Image Processing using PYTHON

Deepayan Chakraborty

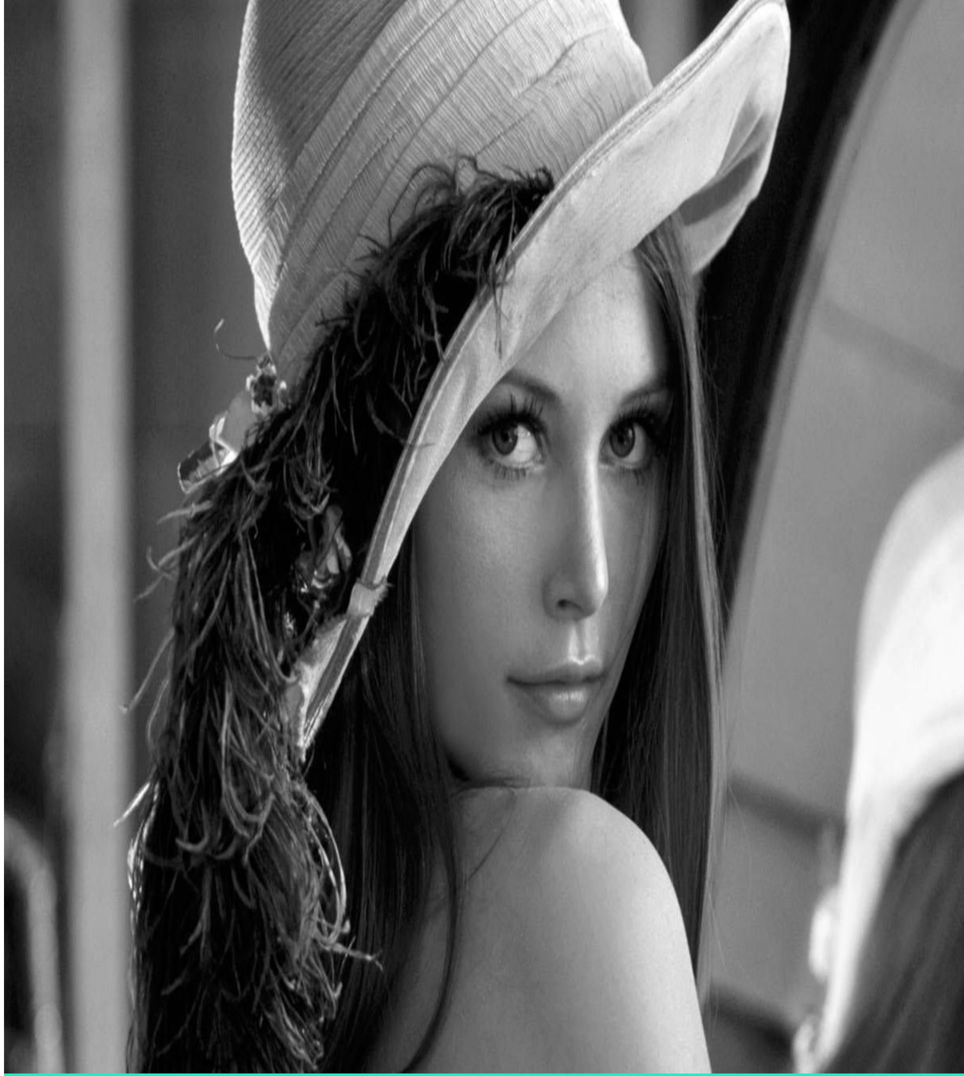
Note

- Basic Image Processing Methods.
- Using OpenCV, PIL, Skimage
- A touch of image recognition using Deep Learning
- Lena Image will be used.

