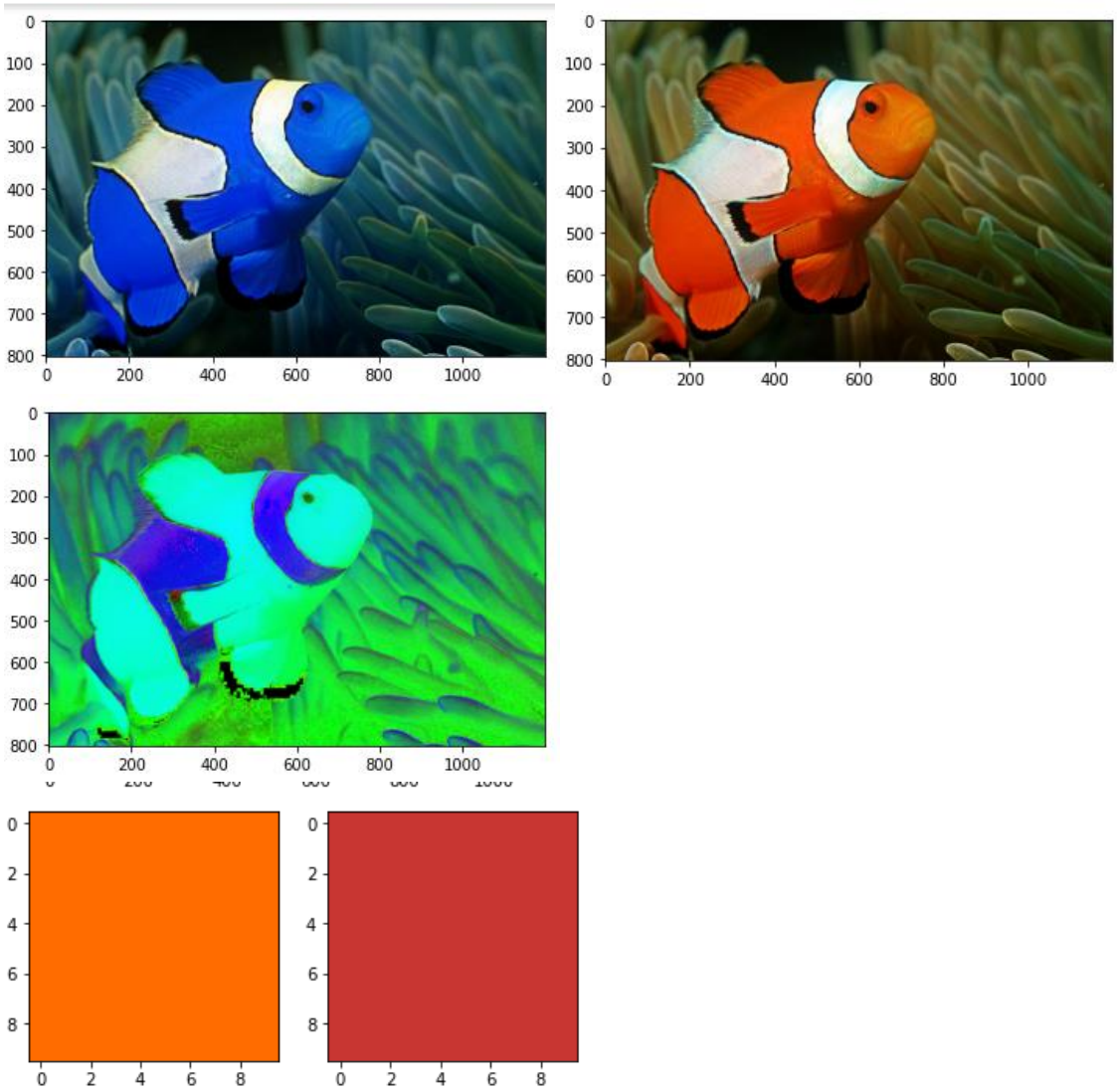

cdf
histogram
14000
12000
10000
8000
6000
4000
2000
0
0
50
100
150
200
250

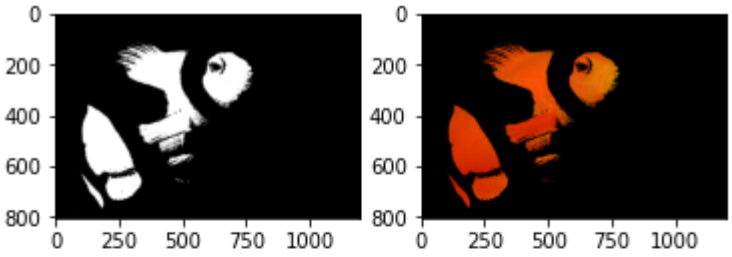
Practical 5b) - Color Segmentation on Nemo.

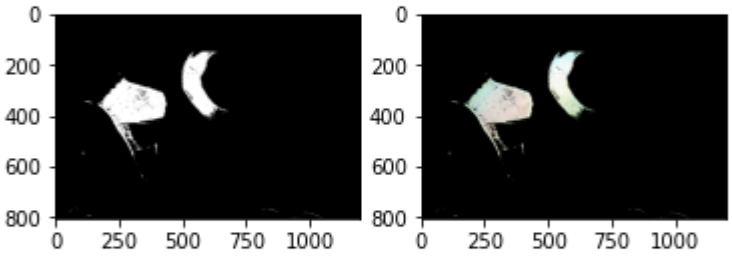
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
from matplotlib.colors import hsv_to_rgb

img = cv2.imread("1200px-Clown_fish_in_the_Andaman_Coral_Reef.jpg")
plt.imshow(img)
plt.show()

nemo = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(nemo)
plt.show()

hsv_nemo = cv2.cvtColor(nemo, cv2.COLOR_RGB2HSV)
plt.imshow(hsv_nemo)
plt.show()


light_orange = (1,190,200)
dark_orange = (18,255,255)

lo_square = np.full((10,10,3),light_orange , dtype=np.uint8)/255.0
do_square = np.full((10,10,3),dark_orange , dtype=np.uint8)/255.0

plt.subplot(1,2,1)
plt.imshow(hsv_to_rgb(do_square))

plt.subplot(1,2,2)
plt.imshow(hsv_to_rgb(lo_square))

plt.show()

mask = cv2.inRange(hsv_nemo, light_orange, dark_orange)
result = cv2.bitwise_and(nemo,nemo, mask=mask)

plt.subplot(1,2,1)
plt.imshow(mask, cmap="gray")

plt.subplot(1,2,2)
plt.imshow(result)
plt.show()

light_white = (0,0,200)
dark_white = (145,60,255)

mask_white = cv2.inRange(hsv_nemo, light_white, dark_white)
result_white = cv2.bitwise_and(nemo,nemo,mask=mask_white)

plt.subplot(1,2,1)
plt.imshow(mask_white,cmap="gray")

plt.subplot(1,2,2)
plt.imshow(result_white)
```

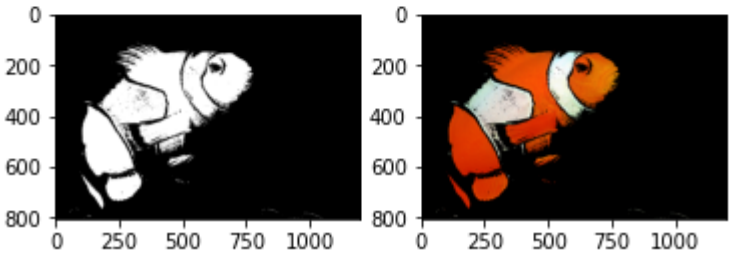
```
plt.show()

final_mask = mask + mask_white
final_result = cv2.bitwise_and(nemo,nemo,mask=final_mask)

plt.subplot(1,2,1)
plt.imshow(final_mask,cmap="gray")

plt.subplot(1,2,2)
plt.imshow(final_result)
plt.show()

kernel = np.ones((20,20),np.float32)/400
mean = cv2.filter2D(final_result,-1,kernel)
plt.imshow(mean)
plt.show()

gauss=cv2.GaussianBlur(final_result,(21,21),7)
plt.imshow(gauss)
plt.show()

medi=cv2.medianBlur(final_result,11)
plt.imshow(medi)
plt.show()
```

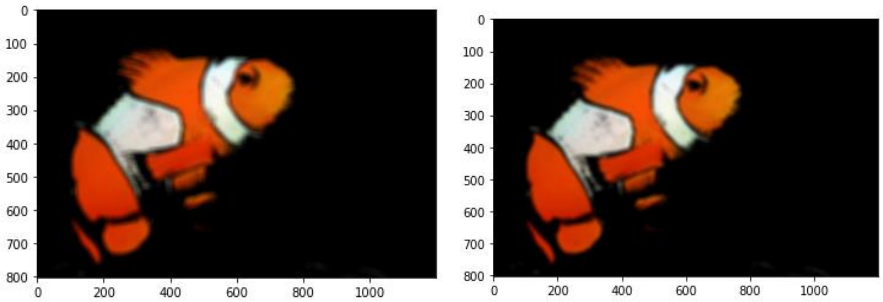
Output –

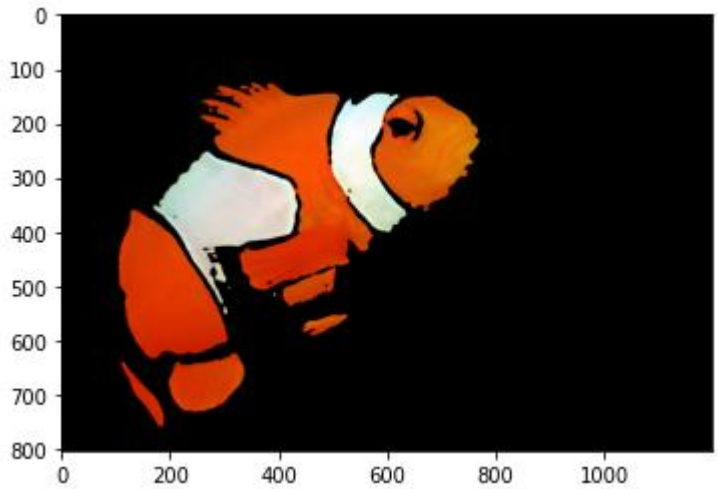



0
200
400
600
800
0 250 500 750 1000


0
200
400
600
800
0 250 500 750 1000


0
200
400
600
800
0 250 500 750 1000


0
100
200
300
400
500
600
700
800
0 200 400 600 800 1000


0
100
200
300
400
500
600
700
800
0 200 400 600 800 1000

Assignment Exercise 5

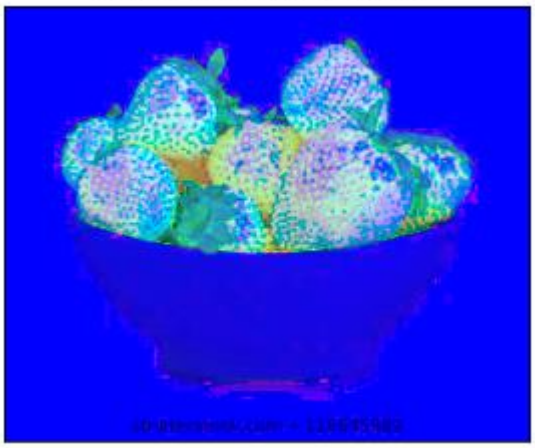
1. Extracting color of Strawberry.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
from matplotlib.colors import hsv_to_rgb

img = cv2.imread("strawberries.webp")
plt.subplot(111)
plt.title('(a) Original Image of Strawberries on bowl (OPEN CV)')
plt.xticks([]),plt.yticks([])
plt.imshow(img)
plt.show()

rbg_strawberry = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.subplot(111)
plt.title('(b) RGB Strawberries on bowl')
plt.xticks([]),plt.yticks([])
plt.imshow(rbg_strawberry)
plt.show()

hsv_strawberry = cv2.cvtColor(rbg_strawberry, cv2.COLOR_RGB2HSV)
plt.subplot(111)
plt.title('(c) HSV Strawberries on bowl')
plt.xticks([]),plt.yticks([])
plt.imshow(hsv_strawberry)
plt.show()
```

Output –

(b) RGB Strawberries on bowl

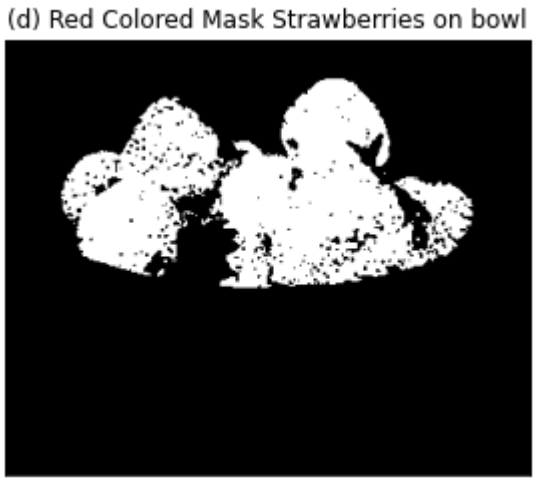
(c) HSV Strawberries on bowl

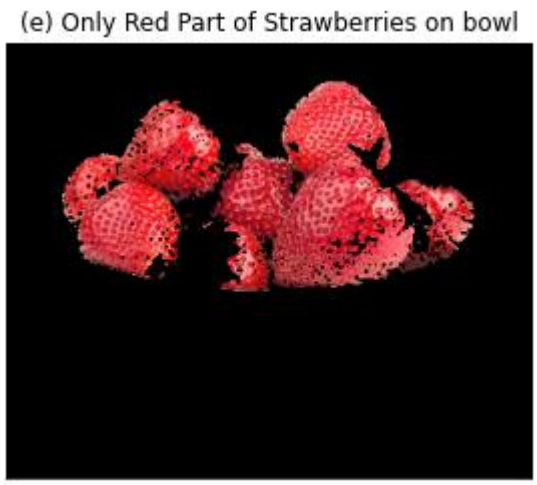


```
mask_lowerRed = cv2.inRange(hsv_strawberry, (0,50,20), (5,255,255))
mask_higherRed = cv2.inRange(hsv_strawberry, (175,50,20), (180,255,255))

mask_Red = cv2.bitwise_or(mask_lowerRed, mask_higherRed )
plt.subplot(111)
plt.title('(d) Red Colored Mask Strawberries on bowl')
plt.xticks([]),plt.yticks([])
plt.imshow(mask_Red,cmap="gray")
plt.show()
#cv2_imshow(mask)

redPart_Strawberry = cv2.bitwise_and(rbg_strawberry, rbg_strawberry, mask=mask_Red)
plt.subplot(111)
plt.title('(e) Only Red Part of Strawberries on bowl')
plt.xticks([]),plt.yticks([])
plt.imshow(redPart_Strawberry)
plt.show()
#cv2_imshow(croped)

#mask_Green= cv2.inRange(hsv_strawberry, (45,90,20), (70,255,255))
mask_Green= cv2.inRange(hsv_strawberry, (36,25,20), (70,255,255))
greenPart_Strawberry = cv2.bitwise_and(rbg_strawberry,rbg_strawberry,
mask=mask_Green)
plt.subplot(111)
plt.title('(f) Only Green Part of Strawberries on bowl')
plt.xticks([]),plt.yticks([])
plt.imshow(greenPart_Strawberry)
plt.show()
```

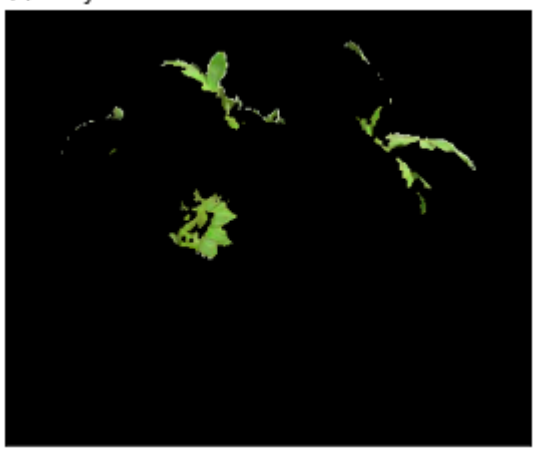
```
mask_final = cv2.bitwise_or(mask_Red, mask_Green )
result_final = cv2.bitwise_and(rbg_strawberry,rbg_strawberry, mask=mask_final)
plt.subplot(111)
plt.title('(g) Red & Green Part of Strawberries on bowl')
plt.xticks([]),plt.yticks([])
plt.imshow(result_final)
plt.show()

plt.subplot(121)
plt.title('(h) BEFORE')
plt.xticks([]),plt.yticks([])
plt.imshow(img)

plt.subplot(122)
plt.title('(i) AFTER ')
plt.xticks([]),plt.yticks([])
plt.imshow(result_final)
plt.show()
```

Output –





(f) Only Green Part of Strawberries on bowl



(g) Red & Green Part of Strawberries on bowl

(h) BEFORE

(i) AFTER

2. Color Segmentation in Extracting Lena's Hat from Lena's Image.

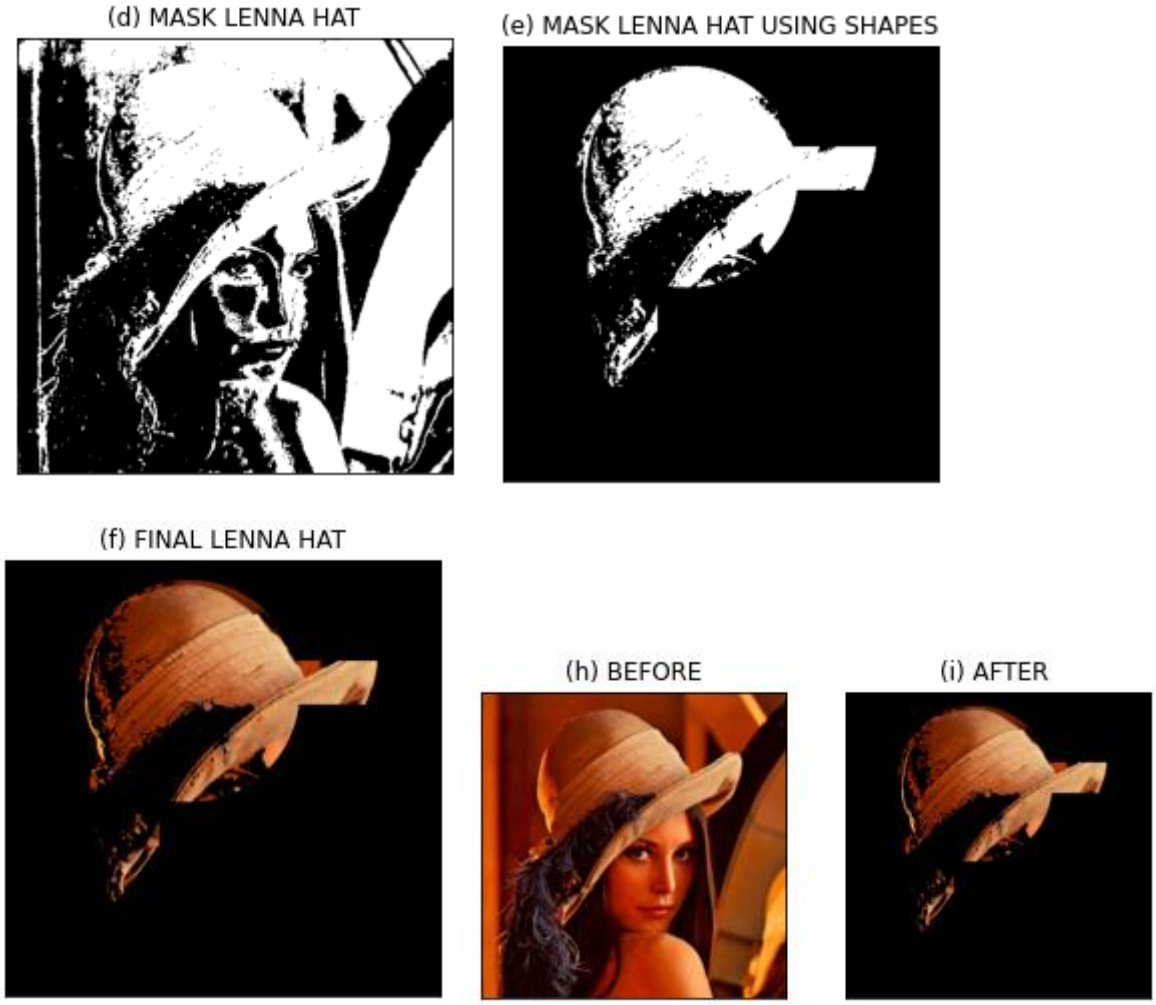
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
from matplotlib.colors import hsv_to_rgb

img = cv2.imread("lenna_RGB.tif")
plt.subplot(111)
plt.title('(a) Original Image of Lenna (OPEN CV)')
plt.xticks([]),plt.yticks([])
plt.imshow(img)
plt.show()

rbg_lenna = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.subplot(111)
plt.title('(b) RGB Lenna')
plt.xticks([]),plt.yticks([])
plt.imshow(rbg_lenna)
plt.show()

hsv_lenna = cv2.cvtColor(rbg_lenna, cv2.COLOR_RGB2HSV)
plt.subplot(111)
plt.title('(c) HSV Lenna')
plt.xticks([]),plt.yticks([])
plt.imshow(hsv_lenna)
plt.show()
```

Output –


