NATIONAL INSTITUTE OF TECHNOLOGY WARANGAL

Warangal– 506 004, Telangana State, India

Department of Computer Science and Engineering

M.Tech., I Semester

Computer Vision and Image Processing

Assignment-1

Roll Number:	***CS21213***
Name:	***Ankur Tiwari***
Date of Submission:	***17-10-2021***

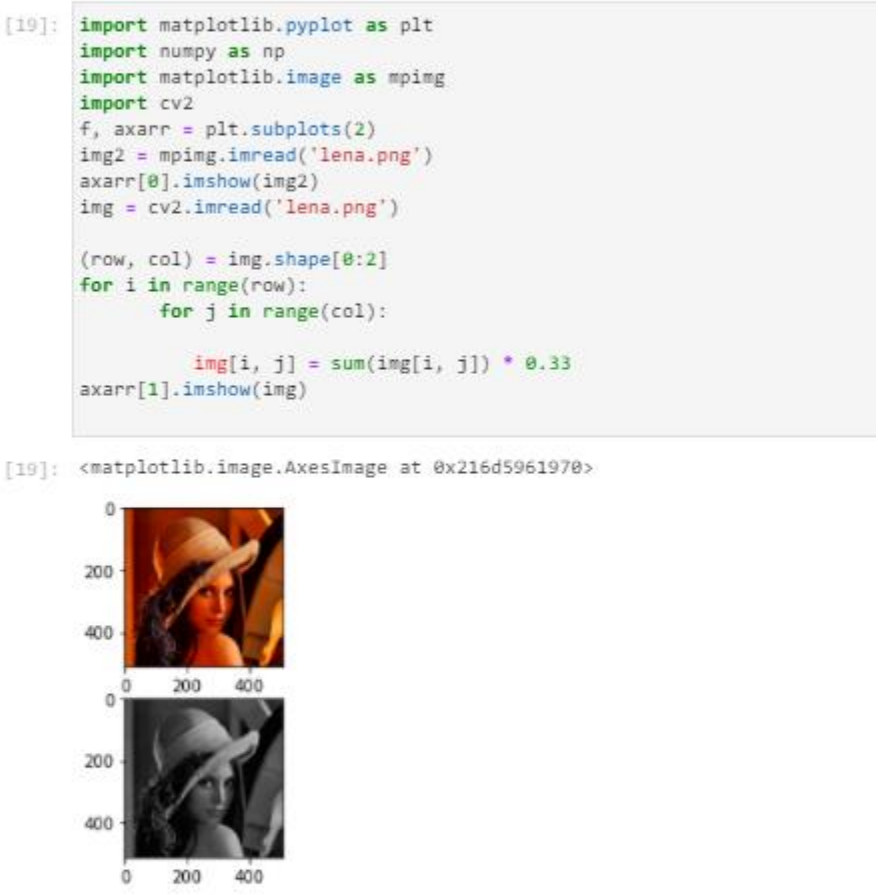
INDEX

TASK-1 : CONVERT COLOR IMAGE TO GRAY IMAGE AND DISPLAY BOTH THEM USING SUBPLOTS

CODES:

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.image as mpimg
import cv2
f, axarr = plt.subplots(2)
img2 = mpimg.imread('lena.png')
axarr[0].imshow(img2)
img = cv2.imread('lena.png')
(row, col) = img.shape[0:2]
for i in range(row):
      for j in range(col):
  img[i, j] = sum(img[i, j]) * 0.33  #grayimage
axarr[1].imshow(img)
```

SCREENSHOT JUPYTER NOTEBOOK :



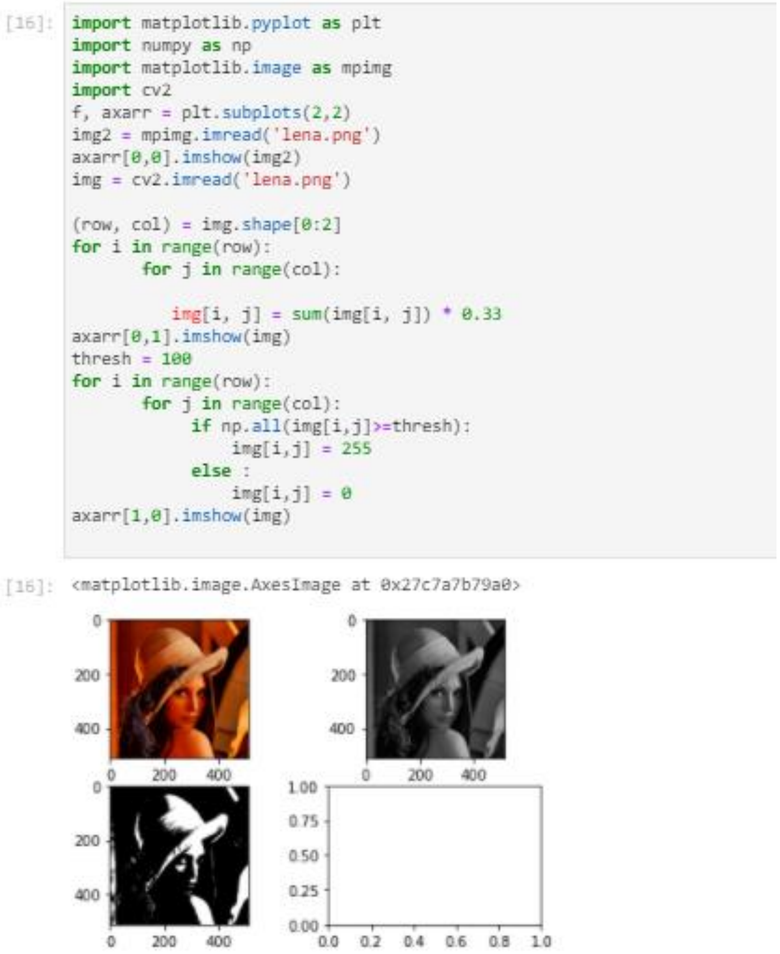
```
[19]: import matplotlib.pyplot as plt
      import numpy as np
      import matplotlib.image as mpimg
      import cv2
      f, axarr = plt.subplots(2)
      img2 = mpimg.imread('lena.png')
      axarr[0].imshow(img2)
      img = cv2.imread('lena.png')

      (row, col) = img.shape[0:2]
      for i in range(row):
            for j in range(col):

                img[i, j] = sum(img[i, j]) * 0.33
      axarr[1].imshow(img)

[19]: <matplotlib.image.AxesImage at 0x216d5961970>
```

TASK-2 :CONVERT TASK-1 RESULT INTO BINARY IMAGE AND DISPLAY ALL THREE IMAGE USING SUBPLOTS

CODE :

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.image as mpimg
import cv2

f, axarr = plt.subplots(2,2)
img2 = mpimg.imread('lena.png')
axarr[0,0].imshow(img2)
img = cv2.imread('lena.png')
(row, col) = img.shape[0:2]
for i in range(row):
    for j in range(col):
      img[i, j] = sum(img[i, j]) * 0.33

axarr[0,1].imshow(img)
thresh = 100         #threshhold value
for i in range(row):
    for j in range(col):
        if np.all(img[i,j]>=thresh):
          img[i,j] = 255
        else :
          img[i,j] = 0
axarr[1,0].imshow(img)
```

SCREENSHOT JUPYTER NOTEBOOK:



```
[16]: import matplotlib.pyplot as plt
      import numpy as np
      import matplotlib.image as mpimg
      import cv2
      f, axarr = plt.subplots(2,2)
      img2 = mpimg.imread('lena.png')
      axarr[0,0].imshow(img2)
      img = cv2.imread('lena.png')

      (row, col) = img.shape[0:2]
      for i in range(row):
             for j in range(col):

                img[i, j] = sum(img[i, j]) * 0.33
      axarr[0,1].imshow(img)
      thresh = 100
      for i in range(row):
             for j in range(col):
                  if np.all(img[i,j]>=thresh):
                      img[i,j] = 255
                  else :
                      img[i,j] = 0
      axarr[1,0].imshow(img)
```

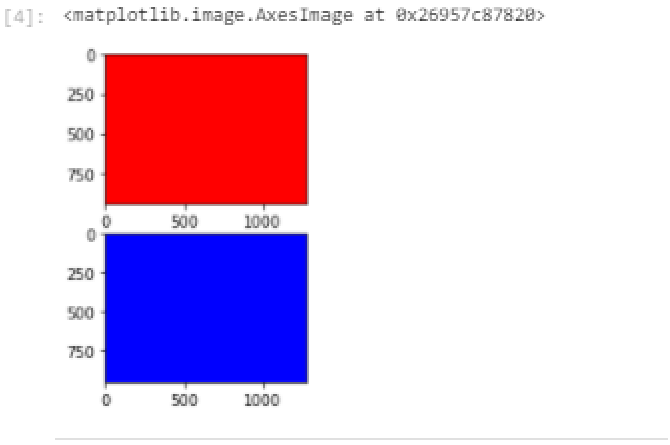
```
[16]: <matplotlib.image.AxesImage at 0x27c7a7b79a0>
```

TASK-3 : CHANGE THE GIVEN PURE RECTANGLE TO PURE BLUE RECTANGLE AND DISPLAY BOTH THE ORIGANAL AND PROCESSED IMAGE USING SUBPLOT

CODE:

```
        import matplotlib.pyplot as plt
        import numpy as np
        import matplotlib.image as mpimg
        import cv2
        f, axarr = plt.subplots(2)
        img2 = mpimg.imread('red.png')
        axarr[0].imshow(img2)
        img = cv2.imread('red.png')
        #blueimage
        (row, col) = img.shape[0:2]
        for i in range(row):
            for j in range(col):
                img[i,j] = (0,0,255)      #increase blue values

axarr[1].imshow(img)
```

SCREENSHOT JUPYTER NOTEBOOK:



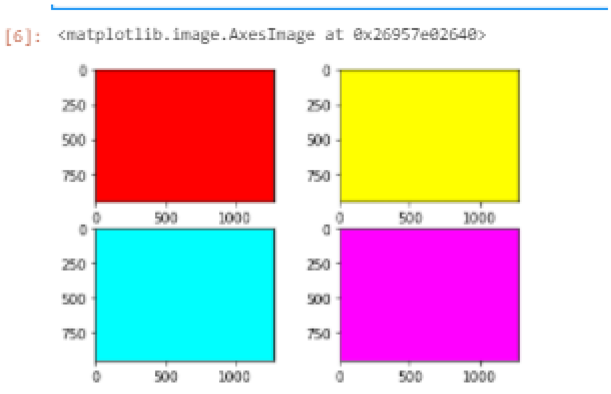
```
[4]: <matplotlib.image.AxesImage at 0x26957c87820>
```

TASK-3 : ***CHANGE THE GIVEN PURE RECTANGLE TO PURE YELLOW, CYAN, AND MAGENTA RECTANGLE AND DISPLAY ALL OF FOUR THE ORIGANAL AND PROCESSED IMAGE USING SUBPLOT***

CODE :

```
    import matplotlib.pyplot as plt
    import numpy as np
    import matplotlib.image as mpimg
    import cv2
    f, axarr = plt.subplots(2,2)
    img2 = mpimg.imread('red.png')
    axarr[0,0].imshow(img2)
    img = cv2.imread('red.png')
    #yellow
    (row, col) = img.shape[0:2]
    for i in range(row):
        for j in range(col):
            img[i,j] = (255,255,0)
      #cyan
    axarr[0,1].imshow(img)
    for i in range(row):
        for j in range(col):
            img[i,j] = (0,255,255)
      #magenta
    axarr[1,0].imshow(img)
    for i in range(row):
        for j in range(col):
            img[i,j] = (255,0,255)
   axarr[1,1].imshow(img)
```

SCREENSHOT JUPYTER NOTEBOOK:



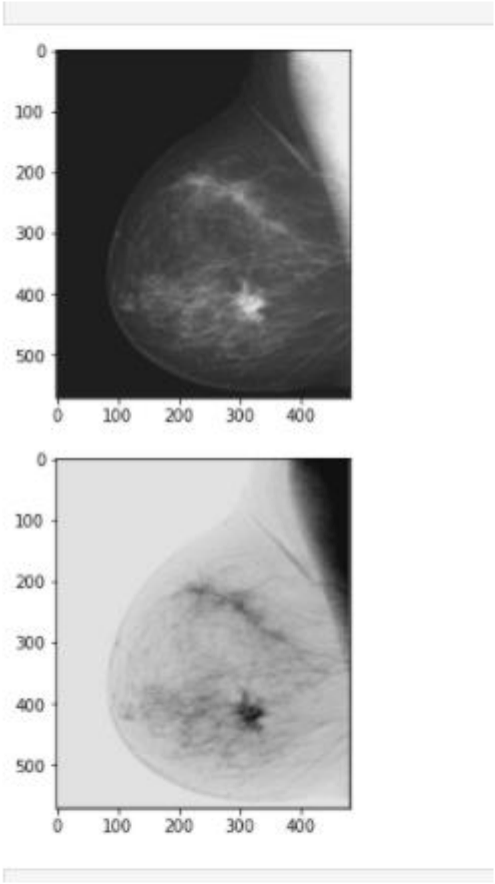
```
[6]: <matplotlib.image.AxesImage at 0x26957e02640>
```