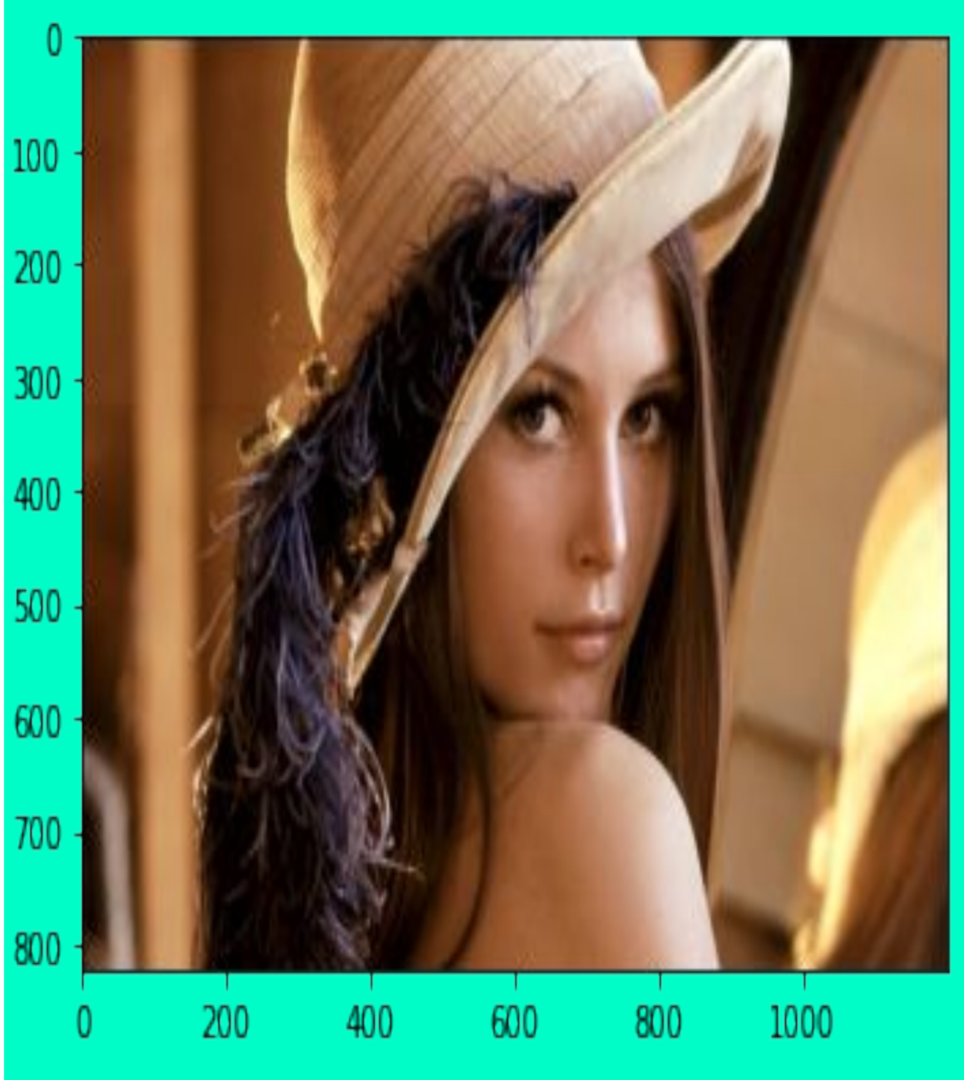
Read and Display Image

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
# Load an color image in Colour
img = cv2.imread('Lena.jpg',cv2.IMREAD_COLOR)
#img = cv2.imread('Lena.jpg',1)
# Load an color image in grayscale
img = cv2.imread('Lena.jpg',cv2.IMREAD_GRAYSCALE)
#img = cv2.imread('Lena.jpg',0)
# Load an color image as it is
img =
cv2.imread('Lena.jpg',cv2.IMREAD_UNCHANGED)#Reads in
BGR format
#img = cv2.imread('Lena.jpg',-1)
cv2_imshow(img)#Converts BGR to RGB then
display #For Colab only
#For Jupyter
# cv2.imshow('image',img)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```

```
#Import required library
from PIL import Image

#Open Image
#im = Image.open("Lena.jpg")
im=Image.open('Lena.jpg').convert('L')
im
#im.show()
#im.save('Lena.png')
# im.size
# im.format
```



```python
import matplotlib.pyplot as plt
img1=plt.imread('Lena.jpg')
# The image data. The returned array has shape

#      - (M, N) for grayscale images.
#      - (M, N, 3) for RGB images.
#      - (M, N, 4) for RGBA images.
#img1.shape
plt.imshow(img1)#cmap can be specified for grayscale image
```



Converting to grayscale Image

After Loading the image in colour format.... .

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

```
gray_1 = im.convert('L')
gray_1
```

```
from skimage.color import rgb2gray
gray_2=rgb2gray(img1)
plt.imshow(gray_2,cmap=plt.cm.gray)
```