```
mask_lowerHat = cv2.inRange(hsv_lenna, (0,50,100), (20,200,255))
mask_higherHat = cv2.inRange(hsv_lenna, (30,50,20), (45,240,255))

mask_Hat = cv2.bitwise_or(mask_lowerHat, mask_higherHat )

orangePart = cv2.inRange(hsv_lenna, (17,200,20), (30,240,255))
brownPart = cv2.inRange(hsv_lenna, (7,100,20), (20,255,200))

mask_Hat_1 = cv2.bitwise_or(mask_Hat, orangePart )
mask_Hat_2 = cv2.bitwise_or(mask_Hat_1, brownPart )

image1 = np.zeros((512, 512), dtype='uint8')
cv2.rectangle(image1, (100, 200), (180, 400), 255, -1)
cv2.rectangle(image1, (290, 120), (460, 170), 255, -1)
cv2.circle(image1, (215, 155), 130, 255, -1)
#cv2_imshow(image1)

plt.subplot(111)
plt.title('(d) MASK LENNA HAT')
plt.xticks([]),plt.yticks([])
plt.imshow(mask_Hat_2,cmap="gray")
plt.show()

plt.subplot(111)
plt.title('(e) MASK LENNA HAT USING SHAPES')
plt.xticks([]),plt.yticks([])
mask_lenna = cv2.bitwise_and(mask_Hat_2, mask_Hat_2, mask=image1)
plt.imshow(mask_lenna,cmap="gray")
```

```
plt.show()

final_lenna = cv2.bitwise_and(rbg_lenna, rbg_lenna, mask=mask_lenna)
plt.subplot(111)
plt.title('(f) FINAL LENNA HAT')
plt.xticks([]),plt.yticks([])
plt.imshow(final_lenna)
plt.show()

plt.subplot(121)
plt.title('(h) BEFORE')
plt.xticks([]),plt.yticks([])
plt.imshow(rbg_lenna)

plt.subplot(122)
plt.title('(i) AFTER ')
plt.xticks([]),plt.yticks([])
plt.imshow(final_lenna)
plt.show()
```

Output –



Practical 6

- cv2.Canny() - finds edges in the input image and marks them in the output map edges.

- Image Scaling – cv2.resize()

Image resizing refers to the scaling of images. Scaling comes in handy in many image processing as well as machine learning applications. It helps in reducing the number of pixels from an image and that has several advantages e.g. It can reduce the time of training of a neural network as more is the number of pixels in an image more is the number of input nodes that in turn increases the complexity of the model.
It also helps in zooming in images. Many times we need to resize the image i.e. either shrink it or scale up to meet the size requirements. OpenCV provides us several interpolation methods for resizing an image.

Choice of Interpolation Method for Resizing –
cv2.INTER_AREA: This is used when we need to shrink an image.
cv2.INTER_CUBIC: This is slow but more efficient.
cv2.INTER_LINEAR: This is primarily used when zooming is required. This is the default interpolation technique in OpenCV.
Note: One thing to keep in mind while using the cv2.resize() function is that the tuple passed for determining the size of the new image ((1050, 1610) in this case) follows the order (width, height) unlike as expected (height, width).

- Rotate Image –

Rotating images with OpenCV is easy, but sometimes simple rotation tasks cropped/cut sides of an image, which leads to a half image. Now, In this tutorial, We will explore a solution to safely rotate an image without cropping/cutting sides of an image so that the entire image will include in rotation, and also compare the conventional rotation method with the modified rotation version.

Step-by-step approach:

1. In-order to rotate an image without cutting off sides, we will create an explicit function named ModifedWay() which will take the image itself and the angle to which the image is to be rotated as an argument.
2. In the function, first, get the height and width of the image.
3. Locate the center of the image.
4. Then compute the 2D rotation matrix
5. Extract the absolute sin and cos values from the rotation matrix.
6. Get the new height and width of the image and update the values of the rotation matrix to ensure that there is no cropping.
7. Finally, use the wrapAffine() method to perform the actual rotation of the image.

- Image Translation – Translating an image means shifting it within a given frame of reference.

If the shift is (x, y) then matrix would be
M = [1 0 x]
[0 1 y]
Let's shift by (100, 50). Then we would be writing it as -
M = np.float32([[1, 0, 100], [0, 1, 50]])

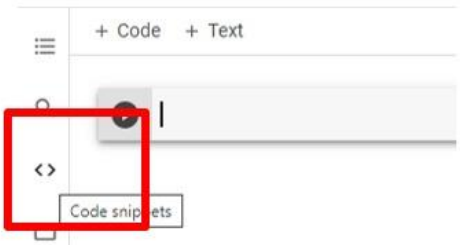
- Harris Corner Detection –

Harris Corner detection algorithm was developed to identify the internal corners of an image. The corners of an image are basically identified as the regions in which there are variations in large intensity of the gradient in all possible dimensions and directions. Corners extracted can be a part of the image features, which can be matched with features of other images, and can be used to extract accurate information. Harris Corner Detection is a method to extract the corners from the input image and to extract features from the input image.

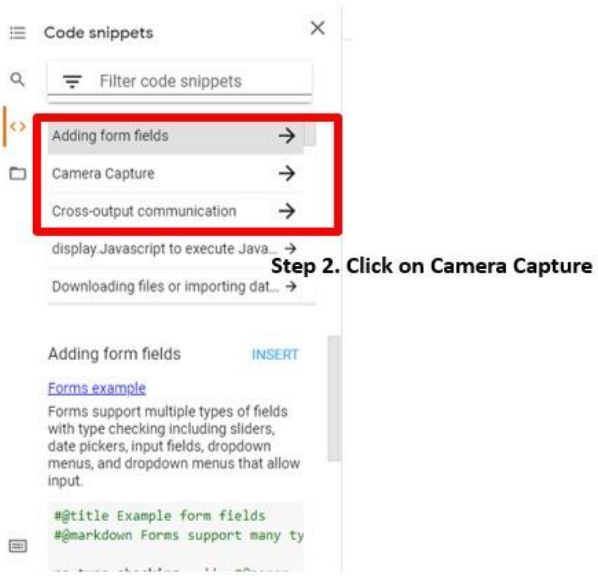
Syntax: cv2.cornerHarris(src, dest, blockSize, kSize, freeParameter, borderType)

Parameters:
src – Input Image (Single-channel, 8-bit or floating-point)
dest – Image to store the Harris detector responses. Size is same as source image
blockSize – Neighborhood size (for each pixel value blockSize * blockSize neighbourhood is considered)
ksize – Aperture parameter for the Sobel() operator
freeParameter – Harris detector free parameter
borderType – Pixel extrapolation method (the extrapolation mode used returns the coordinate of the pixel corresponding to the specified extrapolated pixel)

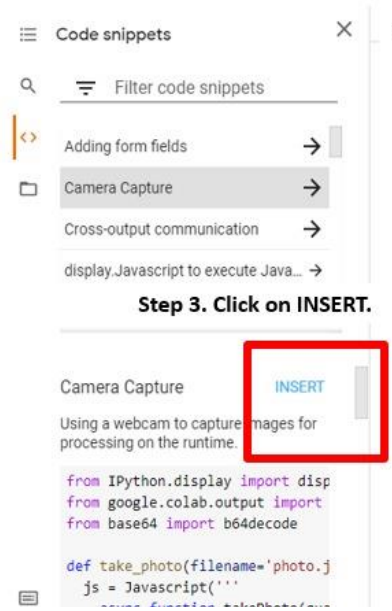
- Steps to capture live photo in Google colab –



Step 1. Click on Code Snippet



Step 2. Click on Camera Capture

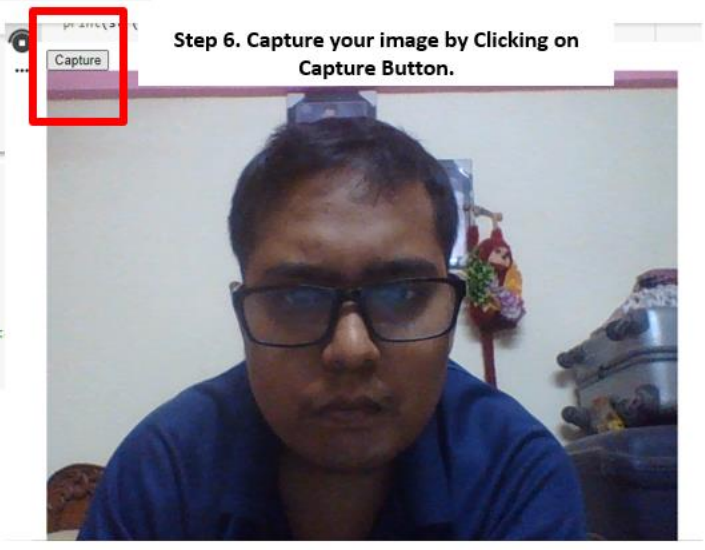


Step 3. Click on INSERT.

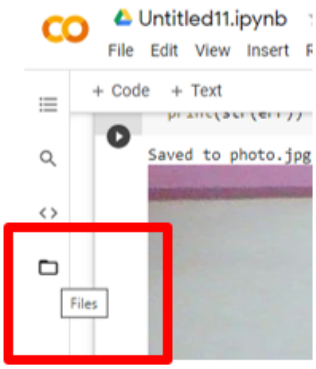
Step 4. After INSERT operation, Execute this block.

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
Run cell (Ctrl+Enter)
cell has not been executed in this session
def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.
```

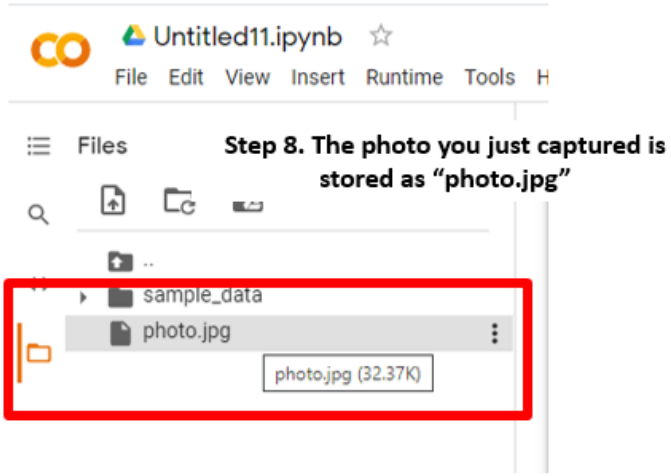
Step 5. Execute 2nd Block.



Step 6. Capture your image by Clicking on Capture Button.



Step 7. In order to check, where your photo is stored, you need to click on Files.



Step 8. The photo you just captured is stored as "photo.jpg"

- Cv2.dilate() –

1. Basics of dilation:

   Increases the object area
   Used to accentuate features

2. Working of dilation:

   A kernel(a matrix of odd size(3,5,7) is convolved with the image
   A pixel element in the original image is '1' if atleast one pixel under the kernel is '1'.
   It increases the white region in the image or size of foreground object increases.

The first parameter is the original image, kernel is the matrix with which image is convolved and third parameter is the number of iterations, which will determine how much you want to erode/dilate a given image.

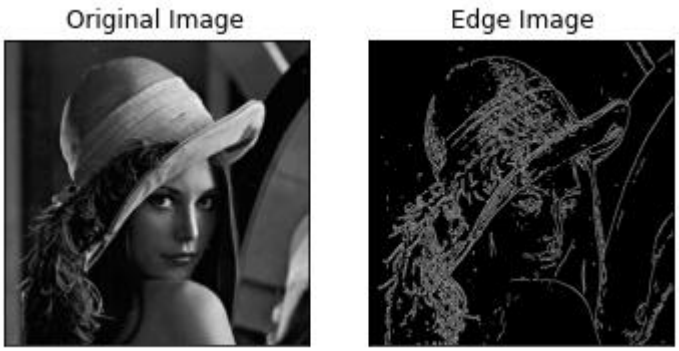
3. Uses of Dilation

   In cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases.
   It is also useful in joining broken parts of an object.

Practical 6a) - Canny Edge Detection. (Code – 1)

```
#Canny Edge detection
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

img = cv2.imread("Fig0646(a)(lenna_original_RGB).tif",0)
edges = cv2.Canny(img,70,100)

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("Edge Image"), plt.xticks([]), plt.yticks([])

plt.show()
```

Output –



Practical 6a) - Canny Edge Detection. (Code – 2)

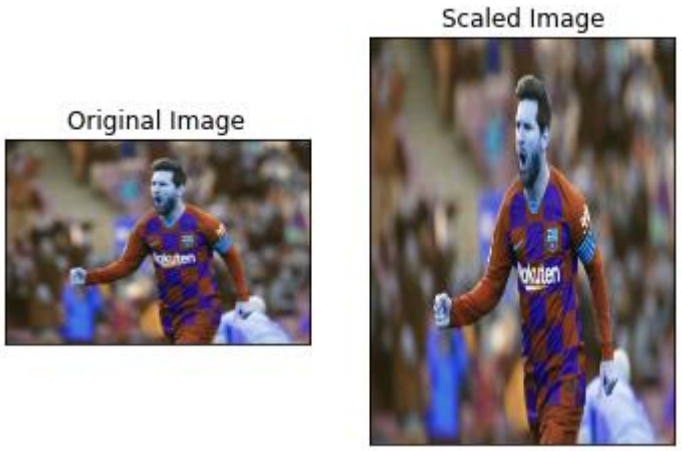
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

img = cv2.imread("m.jpg",0)
edges = cv2.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("Edge Image"), plt.xticks([]), plt.yticks([])

plt.show()
```

Output –

Practical 6b) - Image Scaling.

```
# Image scaling
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt
img = cv2.imread("m.jpg")
res = cv2.resize(img, None, fx=2, fy=4, interpolation = cv2.INTER_AREA)

plt.subplot(121), plt.imshow(img,cmap="gray")
plt.title("Original Image"), plt.xticks([]), plt.yticks([])

plt.subplot(122), plt.imshow(res,cmap="gray")
plt.title("Scaled Image"), plt.xticks([]), plt.yticks([])

plt.show()

#OR

height, width = img.shape[:2]
res= cv2.resize(img, (2*width, 2*height), interpolation =cv2.INTER_CUBIC)
cv2_imshow(res)
```

Output –



Practical 6c) - Rotate Image.

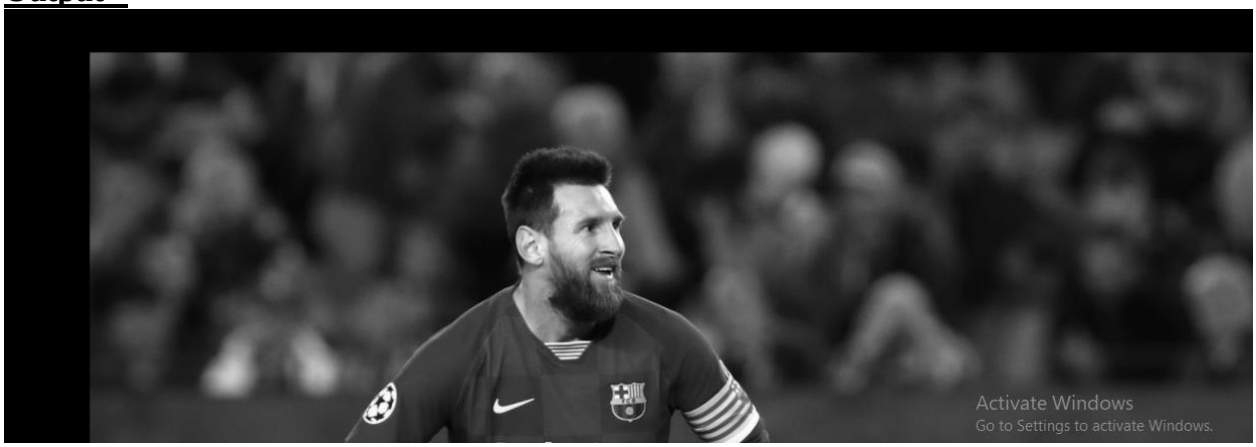
```
#Rotate Image
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt

img = cv2.imread("m.jpg",0)
rows, cols = img.shape

M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
dst = cv2.warpAffine(img,M,(cols,rows))

cv2_imshow(dst)
```

Output –

Practical 6d) - Image Translation.

```
#Image Translation
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt

img = cv2.imread("messi.jpg",0)
rows, cols = img.shape

M = np.float32([[1,0,100],[0,1,50]])
dst = cv2.warpAffine(img,M,(cols,rows))

cv2_imshow(dst)
```

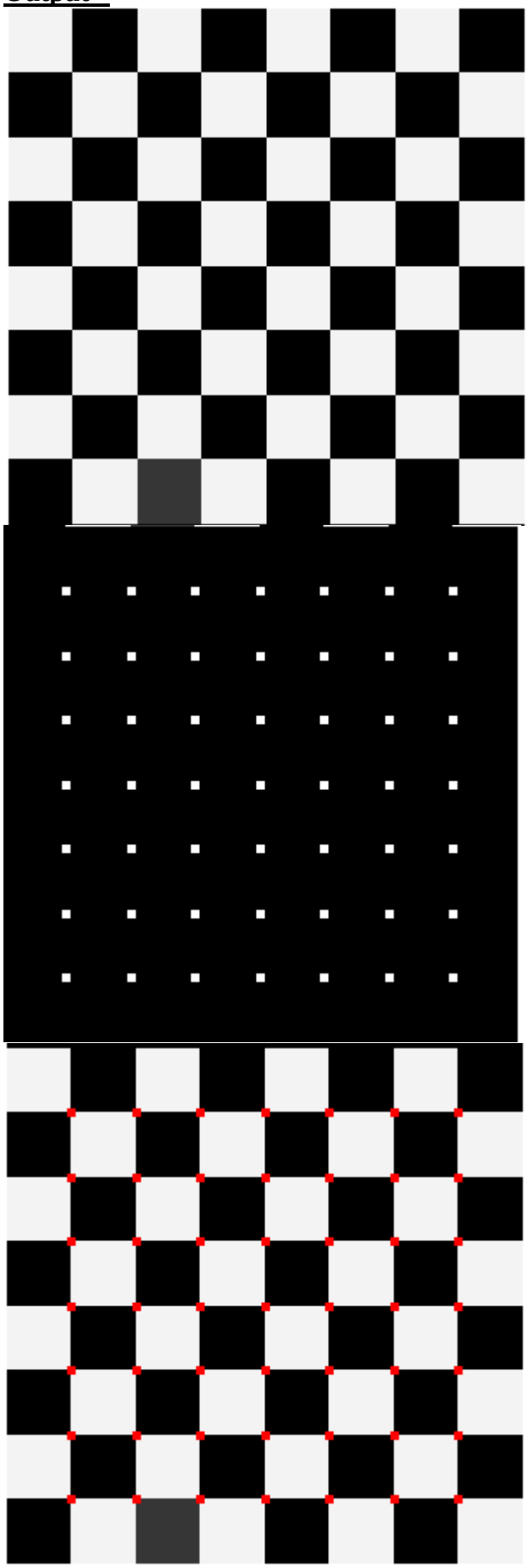
Output –



Practical 6e) - Harris Corner Detection.