TASK 4 – IMAGE NEGATIVE

CODES:

```
import cv2
import matplotlib.pyplot as plt
f, axarr = plt.subplots(2)
img_bgr = cv2.imread('breast.jpg', 1)
#plt.imshow(img_bgr)
axarr[0].imshow(img_bgr)
#plt.show()
height, width, _ = img_bgr.shape

for i in range(0, height - 1):
    for j in range(0, width - 1):

        pixel = img_bgr[i, j]
        pixel[0] = 255 - pixel[0]
        pixel[1] = 255 - pixel[1]
        pixel[2] = 255 - pixel[2]
        img_bgr[i, j] = pixel

#plt.imshow(img_bgr)
axarr[1].imshow(img_bgr)
#plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:

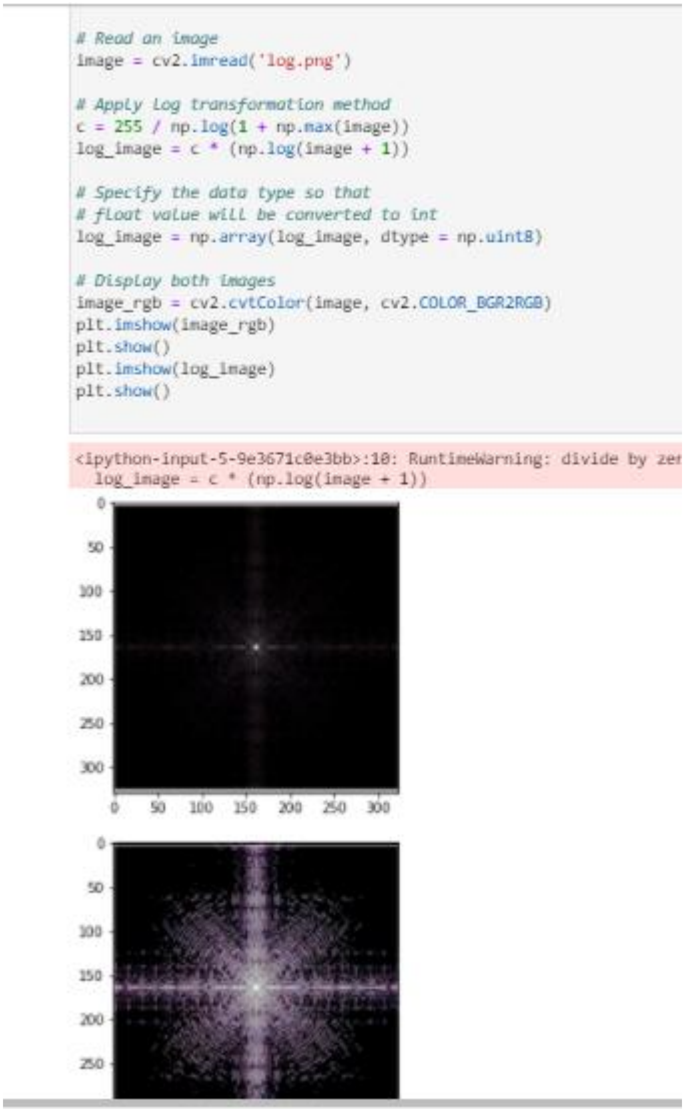


TASK 5 – LOG TRANSFORMATION

CODES:

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

# Read an image
image = cv2.imread('log.png')

# Apply log transformation method
c = 255 / np.log(1 + np.max(image))
log_image = c * (np.log(image + 1))

# Specify the data type so that
# float value will be converted to int
log_image = np.array(log_image, dtype = np.uint8)

# Display both images
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()
plt.imshow(log_image)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:



```
# Read an image
image = cv2.imread('log.png')

# Apply log transformation method
c = 255 / np.log(1 + np.max(image))
log_image = c * (np.log(image + 1))

# Specify the data type so that
# float value will be converted to int
log_image = np.array(log_image, dtype = np.uint8)

# Display both images
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()
plt.imshow(log_image)
plt.show()
```

```
<ipython-input-5-9e3671c0e3bb>:10: RuntimeWarning: divide by zer
  log_image = c * (np.log(image + 1))
```

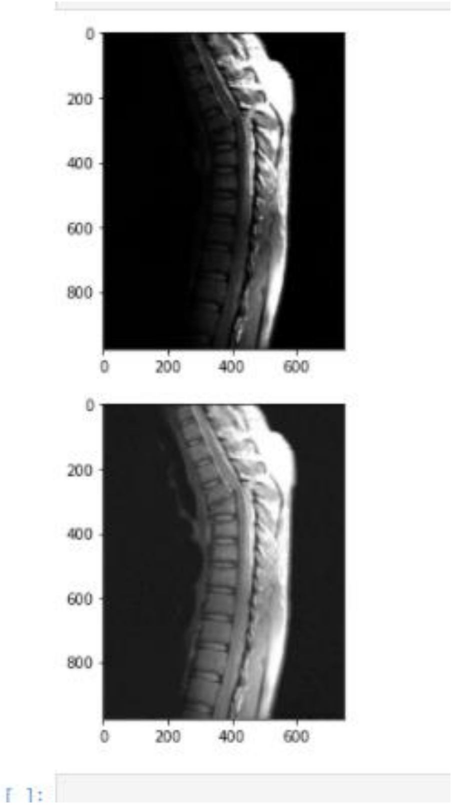
TASK 6-7 – POWER-LAW GAMMA TRANSFORMATION

CODES:

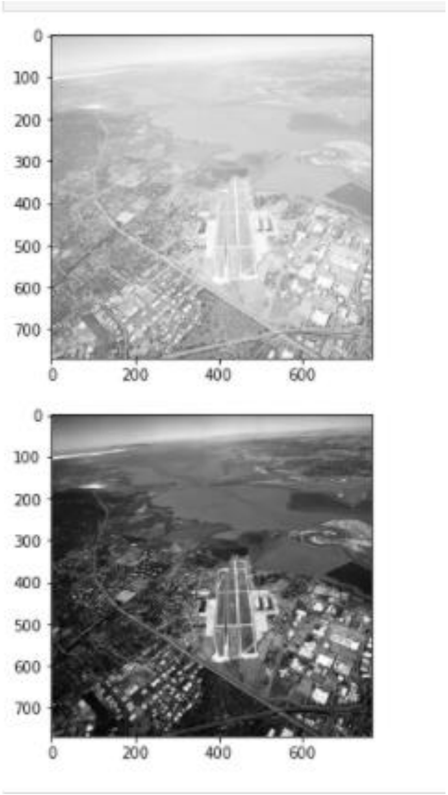
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

img = cv2.imread('Fig0308(a)(fractured_spine).tif')
plt.imshow(img)
plt.show()
# gamma value 0.4
for gamma in [0.4]:
        gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')
plt.imshow(gamma_corrected)
plt.show()

img1 = cv2.imread('city.tif')
plt.imshow(img1)
plt.show()

# gamma value 5
for gamma in [5]:
        gamma_corrected = np.array(255*(img1 / 255) ** gamma, dtype = 'uint8')
plt.imshow(gamma_corrected)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:

GAMMA VALUE = 0.4



GAMMA VALUE = 5.0

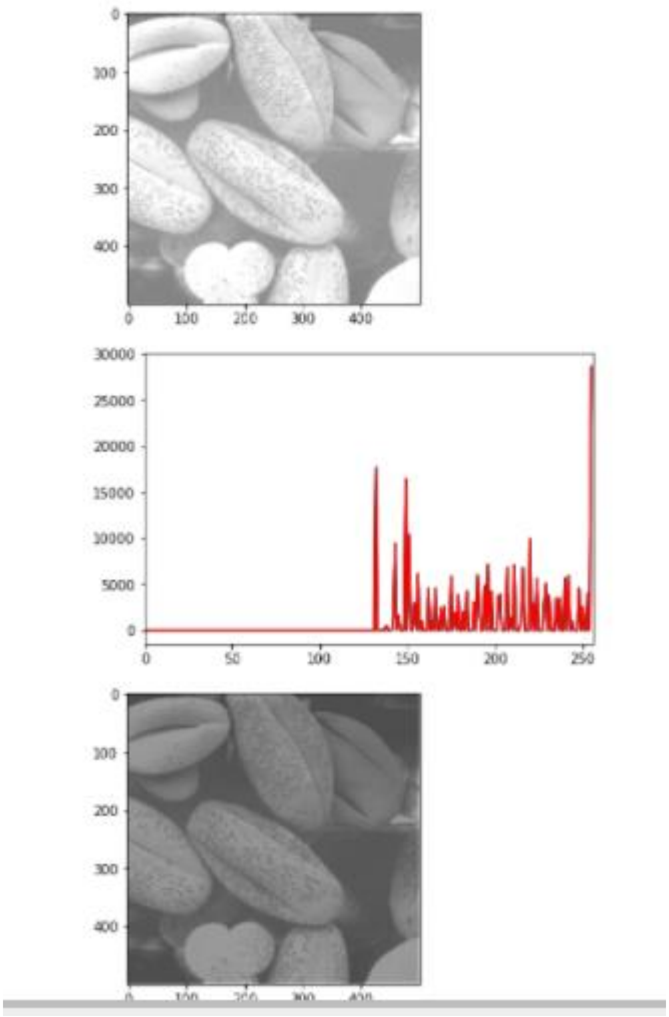


TASK 8 – CONSTRACT STRETCHING

CODES:

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

image = cv2.imread('E2.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()

color = ('b', 'g', 'r')
  for i, col in enumerate(color):
      histr = cv2.calcHist([image],
                    [i], None,
                    [256],
                    [0, 256])

    plt.plot(histr, color = col)

    plt.xlim([0, 256])
     plt.show()

maxiI = 250
miniI = 2

maxoI = 150
minoI = 0

stretched_image = image.copy()
height, width, _ = image.shape
  for i in range(0, height - 1):
    for j in range(0, width - 1):

        pixel = stretched_image[i, j]

        pixel[0] = (pixel[0] - miniI) * ((maxoI-minoI) / (maxiI-miniI)) + minoI
        pixel[1] = (pixel[1] - miniI) * ((maxoI-minoI) / (maxiI-miniI)) + minoI
        pixel[2] = (pixel[2] - miniI) * ((maxoI-minoI) / (maxiI-miniI)) + minoI
        stretched_image[i, j] = pixel

plt.imshow(stretched_image)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:



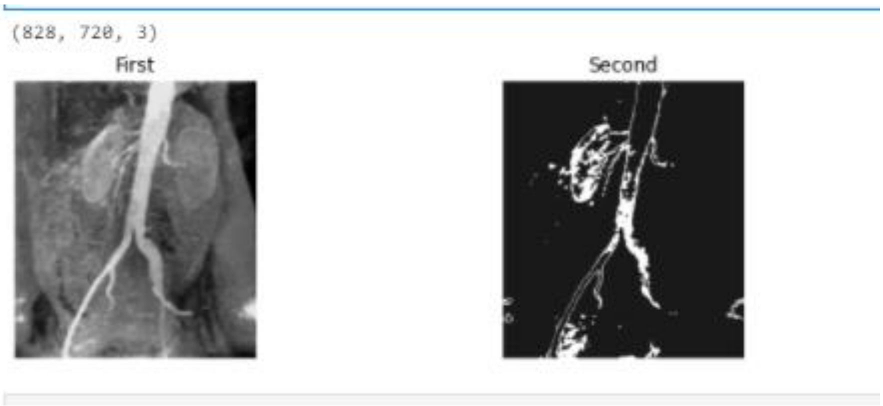
Task -9 intensity-level slicing

Codes:

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

img = cv2.imread('Fig0312(a)(kidney).tif')
fig = plt.figure(figsize=(10, 7))

# setting values to rows and column variables
rows = 2
columns = 2
# Adds a subplot at the 1st position
fig.add_subplot(rows, columns, 1)

# showing image
plt.imshow(img)
plt.axis('off')
plt.title("First")

def intensitylevelslicing( r,r1,s1,r2,s2):
    if(0<=r and r<r1):
        return s1
    elif(r1<=r and r<r2):
        return s2
    else:
        return  s1



r1=153
s1=25
r2=193
s2=255

h=img.shape[1];
w=img.shape[0];
c=img.shape[2];

a = np.uint8(np.zeros((w,h,c)))

for i in range(w):
    for j in range(h):
        for k in range(c):
         a[i,j,k]=intensitylevelslicing(img[i,j,k],r1,s1,r2,s2)

print(a.shape)
```

```
#cv2.imshow("intensity level sliced",a)

fig.add_subplot(rows, columns, 2)

# showing image
plt.imshow(a)
plt.axis('off')
plt.title("Second")

plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:



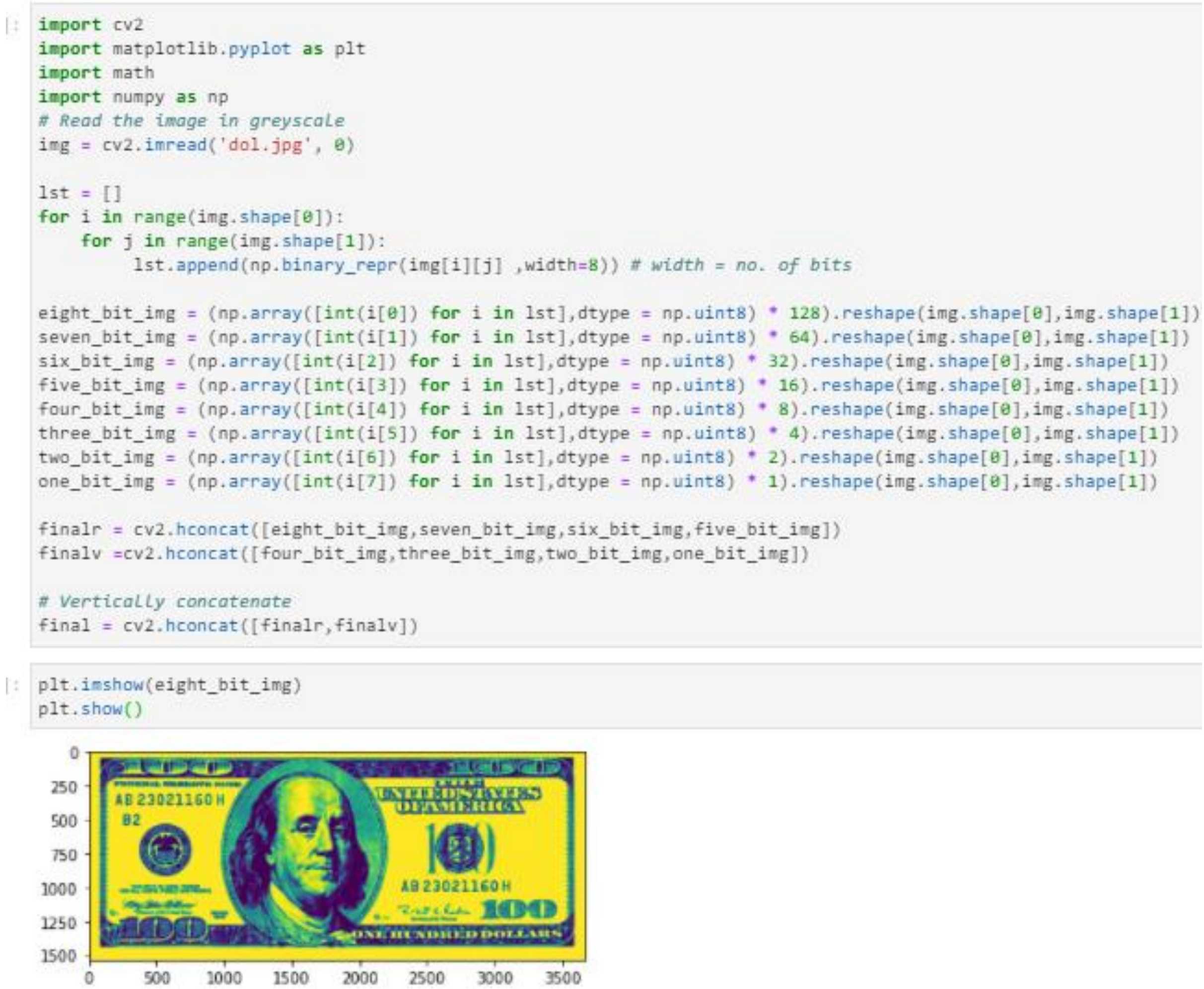
```
(828, 720, 3)
```

TASK 10-11 - BIT-PLANE SLICING

CODE:

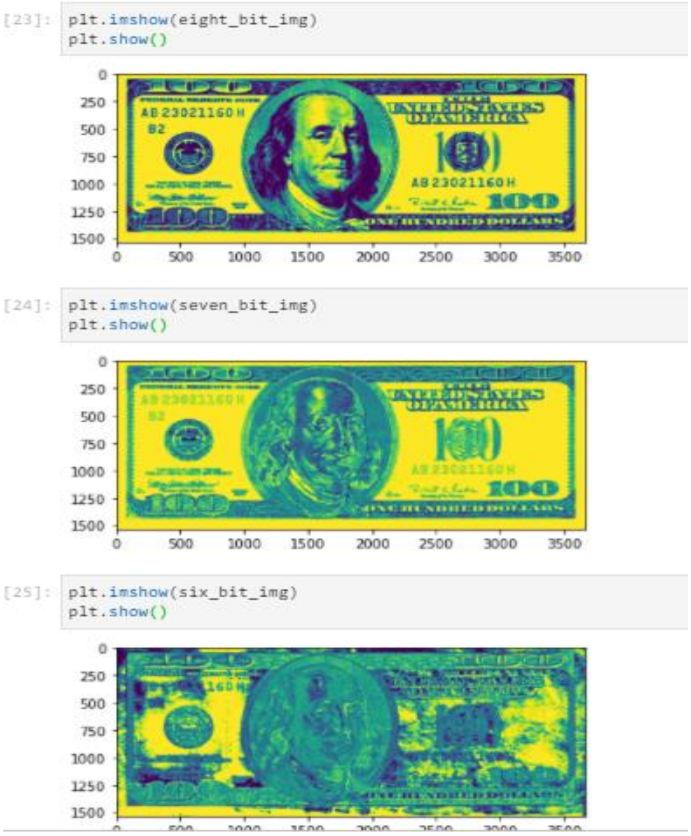
```
import cv2
import matplotlib.pyplot as plt
import math
import numpy as np
# Read the image in greyscale
img = cv2.imread('dol.jpg', 0)