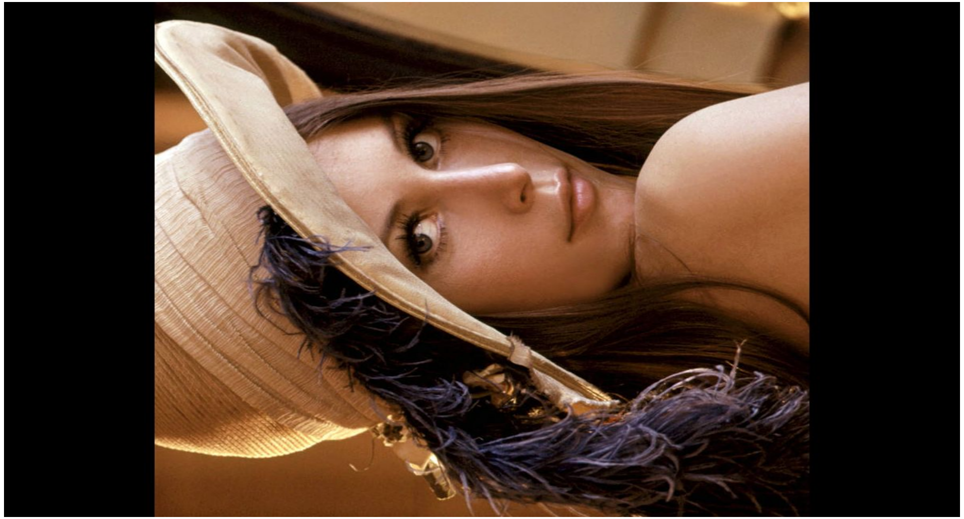
Geometric Transformation

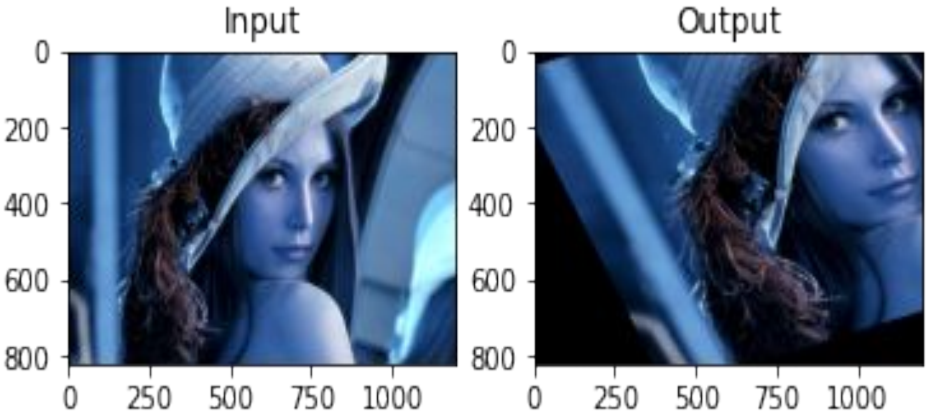
- Using Scaling, Translation, Rotation, Affine Transformation.
- Described For three methods separately.

Using OPENCV

```
#Scaling
height, width = img.shape[:2]
res = cv2.resize(img,(2*width, 2*height),
interpolation = cv2.INTER_CUBIC)#Preferable
interpolation methods are cv2.INTER_AREA for
shrinking and cv2.INTER_CUBIC (slow) &
cv2.INTER_LINEAR for zooming. By default,
interpolation method used is cv2.INTER_LINEAR
for all resizing purposes.
cv2_imshow(res)
```

```
#Translation
rows,cols = img.shape[:2]
M = np.float32([[1,0,100],[0,1,50]])
 #If you know the shift in (x,y) direction, let it be
(t_x,t_y), you can create the transformation matrix
M.
dst = cv2.warpAffine(img,M,(cols,rows))
#Third argument of the cv2.warpAffine() function is
the size of the output image, which should be in the
form of (width, height). Remember width = number of
columns, and height = number of rows.
cv2_imshow(dst)
```





```
#Rotation
rows,cols = img.shape[:2]

M =
cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
dst = cv2.warpAffine(img,M,(cols,rows))
cv2_imshow(dst)

#Affine Transformation
rows,cols,ch = img.shape

pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])

M = cv2.getAffineTransform(pts1,pts2)

dst = cv2.warpAffine(img,M,(cols,rows))

plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
```

Using PIL

```
#Scaling using PIL
width, height = im.size[:2]
newsize = (round(.5*height), round(.5*width))
im1 = im.resize(newsize)
# Shows the image in image viewer
im1

#Rotate using PIL
im.rotate(45)#for local useim.rotate(45).show

#Translation and Affine Transformation
using PIL
#For each pixel (x, y), the output will be calculate
as (ax+by+c, dx+ey+f)
# for translation, you only have to look at the c an
f values of your matrix
a = 1
b = 0
c = -10 #left/right (i.e. 5/-5)
d = 0
e = 1
f = -10 #up/down (i.e. 5/-5)
img = im.transform(im.size[:2], Image.AFFINE, (a, b
```

Using SKimage

```
# Scaling using Skimage
import matplotlib.pyplot as plt

from skimage import data, color
from skimage.transform import resize

image = img1
image_resized = resize(image, (image.shape[0]/2 // 4,
image.shape[1] // 4,image.shape[2]))

plt.imshow(image_resized)
```

Assignment!!!!!