```
#Harris Corner detection
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

#Original Image
img = cv2.imread("FigP0314(b).tif")
cv2_imshow(img)

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

gray = np.float32(gray)
dst = cv2.cornerHarris(gray,2,3,0.04)

#result is dilated for marking the corners, not important
dst = cv2.dilate(dst, None)

# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0, 0, 255]
#diated image
cv2_imshow(dst)
#corner detection output
cv2_imshow(img)
```

Output –



Practical 6f) - Capturing Live Photo from Webcam in Google Colab.

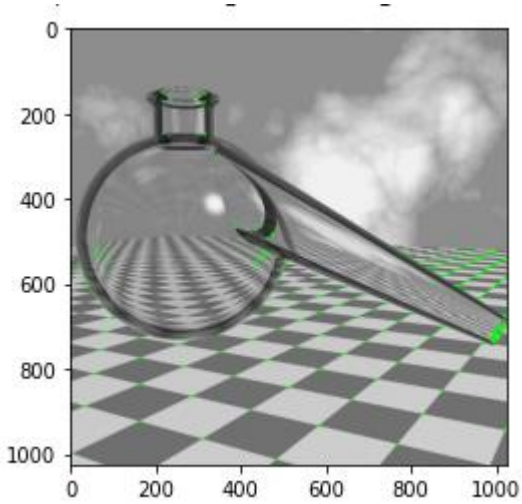
```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode

def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename

from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
  # Errors will be thrown if the user does not have a webcam or if they do not
  # grant the page permission to access it.
  print(str(err))
```

Output –

Practical 6g) - Corner Detection with Harris Corner.

```
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
import cv2
import numpy as np

image = cv2.imread('ray_traced_bottle_original.tif')

# Make a copy of the image
image_copy = np.copy(image)

# Convert image in RGB
image_copy = cv2.cvtColor(image_copy, cv2.COLOR_BGR2RGB)

plt.imshow(image_copy)

# Convert to gray scale image
gray = cv2.cvtColor(image_copy, cv2.COLOR_RGB2GRAY)
gray = np.float32(gray)

# Detect corners
dst = cv2.cornerHarris(gray, 2, 3, 0.04)

# Dilate corner image to enhance corner points
dst = cv2.dilate(dst, None)

plt.imshow(dst, cmap="gray")
# This value vary depending on the image and how many corners you want to detect
# try changing this free parameter, 0.1, to be larger or smaller and see what happens
thresh = 0.1*dst.max()

# Create an image copy to draw corners on
corner_image = np.copy(image_copy)

# Iterate through all the corners and draw them on the image (if they pass the threshold)
for j in range(0, dst.shape[0]):
  for i in range(0, dst.shape[1]):
    if(dst[j,i] > thresh):
      # image, center pt, radius, color, thickness
      cv2.circle(corner_image, (i,j), 1, (0, 255, 0), 1)

plt.imshow(corner_image)
```

Output –


0
200
400
600
800
1000
0
200
400
600
800
1000

Assignment Exercise 6

Live Image Sketch

```
from IPython.display import display, Javascript
from google.colab.output import eval_js
from base64 import b64decode

def take_photo(filename='photo.jpg', quality=0.8):
  js = Javascript('''
    async function takePhoto(quality) {
      const div = document.createElement('div');
      const capture = document.createElement('button');
      capture.textContent = 'Capture';
      div.appendChild(capture);

      const video = document.createElement('video');
      video.style.display = 'block';
      const stream = await navigator.mediaDevices.getUserMedia({video: true});

      document.body.appendChild(div);
      div.appendChild(video);
      video.srcObject = stream;
      await video.play();

      // Resize the output to fit the video element.
      google.colab.output.setIframeHeight(document.documentElement.scrollHeight, true);

      // Wait for Capture to be clicked.
      await new Promise((resolve) => capture.onclick = resolve);

      const canvas = document.createElement('canvas');
      canvas.width = video.videoWidth;
      canvas.height = video.videoHeight;
      canvas.getContext('2d').drawImage(video, 0, 0);
      stream.getVideoTracks()[0].stop();
      div.remove();
      return canvas.toDataURL('image/jpeg', quality);
    }
    ''')
  display(js)
  data = eval_js('takePhoto({})'.format(quality))
  binary = b64decode(data.split(',')[1])
  with open(filename, 'wb') as f:
    f.write(binary)
  return filename

from IPython.display import Image
try:
  filename = take_photo()
  print('Saved to {}'.format(filename))

  # Show the image which was just taken.
  display(Image(filename))
except Exception as err:
```

```
# Errors will be thrown if the user does not have a webcam or if they do not
# grant the page permission to access it.
print(str(err))
```

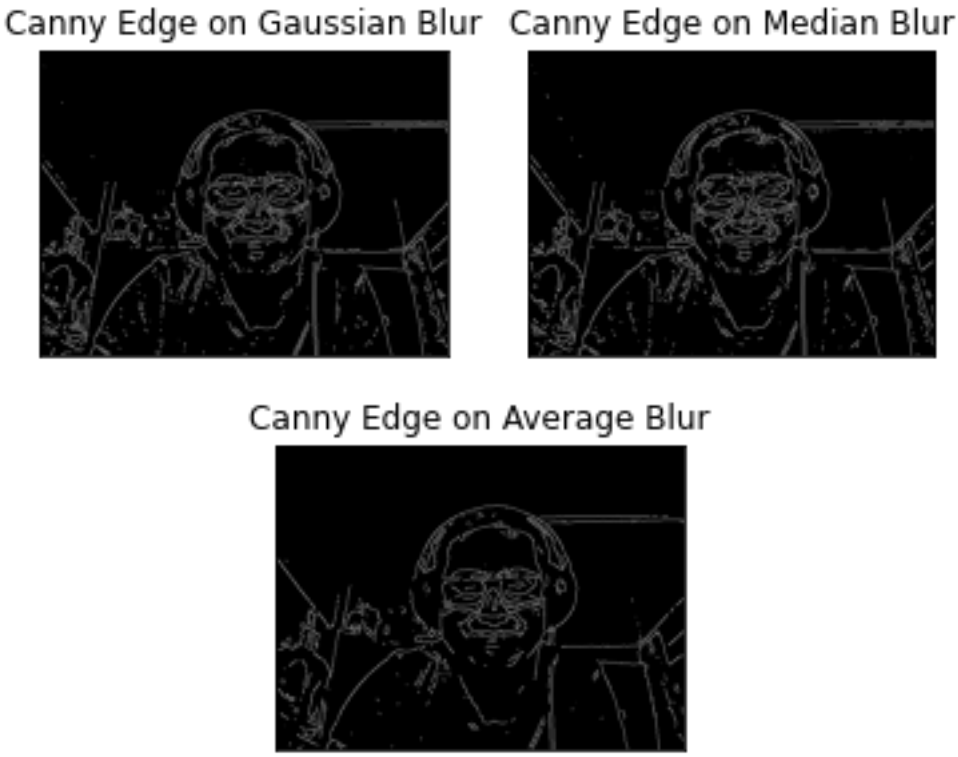
- **Outputs**

1) Original Image captured via Google Colab.

2) Captured Image (Converted to Grayscale RGB) & Gaussian Blur (5x5)

3) Median Blur & Average Blur.

4) Applying Canny Edge detection algorithm on all of the above image smoothing.



5) Big image of canny algorithm applied to Gaussian Blur.

6) Big image of canny algorithm applied to Median Blur.

7) Big image of canny algorithm applied to Average Blur.

Practical 7

- Template Matching

Template matching is a technique for finding areas of an image that are similar to a patch (template).A patch is a small image with certain features. The goal of template matching is to find the patch/template in an image.
To find it, the user has to give two input images: Source Image (S) – The image to find the template in and Template Image (T) – The image that is to be found in the source image.

It is basically a method for searching and finding the location of a template image in a larger image. The idea here is to find identical regions of an image that match a template we provide, giving a threshold

The threshold depends on the accuracy with which we want to detect the template in the source image. For instance, if we are applying face recognition and we want to detect the eyes of a person, we can provide a random image of an eye as the template and search the source (the face of a person).
In this case, since "eyes" show a large amount of variations from person to person, even if we set the threshold as 50%(0.5), the eye will be detected.
In cases where almost identical templates are to be searched, the threshold should be set high.(t>=0.8)

How Template Matching Works?

1. The template image simply slides over the input image (as in 2D convolution)
2. The template and patch of input image under the template image are compared.
3. The result obtained is compared with the threshold.
4. If the result is greater than threshold, the portion will be marked as detected.
5. In the function cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED) the first parameter is the main image, second parameter is the template to be matched and third parameter is the method used for matching.

Template Matching is a method for searching and finding the location of a template image in a larger image. OpenCV comes with a function cv.matchTemplate() for this purpose. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. Several comparison methods are implemented in OpenCV. (You can check docs for more details). It returns a grayscale image, where each pixel denotes how much does the neighborhood of that pixel match with template.

If input image is of size (WxH) and template image is of size (wxh), output image will have a size of (W-w+1, H-h+1). Once you got the result, you can use cv.minMaxLoc() function to find where is the maximum/minimum value. Take it as the top-left corner of rectangle and take (w,h) as width and height of the rectangle. That rectangle is your region of template.

- Drawing Contours

Contours are defined as the line joining all the points along the boundary of an image that are having the same intensity. Contours come handy in shape analysis, finding the size of the object of interest, and object detection.

OpenCV has findContour() function that helps in extracting the contours from the image. It works best on binary images, so we should first apply thresholding techniques, Sobel edges, etc.

We see that there are three essential arguments in cv2.findContours() function. First one is source image, second is contour retrieval mode, third is contour approximation method and it outputs the image, contours, and hierarchy. 'contours' is a Python list of all the contours in the image. Each individual contour is a Numpy array of (x, y) coordinates of boundary points of the object.

Contours Approximation Method –
Above, we see that contours are the boundaries of a shape with the same intensity. It stores the (x, y) coordinates of the boundary of a shape. But does it store all the coordinates? That is specified by this contour approximation method.
If we pass cv2.CHAIN_APPROX_NONE, all the boundary points are stored. But actually, do we need all the points? For eg, if we have to find the contour of a straight line. We need just two endpoints of that line. This is what cv2.CHAIN_APPROX_SIMPLE does. It removes all redundant points and compresses the contour, thereby saving memory.

- cv2.drawContours Method -

Draw Contours on different shapes
Draws the contours outlines or filled color .
It can also be used to draw any shape provided you have its boundary points.
1st Argument - source image,
2nd Argument - the contours which should be passed as a Python list,
3rd Argument - index of contours (useful when drawing individual contour. To draw all contours, pass -1),
4th Argument - Color,
5th Argument - Thickness.

- Hough Circles

The Hough Circles function in OpenCV has the following parameters which can be altered according to the image.

Detection Method: OpenCV has an advanced implementation, HOUGH_GRADIENT, which uses gradient of the edges instead of filling up the entire 3D accumulator matrix, thereby speeding up the process.

dp: This is the ratio of the resolution of original image to the accumulator matrix.
minDist: This parameter controls the minimum distance between detected circles.
Param1: Canny edge detection requires two parameters — minVal and maxVal. Param1 is the higher threshold of the two. The second one is set as Param1/2.
Param2: This is the accumulator threshold for the candidate detected circles. By increasing this threshold value, we can ensure that only the best circles, corresponding to larger accumulator values, are returned.
minRadius: Minimum circle radius.
maxRadius: Maximum circle radius.

- cv2.approxPolyDP method -

This function calculates and approximates a polygonal curve with specified precision. approximates a contour shape to another shape with less number of vertices depending upon the precision we specify.

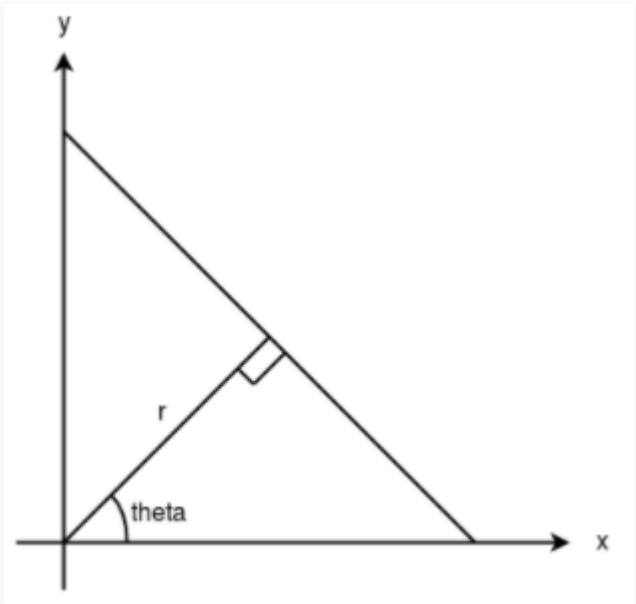
- cv2.boundingRect method –
  It gives the boundary points of the rectangle.
  It returns x and y coordinate and width, height of each shape.

- Cv2.putText method –

1st Argument - Source Image,
2nd Argument - Text on Image,
3rd Argument - Coordinates of text,
4th Argument - Font Face,
5th Argument - Font Scale,
6th Argument - Font Color,
7th Argument - Font Thickness,
8th Argument - Line Type
Example –
cv2.putText(image, text, coordinates , cv2.FONT_HERSHEY_SIMPLEX , 0.5, (0, 0, 0) , 1, cv2.LINE_AA)

- Hough Lines

Basics of Houghline Method
A line can be represented as $y = mx + c$ or in parametric form, as $r = x\cos\theta + y\sin\theta$ where r is the perpendicular distance from origin to the line, and θ is the angle formed by this perpendicular line and horizontal axis measured in counter-clockwise (That direction varies on how you represent the coordinate system. This representation is used in OpenCV).



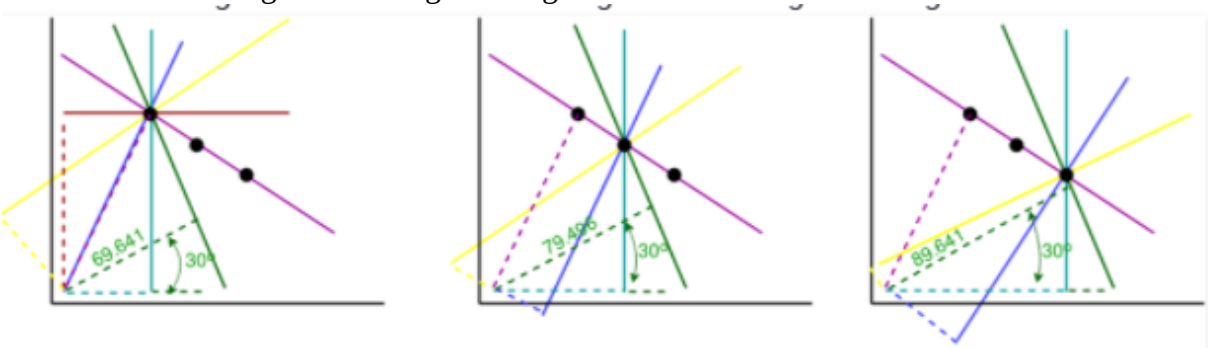
So Any line can be represented in these two terms, (r, θ).

Working of Houghline method:

- First it creates a 2D array or accumulator (to hold values of two parameters) and it is set to zero initially.
- Let rows denote the r and columns denote the (θ)theta.
- Size of array depends on the accuracy you need. Suppose you want the accuracy of angles to be 1 degree, you need 180 columns(Maximum degree for a straight line is 180).
- For r, the maximum distance possible is the diagonal length of the image. So taking one pixel accuracy, number of rows can be diagonal length of the image.

Example:
Consider a 100×100 image with a horizontal line at the middle. Take the first point of the line. You know its (x,y) values. Now in the line equation, put the values θ(theta) = 0,1,2,….,180 and check the r you get. For every (r, 0) pair, you increment value by one in the accumulator in its corresponding (r,0) cells. So now in accumulator, the cell (50,90) = 1 along with some other cells.

Now take the second point on the line. Do the same as above. Increment the values in the cells corresponding to (r,0) you got. This time, the cell (50,90) = 2. We are actually voting the (r,0) values. You continue this process for every point on the line. At each point, the cell (50,90) will be incremented or voted up, while other cells may or may not be voted up. This way, at the end, the cell (50,90) will have maximum votes. So if you search the accumulator for maximum votes, you get the value (50,90) which says, there is a line in this image at distance 50 from origin and at angle 90 degrees.



Angle	Dist.
0	40
30	69.6
60	81.2
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
120	23.4
150	-19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
120	6.0
150	-39.6

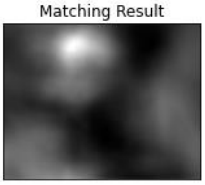
Everything explained above is encapsulated in the OpenCV function, cv2.HoughLines(). It simply returns an array of (r, 0) values. r is measured in pixels and 0 is measured in radians.

Practical 7a) – Template Matching

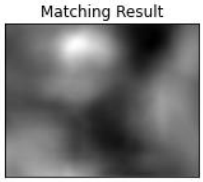
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow


img = cv2.imread("messi_2.PNG",0)
img2= img.copy()
template = cv2.imread("messi_face.PNG",0)

cv2_imshow(template)

w, h = template.shape[::-1]

#All the 6 method for comparision in a list
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']


for meth in methods :
        img = img2.copy()
        method = eval(meth)

        #Apply template matching
        res = cv2.matchTemplate(img, template, method)
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

        #If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
        if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
                top_left = min_loc
        else :
                top_left = max_loc
        bottom_right = (top_left[0] + w, top_left[1] + h)

        cv2.rectangle(img, top_left, bottom_right, 255, 2)

        plt.subplot(121), plt.imshow(res, cmap="gray")
        plt.title("Matching Result"), plt.xticks([]), plt.yticks([])

        plt.subplot(122), plt.imshow(img, cmap="gray")
        plt.title("Detected Point"), plt.xticks([]), plt.yticks([])
        plt.suptitle(meth)
        plt.show()
```

Output –

cv2.TM_CCOEFF

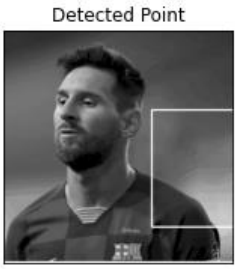


cv2.TM_CCOEFF_NORMED



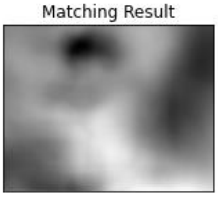
cv2.TM_CCORR

cv2.TM_CCORR_NORMED



cv2.TM_SQDIFF

cv2.TM_SQDIFF_NORMED

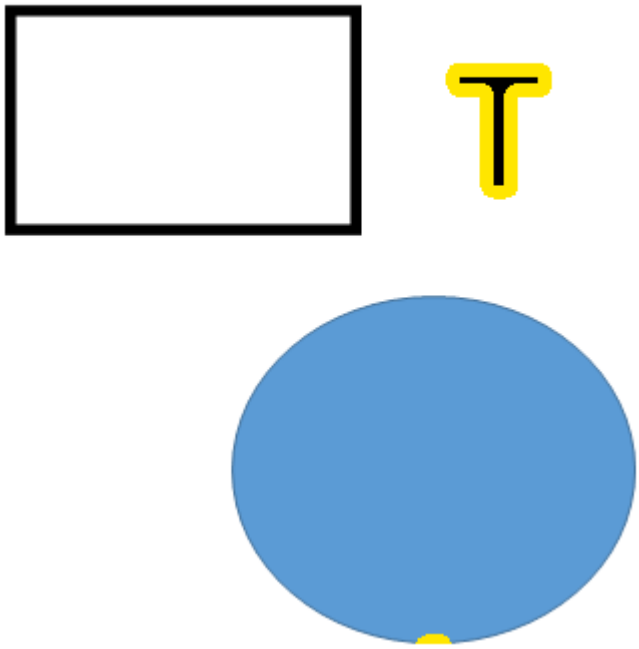


Practical 7b) - Drawing Contours.