lst = []
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
         lst.append(np.binary_repr(img[i][j] ,width=8))

eight_bit_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8) *
128).reshape(img.shape[0],img.shape[1])

seven_bit_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8) *
64).reshape(img.shape[0],img.shape[1])

six_bit_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) *
32).reshape(img.shape[0],img.shape[1])

five_bit_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8) *
16).reshape(img.shape[0],img.shape[1])

four_bit_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8) *
8).reshape(img.shape[0],img.shape[1])

three_bit_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8) *
4).reshape(img.shape[0],img.shape[1])

two_bit_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) *
2).reshape(img.shape[0],img.shape[1])

one_bit_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) *
1).reshape(img.shape[0],img.shape[1])

finalr = cv2.hconcat([eight_bit_img,seven_bit_img,six_bit_img,five_bit_img])
finalv =cv2.hconcat([four_bit_img,three_bit_img,two_bit_img,one_bit_img])

# Vertically concatenate
final = cv2.hconcat([finalr,finalv])

plt.imshow(eight_bit_img)
plt.show()
plt.imshow(seven_bit_img)
plt.show()
plt.imshow(five_bit_img)
plt.show()
#image-reconstruction
new_img = eight_bit_img+seven_bit_img+six_bit_img+five_bit_img
plt.imshow(new_img)
plt.show()
```

<u>SCREENSHOT JUPYTER NOTEBOOK:</u>



```
import cv2
import matplotlib.pyplot as plt
import math
import numpy as np
# Read the image in greyscale
img = cv2.imread('dol.jpg', 0)

lst = []
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        lst.append(np.binary_repr(img[i][j] ,width=8)) # width = no. of bits

eight_bit_img = (np.array([int(i[0]) for i in lst],dtype = np.uint8) * 128).reshape(img.shape[0],img.shape[1])
seven_bit_img = (np.array([int(i[1]) for i in lst],dtype = np.uint8) * 64).reshape(img.shape[0],img.shape[1])
six_bit_img = (np.array([int(i[2]) for i in lst],dtype = np.uint8) * 32).reshape(img.shape[0],img.shape[1])
five_bit_img = (np.array([int(i[3]) for i in lst],dtype = np.uint8) * 16).reshape(img.shape[0],img.shape[1])
four_bit_img = (np.array([int(i[4]) for i in lst],dtype = np.uint8) * 8).reshape(img.shape[0],img.shape[1])
three_bit_img = (np.array([int(i[5]) for i in lst],dtype = np.uint8) * 4).reshape(img.shape[0],img.shape[1])
two_bit_img = (np.array([int(i[6]) for i in lst],dtype = np.uint8) * 2).reshape(img.shape[0],img.shape[1])
one_bit_img = (np.array([int(i[7]) for i in lst],dtype = np.uint8) * 1).reshape(img.shape[0],img.shape[1])

finalr = cv2.hconcat([eight_bit_img,seven_bit_img,six_bit_img,five_bit_img])
finalv =cv2.hconcat([four_bit_img,three_bit_img,two_bit_img,one_bit_img])

# Vertically concatenate
final = cv2.hconcat([finalr,finalv])
```

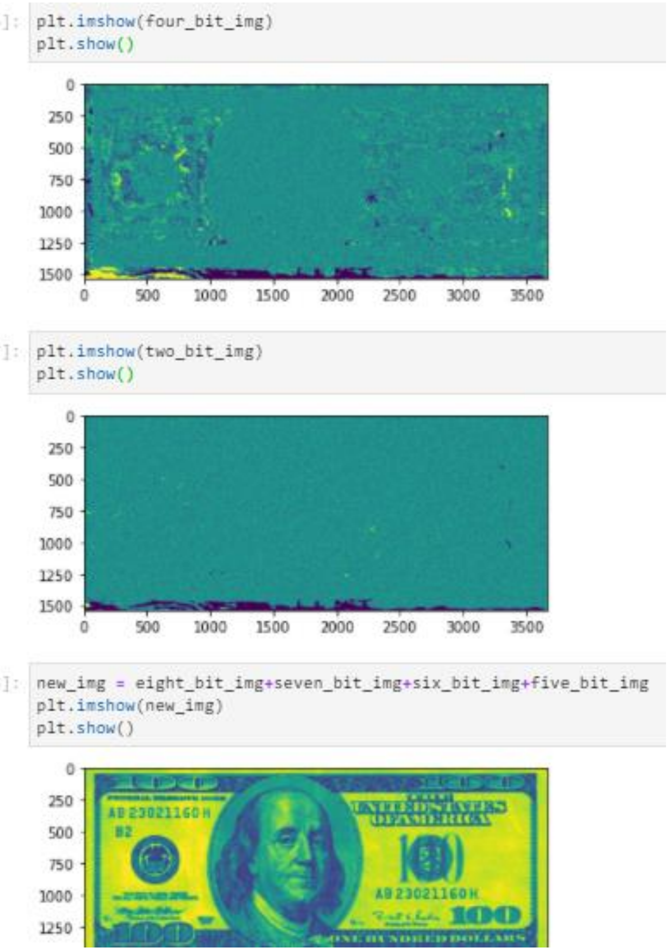
```
plt.imshow(eight_bit_img)
plt.show()
```