**Apply Skimage for other types of transformations

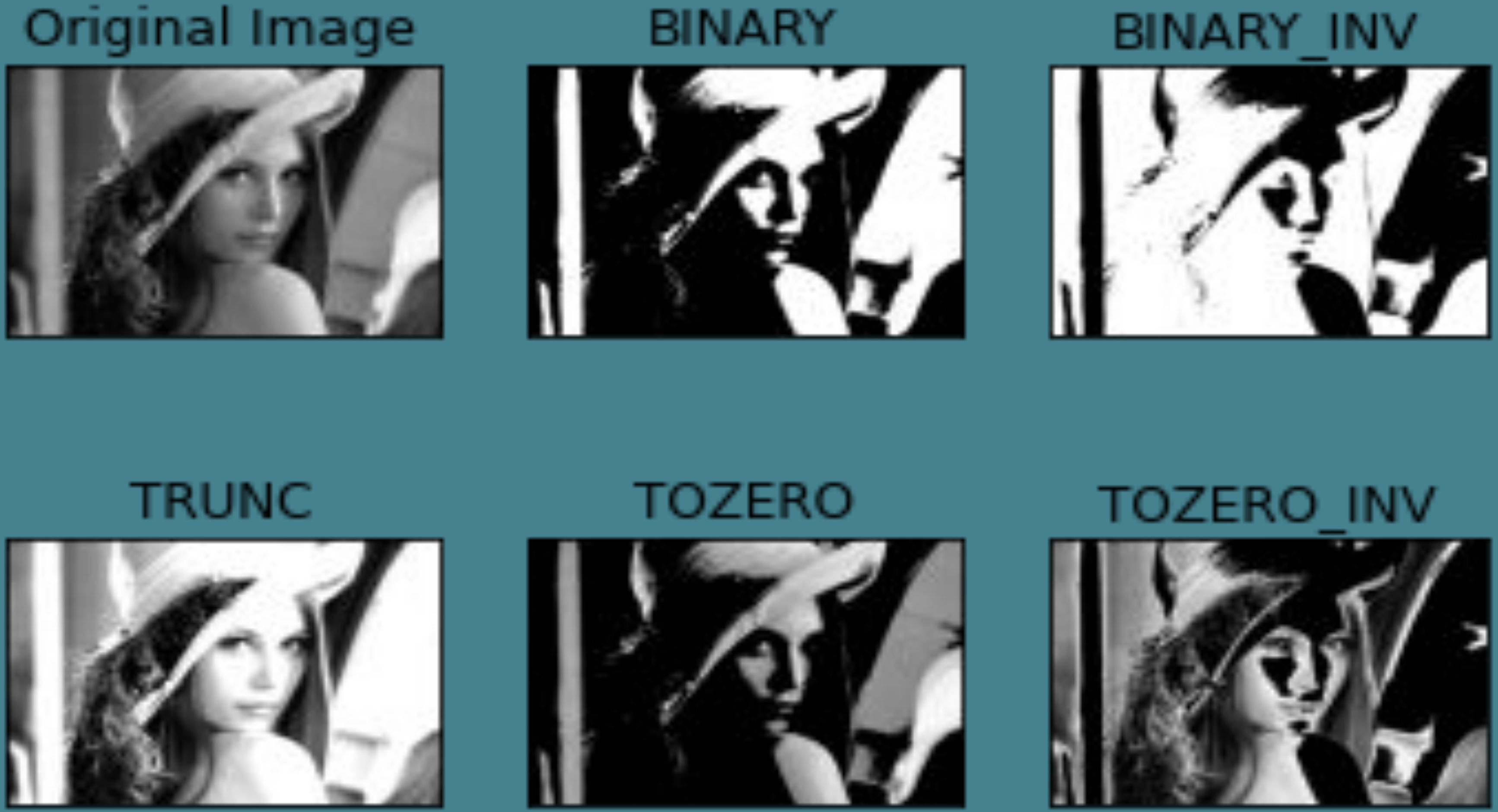
Binarization of Image

Grayscale to binary…. .