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
#load an image
image = cv2.imread("CONTOURS.png")

#Changing the color-space
imgray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

#Find edges
edges = cv2.Canny(imgray, 10, 100)

#Find Comtours
contours, hierarchy = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

#Find Number of contours
print("Number of Contours is :"+str(len(contours)))

#Draw yellow border around two contours
cv2.drawContours(image, contours, 0, (0,230,255), 6)
cv2.drawContours(image, contours, 2, (0,230,255), 6)

#Show the image with contours
cv2_imshow(image)
```

Output –

```
Number of Contours is :5
```



Practical 7c) - Detecting Circles with Hough Circles.

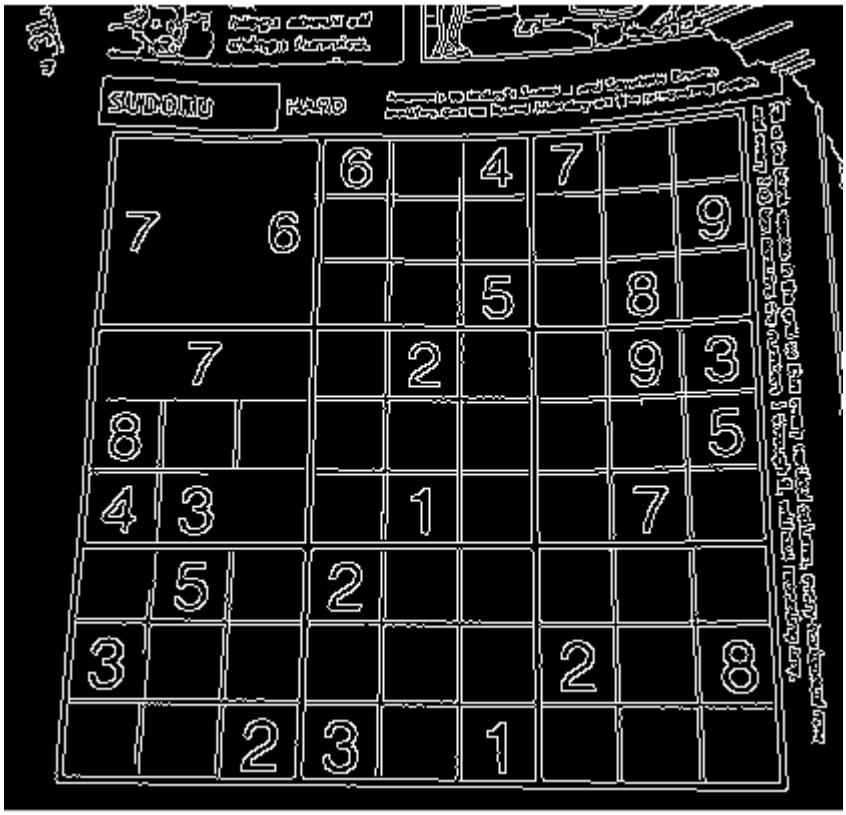
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
#Read image
img = cv2.imread("alien_1.webp", cv2.IMREAD_COLOR)

#Convert to grayscale.
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#Blur using 3 * 3 kernel
gray_blurred = cv2.blur(gray,(3,3))

#Apply Hogh transform on the blurred image.
detected_circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, 1 , 20,
param1 = 50, param2 = 30, minRadius = 1, maxRadius = 40)

#Draw circles that are detected.
if detected_circles is not None:

    #Convert the circle parameters a,b and r to integers.
    detected_circles = np.uint16(np.around(detected_circles))

    for pt in detected_circles[0,:]:
        a,b,r = pt[0], pt[1], pt[2]

        #Draw the circumference of the circle
        cv2.circle(img , (a,b) , r , (0,255,0), 2)

        #Draw a small circle (of radius 1) to show the center.
        cv2.circle(img, (a,b), 1, (0,0,255), 3)
        cv2_imshow(img)
```

Output –

Practical 7d) - Detecting Lines with Hough Lines.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow
%matplotlib inline

img= cv2.imread('sudoku-original.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges= cv2.Canny(gray,50,150,apertureSize = 3)
cv2_imshow(edges)
lines = cv2.HoughLines(edges,1,np.pi/180,70)

for rho,theta in lines[0]:
  a=np.cos(theta)
  b=np.sin(theta)
  x0=a*rho
  y0=b*rho
  x1=int(x0 + 1000*(-b))
  y1=int(y0 + 1000*(a))
  x2=int(x0 - 1000*(-b))
  y2=int(y0 - 1000*(a))

  cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)

cv2.imwrite('houghlines3.jpg',img)
plt.subplot(131),plt.imshow(gray,cmap='gray')
plt.title('Original Image'),plt.xticks([]), plt.yticks([])
plt.subplot(132),plt.imshow(edges,cmap='gray')
plt.title('Edge Image'),plt.xticks([]), plt.yticks([])
plt.subplot(133),plt.imshow(img,cmap='gray')
plt.title('Houghlines'),plt.xticks([]), plt.yticks([])
plt.show()
```

Output –



Original Image

Edge Image

Houghlines

Assignment Exercise 7

Shape detection from image

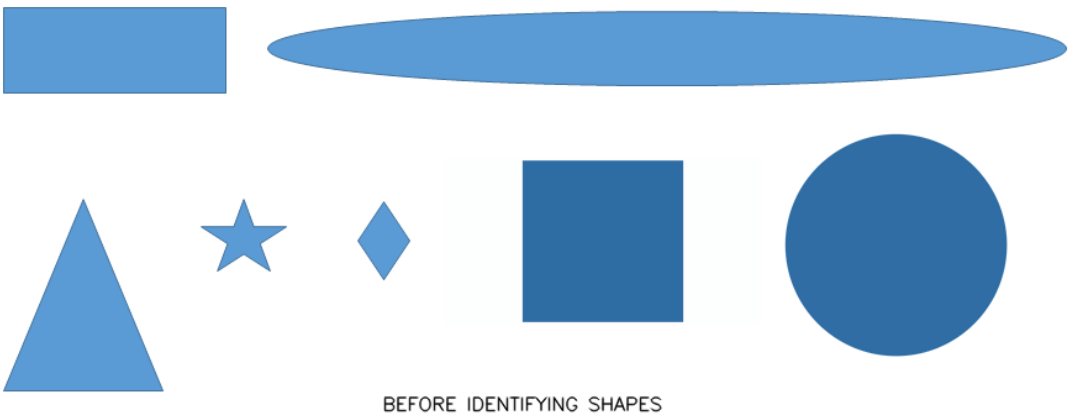
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

#Reading image.
image = cv2.imread("shapes.png")

#Making Copy of the original Image.
original_image = image.copy()

#Converting Input image into grayscale Image.
imgray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

#Common Function to draw contours on shape and put text respectively.
def drawingContoursAndPutText(approx, color, cordinates, text):

  cv2.drawContours(image, [approx], 0, color, cv2.FILLED)
  '''
    1st Arguement - Source Image,
    2nd Arguement - Text on Image,
    3rd Arguement - Cordinates of text,
    4th Arguement - Font Face,
    5th Arguement - Font Scale,
    6th Arguement - Font Color,
    7th Arguement - Font Thickness,
    8th Arguement - Line Type
  '''
  cv2.putText(image, text, cordinates , cv2.FONT_HERSHEY_SIMPLEX , 0.5, (0, 0, 0) , 1,
cv2.LINE_AA)

#_,threshold = cv2.threshold(image, 240, 255,cv2.THRESH_BINARY)

edges = cv2.Canny(imgray, 10, 100)

contours, hierarchy = cv2.findContours(edges, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours :

    '''
      approxPolyDP() -
      This function calculates and approximates a polygonal curve with specified precision.
      approximates a contour shape to another shape with less number of vertices depending
upon the precision we specify.
    '''
    approx = cv2.approxPolyDP(cnt, 0.009*cv2.arcLength(cnt,True), True)

    '''
      drawContours() -
      Draw Contours on different shapes
      Draws the contours outlines or filled color .
      It can also be used to draw any shape provided you have its boundary points.
      1st Argument - source image,
```

```
    2nd Argument - the contours which should be passed as a Python list,
    3rd Argument - index of contours (useful when drawing individual contour. To draw all
contours, pass -1),
    4th Argument - Color,
    5th Arguement - Thickness.
  '''
  cv2.drawContours(image, [approx], 0, (0,0,0), 4)
  #print(len(approx))

  #Triangle
  if len(approx) == 3:
    drawingContoursAndPutText(approx, (255,0,0), (125, 395), 'Triangle')

  elif len(approx) == 4:
    '''
      BoundingRect() -
      It gives the boundary points of the rectangle.
      It returns x and y cordinate and width, height of each shape.
    '''
    (x, y, w, h) = cv2.boundingRect(approx)

    '''
      Calculating Aspect Ratio's of different shapes to identify shapes.
    '''
    aspectRatio = float(w)/h
    #print(aspectRatio)

    if aspectRatio > 0.95 and aspectRatio < 1.05:
      #Square
      drawingContoursAndPutText(approx, (0,255,255), (515, 345), 'Square')

    elif aspectRatio > 2.5 :
      #Rectangle
      drawingContoursAndPutText(approx, (128,0,128), (155, 155), 'Rectangle')

    elif aspectRatio < 0.7:
      #Diamond
      drawingContoursAndPutText(approx, (42, 42, 165), (360, 310), 'Diamond')

  #Star
  elif len(approx) == 10:
    drawingContoursAndPutText(approx, (0,165,255), (250, 295), 'Star')

  #Circle
  elif len(approx) == 16 :
    drawingContoursAndPutText(approx, (0,255,0), (760, 375), 'Circle')

  elif len(approx) == 7:
  #Ellipse
    drawingContoursAndPutText(approx, (203,192,255), (600, 155),  'Ellipse')

'''
cv2_imshow(original_image)
cv2.putText(original_image, "BEFORE IDENTIFYING SHAPES", (380,390) ,
cv2.FONT_HERSHEY_SIMPLEX , 0.5, (0, 0, 0) , 1, cv2.LINE_AA)
```

```
print("============================================================
==========================================================
==============")

cv2_imshow(image)
cv2.putText(image, "AFTER IDENTIFYING SHAPES", (460,400) ,
cv2.FONT_HERSHEY_SIMPLEX , 0.5, (0, 0, 0) , 1, cv2.LINE_AA)
print("============================================================
==========================================================
==============")
#cv2_imshow(threshold)
'''

cv2_imshow(original_image)
cv2.putText(original_image, "BEFORE IDENTIFYING SHAPES", (380,390) ,
cv2.FONT_HERSHEY_SIMPLEX , 0.5, (0, 0, 0) , 1, cv2.LINE_AA)
```

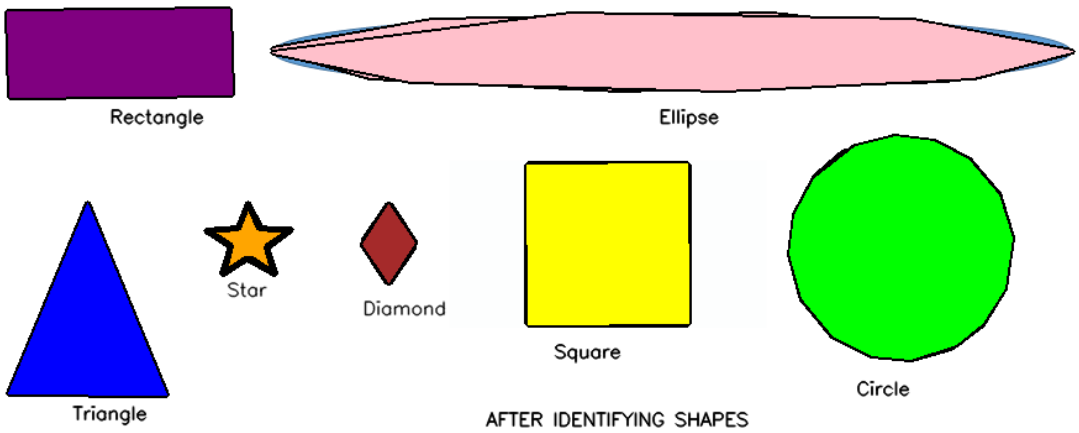
Output –

1) Original Image.




```
cv2_imshow(image)
cv2.putText(image, "AFTER IDENTIFYING SHAPES", (380,390) ,
cv2.FONT_HERSHEY_SIMPLEX , 0.5, (0, 0, 0) , 1, cv2.LINE_AA)
```

2) After Identifying Shapes.


Rectangle
Ellipse
Star
Diamond
Square
Circle
Triangle
AFTER IDENTIFYING SHAPES

Practical 8

- ORB Algorithm

ORB is a fusion of FAST keypoint detector and BRIEF descriptor with some added features to improve the performance. FAST is Features from Accelerated Segment Test used to detect features from the provided image. It also uses a pyramid to produce multiscale-features. Now it doesn't compute the orientation and descriptors for the features, so this is where BRIEF comes in the role.

ORB uses BRIEF descriptors but as the BRIEF performs poorly with rotation. So what ORB does is to rotate the BRIEF according to the orientation of keypoints. Using the orientation of the patch, its rotation matrix is found and rotates the BRIEF to get the rotated version. ORB is an efficient alternative to SIFT or SURF algorithms used for feature extraction, in computation cost, matching performance, and mainly the patents. SIFT and SURF are patented and you are supposed to pay them for its use. But ORB is not patented.

Algorithm

- Take the query image and convert it to grayscale.
- Now Initialize the ORB detector and detect the keypoints in query image and scene.
- Compute the descriptors belonging to both the images.
- Match the keypoints using Brute Force Matcher.
- Show the matched images.

- Homography –

Homography is a transformation that maps the points in one point to the corresponding point in another image. The homography is a 3×3 matrix :

$$\mathbf{H} = \begin{vmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{vmatrix}$$

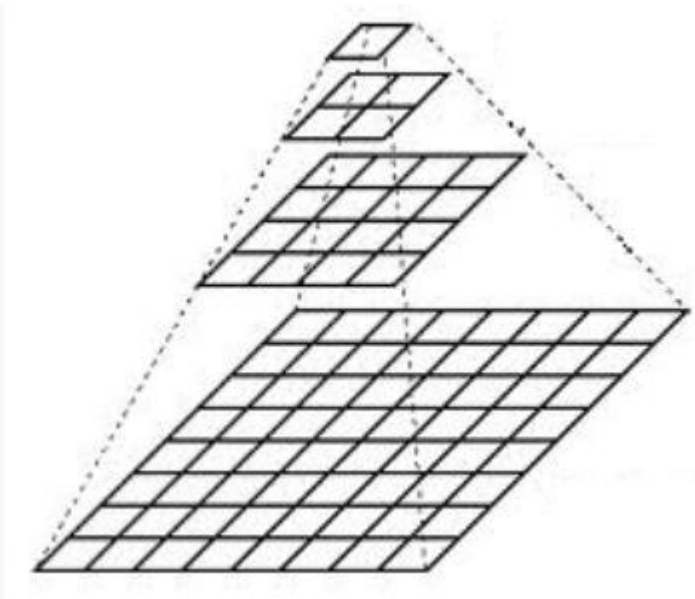
If 2 points are not in the same plane then we have to use 2 homographs. Similarly, for n planes, we have to use n homographs. If we have more homographs then we need to handle all of them properly. So that is why we use feature matching.
Homography : To detect the homography of the object we have to obtain the matrix and use function `findHomography()` to obtain the homograph of the object.

- Feature Matching : Feature matching means finding corresponding features from two similar datasets based on a search distance. Now will be using sift algorithm and flann type feature matching.

- Image Pyramids-

Image Pyramids are one of the most beautiful concept of image processing.Normally, we work with images with default resolution but many times we need to change the resolution (lower it) or resize the original image in that case image pyramids comes handy.

The **pyrUp()** function increases the size to double of its original size and **pyrDown()** function decreases the size to half. If we keep the original image as a base image and go on applying **`pyrDown`** function on it and keep the images in a vertical stack, it will look like a pyramid. The same is true for upscaling the original image by **pyrUp** function.



Once we scale down and if we rescale it to the original size, we lose some information and the resolution of the new image is much lower than the original one.

- Erosion

cv2.erode() method is used to perform erosion on the image. The basic idea of erosion is just like soil erosion only, it erodes away the boundaries of foreground object (Always try to keep foreground in white). It is normally performed on binary images. It needs two inputs, one is our original image, second one is called structuring element or kernel which decides the nature of operation. A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).

Syntax: cv2.erode(src, kernel[, dst[, anchor[, iterations[, borderType[, borderValue]]]]])

Parameters:
src: It is the image which is to be eroded .
kernel: A structuring element used for erosion. If element = Mat(), a 3 x 3 rectangular structuring element is used. Kernel can be created using getStructuringElement.
dst: It is the output image of the same size and type as src.
anchor: It is a variable of type integer representing anchor point and it's default value Point is (-1, -1) which means that the anchor is at the kernel center.
borderType: It depicts what kind of border to be added. It is defined by flags like cv2.BORDER_CONSTANT, cv2.BORDER_REFLECT, etc.
iterations: It is number of times erosion is applied.
borderValue: It is border value in case of a constant border.
Return Value: It returns an image.

- Dilation

Morphological operations are a set of operations that process images based on shapes. They apply a structuring element to an input image and generate an output image.
The most basic morphological operations are two: Erosion and Dilation.

Basics of Erosion:
Erodes away the boundaries of foreground object
Used to diminish the features of an image.

Working of erosion:
1. A kernel(a matrix of odd size(3,5,7) is convolved with the image.
2. A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero).
3. Thus all the pixels near boundary will be discarded depending upon the size of kernel.
4. So the thickness or size of the foreground object decreases or simply white region decreases in the image.

Basics of dilation:
Increases the object area
Used to accentuate features

Working of dilation:
1. A kernel(a matrix of odd size(3,5,7) is convolved with the image
2. A pixel element in the original image is '1' if atleast one pixel under the kernel is '1'.
3. It increases the white region in the image or size of foreground object increases.

Uses of Erosion and Dilation:
Erosion:
It is useful for removing small white noises.
Used to detach two connected objects etc.

Dilation:
In cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases.
It is also useful in joining broken parts of an object.

- Morphological Operations

Morphological operations are used to extract image components that are useful in the representation and description of region shape. Morphological operations are some basic tasks dependent on the picture shape. It is typically performed on binary images. It needs two data sources, one is the input image, the second one is called structuring component. Morphological operators take an input image and a structuring component as input and these elements are then combines using the set operators. The objects in the input image are processed depending on attributes of the shape of the image, which are encoded in the structuring component.
Opening is similar to erosion as it tends to remove the bright foreground pixels from the edges of regions of foreground pixels. The impact of the operator is to safeguard foreground region that has similarity with the structuring component, or that can totally contain the structuring component while taking out every single other area of foreground pixels. Opening operation is used for removing internal noise in an image.