```
[23]: plt.imshow(eight_bit_img)
      plt.show()
```

```
[24]: plt.imshow(seven_bit_img)
      plt.show()
```

```
[25]: plt.imshow(six_bit_img)
      plt.show()
```



```
]: plt.imshow(four_bit_img)
   plt.show()
```

```
]: plt.imshow(two_bit_img)
   plt.show()
```

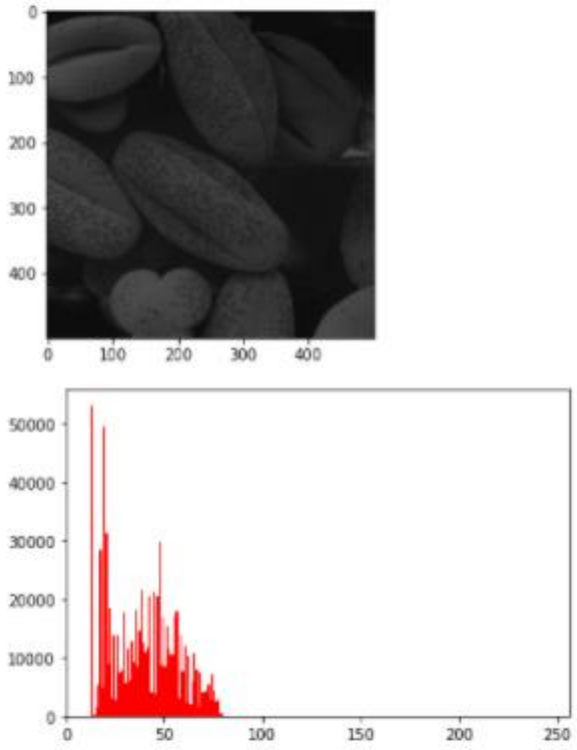
```
]: new_img = eight_bit_img+seven_bit_img+six_bit_img+five_bit_img
   plt.imshow(new_img)
   plt.show()
```

TASK 12 – DISPLAYING IMAGE AND ITS HISTOGRAM

CODES:

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

img = cv2.imread('Fig0320(4)(bottom_left).tif')
# Display the images
plt.imshow(img)
plt.show()

#display the histogram
hist,bins = np.histogram(img.flatten(),256,[0,256])
plt.hist(img.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:



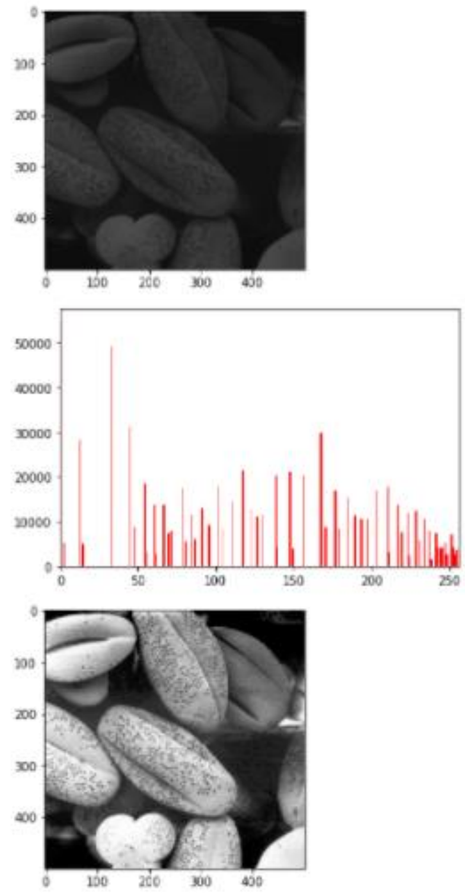
TASK 13 – HISTOGRAM EQUALIZATION

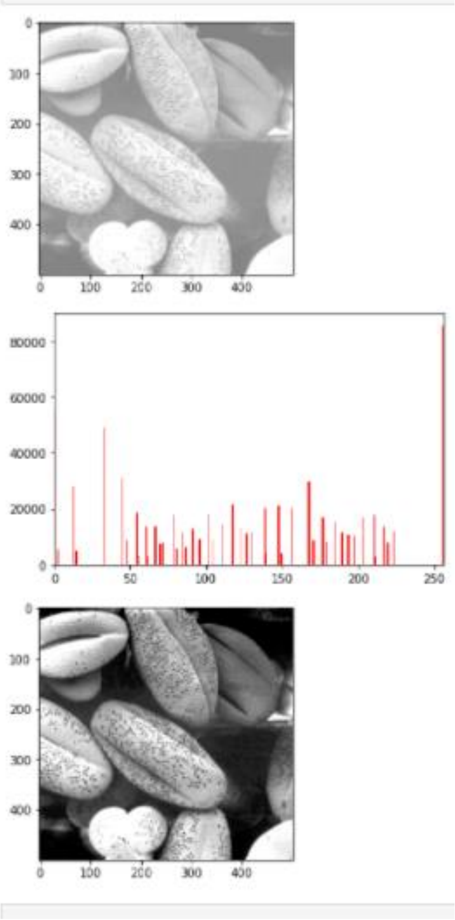
CODE:

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

def histogram_equalization(img_in):

# segregate color streams
   b,g,r = cv2.split(img_in)
   h_b, bin_b = np.histogram(b.flatten(), 256, [0, 256])
   h_g, bin_g = np.histogram(g.flatten(), 256, [0, 256])
   h_r, bin_r = np.histogram(r.flatten(), 256, [0, 256])

# calculate cdf
   cdf_b = np.cumsum(h_b)
   cdf_g = np.cumsum(h_g)
   cdf_r = np.cumsum(h_r)

# mask all pixels with value=0 and replace it with mean of the pixel values
   cdf_m_b = np.ma.masked_equal(cdf_b,0)
   cdf_m_b = (cdf_m_b - cdf_m_b.min())*255/(cdf_m_b.max()-cdf_m_b.min())
   cdf_final_b = np.ma.filled(cdf_m_b,0).astype('uint8')

   cdf_m_g = np.ma.masked_equal(cdf_g,0)
   cdf_m_g = (cdf_m_g - cdf_m_g.min())*255/(cdf_m_g.max()-cdf_m_g.min())
   cdf_final_g = np.ma.filled(cdf_m_g,0).astype('uint8')

   cdf_m_r = np.ma.masked_equal(cdf_r,0)
   cdf_m_r = (cdf_m_r - cdf_m_r.min())*255/(cdf_m_r.max()-cdf_m_r.min())
   cdf_final_r = np.ma.filled(cdf_m_r,0).astype('uint8')

# merge the images in the three channels
   img_b = cdf_final_b[b]
   img_g = cdf_final_g[g]
   img_r = cdf_final_r[r]

   img_out = cv2.merge((img_b, img_g, img_r))

# validation
   equ_b = cv2.equalizeHist(b)
   equ_g = cv2.equalizeHist(g)
   equ_r = cv2.equalizeHist(r)

   equ = cv2.merge((equ_b, equ_g, equ_r))
```

```
    #print(equ)
    #cv2.imwrite('output_name.png', equ)
    return img_out

img = cv2.imread('Fig0320(4)(bottom_left).tif')
img_out = histogram_equalization(img)

# Display the images
plt.imshow(img)
plt.show()

#display the histogram
hist,bins = np.histogram(img_out.flatten(),256,[0,256])
plt.hist(img_out.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.show()
plt.imshow(img_out)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:



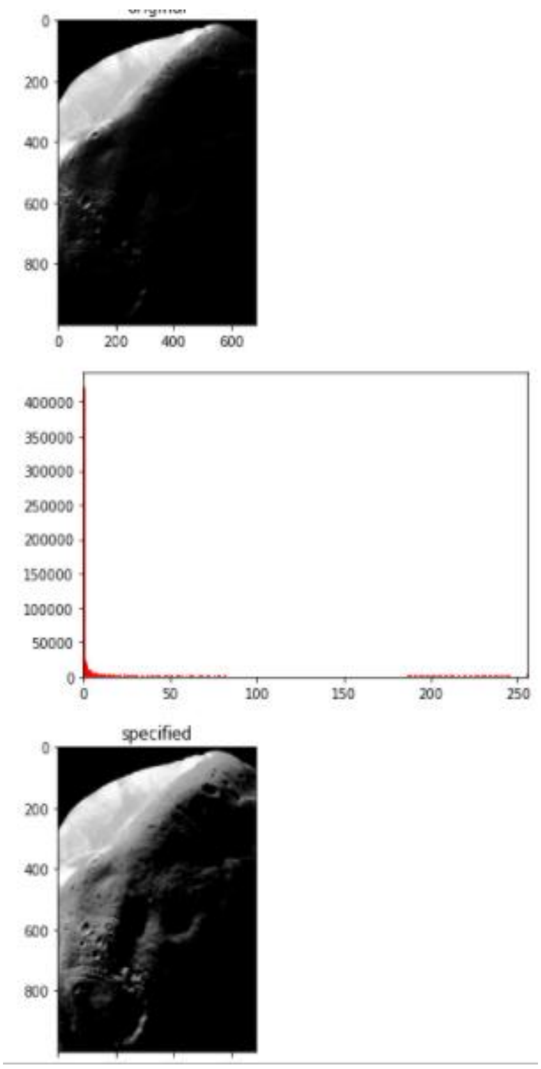


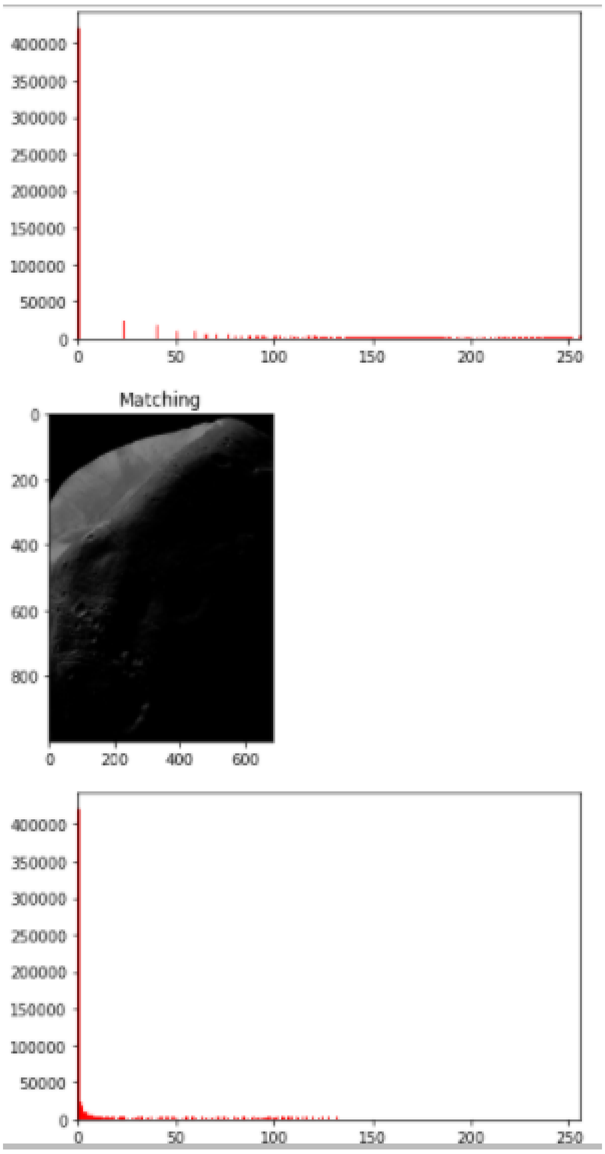
TASK 14 – HISTOGRAM MATCHING

CODE:

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

def find_nearest_above(my_array, target):
    diff = my_array - target
    mask = np.ma.less_equal(diff, -1)
    # We need to mask the negative differences
    # since we are looking for values above
    if np.all(mask):
        c = np.abs(diff).argmin()
        return c # returns min index of the nearest if target is greater than any value
    masked_diff = np.ma.masked_array(diff, mask)
    return masked_diff.argmin()


def hist_match(original, specified):
    oldshape = original.shape
    original = original.ravel()
    specified = specified.ravel()

    # get the set of unique pixel values and their corresponding indices and counts
    s_values, bin_idx, s_counts = np.unique(original, return_inverse=True,return_counts=True)
    t_values, t_counts = np.unique(specified, return_counts=True)

    # Calculate s_k for original image
    s_quantiles = np.cumsum(s_counts).astype(np.float64)
    s_quantiles /= s_quantiles[-1]

    # Calculate s_k for specified image
    t_quantiles = np.cumsum(t_counts).astype(np.float64)
    t_quantiles /= t_quantiles[-1]

    # Round the values
    sour = np.around(s_quantiles*255)
    temp = np.around(t_quantiles*255)

    # Map the rounded values
    b=[]
    for data in sour[:]:
        b.append(find_nearest_above(temp,data))
    b= np.array(b,dtype='uint8')
    return b[bin_idx].reshape(oldshape)

 # Load the images in greyscale
original = cv2.imread('Fig0323(a)(mars_moon_phobos).tif',0)
img = cv2.imread('Fig0323(a)(mars_moon_phobos).tif')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
specified = cv2.equalizeHist(gray)

# perform Histogram Matching
a = hist_match(original, specified)



# Display the images
image_rgb1 = cv2.cvtColor(original, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb1)
plt.title("original")
plt.show()
#display the histogram
hist,bins = np.histogram(original.flatten(),256,[0,256])
plt.hist(original.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.show()
gray1 = cv2.cvtColor(specified, cv2.COLOR_GRAY2RGB)
plt.imshow(gray1)
plt.title("specified")
plt.show()
#display the histogram
hist,bins = np.histogram(specified.flatten(),256,[0,256])
plt.hist(specified.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.show()
image_rgb1 = cv2.cvtColor(a, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb1)
plt.title("Matching")
plt.show()
#display the histogram
hist,bins = np.histogram(a.flatten(),256,[0,256])
plt.hist(a.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.show()
```