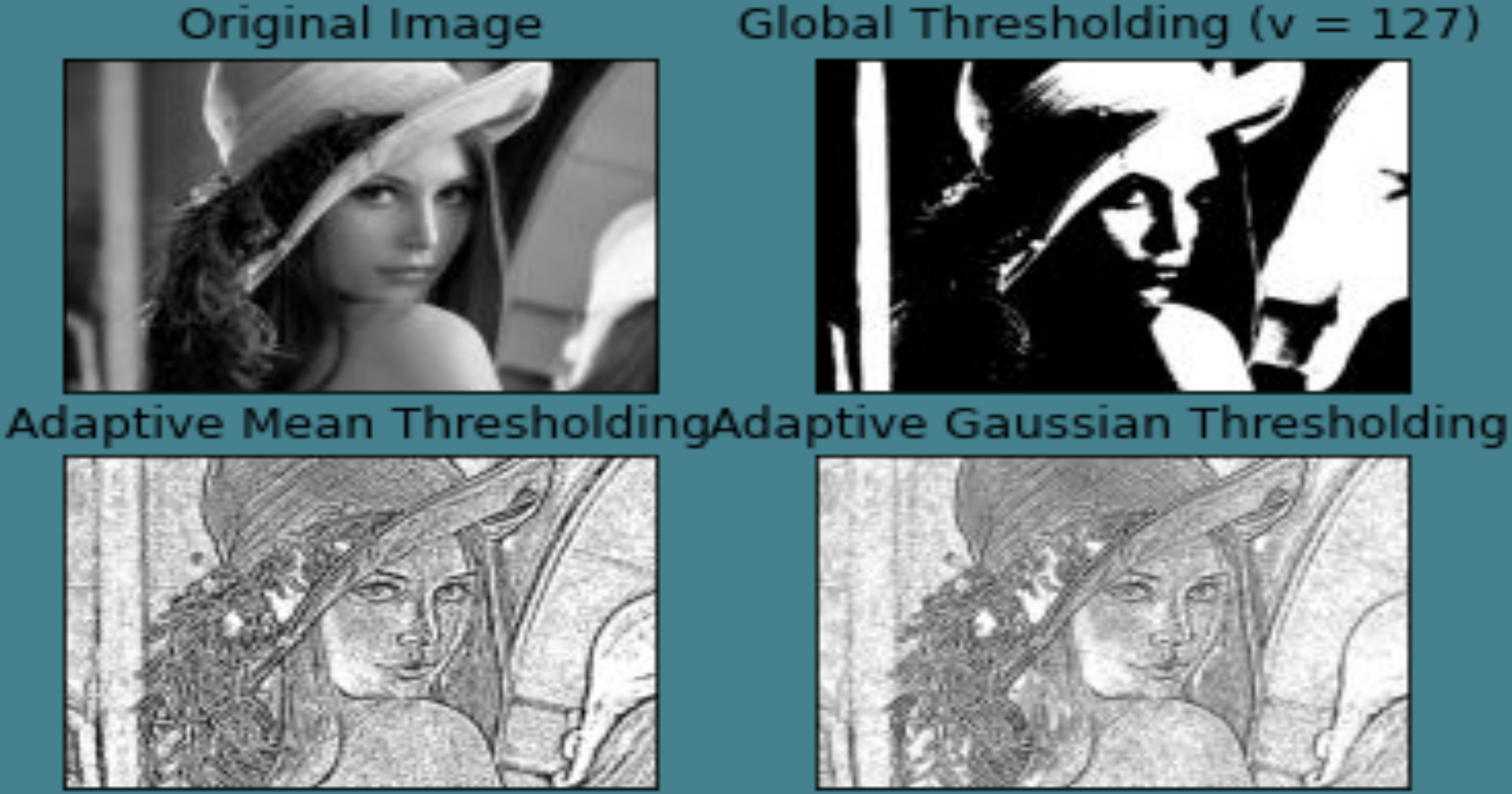
Thresholding

Simple, Adaptive, Otsu…. .

```
#Simple Thresholding
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = gray
ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH_BINARY_INV)
ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','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([])#current tick locations and labels of the x-axis.

plt.show()
```