Opening is erosion operation followed by dilation operation.

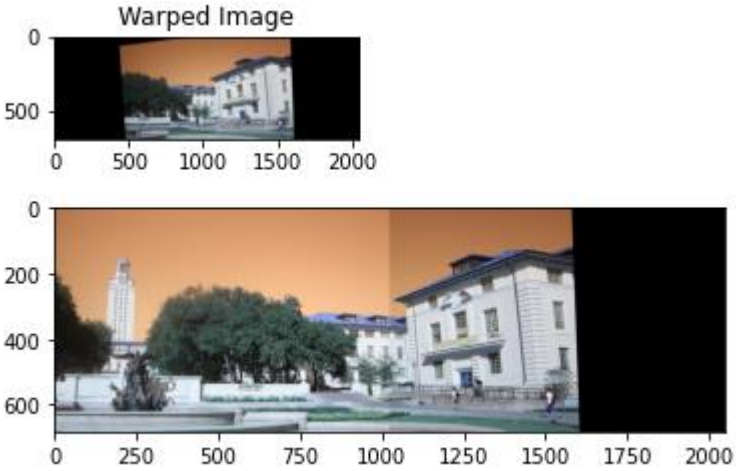
$$A \circ B = (A \ominus B) \oplus B$$

Syntax: cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)
Parameters:
-> image: Input Image array.
-> cv2.MORPH_OPEN: Applying the Morphological Opening operation.
-> kernel: Structuring element.

Practical 8a) – Image Stitching.

```
#Image Stiching
import numpy as np
import cv2
import matplotlib.pyplot as plt
from random import randrange
from google.colab.patches import cv2_imshow
img_ = cv2.imread('right.jpeg')
img1 = cv2.cvtColor(img_,cv2.COLOR_BGR2GRAY)
img = cv2.imread('left.jpeg')
img2 = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2_imshow(img1)
cv2_imshow(img2)
orb = cv2.ORB_create(nfeatures=1500)
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=False)
matches = bf.knnMatch(des1,des2,k=2)

#APPLY RATIO TEST
good = []
for m in matches:
  if m[0].distance < 0.5*m[1].distance:
    good.append(m)
    matches = np.asarray(good)
if len(matches[:,0]) >= 4:
  src = np.float32([ kp1[m.queryIdx].pt for m in matches[:,0] ]).reshape(-1,1,2)
  dst = np.float32([ kp2[m.trainIdx].pt for m in matches[:,0] ]).reshape(-1,1,2)
  H, masked = cv2.findHomography(src, dst, cv2.RANSAC, 5.0)
  #print (H)
else:
  raise AssertionError('Cant find enough keypoints.')
dst = cv2.warpPerspective(img_,H,(img.shape[1] + img_.shape[1], img.shape[0]))
plt.subplot(122),plt.imshow(dst),plt.title('Warped Image')
plt.show()
plt.figure()
dst[0:img.shape[0], 0:img.shape[1]] = img
cv2.imwrite('output.jpg',dst)
plt.imshow(dst)
plt.show()
```

Output –


Warped Image
0
500
0
500
1000
1500
2000

0
200
400
600
0
250
500
750
1000
1250
1500
1750
2000

Practical 8b) – Image Pyramid Scheme. (Code -1)

```
#Image Pyramids Scheme
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

img = cv2.imread("jeff-bezos.jpg")
G = img.copy()
gpimg = [G]
for i in range(6) :
        G = cv2.pyrDown(G)
        gpimg.append(G)
        cv2_imshow(gpimg[i])
```

Output –

Practical 8b) – Image Pyramid Scheme. (Code -2)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

img = cv2.imread("jeff-bezos.jpg")
G = img.copy()
gpimg = [G]
for i in range(6):
        G = cv2.pyrDown(G)
        gpimg.append(G)
high_reso = cv2.pyrUp(gpimg[1])
cv2_imshow(high_reso)
```

Output –

Practical 8c) – Laplacian Pyramid Scheme.

```
#Laplacian
lpimg = [gpimg[5]]
cv2_imshow(lpimg[0])
for i in range(6):
        GE = cv2.pyrUp(gpimg[i])
        GE = cv2.resize(GE, gpimg[i-1].shape[-2::-1])
        L = cv2.subtract(gpimg[i-1],GE)
        lpimg.append(L)
for i in range(6, 0, -1):
        cv2_imshow(lpimg[i])
```

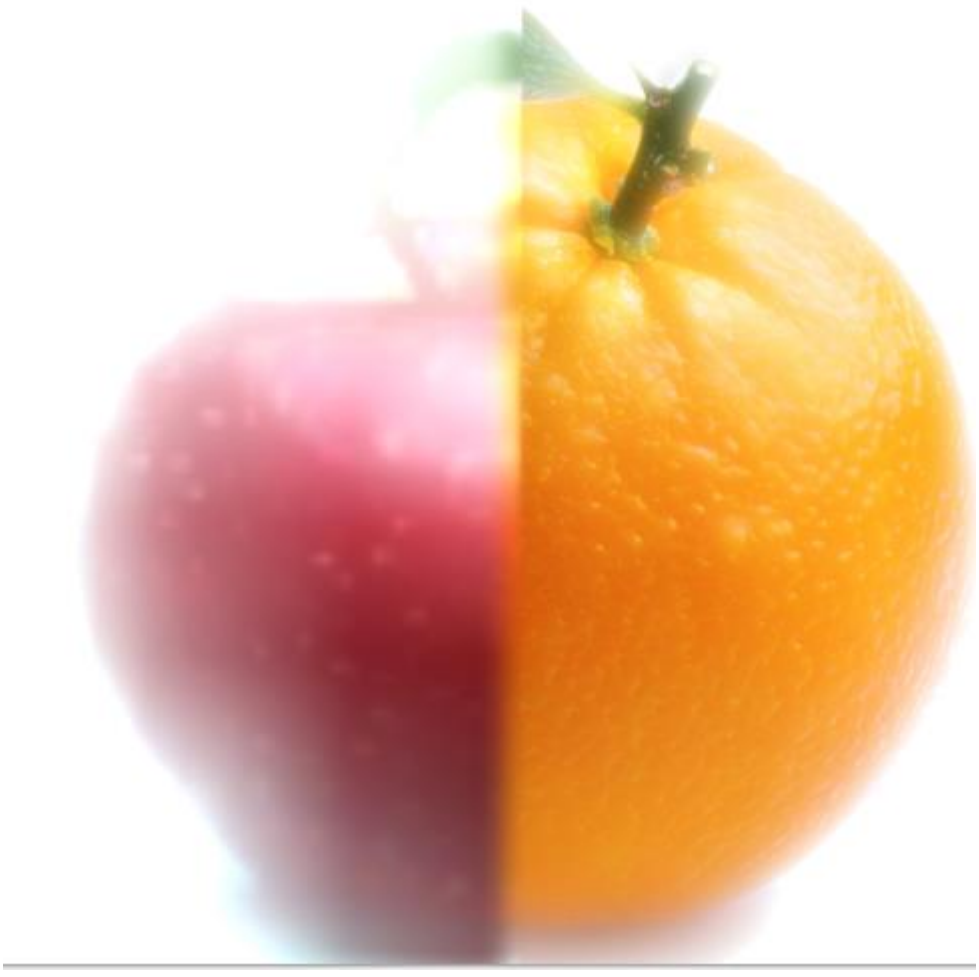
Output –

Practical 8d) – Image Blending.

```
# Image Blending

import cv2
import numpy as np

A = cv2.imread('apple.jpg')
B = cv2.imread('orange.jpg')

# generate Gaussian pyramid for A
G = A.copy()
print(G)
gpA = [G]
for i in range(6):
  G = cv2.pyrDown(G)
  gpA.append(G)

# generate Gaussian pyramid for B
G = B.copy()
print(G)
gpB = [G]
for i in range(6):
  G = cv2.pyrDown(G)
  gpB.append(G)

# generate laplacian pyramid for A
lpA = [gpA[5]]

for i in range(6, 0, -1):
  GE = cv2.pyrUp(gpA[i])
  GE = cv2.resize(GE, gpA[i - 1].shape[-2::-1])
  L = cv2.subtract(gpA[i-1], GE)
  lpA.append(L)

# generate laplacian pyramid for B
lpB = [gpB[5]]

for i in range(6, 0, -1):
  GE = cv2.pyrUp(gpB[i])
  GE = cv2.resize(GE, gpB[i - 1].shape[-2::-1])
  L = cv2.subtract(gpB[i-1], GE)
  # print(L.shape)
  lpB.append(L)


# Now add left and right halves of images in each level
LS = []
lpAc = []

for i in range(len(lpA)):
  b = cv2.resize(lpA[i], lpB[i].shape[-2::-1])
  lpAc.append(b)

print(len(lpAc))
print(len(lpB))
```

```
j = 0

for i in zip(lpAc, lpB):
  la, lb = i
  rows, cols, dpt = la.shape
  ls = np.hstack((la[:, 0:cols//2], lb[:, cols//2:]))
  j = j + 1
  LS.append(ls)

ls_ = LS[0]
for i in range(1, 6):
  ls_ = cv2.pyrUp(ls_)
  ls_ = cv2.resize(ls_, LS[i].shape[-2::-1])
  ls_ = cv2.add(ls_, LS[i])

# image with direct connecting each other
B = cv2.resize(B, A.shape[-2::-1])
real = np.hstack((A[:, :cols//2], B[:, cols//2:]))

cv2.imwrite('Pyramid_blending2.jpg', ls_)
cv2.imwrite('Direct_blending.jpg', real)

cv2_imshow(ls_)
cv2_imshow(real)
```

Output –



Practical 8e) – Morphological Transformation – Erosion.

```
#morphological transformation - Erosion
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
img = cv2.imread("j.png",0)
cv2_imshow(img)
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(img, kernel, iterations = 1)
cv2_imshow(erosion)
```

Output –

Input Image	Output Image
	

Practical 8f) – Morphological Transformation – Dilation.

```
#Morphological Transformation - Dilation
dilation = cv2.dilate(img, kernel, iterations = 1)
cv2_imshow(dilation)
```

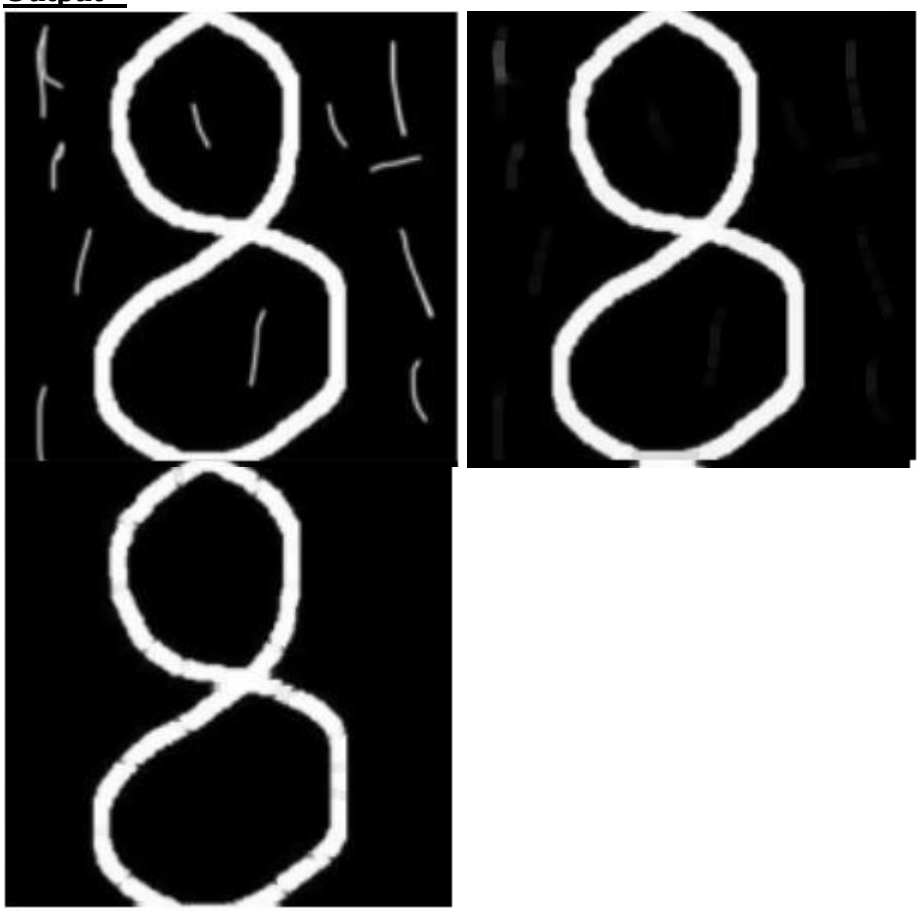
Output –

Input Image	Output Image
	

Practical 8g) – Morphological Transformation – Opening & Closing.

```
#Morphological Transformation - Opening & Closing
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

img = cv2.imread("open.png",0)
cv2_imshow(img)
#Opening -> Erosion -> Dilation
kernel = np.ones((5,5),np.uint8)
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
cv2_imshow(opening)
closing = cv2.imread("closing.png")
closing = cv2.morphologyEx(closing, cv2.MORPH_CLOSE, kernel)
#Closing -> Dilation -> Erosion
cv2_imshow(closing)
```

Output –



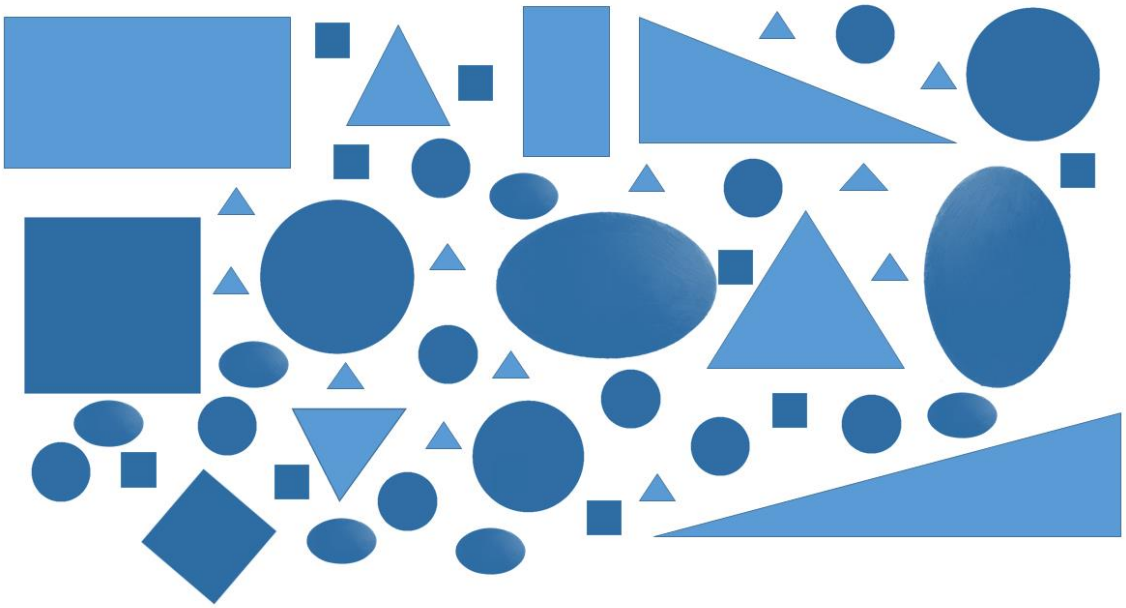
Assignment Exercise 8

Detect Blobs and Show Count.

```
# Standard imports
import cv2
import numpy as np;
from google.colab.patches import cv2_imshow

# Read image
image = cv2.imread("countShape.png")

# Set up the detector with default parameters.
detector = cv2.SimpleBlobDetector_create()

# Detect blobs.
keypoints = detector.detect(image)

# Draw blobs on our image as red circles
blobs = cv2.drawKeypoints(image, keypoints, np.array([]), (0, 0, 255),
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# Show blobs
text = "There are total " + str(len(keypoints)) + " blobs."
cv2.putText(blobs, text, (480,745) , cv2.FONT_HERSHEY_SIMPLEX , 0.5, (0, 0, 0) , 1,
cv2.LINE_AA)
cv2_imshow(blobs)
```

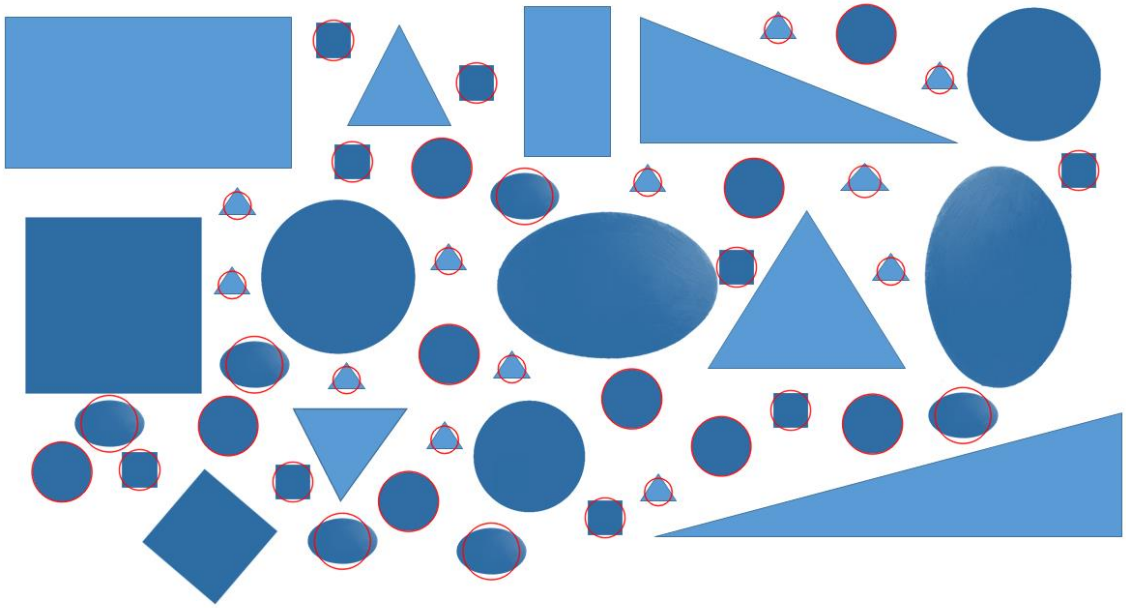
- **Outputs**

3) Original Image.



