Assignment 1

Neeraj Kumar (2020BSZ8607) bsz208607@iitd.ac.in
Krishna Chaitanya Reddy Tamataam(2018MT60785) mt6180785maths.@iitd.ac.in
Chayan Kumar Paul(2020EEZ7528) eez207528.@iitd.ac.in
IIT Delhi

1st, Septemeber 2020

1 Question 1

1.1 Python Code

```
from PIL import Image
import math
import numpy as np
import cv2
import matplotlib.pyplot as plt
import imutils


# use to translate the image
def translate(img, translate_x, translate_y):


    img_translate = np.zeros((img.shape[0] + translate_x, img.shape[1] +
        translate_x), dtype=np.int8)
    # create transformation matrix
    tranformation_mat = np.array(((1, 0, translate_x), (0, 1, translate_y), (0,
        0, 1)))
    tranformation_mat = np.linalg.inv(tranformation_mat)


    # we compute old and new origin
    old_image_centre_x, new_image_centre_x, old_image_centre_y,
        new_image_centre_y = img.shape[1]/2, img_translate.shape[1]/2, img.shape
        [0]/2, img_translate.shape[0]/2
    # we will compute offset as there will be the change of origin on
        translation
    orgin_offset = [new_image_centre_x - old_image_centre_x, new_image_centre_y
        - old_image_centre_y]


    for i in range(img_translate.shape[0]):
        for j in range(img_translate.shape[1]):
            # find inverse-mapped coordinates, currently in new origin
            arr = np.dot(tranformation_mat, np.array((j - new_image_centre_x +
                orgin_offset[0], new_image_centre_y - i + orgin_offset[1], 1)).T)
            # convert from new origin to previous origin (because need intensity
                values from input image)
```

```
            temp = arr[0]
            arr[0] = old_image_centre_y - arr[1]
            arr[1] = old_image_centre_x + temp
            # put only if lie in this range, the new image should be black in
                other places, to account for the transformation
            #
            if arr[0] >= 0 and arr[0] < img.shape[0] and arr[1] >= 0 and arr[1]
                < img.shape[1]:
                 img_translate[i][j] = img[int(arr[0])][int(arr[1])]
    # return new translated image
    return img_translate



def scale(img, scale_x, scale_y):
    # create image with sclar in horizontal and vertical direction with the
        defined value and fill the image with zero
    img_s = np.zeros((int(img.shape[0]*scale_x), int(img.shape[1]*scale_y)),
        dtype = np.int8)

    # We prevent thr black spot by inverse mapping it
    for i in range(img_s.shape[0]):
        for j in range(img_s.shape[1]):

            img_s[i][j] = img[int(i/scale_x)][int(j/scale_y)]
    # final scaled image
    return img_s



image_name='sample1.jpg'
gray_scale_array = cv2.imread(image_name, 0)


scale_x = 2
scale_y = 3
translate_x = 32
translate_y = 32
theta = 75



plt.imshow(gray_scale_array,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)

#scsled the original image
img_scale = scale(gray_scale_array, scale_x, scale_y)
img_scale_gray = Image.fromarray(img_scale, mode = 'L')
plt.imshow(img_scale_gray,cmap='gray', vmin=0, vmax=255)

plt.show(block=False)

#translated the function with 6 untis horizontal and 7 untis vertical
img_translate = translate(img_scale, translate_x, translate_y)

img_translate_gray = Image.fromarray(img_translate, mode = 'L')


plt.imshow(img_translate_gray,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)
plt.imsave('anv.jpg',img_translate_gray)
```

```
#image is rotated with 75 degree counter clockwise
img = cv2.imread('anv.jpg', 0)
# used function from imutisl
img_rotate_gray = imutils.rotate(img, theta)

plt.imshow(img_rotate_gray ,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)
```

1.2 Results

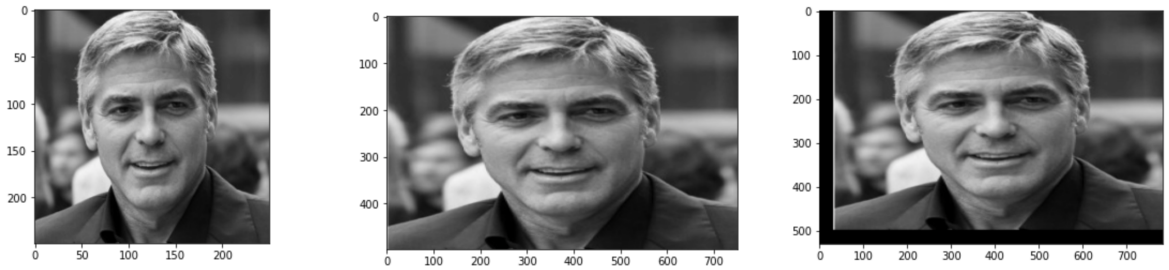

Figure 1: **Left: original Image(f(x,y)) , Centre : g1(x,y)= 3 times larger along y axis and 2 times along x axis , Right: g(x,y) = 6 units horizontal and 7 units vertical distance from f(x,y) performed on g1(x,y)**