Original Image
BINARY
BINARY_INV
TRUNC
TOZERO
TOZERO_INV

```
#Adaptive Thresholding
import cv2
import numpy as np
from matplotlib import pyplot as plt


ret,th1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
            cv2.THRESH_BINARY,11,2)#threshold value is the mean of neighbourhood area.
th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
            cv2.THRESH_BINARY,11,2)# threshold value is the weighted sum of neighbourhood
values where weights are a gaussian window.

titles = ['Original Image', 'Global Thresholding (v = 127)',
            'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]
for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

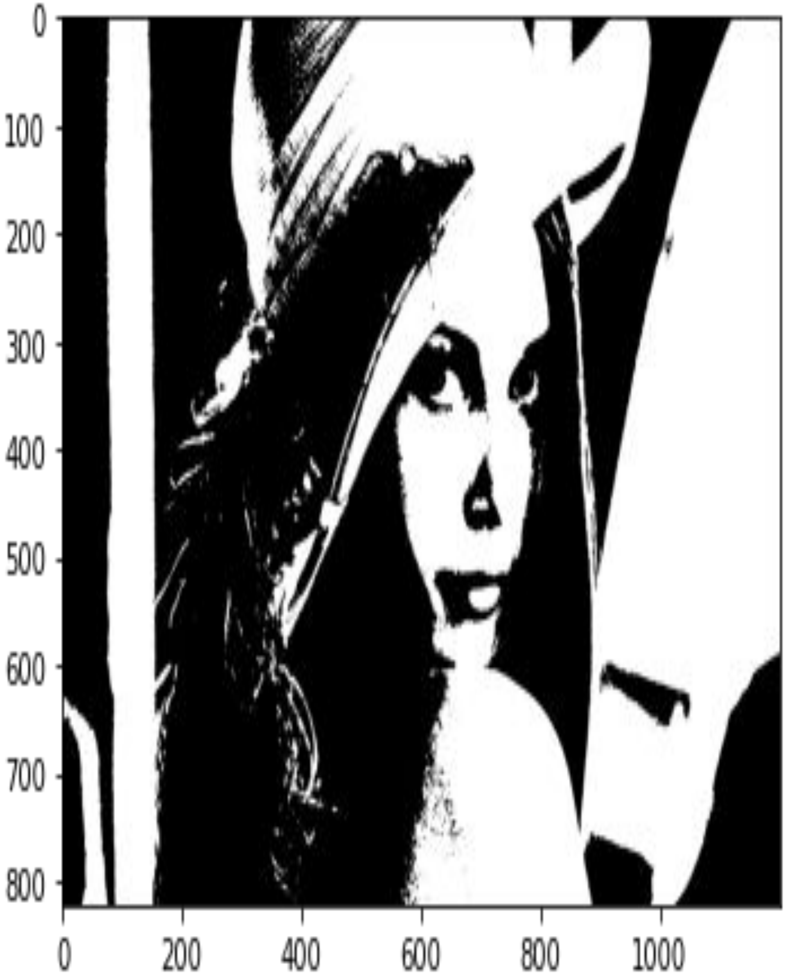


Original Image

Global Thresholding (v = 127)

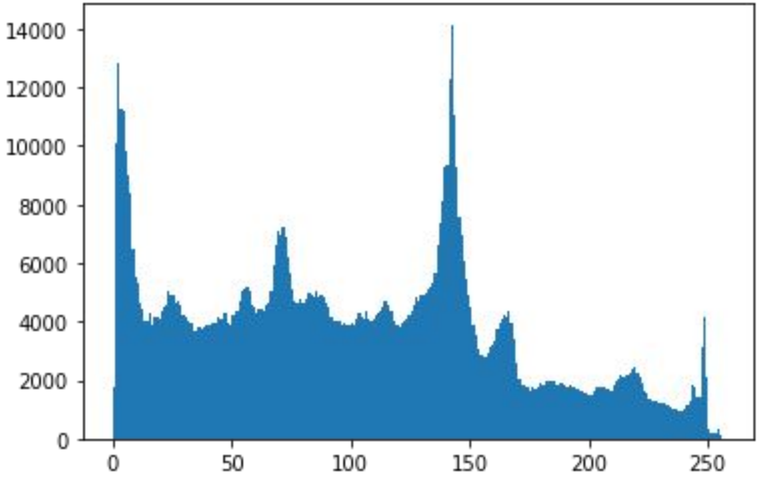
Adaptive Mean Thresholding

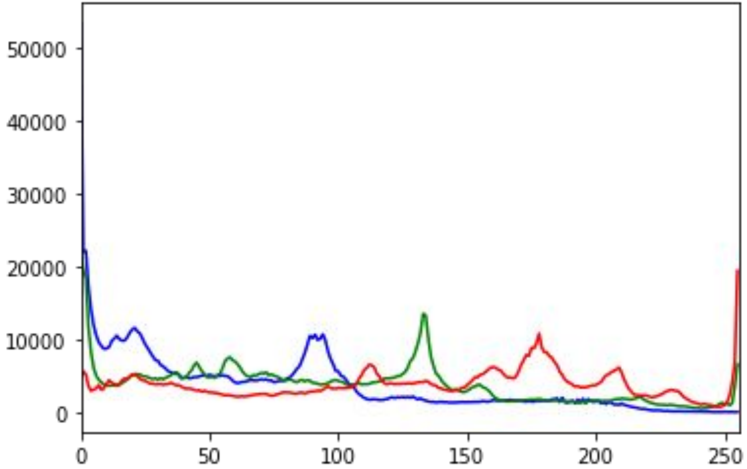
Adaptive Gaussian Thresholding



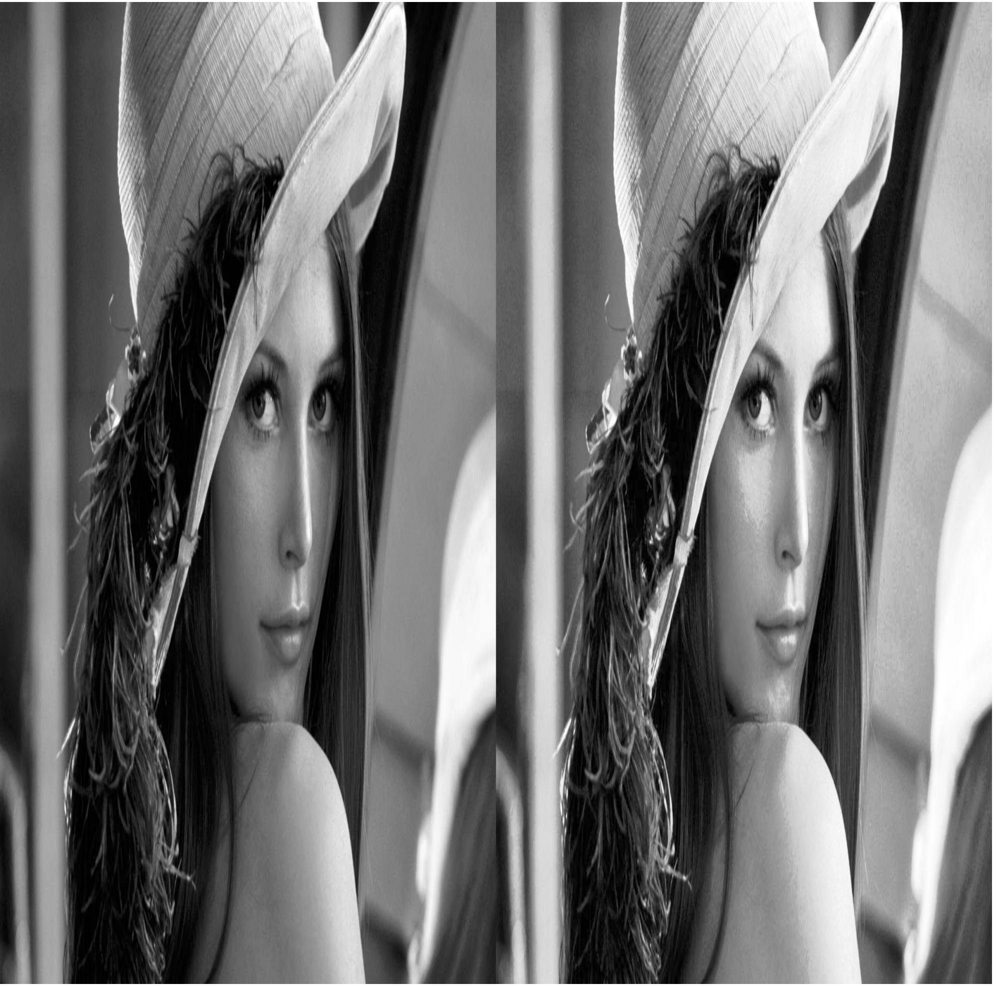
```
#Otsu's Thresholding
ret2,th2 =
cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plt.imshow(th2,cmap='gray')
```

Histogram

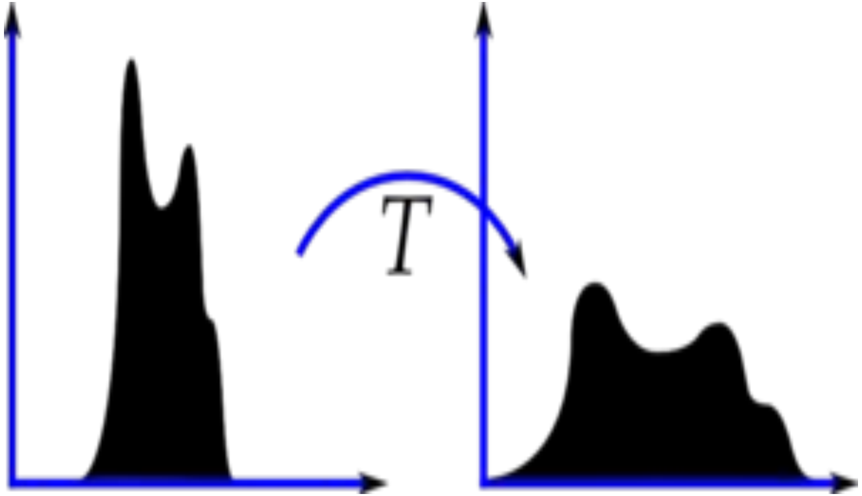