<u>SCREENSHOT JUPYTER NOTEBOOK:</u>


original
0
200
400
600
800
0
200
400
600
400000
350000
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250
specified
0
200
400
600
800

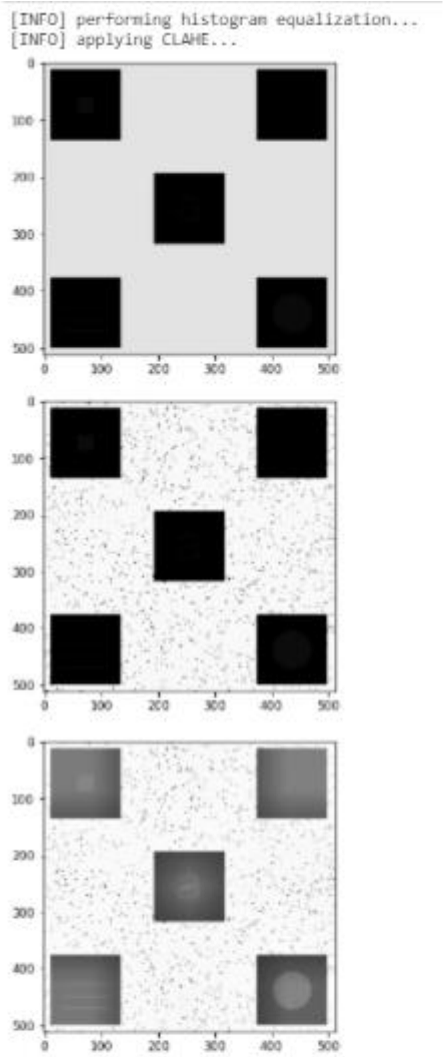

400000
350000
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250
Matching
0
200
400
600
800
0
200
400
600
400000
350000
300000
250000
200000
150000
100000
50000
0
0
50
100
150
200
250

TASK 15 LOCAL HISTOGRAM EQULIZATION

CODE:

```
import cv2
import matplotlib.pyplot as plt
import math
import numpy as np
image = cv2.imread("Fig0326(a)(embedded_square_noisy_512).tif")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# apply histogram equalization
print("[INFO] performing histogram equalization...")
gequalized = cv2.equalizeHist(gray)
# apply CLAHE ( Local Histogram Equalization)
print("[INFO] applying CLAHE...")
clahe = cv2.createCLAHE(clipLimit=256,tileGridSize=(3,3))
equalized = clahe.apply(gray)
gray1 = cv2.cvtColor(gray, cv2.COLOR_GRAY2RGB)
plt.imshow(gray1)
plt.show()
gray2 = cv2.cvtColor(gequalized, cv2.COLOR_GRAY2RGB)
plt.imshow( gray2)
plt.show()
gray3 = cv2.cvtColor(equalized, cv2.COLOR_GRAY2RGB)
plt.imshow(gray3)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:



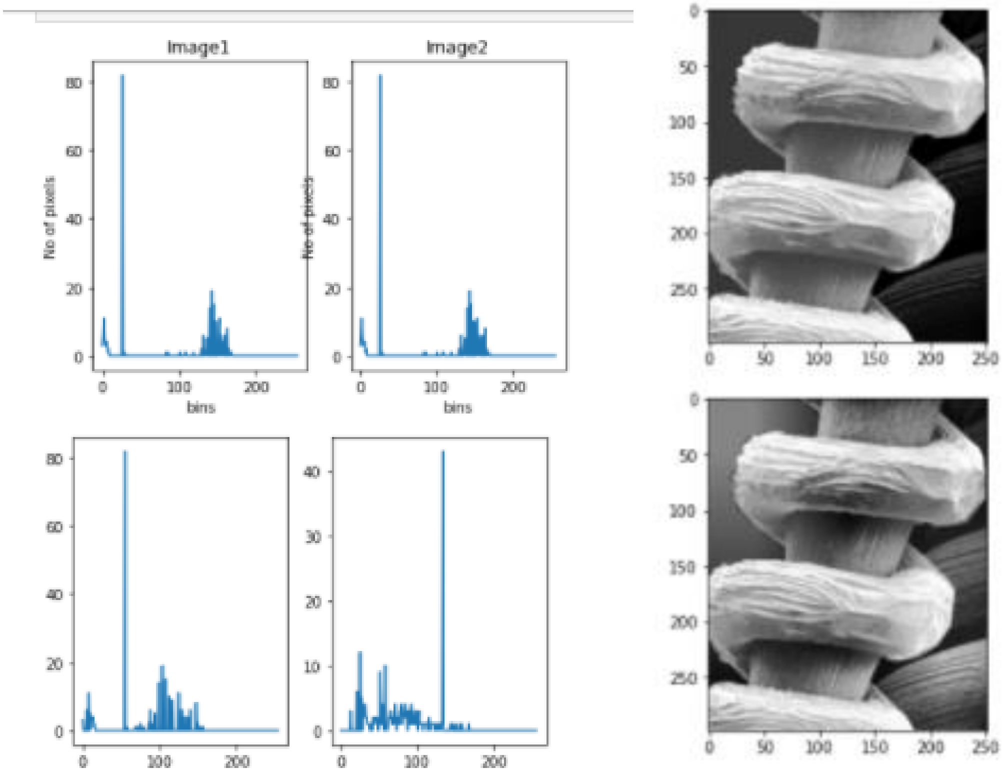
```
[INFO] performing histogram equalization...
[INFO] applying CLAHE...
```

Task 16 -Using Histogram Statistics for Image Enhancement

Codes –

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img= cv2.imread('Fig0327(a)(tungsten_original).tif')
img1=cv2.imread('Fig0327(a)(tungsten_original).tif')
gray_img=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
gray_img1=cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
hist=cv2.calcHist(gray_img,[0],None,[256],[0,256])
hist1=cv2.calcHist(gray_img1,[0],None,[256],[0,256])
plt.subplot(121)
plt.title("Image1")
plt.xlabel('bins')
plt.ylabel("No of pixels")
plt.plot(hist)
plt.subplot(122)
plt.title("Image2")
plt.xlabel('bins')
plt.ylabel("No of pixels")
plt.plot(hist1)
plt.show()
#equalization
gray_img_eqhist=cv2.equalizeHist(gray_img)
hist=cv2.calcHist(gray_img_eqhist,[0],None,[256],[0,256])
#localhistogram
clahe = cv2.createCLAHE(clipLimit=256,tileGridSize=(3,3))
equalized = clahe.apply(gray_img1)
hist1=cv2.calcHist(equalized,[0],None,[256],[0,256])
plt.subplot(121)
plt.plot(hist)
plt.subplot(122)
plt.plot(hist1)
plt.show()
gray_=cv2.cvtColor(gray_img_eqhist,cv2.COLOR_GRAY2RGB)
plt.imshow(gray_)
plt.show()
gray_=cv2.cvtColor(equalized,cv2.COLOR_GRAY2RGB)
plt.imshow(gray_)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:

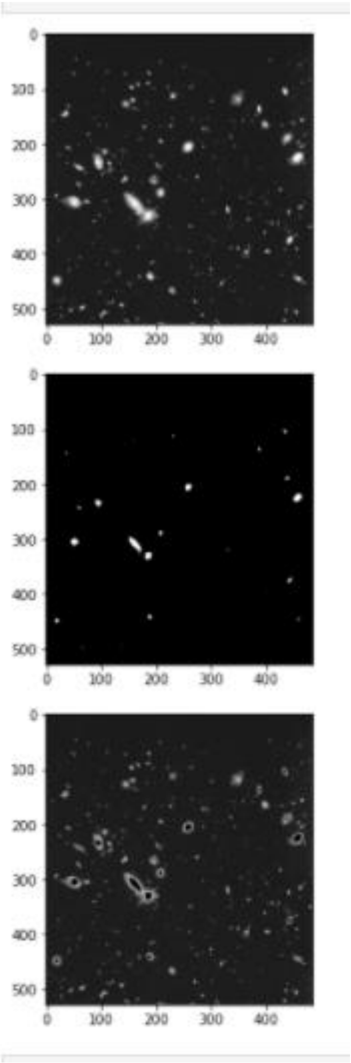


TASK 17 - Image Smoothing and thresholding

Codes:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread("coin.jpg")
image_rgb1 = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb1)
plt.show()
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
#7-7gaussianmasking
blurred = cv2.GaussianBlur(gray, (7, 7), 0)
(T, threshInv) = cv2.threshold(blurred, 200, 255,cv2.THRESH_BINARY_INV)
cv2.imshow("Threshold Binary Inverse", threshInv)
(T, thresh) = cv2.threshold(blurred, 200, 255, cv2.THRESH_BINARY)
image_rgb1 = cv2.cvtColor(thresh, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb1)
plt.show()
# thresholding
masked = cv2.bitwise_and(image, image, mask=threshInv)
image_rgb1 = cv2.cvtColor(masked, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb1)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:

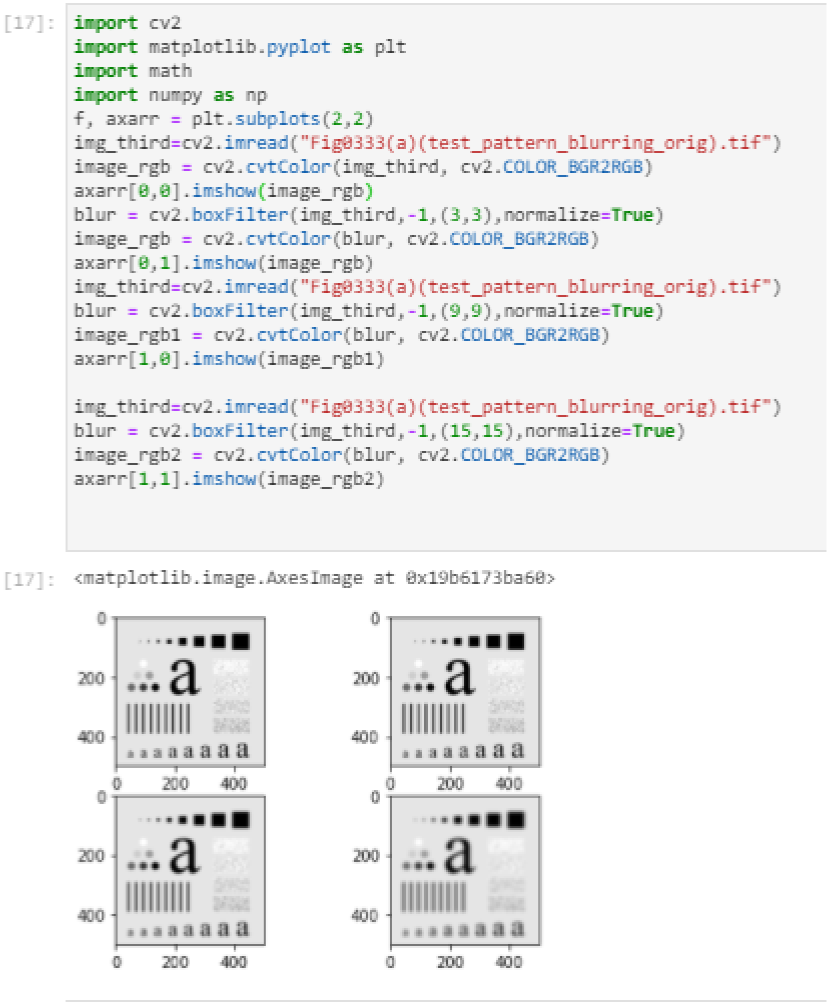


TASK 18 – IMAGE SMOOTHING

CODE:

```
import cv2
import matplotlib.pyplot as plt
import math
import numpy as np
f, axarr = plt.subplots(2,2)
img_third=cv2.imread("Fig0333(a)(test_pattern_blurring_orig).tif")
image_rgb = cv2.cvtColor(img_third, cv2.COLOR_BGR2RGB)
axarr[0,0].imshow(image_rgb)
#3-3
blur = cv2.boxFilter(img_third,-1,(3,3),normalize=True)
image_rgb = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)
axarr[0,1].imshow(image_rgb)
#9-9
img_third=cv2.imread("Fig0333(a)(test_pattern_blurring_orig).tif")
blur = cv2.boxFilter(img_third,-1,(9,9),normalize=True)
image_rgb1 = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)
axarr[1,0].imshow(image_rgb1)
#15-15
img_third=cv2.imread("Fig0333(a)(test_pattern_blurring_orig).tif")
blur = cv2.boxFilter(img_third,-1,(15,15),normalize=True)
image_rgb2 = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)
axarr[1,1].imshow(image_rgb2)
```

SCREENSHOT JUPYTER NOTEBOOK:



```
[17]: import cv2
      import matplotlib.pyplot as plt
      import math
      import numpy as np
      f, axarr = plt.subplots(2,2)
      img_third=cv2.imread("Fig0333(a)(test_pattern_blurring_orig).tif")
      image_rgb = cv2.cvtColor(img_third, cv2.COLOR_BGR2RGB)
      axarr[0,0].imshow(image_rgb)
      blur = cv2.boxFilter(img_third,-1,(3,3),normalize=True)
      image_rgb = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)
      axarr[0,1].imshow(image_rgb)
      img_third=cv2.imread("Fig0333(a)(test_pattern_blurring_orig).tif")
      blur = cv2.boxFilter(img_third,-1,(9,9),normalize=True)
      image_rgb1 = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)
      axarr[1,0].imshow(image_rgb1)

      img_third=cv2.imread("Fig0333(a)(test_pattern_blurring_orig).tif")
      blur = cv2.boxFilter(img_third,-1,(15,15),normalize=True)
      image_rgb2 = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)
      axarr[1,1].imshow(image_rgb2)
```

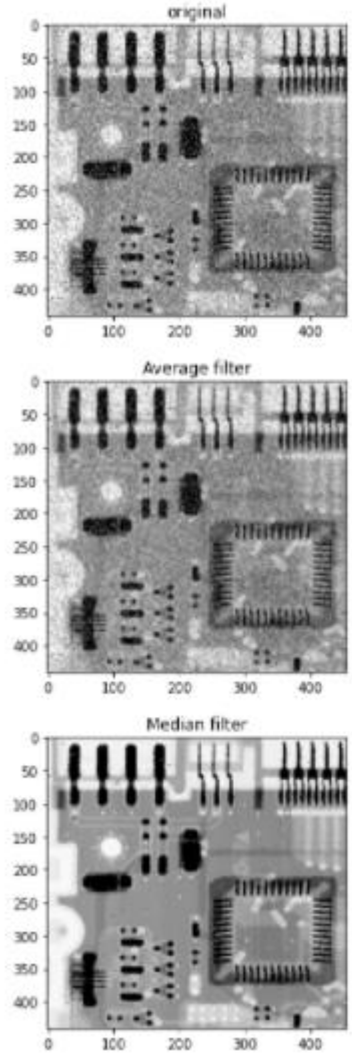
```
[17]: <matplotlib.image.AxesImage at 0x19b6173ba60>
```

TASK 19 - Averaging and Median Filter:

CODE:

```
import cv2
import matplotlib.pyplot as plt
import math
import numpy as np
img_third=cv2.imread("Fig0335(a)(ckt_board_saltpep_prob_pt05).tif")
image_rgb = cv2.cvtColor(img_third, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()
blur = cv2.boxFilter(img_third,-1,(3,3),normalize=True)
#average
image_rgb = cv2.cvtColor(blur, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()
median = cv2.medianBlur(src=img_third, ksize=3)
#median
image_rgb = cv2.cvtColor(median, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()
```

<u>SCREENSHOT JUPYTER NOTEBOOK:</u>

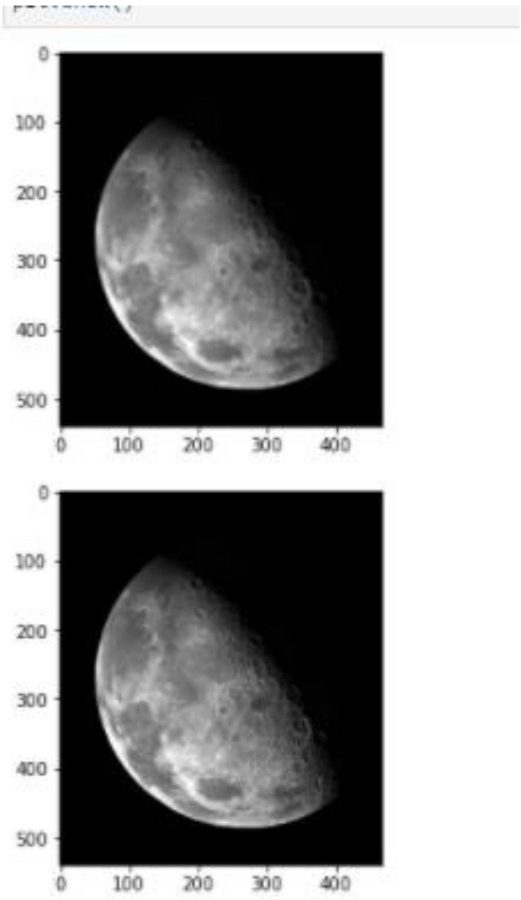


TASK 20 - Image Sharpening

CODES:

```
import cv2
import matplotlib.pyplot as plt
import math
import numpy as np
img_third=cv2.imread("Fig0338(a)(blurry_moon).tif")
image_rgb = cv2.cvtColor(img_third, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()
# Creating our sharpening filter
filter = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
# Applying cv2.filter2D function
sharpen_img_1=cv2.filter2D(img_third,-1,filter)
image_rgb = cv2.cvtColor(sharpen_img_1, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:

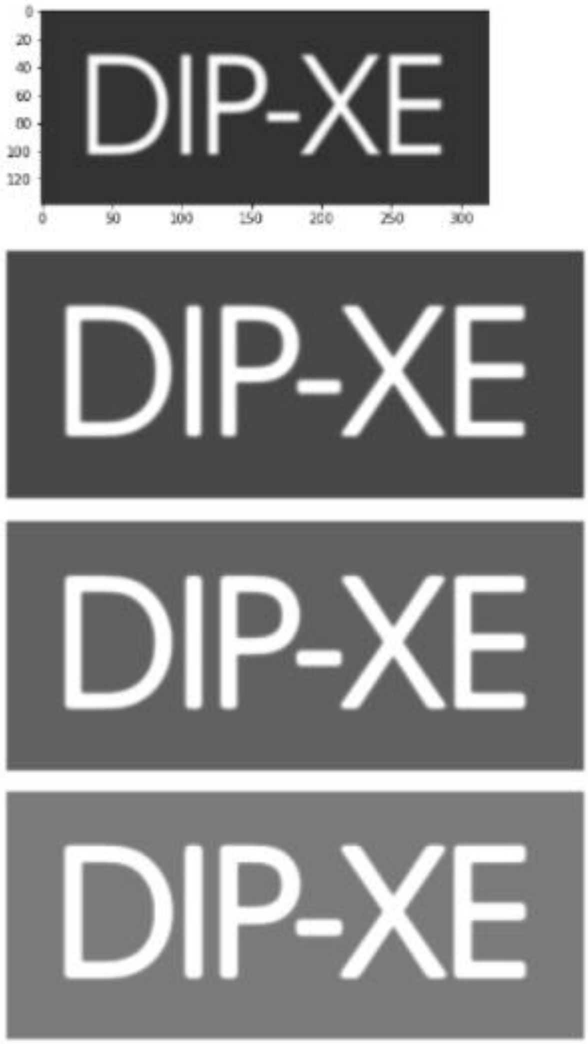


TASK 21 - High boost filtering:

CODES:

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

def highBoost(image,order,cutoff,a):
  img=np.asarray(cv2.imread(image,0))

  # Get the fourier transform of the image
  ft=np.fft.fft(img)

  # Shift the fourier transform
  sft=np.fft.fftshift(ft)

  # fourier transform
  magnitude_spectrum = sft

  rows,cols=img.shape
  butterFlt=np.zeros(img.shape)
  result=np.zeros(img.shape)

  # Computing the highpass and highboost filter
  for i in range(-int(rows/2),int(rows/2)):
    for j in range(-int(cols/2),int(cols/2)):
      distance=math.sqrt(pow(i,2)+pow(j,2))
      if(distance==0):
        one=(cutoff)
      else:
        one=(cutoff/distance)
      two=2*order
      demo=1+pow(one,two)
      butterFlt[i,j]=(a-1)+(1/demo)

  result=np.multiply(magnitude_spectrum,butterFlt)

  # Applying inverse fourier transform to transform the image back to the spatial domain
  f_ishift = np.fft.ifftshift(result)
  back = np.fft.ifft(f_ishift)
  img_back = np.abs(back)

  x=Image.fromarray(img_back)
  fig = plt.figure(frameon=False)

  ax = plt.Axes(fig, [0., 0., 1., 1.])
  ax.set_axis_off()
  fig.add_axes(ax)
  plt.imshow(x)
  return x

def main():
  img_third=cv2.imread("DIP.png")
  image_rgb = cv2.cvtColor(img_third, cv2.COLOR_BGR2RGB)
  plt.imshow(image_rgb)
  plt.show()

  Moon_1=highBoost('DIP.png',1,50,1.5)
  plt.savefig('HB_1.jpg')
  # DIP2
  Moon_2=highBoost('DIP.png',1,50,2)
  plt.savefig('HB_2.jpg')
  # DIP3
  Moon_3=highBoost('DIP.png',1,50,2.5)
  plt.savefig('HB_3.jpg')

if __name__=='__main__':
  main()
```

SCREENSHOT JUPYTER NOTEBOOK:

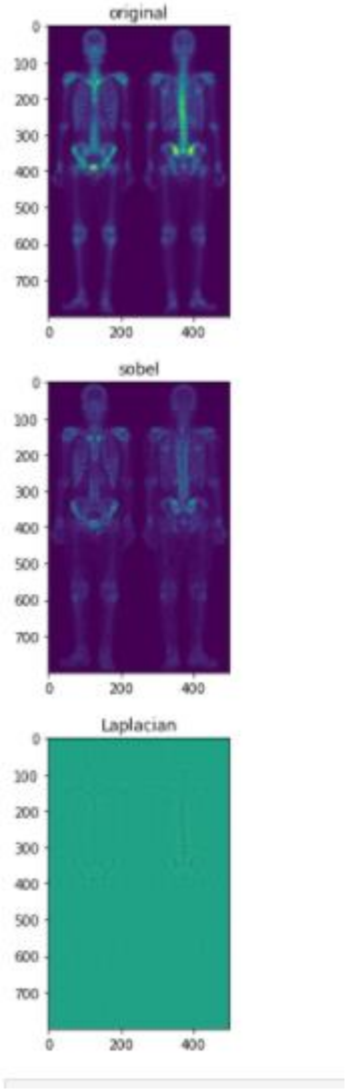


Task 22 – sobel gradient

Code:

```
import argparse
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('bone33.tif')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ksize = 3

gX = cv2.Sobel(gray, ddepth=cv2.CV_32F, dx=1, dy=0, ksize=ksize)
gY = cv2.Sobel(gray, ddepth=cv2.CV_32F, dx=0, dy=1, ksize=ksize)

gX = cv2.convertScaleAbs(gX)
gY = cv2.convertScaleAbs(gY)

# combine the gradient representations into a single image
combined = cv2.addWeighted(gX, 0.5, gY, 0.5, 0)
laplacian = cv2.Laplacian(gray,cv2.CV_32F,ksize=5)
plt.imshow(gray)
plt.title('original')
plt.show()
plt.imshow(combined)
plt.title('sobel')
plt.show()
plt.imshow(laplacian)
plt.title('Laplacian')
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:

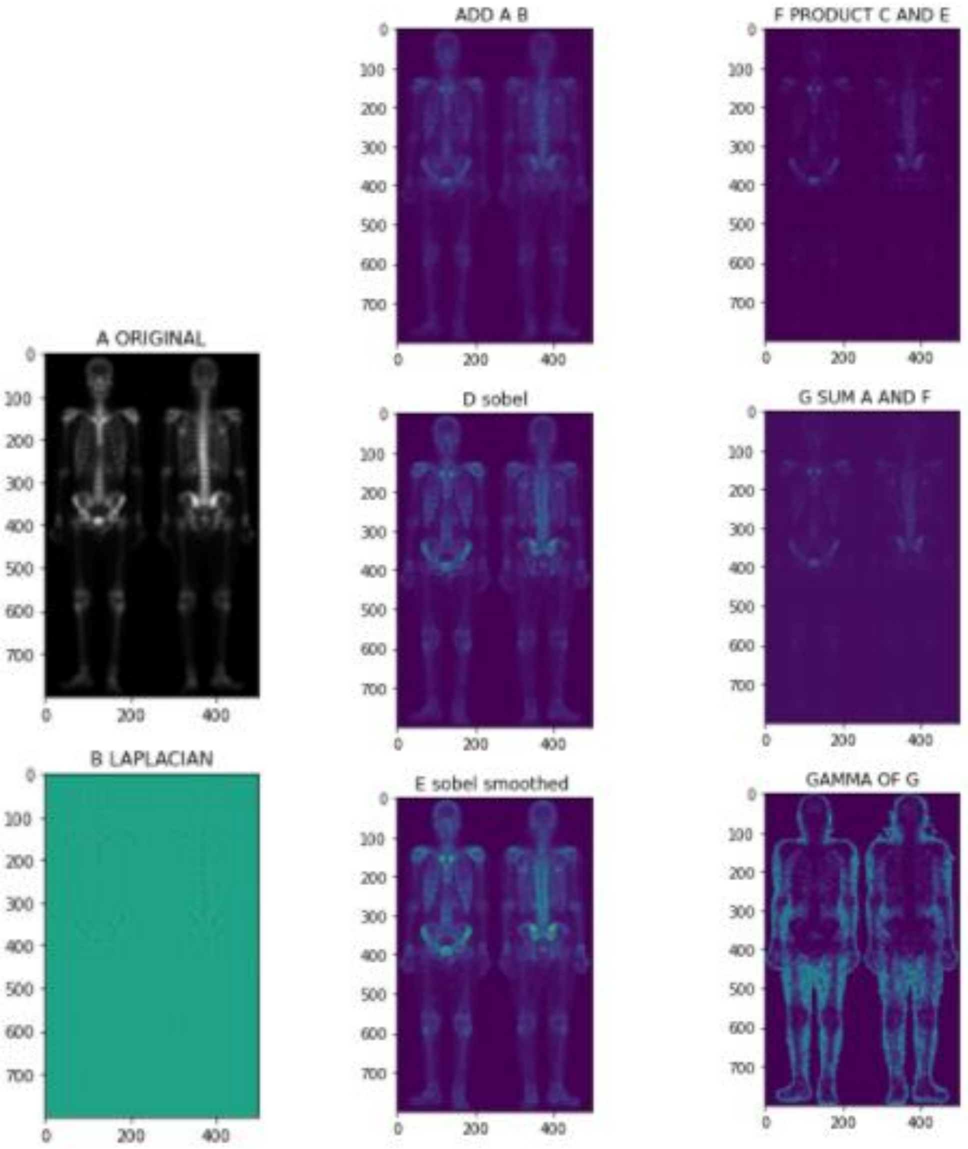


TASK 23 – Combining Spatial Enhancement Methods

Codes-

```
import argparse
import cv2
import numpy as np
from matplotlib import pyplot as plt
image = cv2.imread('bone33.tif')
plt.imshow(image)
plt.title('A ORIGINAL')
plt.show()
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
laplacian = cv2.Laplacian(gray,cv2.CV_32F,ksize=5)
plt.imshow(laplacian)
plt.title('B LAPLACIAN')
plt.show()

(row, col) = image.shape[0:2]
outcome = np.zeros([row, col])
for i in range(0,row):
    for j in range(0,col):
        outcome[i,j] = abs(laplacian[i, j]) + abs(gray[i, j])
plt.imshow(outcome)
plt.title('ADD A B ')
plt.show()
ksize = 3
gX = cv2.Sobel(gray, ddepth=cv2.CV_32F, dx=1, dy=0, ksize=ksize)
gY = cv2.Sobel(gray, ddepth=cv2.CV_32F, dx=0, dy=1, ksize=ksize)

gX = cv2.convertScaleAbs(gX)
gY = cv2.convertScaleAbs(gY)
# combine the gradient representations into a single image
combined = cv2.addWeighted(gX, 0.5, gY, 0.5, 0)
plt.imshow(combined)
plt.title('D sobel')
plt.show()
blur = cv2.boxFilter(combined,-1,(5,5),normalize=True)
plt.imshow(blur)
plt.title('E sobel smoothed ')
plt.show()
(row, col) = image.shape[0:2]
product = np.zeros([row, col])
for i in range(0,row):
    for j in range(0,col):
        product[i,j] = abs(blur[i, j]) * abs(outcome[i, j])
plt.imshow(product)
plt.title('F PRODUCT C AND E')
plt.show()
g = np.zeros([row, col])
for i in range(0,row):
    for j in range(0,col):
        g[i,j] = abs(gray[i, j]) + abs(product[i, j])
filter = np.array([[0, -1, 0], [-1, 15, -1], [0, -1, 0]])
# Applying cv2.filter2D function on our Cybertruck image
sharpen=cv2.filter2D(g,-1,filter)
plt.imshow(sharpen)
plt.title(' G SUM A AND F')
plt.show()
for gamma in [5]:
        gamma_corrected = np.array(255*(sharpen/ 255) ** gamma, dtype = 'uint8')
plt.imshow(gamma_corrected)
plt.title('GAMMA OF G')
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:



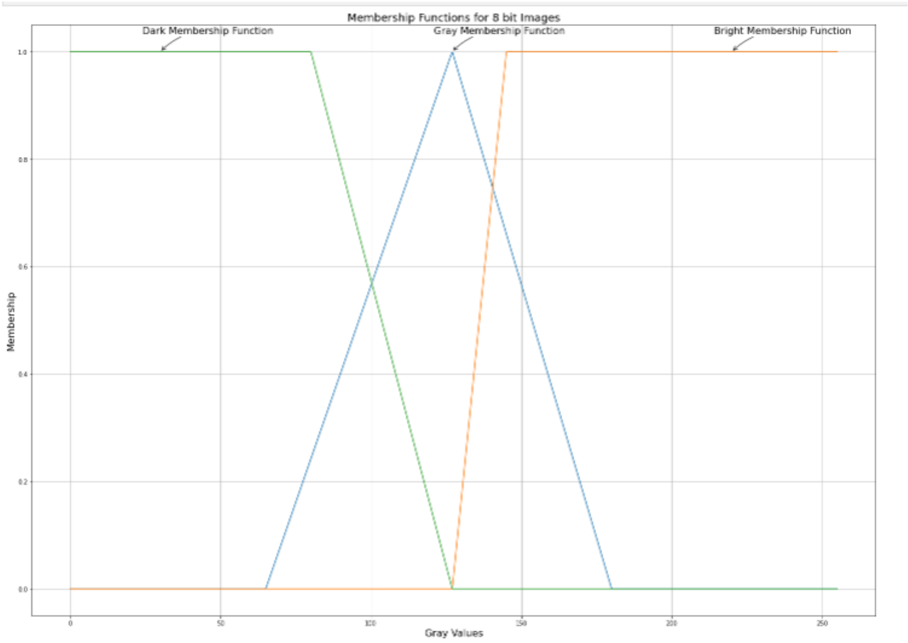
Task 24 – fuzzy rule-based contrast enhancement

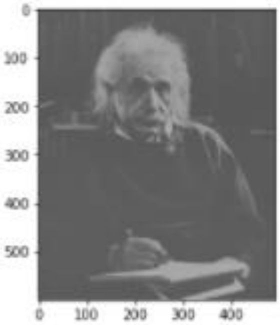
Code:

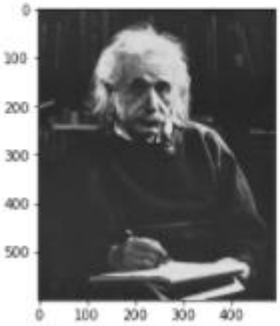
```
import numpy as np
from matplotlib.pyplot import imread
from matplotlib.pyplot import imsave
import sys

def enhance_grayscale_8bit_image(image)
    dark_color = 0
    gray_color = 127
    bright_color = 255

    # The membership parameters can be modified, if the result
    gray_membership_function = np.vectorize(
        triangular_membership_function(65, gray_color, 190))
    bright_membership_function = np.vectorize(
        sigma_membership_function(gray_color, 145))
    dark_membership_function = np.vectorize(
        inverse_sigma_membership_function(80, gray_color))

    dark_image_part = dark_membership_function(image)
    gray_image_part = gray_membership_function(image)
    bright_image_part = bright_membership_function(image)

    enhanced_image = (dark_image_part * dark_color +
                      gray_image_part * gray_color +
                      bright_image_part * bright_color) / \
        (dark_image_part + gray_image_part + bright_image_part)

    enhanced_image = enhanced_image.astype(np.uint8)
    return enhanced_image
def triangular_membership_function(triangle_start, triangle_peak, triangle_end):
    def membership_function(parameter):
        if parameter < triangle_start:
            return 0

        if triangle_start <= parameter and parameter < triangle_peak:
            return (parameter - triangle_start) / (triangle_peak - triangle_start)

        if triangle_peak <= parameter and parameter < triangle_end:
            return 1 - (parameter - triangle_peak) / (triangle_end - triangle_peak)

        # triangle_end <= parameter
        return 0

    return membership_function
```

```
def sigma_membership_function(sigma_start, sigma_end):
    def membership_function(parameter):
        if parameter < sigma_start:
            return 0

        if sigma_start <= parameter and parameter < sigma_end:
            return (parameter - sigma_start) / (sigma_end - sigma_start)

        # sigma_end <= parameter
        return 1

    return membership_function

def inverse_sigma_membership_function(sigma_start, sigma_end):
    def membership_function(parameter):
        if parameter < sigma_start:
            return 1

        if sigma_start <= parameter and parameter < sigma_end:
            return 1 - (parameter - sigma_start) / (sigma_end - sigma_start)

        # sigma_end <= parameter
        return 0
    return membership_function

    import sys
sys.path.append('ee.tif')

# The membership parameters can be modified, if the result
gray_membership_function = triangular_membership_function(65, 127, 180)
bright_membership_function = sigma_membership_function(127, 145)
dark_membership_function = inverse_sigma_membership_function(80, 127)

x = np.linspace(0, 255, 1000)
y1 = [gray_membership_function(param) for param in x]
y2 = [bright_membership_function(param) for param in x]
y3 = [dark_membership_function(param) for param in x]

fig, axs = plt.subplots(figsize=(25, 18))
axs.plot(x, y1, x, y2, x, y3)
axs.set_xlabel('Gray Values', fontsize=16)
axs.set_ylabel('Membership', fontsize=16)
axs.set_title("Membership Functions for 8 bit Images", fontsize=18)
axs.grid()

plt.annotate('Gray Membership Function', xy=(127, 1),
             xycoords='data',
             xytext=(-30, +30),
             textcoords='offset points',
             fontsize=16,
             arrowprops=dict(arrowstyle="->",
             connectionstyle="arc3,rad=.2"))

plt.annotate('Dark Membership Function', xy=(30, 1),
             xycoords='data',
             xytext=(-30, +30),
             textcoords='offset points',
             fontsize=16,
             arrowprops=dict(arrowstyle="->",
             connectionstyle="arc3,rad=.2"))

plt.annotate('Bright Membership Function', xy=(220, 1),
             xycoords='data',
             xytext=(-30, +30),
             textcoords='offset points',
             fontsize=16,
             arrowprops=dict(arrowstyle="->",
             connectionstyle="arc3,rad=.2"))

plt.savefig("fuzzy_functions.svg")

plt.show()
sys.path.append('sys')

image = imread("ee.tif")
enhanced_image = enhance_grayscale_8bit_image(image)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb)
plt.show()
equ = cv2.equalizeHist(image)

#display histogram
image_rgb1 = cv2.cvtColor(equ, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb1)
plt.show()
#fuzzy enhancement
image_rgb1 = cv2.cvtColor(enhanced_image, cv2.COLOR_BGR2RGB)
plt.imshow(image_rgb1)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:







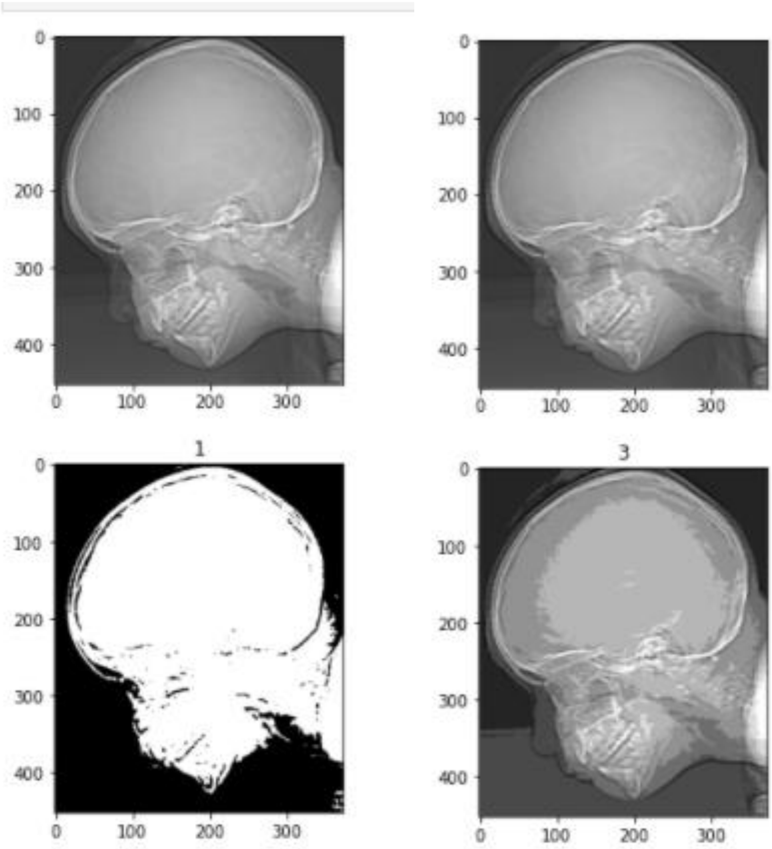
Task 25 : **Write a computer program capable of reducing the number of intensity levels in an image from 256 to 2, in integer powers of 2. The desired number of intensity levels needs to be a variable input to your program.**