```
def countNumberOfBlobs(min, max, shapeName) :

  # Set up the detector and configure its params.
  # Create a detector with the parameters.
  params = cv2.SimpleBlobDetector_Params()

  params.filterByCircularity=1
  params.minCircularity=min
  params.maxCircularity=max

  detector = cv2.SimpleBlobDetector_create(params)
  # Detect blobs
  keypoints = detector.detect(image)

  # Draw blobs on image as red circles
  blobs = cv2.drawKeypoints(image, keypoints, np.array([]), (0, 0, 255),
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

  # Show blobs
  text = "There are total " + str(len(keypoints)) + " "+ shapeName +" in the above image."
  cv2.putText(blobs, text, (480,745) , cv2.FONT_HERSHEY_SIMPLEX , 0.5, (0, 0, 0) , 1,
cv2.LINE_AA)
  cv2_imshow(blobs)
```

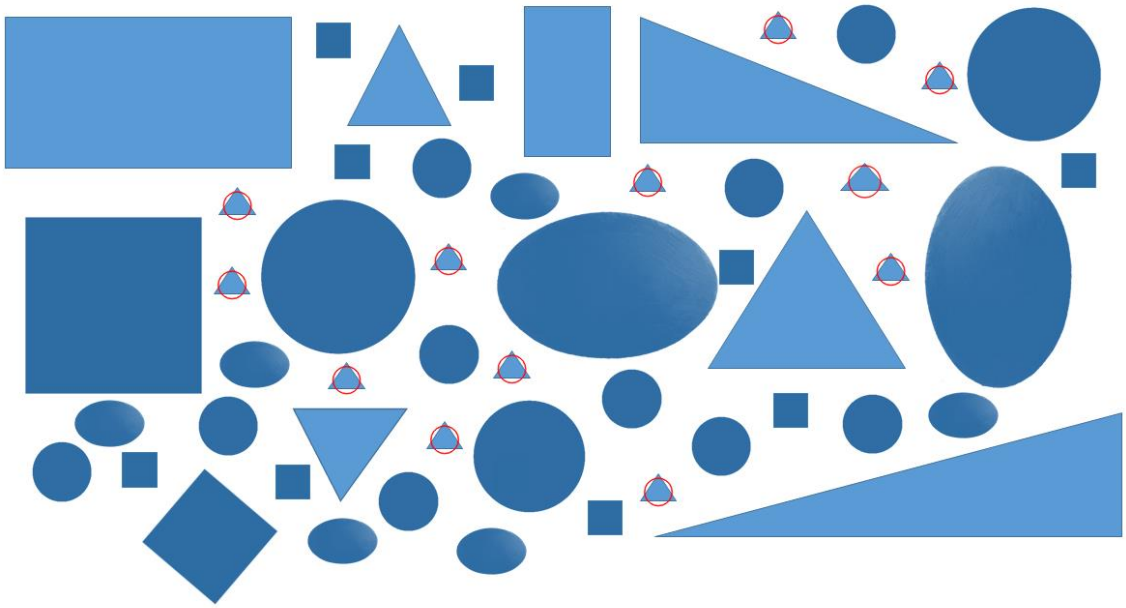
4) After Identifying Blobs.



There are total 37 blobs.

```
# Trianglar Blobs
countNumberOfBlobs(0.4, 0.785, "TRIANGLAR BLOBS")
```

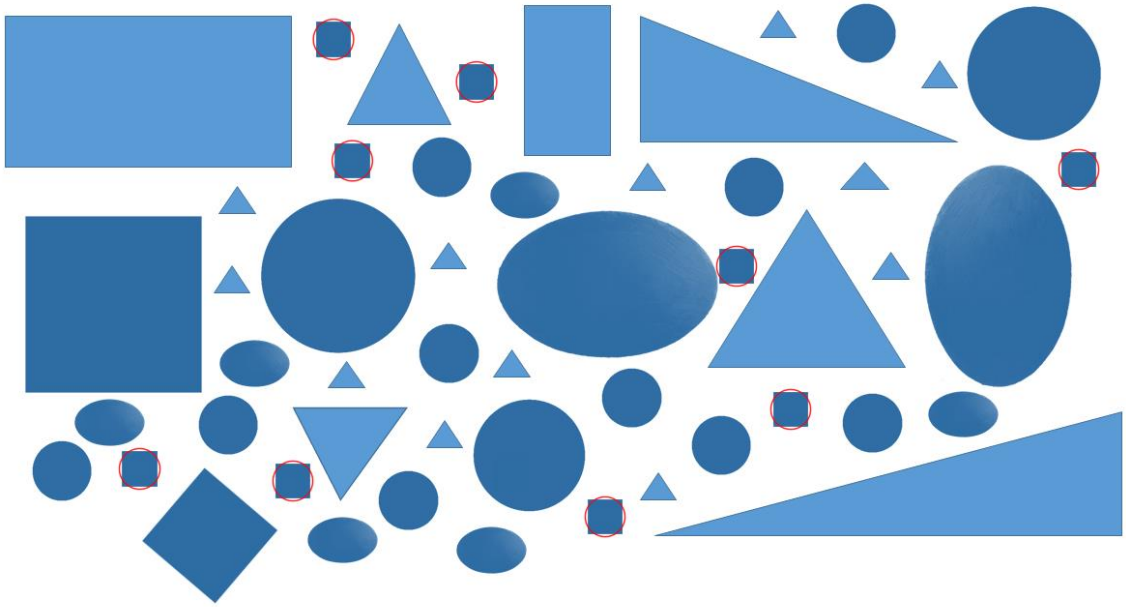
5) Triangular Blobs



There are total 12 TRIANGLAR BLOBS in the above image.

```
# Square Blobs
countNumberOfBlobs(0.79, 0.83, "SQUARE BLOBS")
```

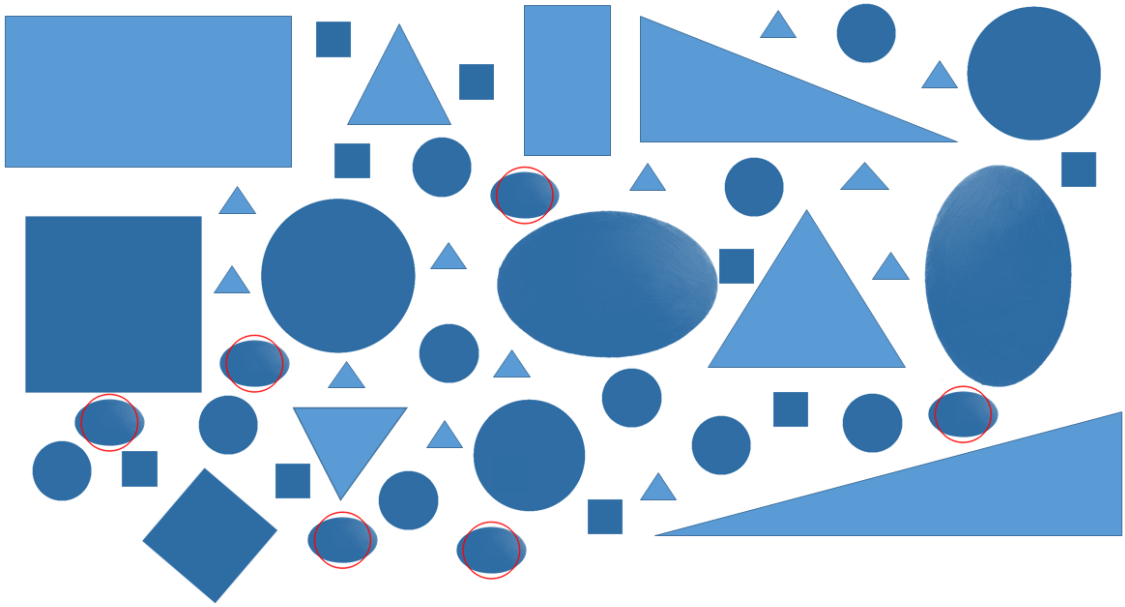
6) Square Blobs



There are total 9 SQUARE BLOBS in the above image.

```
# Oval Blobs
countNumberOfBlobs(0.84, 0.875, "OVAL BLOBS")
```

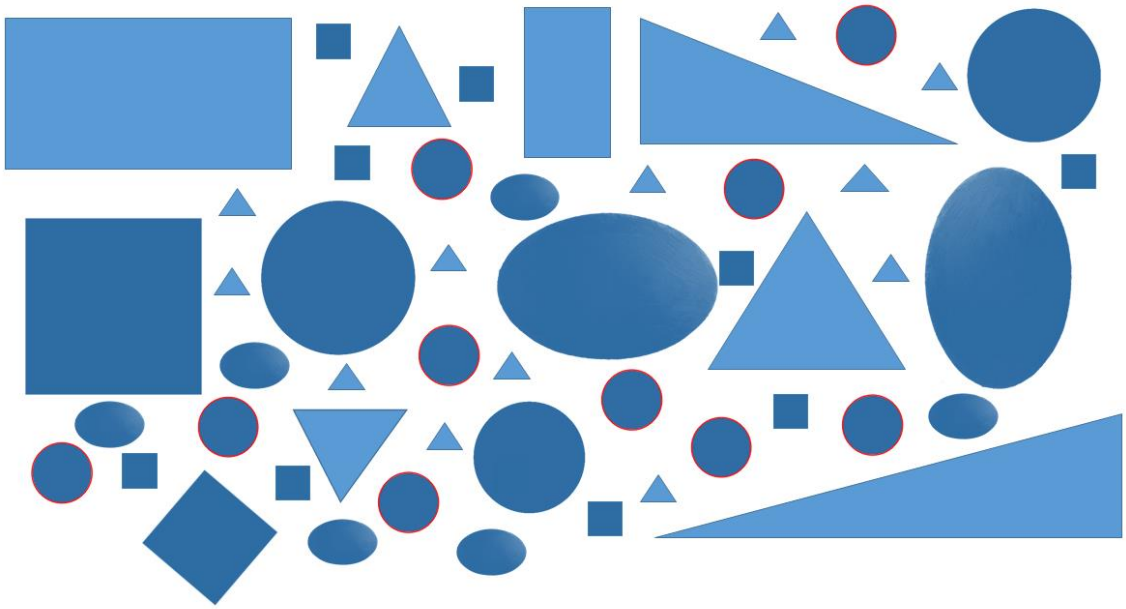
7) Oval Blobs



There are total 6 OVAL BLOBS in the above image.

```
# Circular Blobs
countNumberOfBlobs(0.9, 0.91, "CIRCULAR BLOBS")
```

8) Circle Blobs

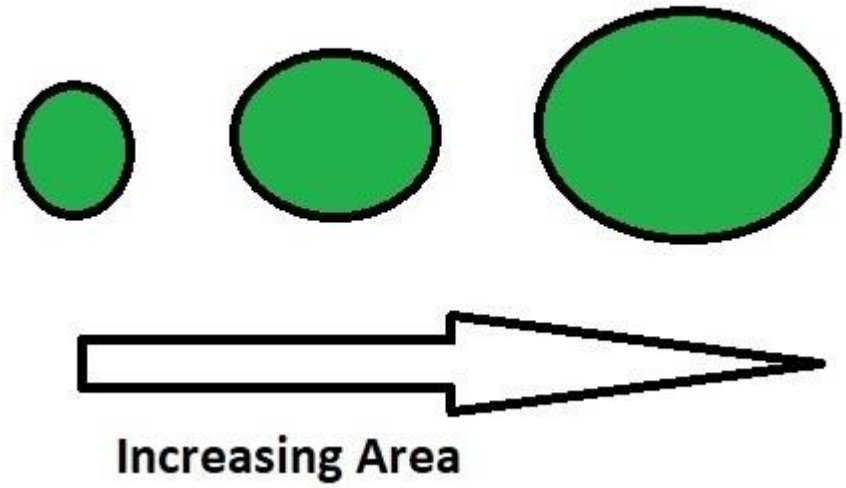


There are total 10 CIRCULAR BLOBS in the above image.

Practical 9

- Blobs –

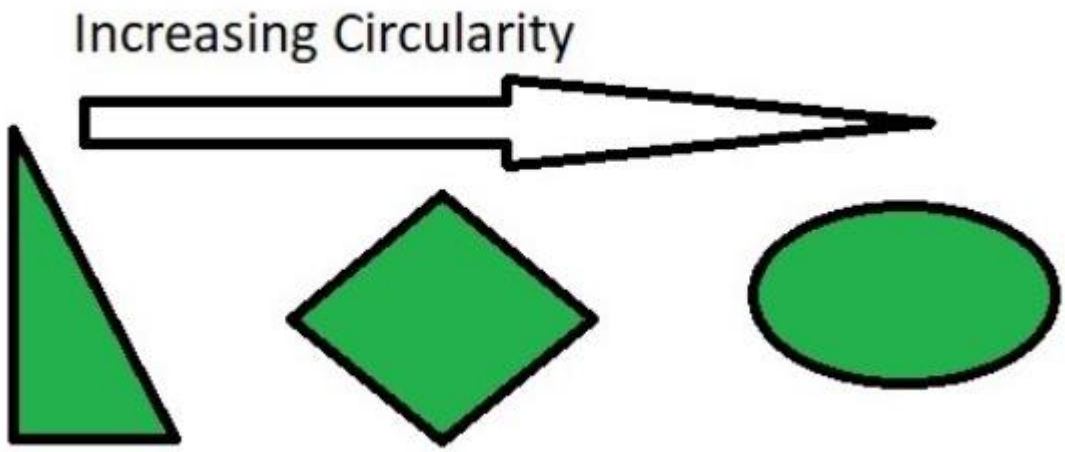
To identify circles, ellipses or in general any shape in which the pixels are connected we use the SimpleBlobDetector() function of OpenCV. In non-technical terms, a blob is understood as a thick liquid drop. Here, we are going to call all shapes as a blob. Our task is to detect and recognize whether the blob is a circle or not.
OpenCV provides a convenient way to detect blobs and filter them based on different characteristics. There are various different parameters that control the identification process and the results. The important parameters used for this project are:
Filter by Area – This is to avoid any identification of any small dots present in the image that can be wrongly detected as a circle.



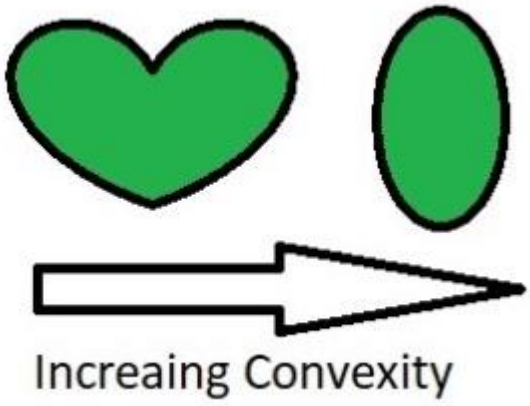
Filter by Circularity – This helps us to identify, shapes that are more similar to a circle.

$$\text{Circularity} = 4 * pi * Area/(perimeter)^2.$$

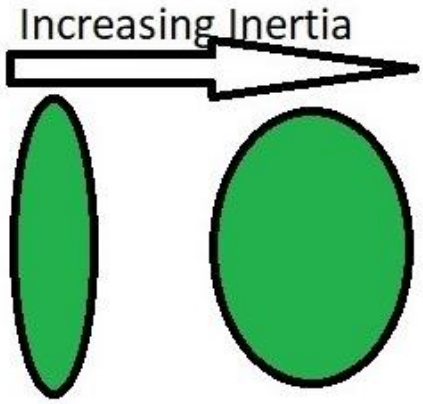
A true circle has circularity of 1, a square has a circularity near 78%.



Filter by Convexity – Concavity in general, destroys the circularity. More is the convexity, the closer it is to a close circle.



Filter by Inertia – Objects similar to a circle has larger inertial.E.g. for a circle, this value is 1, for an ellipse it is between 0 and 1, and for a line it is 0. To filter by inertia ratio, set filterByInertia = 1, and set, 0 <= minInertiaRatio <= 1 and maxInertiaRatio (<=1) appropriately.



- Watershed Algorithm

If we want to extract or define something from the rest of the image, eg. detecting an object from a background, we can break the image up into segments in which we can do more processing on. This is typically called Segmentation.

Morphological operations are some simple operations based on the image shape. It is normally performed on binary images. Two basic morphological operators are Erosion and Dilation. For basic understanding about Dilation and Erosion, refer this article.
In order to process on we'll use OTSU's threshold algorithm where this removes over segmented result due to noise or any other irregularities in the image and implement with OpenCV.

Approach :

- Label the region which we are sure of being the foreground or object with one color (or intensity), Label the region which we are sure of being background or non-object with another color.
- Finally the region which we are not sure of anything, label it with 0. That is our marker. Then apply watershed algorithm.
- Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1.

Now, we need to remove any small white noises in the image i.e. foreground. For that we can use morphological closing. To remove any small holes in the foreground object, we can use morphological closing. To obtain background we dilate the image. Dilation increases object boundary to background.

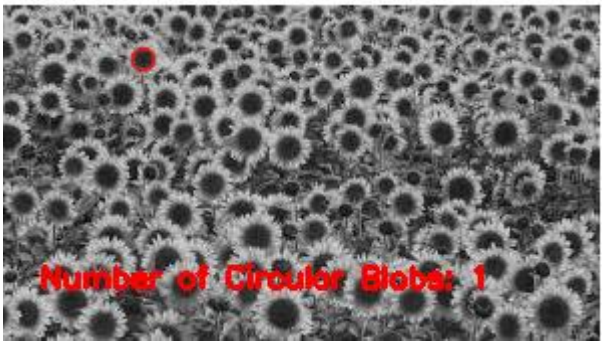
Practical 9a) – Detect Blobs.

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow

im = cv2.imread('sunflower.jpg',cv2.IMREAD_GRAYSCALE)
params = cv2.SimpleBlobDetector_Params()
detector = cv2.SimpleBlobDetector_create(params)

#detector = cv2.SimpleBlobDetector()
keypoints = detector.detect(im)

#Draw detected blobs as red circles
#cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of the
circle corresponds to the size of the blob
im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]), (0,0,255),
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

#Show keypoints
cv2_imshow(im_with_keypoints )
```

Output –

```
import cv2
import numpy as np

# Load image
image = cv2.imread("sunflower.jpg", 0)

# Set our filtering parameters
# Initialize parameter settiing using cv2.SimpleBlobDetector
params = cv2.SimpleBlobDetector_Params()

# Set Area filtering parameters
params.filterByArea = True
params.minArea = 100

# Set Circularity filtering parameters
params.filterByCircularity = True
params.minCircularity = 0.9

# Set Convexity filtering parameters
params.filterByConvexity = True
params.minConvexity = 0.2

# Set inertia filtering parameters
params.filterByInertia = True
params.minInertiaRatio = 0.01

# Create a detector with the parameters
detector = cv2.SimpleBlobDetector_create(params)

# Detect blobs
keypoints = detector.detect(image)

# Draw blobs on our image as red circles
blank = np.zeros((1, 1))
blobs = cv2.drawKeypoints(image, keypoints, blank, (0, 0, 255),
cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

number_of_blobs = len(keypoints)
text = "Number of Circular Blobs: " + str(len(keypoints))
cv2.putText(blobs, text, (20, 140),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2 )

# Show blobs
cv2_imshow(blobs)
```

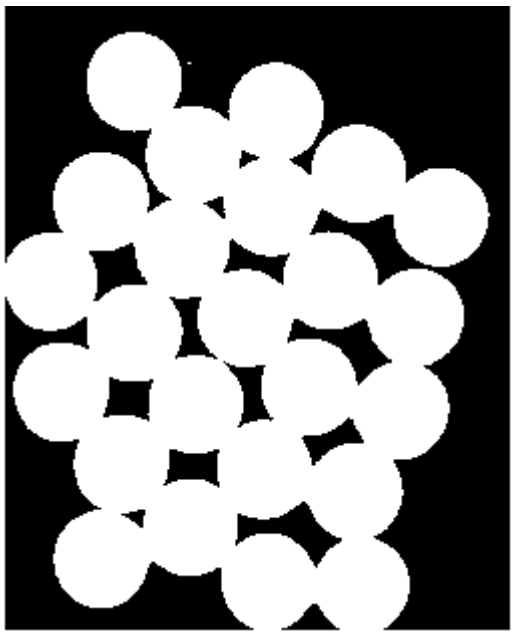
Output –



Practical 9b) – Watershed Algorithm

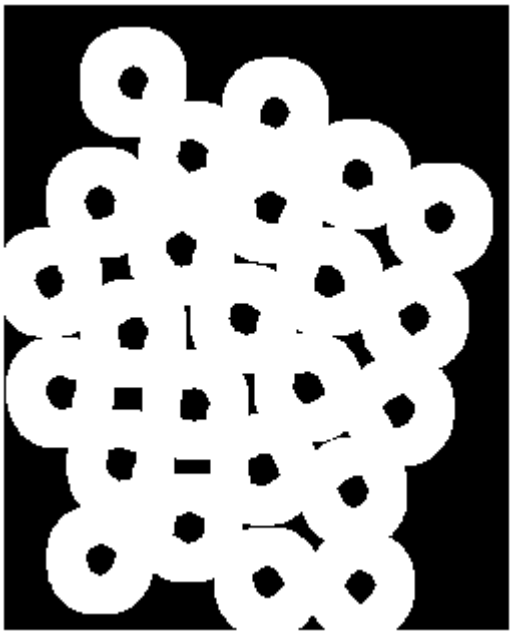
```
# watershed algorithm

import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('water_coins.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret, thresh = cv2.threshold(gray,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
cv2_imshow(thresh)
```