```
#For Grayscale image
import cv2
import numpy as np
from matplotlib import pyplot as plt

plt.hist(gray.ravel(),256,[0,256]); plt.show()

#For Colour image
img = cv2.imread('Lena.jpg')
color = ('b','g','r')
for i,col in enumerate(color):
    histr =
cv2.calcHist([img],[i],None,[256],[0,256])#calcHist
images, channels, mask, histSize, ranges[,
hist[, accumulate]])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

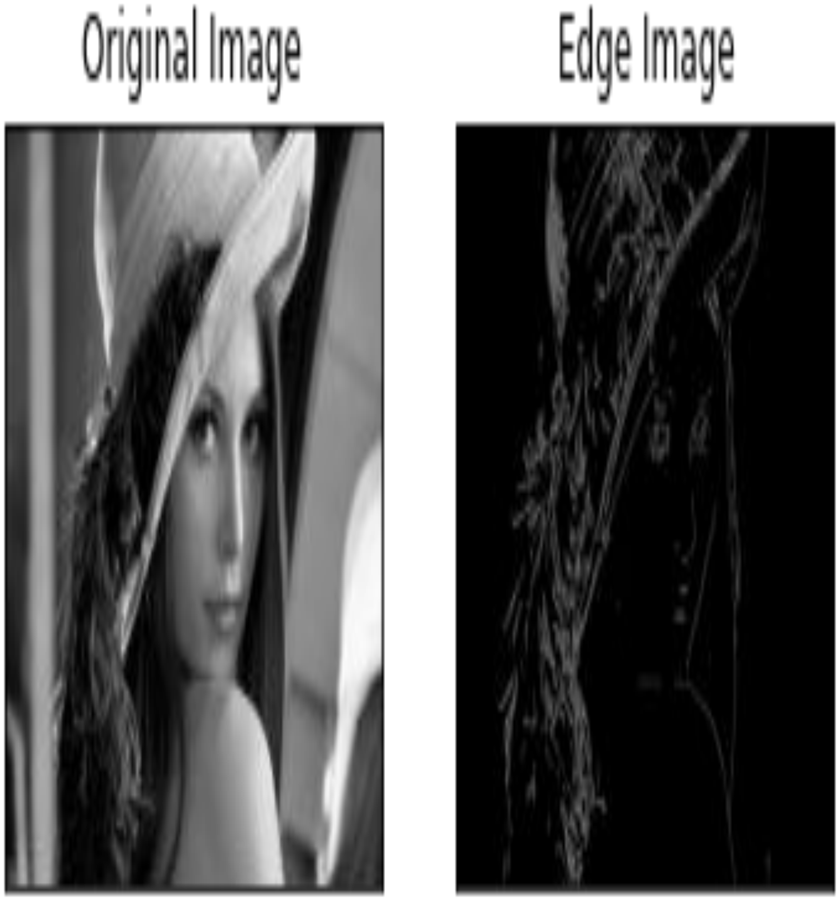


```
#Improve contrasts using Histogram equalization

img = cv2.imread('Lena.jpg',0)
equ = cv2.equalizeHist(img)
res = np.hstack((img,equ)) #stacking images side-by-side.
cv2_imshow(res)
```