Codes:-

```
import cv2
import numpy as np
import pandas as pd
from PIL import Image
from numpy import asarray
from matplotlib import pyplot as plt
img = cv2.imread('Fig0221(a)(ctskull-256).tif')
plt.imshow(img)
plt.show()
img1 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

level= 1  #input intensity level
k=8-level
intensity_level=2**k
img_reduce = np.uint8(np.floor(np.double(img1)/intensity_level))
norm_img=cv2.normalize(img_reduce,None,0,255,norm_type=cv2.NORM_MINMAX)
img2 = cv2.cvtColor(norm_img, cv2.COLOR_GRAY2RGB)
plt.imshow(img2)
plt.title(level)
plt.show()
```

SCREENSHOT JUPYTER NOTEBOOK:

Level: 1 and 3



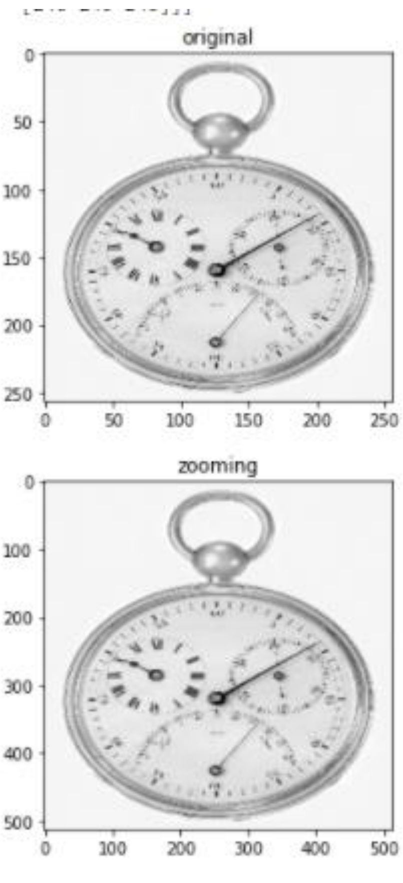
Task 26 : Interpolation for Zooming of an image as per the user’s choice of zooming factor.

Codes:

A) **Nearest neighbor**

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math
img = cv2.imread("Fig0220(a)(chronometer 3692x2812  2pt25 inch 1250 dpi).tif")
arr = cv2.resize(img,(256,256))
cv2.imshow("original",arr)
plt.show()
repetitions = 2
print("\n arr : \n", arr)
img = np.repeat(arr, repetitions, 1)
print("\nRepeating arr : \n", img)
c2 = np.repeat(img, repetitions, 0)
cv2.imshow("zoomed",c2)
cv2.waitKey(0)
```

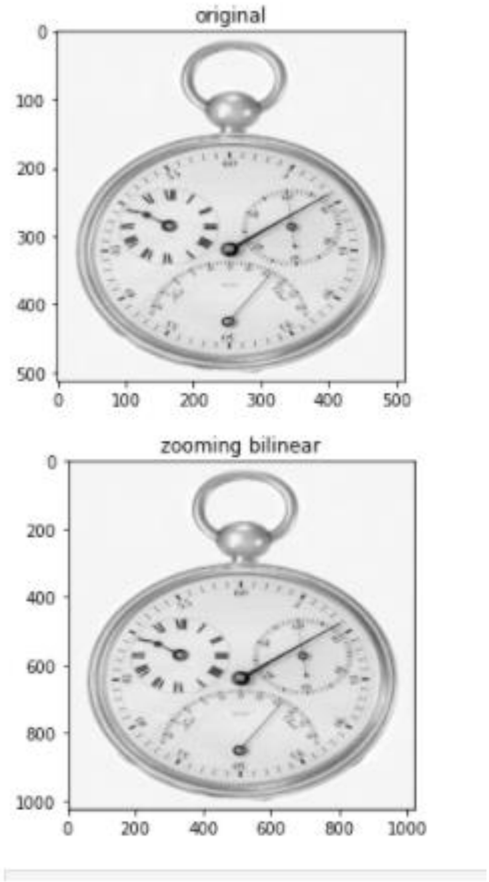
SCREENSHOT JUPYTER NOTEBOOK:



B) Bilinear

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

def bilinear_interpolation(img, scale):
   height,width =img.shape
   m,n = int(height),int(width*scale)
   new_img=np.zeros((m,n))
   for i in range(height):
      for j in range(width):
         p = i
         q = 2*j
         new_img[p,q] = int(img[i,j])
   for i in range(height):
      for j in range(width-1):
         new_img[i,((2*j)+1)] = int( (int(img[i,j])+int(img[i,j+1]))//2)

   new_img = new_img.astype(np.uint8)

   m,n = int(height*scale),int(width*scale)
   final_img = np.zeros((m,n))
   for i in range(int(m//2)):
      for j in range(n):
         p = 2*i
         q = j
         final_img[p,q] = new_img[i,j]

   for i in range(int(m/2-1)):
      for j in range(n):
         final_img[((2*i)+1),j] = int( (int(new_img[i,j])+int(new_img[i+1,j]))//2)

   final_img = final_img.astype(np.uint8)

   return final_img


img = cv2.imread('Fig0220(a)(chronometer 3692x2812  2pt25 inch 1250 dpi).tif')
img = cv2.resize(img,(512,512))
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
bilinear_img = bilinear_interpolation(gray_img, 2.0)
img2 = cv2.cvtColor(bilinear_img, cv2.COLOR_GRAY2RGB)
plt.imshow(img)
plt.title("original")
plt.show()
plt.imshow(img2)
plt.title("zooming bilinear")
plt.show()
```

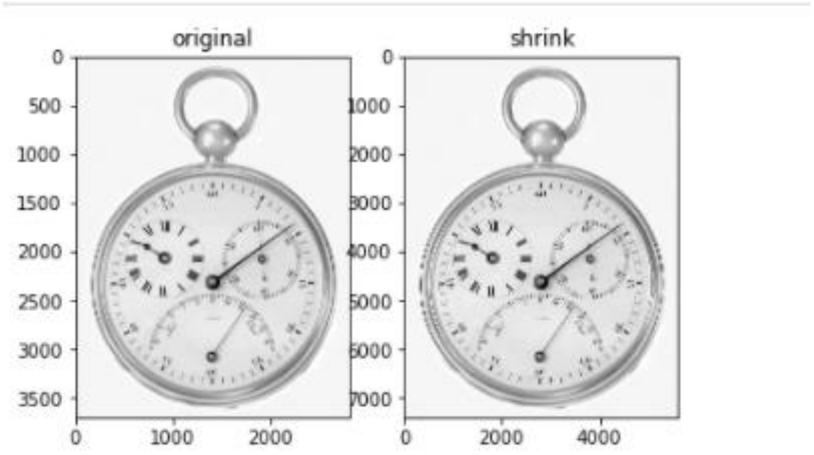
<u>SCREENSHOT JUPYTER NOTEBOOK:</u>



C)Bicubic

```
import numpy as np
import cv2
from math import sqrt,exp

from matplotlib import pyplot as plt

img = cv2.imread("Fig0220(a)(chronometer 3692x2812  2pt25 inch 1250 dpi).tif",0)
x,y=img.shape[0],img.shape[1]
bicubic_filter=np.array([[1,4,6,4,1],[4,16,24,16,4],[6,24,36,24,6],[4,16,24,16,4],[1,4,6,4,1]])
*(1/64)

zoom_factor=2
new_img=np.zeros([x*zoom_factor,y*zoom_factor]).astype('uint8')
x1,y1=new_img.shape[0],new_img.shape[1]
for i in range(x):
  for j in range(y):
    new_img[zoom_factor*i][zoom_factor*j]=img[i][j]

n=cv2.filter2D(new_img,-1,bicubic_filter)

plt.subplot(1,2,1)
plt.title("original")
plt.imshow(img,cmap='gray')
plt.subplot(1,2,2)
plt.imshow(n,cmap='gray')
plt.title("shrink")
plt.show()
```

<u>SCREENSHOT JUPYTER NOTEBOOK:</u>



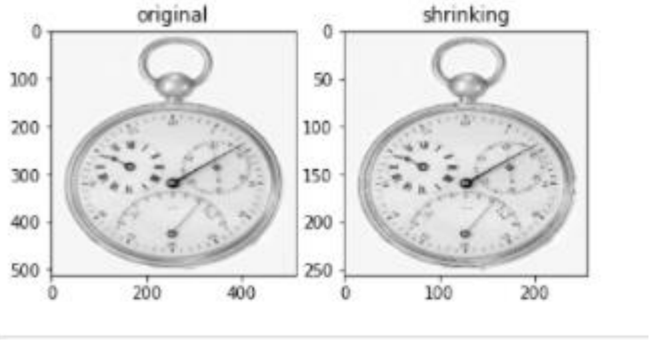
Task 27 : **Interpolation for Shrinking an image as per the user's choice of shrinking factor.**

Codes:

A) Alternative

```
import numpy as np
import cv2
from math import sqrt,exp

from matplotlib import pyplot as plt

img = cv2.imread("Fig0220(a)(chronometer 3692x2812  2pt25 inch 1250 dpi).tif",0)
img = cv2.resize(img,(512,512))
x,y= img.shape[0],img.shape[1]

shrink_factor=2
new_img=np.zeros([round(x/shrink_factor),round(y/shrink_factor)])
x1,y1=new_img.shape[0],new_img.shape[1]
for i in range(0,x1):
  for j in range(0,y1):

    new_img[i,j]=img[i*2][j*2]  #take only even col original image

plt.subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.subplot(1,2,2)
plt.imshow(new_img,cmap='gray')
plt.show()
```

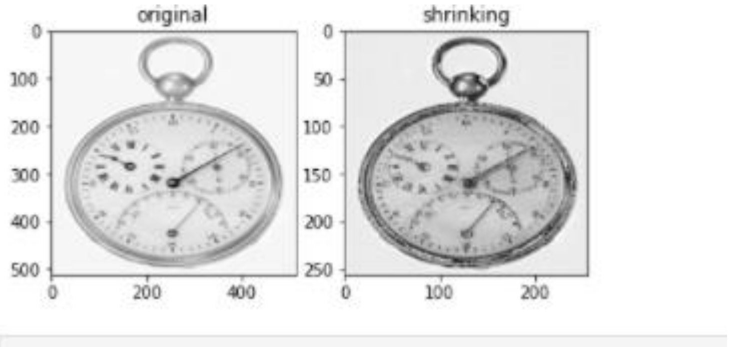
SCREENSHOT JUPYTER NOTEBOOK:



B) Average original image columns

```
import numpy as np
import cv2
from math import sqrt,exp

from matplotlib import pyplot as plt

img = cv2.imread("Fig0220(a)(chronometer 3692x2812  2pt25 inch 1250
dpi).tif",0)
img = cv2.resize(img,(512,512))
x,y= img.shape[0],img.shape[1]

shrink_factor=2
new_img=np.zeros([round(x/shrink_factor),round(y/shrink_factor)])
x1,y1=new_img.shape[0],new_img.shape[1]
for i in range(0,x1):
  for j in range(0,y1):
     val =((img[i*2-1][j*2-1]+img[i*2][j*2])/2)
     new_img[i,j]=val

plt.subplot(1,2,1)
plt.title("original")
plt.imshow(img,cmap='gray')
plt.subplot(1,2,2)
plt.title("shrinking")
plt.imshow(new_img,cmap='gray')
plt.show()
```

<u>SCREENSHOT JUPYTER NOTEBOOK:</u>


original
0
100
200
300
400
500
0
200
400
shrinking
0
50
100
150
200
250
0
100
200