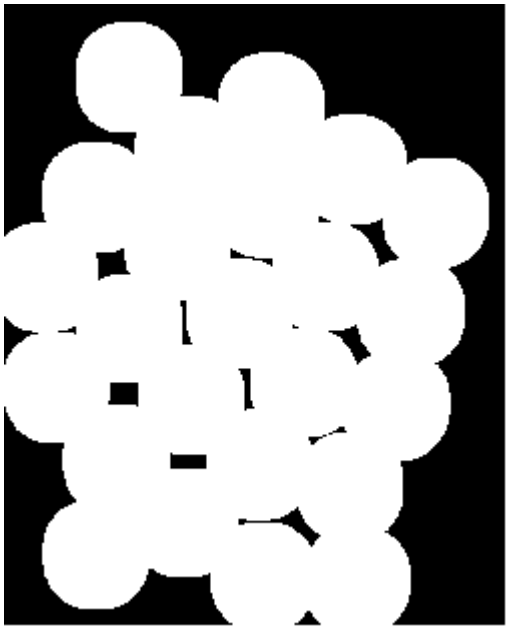
Output –

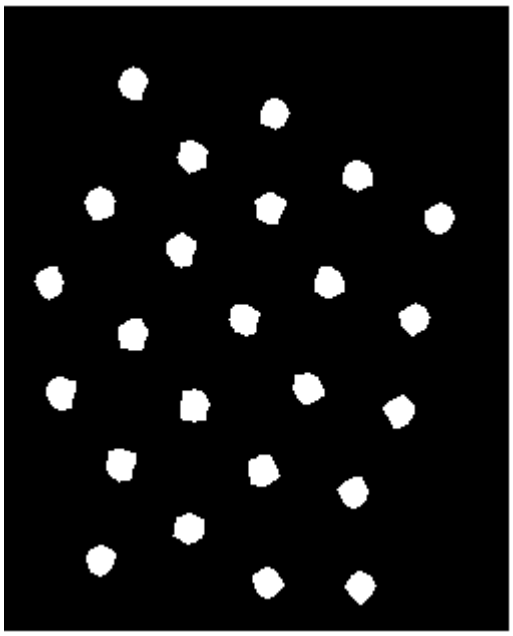


```
# noise removal
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations = 2)

# sure background area
sure_bg = cv2.dilate(opening,kernel,iterations=3)

# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,5)
ret, sure_fg = cv2.threshold(dist_transform,0.7*dist_transform.max(),255,0)

# Finding unknown region
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)

cv2_imshow(unknown)
print("----------------------------------")
cv2_imshow(sure_bg)
print("----------------------------------")
cv2_imshow(sure_fg)
```

Output –

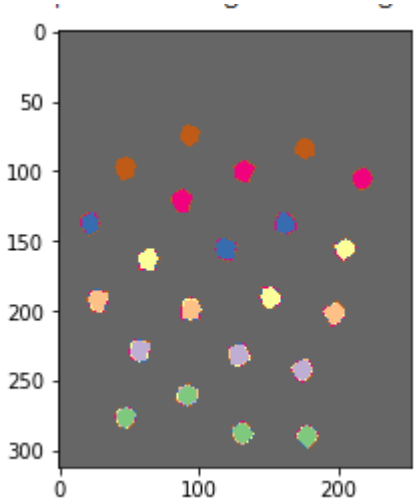






```
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)

# Add one to all labels so that sure background is not 0, but 1
markers = markers+1

# Now, mark the region of unknown with zero
markers[unknown==255] = 0
plt.imshow(markers,cmap='Accent_r')
```

Output –




```
markers = cv2.watershed(img,markers)
img[markers == -1] = [255,0,0]
cv2_imshow(img)
```

Output -

Practical 10

- cv2.CascadeClassifier :- We provide Trained XML classifiers here, which describes some features of some object we want to detect a cascade function is trained from a lot of positive(faces) and negative(non-faces) images.

- detectMultiScale() :- Detects objects of different sizes in the input image. The detected objects are returned as a list of rectangles.

Syntax - cv.CascadeClassifier.detectMultiScale(image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize]]]]])

Parameters

image	Matrix of the type CV_8U containing an image where objects are detected.
objects	Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image.
scaleFactor	Parameter specifying how much the image size is reduced at each image scale.
minNeighbors	Parameter specifying how many neighbors each candidate rectangle should have to retain it.
flags	Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.
minSize	Minimum possible object size. Objects smaller than that are ignored.
maxSize	Maximum possible object size. Objects larger than that are ignored. If maxSize == minSize model is evaluated on single scale.

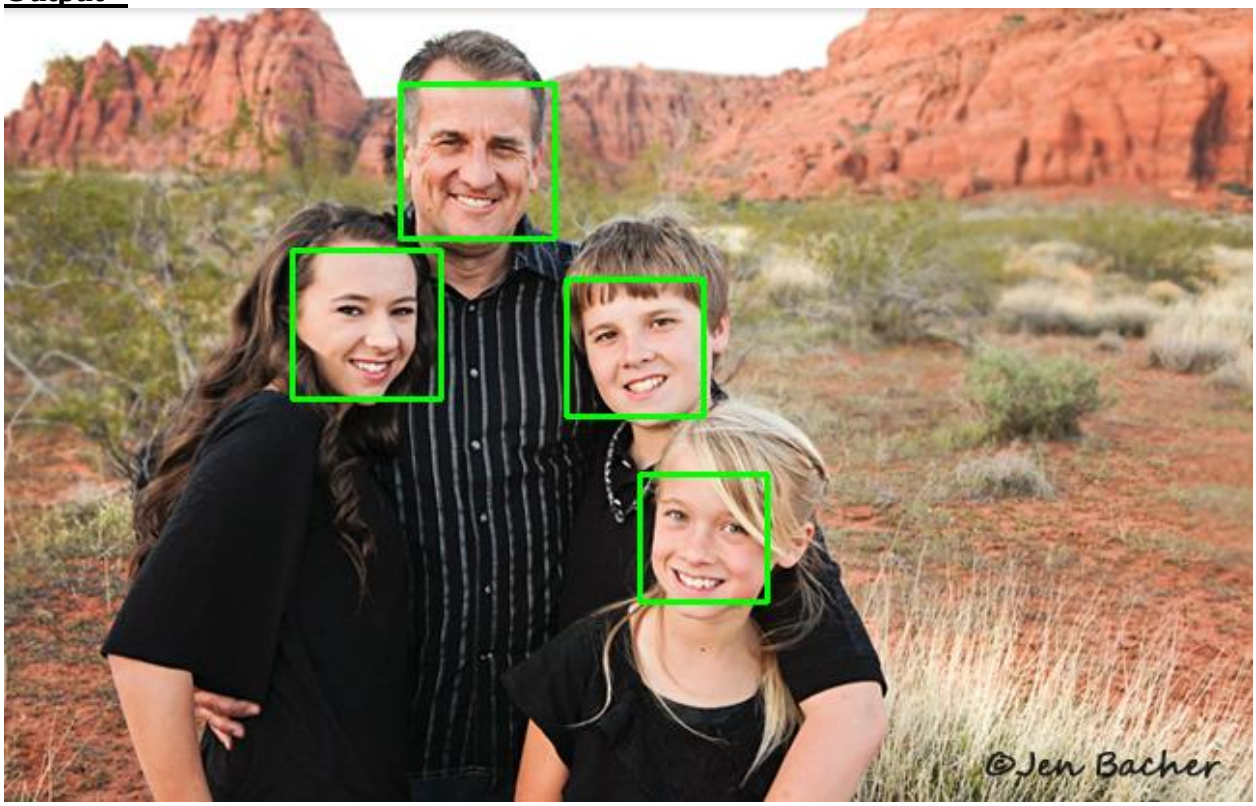
Face Detection from Image.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow

image = cv2.imread("group-image.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
cv2_imshow(gray)
```

Output –

```
faceCascade = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")

#detect faces in the image
faces=faceCascade.detectMultiScale(
   gray,
   scaleFactor=1.1,
   minNeighbors=3,
   minSize=(30,30),
   flags=cv2.CASCADE_SCALE_IMAGE
)

print ("found {0} faces!".format((len(faces))))

for (x,y,w,h) in faces :
        cv2.rectangle(image, (x,y), (x+w, y+h), (0,255,0), 2)"Faces found"
cv2_imshow(image)
```

Output –



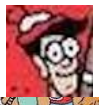
PROJECTS

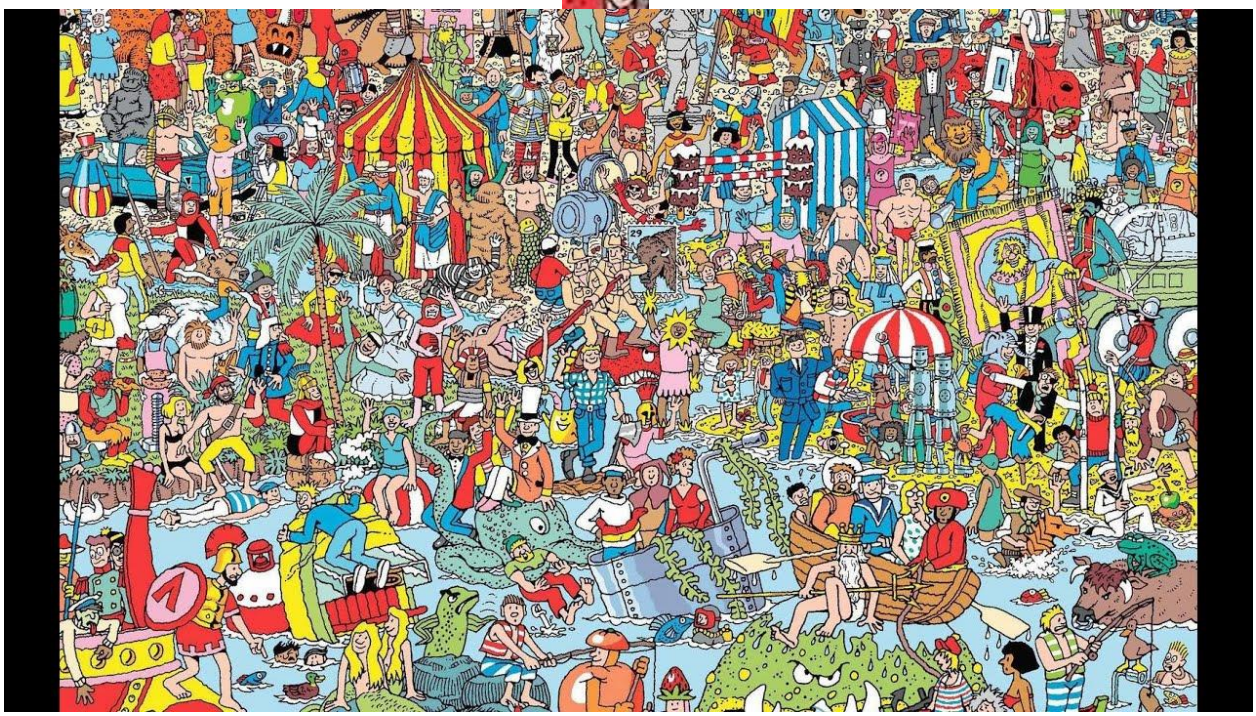
PROJECT 1 - FINDING WALDO FROM THE BEACH.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab.patches import cv2_imshow


img = cv2.imread("beachWaldo.jpg",0)
img2= img.copy()
template = cv2.imread("waldo.png",0)

cv2_imshow(template)

cv2_imshow(img)

w, h = template.shape[::-1]

#All the 6 method for comparision in a list
methods = ['cv2.TM_CCOEFF', 'cv2.TM_CCOEFF_NORMED', 'cv2.TM_CCORR',
'cv2.TM_CCORR_NORMED', 'cv2.TM_SQDIFF', 'cv2.TM_SQDIFF_NORMED']


for meth in methods :
  img = img2.copy()
  method = eval(meth)

  #Apply template matching
  res = cv2.matchTemplate(img, template, method)
  min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

  #If the method is TM_SQDIFF or TM_SQDIFF_NORMED, take minimum
  if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
    top_left = min_loc
  else :
    top_left = max_loc
  bottom_right = (top_left[0] + w, top_left[1] + h)

  cv2.rectangle(img, top_left, bottom_right, 255, 2)

  plt.subplot(121), plt.imshow(res, cmap="gray")
  plt.title("Matching Result"), plt.xticks([]), plt.yticks([])

  #cv2_imshow(res)
  print("--------------------------------------------------------------------------------")
  plt.subplot(122), plt.imshow(img, cmap="gray")
  plt.title("Detected Point"), plt.xticks([]), plt.yticks([])
  cv2_imshow(img)
  plt.suptitle(meth)
  plt.show()
```

- **Outputs**

9) Original Image of Waldo.





10) Detected Waldo Face using various methods.

Method 1 – cv2.TM_CCOEFF

Method 2 – cv2.TM_CCOEFF_NORMED

Method 3 – cv2.TM_CCORR

Method 4 - cv2.TM_CCORR_NORMED

Method 5 - cv2.TM_SQDIFF

Method 6 - cv2.TM_SQDIFF_NORMED

PROJECT 2 – CAR AND PEDESTERIAN DETECTION FROM VIDEO.

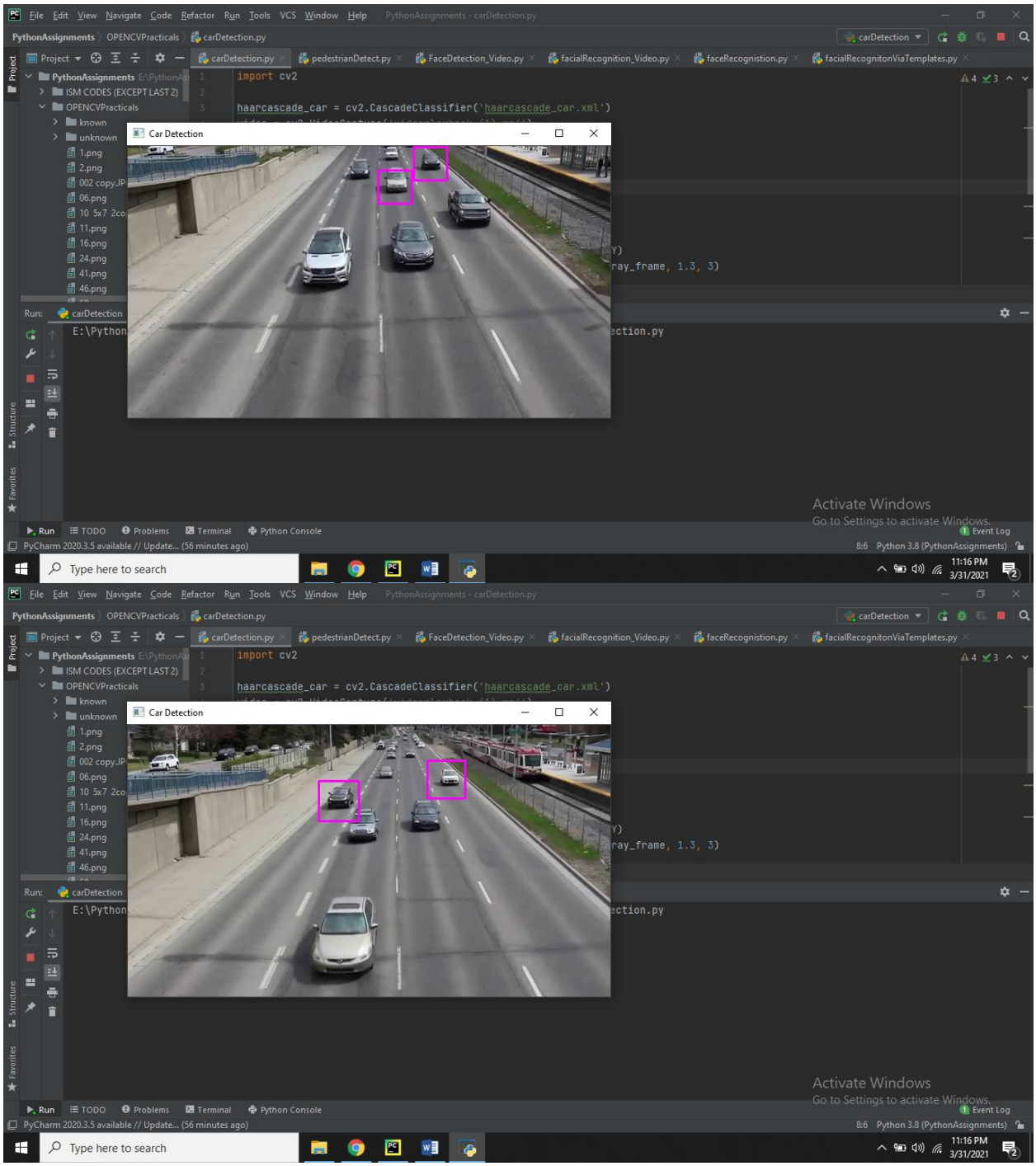
2a) Car Detection

```
import cv2

haarcascade_car = cv2.CascadeClassifier('haarcascade_car.xml')
video = cv2.VideoCapture('videoplayback (1).mp4')

if video.isOpened() == False:
   print("Error opening video")
else:
   while video.isOpened():
      ret, frame = video.read()
      if ret:
         gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
         detected_cars = haarcascade_car.detectMultiScale(gray_frame, 1.3, 3)
         for (x, y, width, height) in detected_cars:
            cv2.rectangle(frame, (x, y), (x+width, y+height), (255, 0, 255), 2)
            cv2.imshow("Car Detection",frame)
         if cv2.waitKey(20) & 0xFF == ord('q'):
            break
      else:
         break
```

- **Outputs**

1. Some frames of original video.

2. Program detecting cars in rectangle.



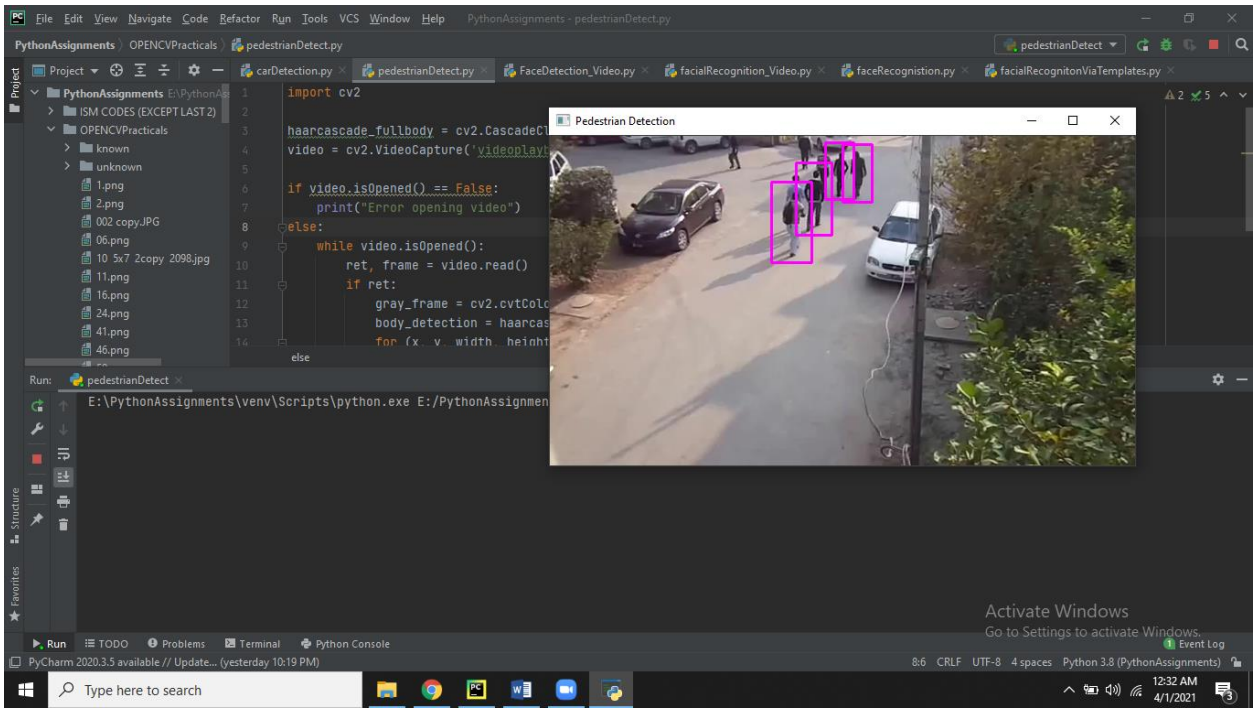
2b) Pedestrian Detection

```
import cv2

haarcascade_fullbody = cv2.CascadeClassifier('haarcascade_fullbody.xml')
video = cv2.VideoCapture('videoplayback (3).mp4')

if video.isOpened() == False:
   print("Error opening video")
else:
   while video.isOpened():
      ret, frame = video.read()
      if ret:
         gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
         body_detection = haarcascade_fullbody.detectMultiScale(gray_frame, 1.2, 3)
         for (x, y, width, height) in body_detection:
            cv2.rectangle(frame, (x, y), (x+width, y+height), (255, 0, 255), 2)
            cv2.imshow('Pedestrian Detection', frame)
         if cv2.waitKey(20) & 0xFF == ord('q'):
            break
      else:
         break
```

- **Outputs**

1) Some frames of original video.