Edge Detection

Using Canny Method…. .



```
img = cv2.imread('Lena.jpg',0)
edges = cv2.Canny(img,100,200)

plt.subplot(121),plt.imshow(img,cmap =
'gray')#subplot(nrows, ncols, index, **kwargs)

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()
```

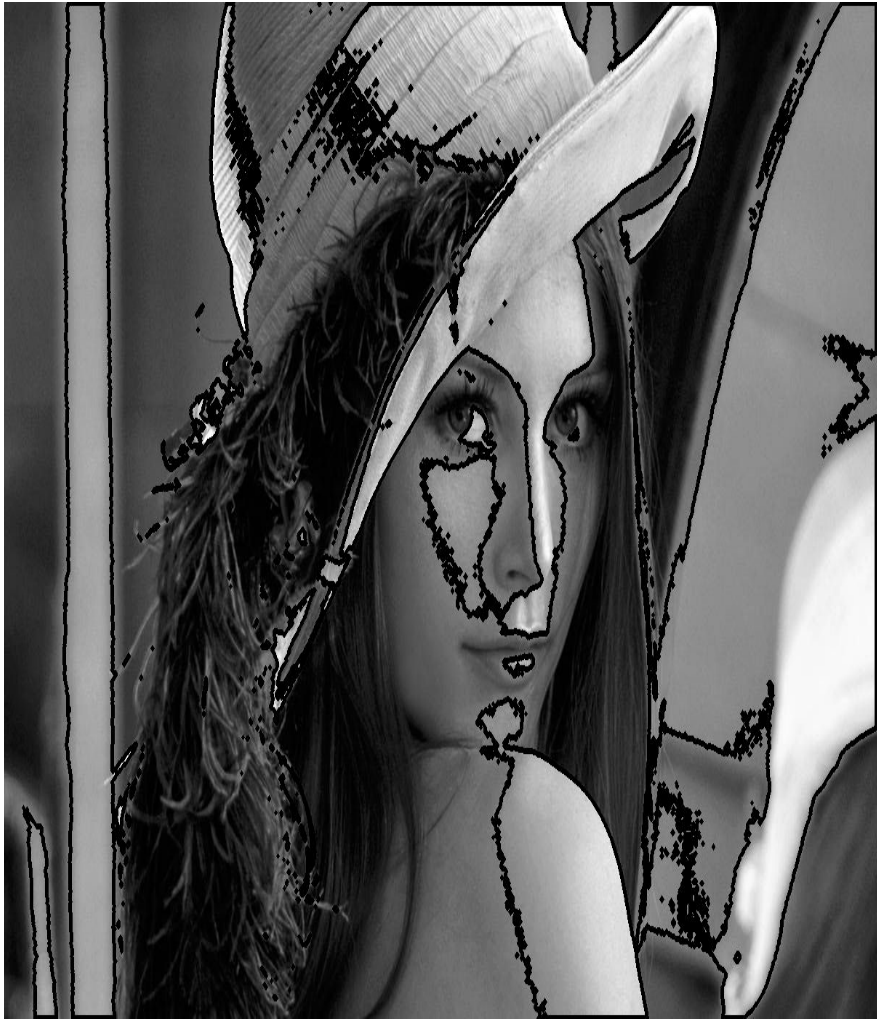
Assignment!!!!!

**Use Sobel Method For Edge Detection

Contour Detection

PS: Contour is the collection of lines joining continuous Pixels with same

intensity.... .



```
import numpy as np
import cv2

im = cv2.imread('Lena.jpg')
imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(imgray,127,255,0)
contours, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_NONE)
#Contour approx method: If we pass
cv2.CHAIN_APPROX_NONE, all the boundary points are
stored
#cv2.RETR_TREE tells OpenCV to compute the
hierarchy (relationship) between contours
img = cv2.drawContours(img, contours, -1, (0,255,0), 3)
cv2_imshow(img)
```

Assignment!!!!!

**Explore Other methods of Contour Detection

Image Recognition (Using Deep Learning)

Described in Colab

Thank You !