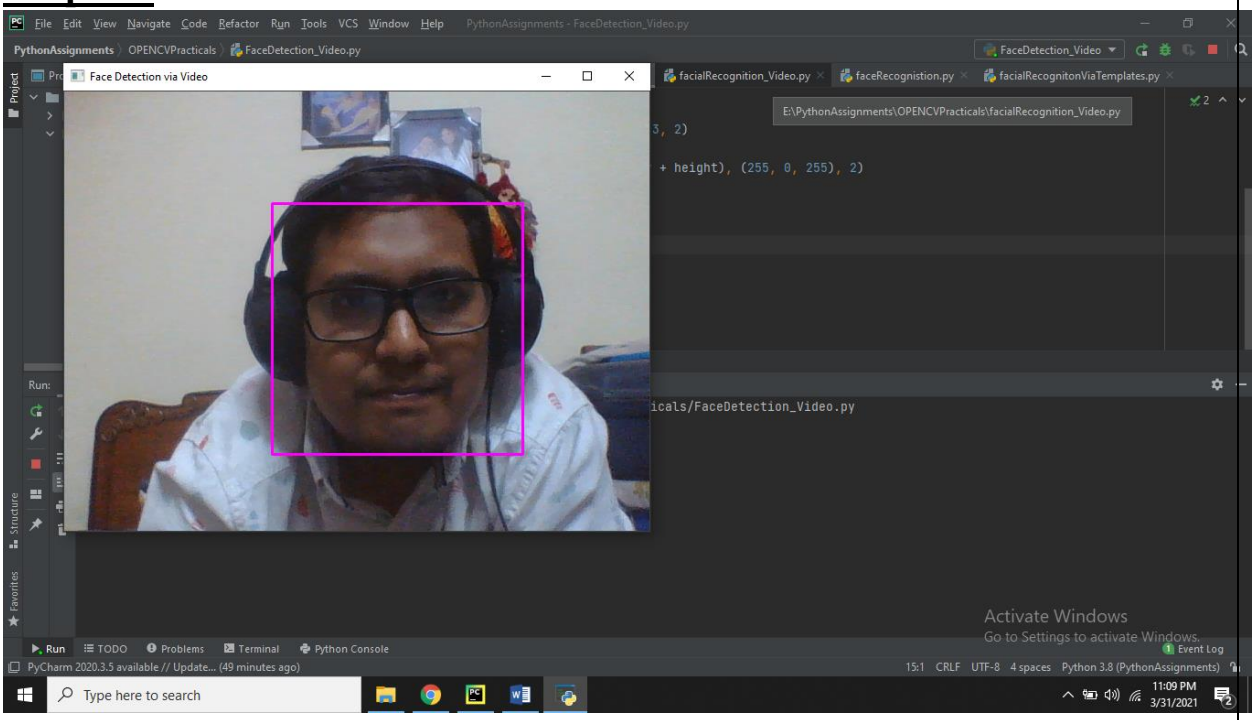
2) Program detecting pedestrian in rectangle.



PROJECT 3A) – FACE DETECTION FROM VIDEO.

```
import cv2

face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
video = cv2.VideoCapture(0)

while True:
    ret, frame = video.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 2)
    for (x, y, width, height) in faces:
        cv2.rectangle(frame, (x, y), (x + width, y + height), (255, 0, 255), 2)
    cv2.imshow('Face Detection via Video', frame)
    if cv2.waitKey(30) & 0xFF == ord('q'):
        break
```

- **Output :-**



PROJECT 3B) – FACE RECOGNITION FROM IMAGE AND VIDEO.

3b 1) Face Recognition from image.

```
import face_recognition
from PIL import Image, ImageDraw
import numpy as np
from google.colab.patches import cv2_imshow
import cv2

# Load a sample picture and learn how to recognize it.
u_image = face_recognition.load_image_file("upendra-min.JPG")
u_face_encoding = face_recognition.face_encodings(u_image)[0]

# Load a second sample picture and learn how to recognize it.
m_image = face_recognition.load_image_file("mrutunjay-min.JPG")
m_face_encoding = face_recognition.face_encodings(m_image)[0]

# Create arrays of known face encodings and their names
known_face_encodings = [
    u_face_encoding,
    m_face_encoding
]
known_face_names = [
    "Upendra",
    "Mrutunjay"
]

# Load an image with an unknown face
unknown_image = face_recognition.load_image_file("combined-min.JPG")

# Find all the faces and face encodings in the unknown image
face_locations = face_recognition.face_locations(unknown_image)
face_encodings = face_recognition.face_encodings(unknown_image, face_locations)


pil_image = Image.fromarray(unknown_image)
draw = ImageDraw.Draw(pil_image)

# Loop through each face found in the unknown image
for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
    # See if the face is a match for the known face(s)
    matches = face_recognition.compare_faces(known_face_encodings, face_encoding)

    name = "Unknown"

    if True in matches:
        first_match_index = matches.index(True)
        name = known_face_names[first_match_index]

    # Draw a box around the face using the Pillow module
    draw.rectangle(((left, top), (right, bottom)), outline=(48, 63, 159))

    # Draw a label with a name below the face
    text_width, text_height = draw.textsize(name)
    draw.rectangle(((left, bottom - text_height - 10), (right, bottom)), fill=(48, 63, 159),
outline=(48, 63, 159))
    draw.text((left + 6, bottom - text_height - 5), name, fill=(255, 255, 255, 0))
```

```
# Remove the drawing library from memory as per the Pillow docs
del draw

pil_image.show()

# You can also save a copy of the new image to disk if you want by uncommenting this line
pil_image.save("image_with_boxes.jpg")
show = cv2.imread("image_with_boxes.jpg")
cv2_imshow(show)
```

- **Outputs**

1) Original Image used.

Mrutunjay	
Upendra	

Combined Photo -

2) After Running the program and getting output –

3) Zoomed output of above.


Upendra

PROJECT 3B) – FACE RECOGNITION FROM IMAGE AND VIDEO.

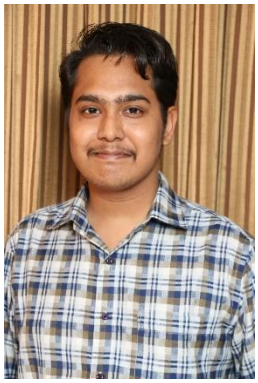
3b 1) Face Recognition from Video.

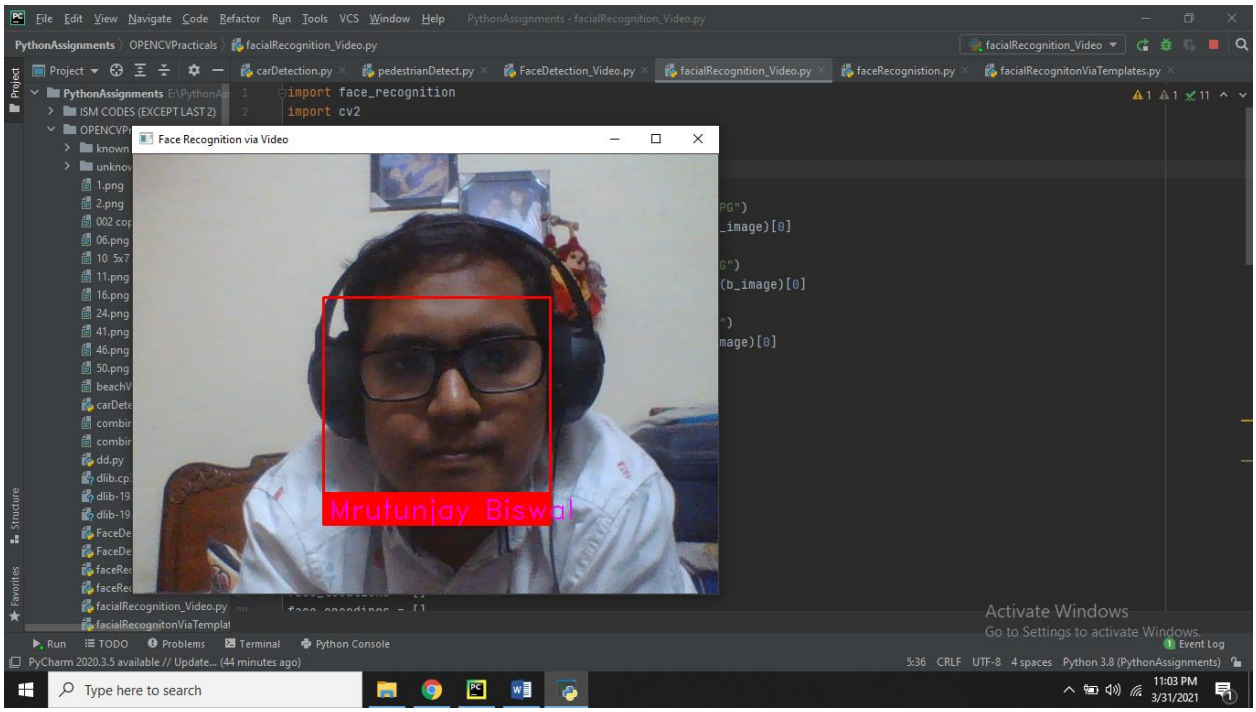
```
import face_recognition
import cv2
import numpy as np

video_capture = cv2.VideoCapture(0)

m_image = face_recognition.load_image_file("mrutunjay-min.JPG")
mrutunjay_face_encoding = face_recognition.face_encodings(m_image)[0]

b_image = face_recognition.load_image_file("IMG_1396-min.JPG")
bijayalaxmi_face_encoding = face_recognition.face_encodings(b_image)[0]

u_image = face_recognition.load_image_file("upendra-min.JPG")
upendra_face_encoding = face_recognition.face_encodings(u_image)[0]

known_face_encodings = [
    mrutunjay_face_encoding,
    bijayalaxmi_face_encoding,
    upendra_face_encoding
]
known_face_names = [
    "Mrutunjay Biswal",
    "Bijayalaxmi Biswal",
    "Upendra Biswal"
]

face_locations = []
face_encodings = []
face_names = []
process_this_frame = True

while True:
    ret, frame = video_capture.read()

    small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

    # Convert the image from BGR color (which OpenCV uses) to RGB color (which
face_recognition uses)
    rgb_small_frame = small_frame[:, :, ::-1]

    # Only process every other frame of video to save time
    if process_this_frame:
        # Find all the faces and face encodings in the current frame of video
        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)

        face_names = []
        for face_encoding in face_encodings:
            # See if the face is a match for the known face(s)
            matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
            name = "Unknown"
```

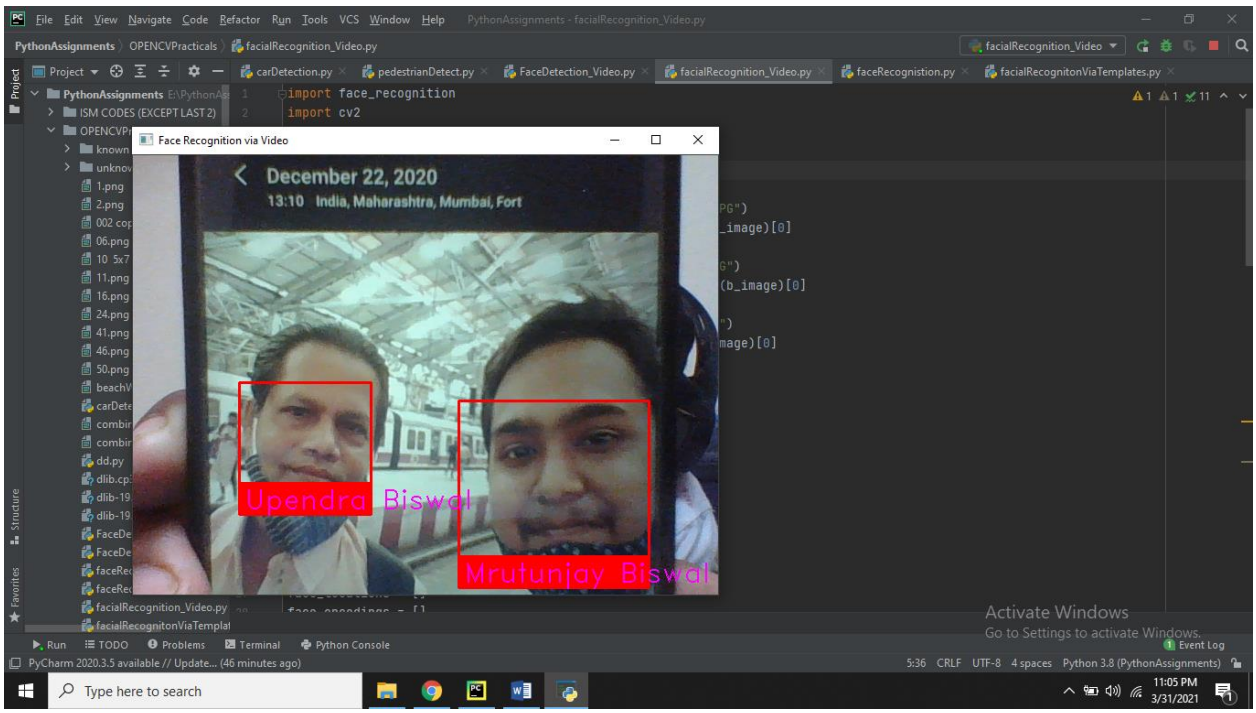
```
            face_distances = face_recognition.face_distance(known_face_encodings,
face_encoding)
            best_match_index = np.argmin(face_distances)
            if matches[best_match_index]:
                name = known_face_names[best_match_index]

            face_names.append(name)

    process_this_frame = not process_this_frame


    # Display the results
    for (top, right, bottom, left), name in zip(face_locations, face_names):
        # Scale back up face locations since the frame we detected in was scaled to 1/4 size
        top *= 4
        right *= 4
        bottom *= 4
        left *= 4

        # Draw a box around the face
        cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

        # Draw a label with a name below the face
        cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
        font = cv2.FONT_HERSHEY_DUPLEX
        cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 0, 255), 1)

    cv2.imshow('Face Recognition via Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

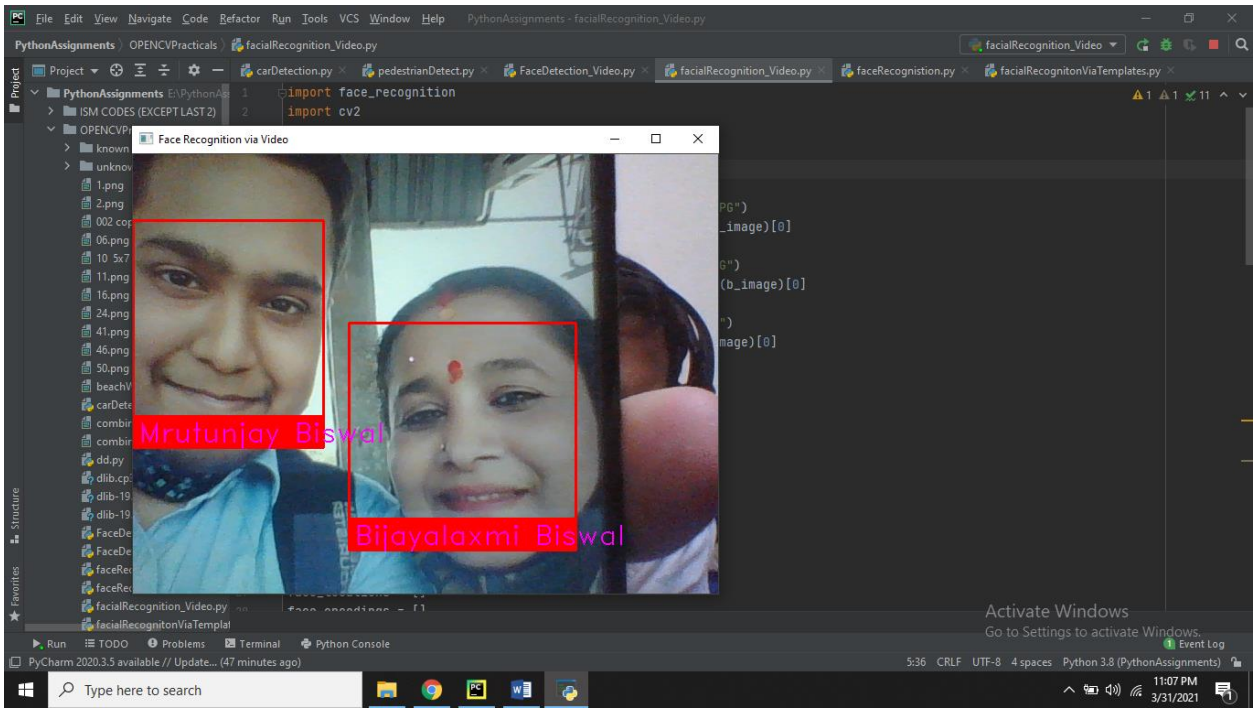
- **Outputs**

1. Original Image used.

Mrutunjay Biswal	
Upendra Biswal	
Bijayalaxmi Biswal	


Face Recognition via Video
Mrutunjay Biswal
Activate Windows
Go to Settings to activate Windows.
11:03 PM
3/31/2021


Face Recognition via Video
December 22, 2020
13:10 India, Maharashtra, Mumbai, Fort
Upendra Biswal
Mrutunjay Biswal
Activate Windows
Go to Settings to activate Windows.
11:05 PM
3/31/2021


Face Recognition via Video
Mrutunjay Biswal
Bijayalaxmi Biswal
import face_recognition
import cv2
Activate Windows
Go to Settings to activate Windows.