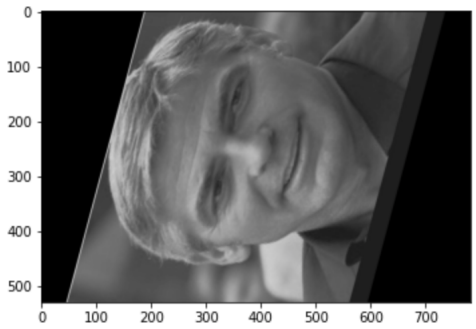

Figure 2: **h(x,y)=75 degree rotated counter clockwise on g(x,y)**

1.3 Observations

In figure 1 , the left side of the image is the original image(f(x,y)). Then the image is made 3 times larger along the yaxis and 2 times larger along the x axis which is shown in the centre of Figure 1. after that the image is translated 6 units horizontal and 7 units vertical distance from f(x,y) which is shown in Right part of Figure 1. The image is denoted as g(x,y).

The image(g(x,y)) is rotated counter clockwise direction to make it h(x,y) as shown in Figure 2.

2 Code2

2.1 a: Negative of image

2.1.1 Matlab Code

```
#calculate the negative of image with new_intensity = l-1-original_intensity
original_image = rgb2gray(imread("sample1.jpg"));
```

```
subplot(1, 2, 1),
 % displaying the RGB image
imshow(original_image);
title("Original image");
 % levels of the 8-bit image
L = 256;
 % finding the negative
negative_image= (L - 1) - original_image;
subplot(1, 2, 2),
 % displaying the negative image
imshow(negative_image);
title("Negative Image")
```

2.1.2 Results

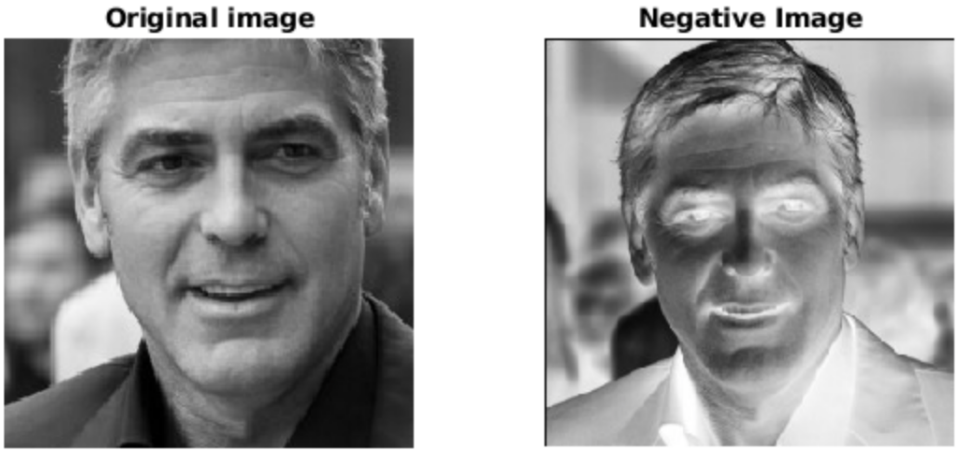

Figure 3: **Left:Actual image , Right: Negative Image**

2.1.3 Observation

In Figure 3, we see the negative image , where the darker regions in original iamge becomes light and lighter one becomes darker. This type of processing is used, for example, in enhancing white or gray detail embedded in dark regions of an image, especially when the black areas are dominant in size.

2.2 a: Log and Antilog of image

2.2.1 Matlab Code

```
#log of the image is calculated using s = log(r+1) .* ((L-1)/log(L))  and anti
    log is calculated using$s2 = (exp(r) .^ (log(L) / (L-1)))-1

L = 256;
image = rgb2gray(imread('sample1.jpg'));
log_image = uint8(log(double(image)+1) .* ((L - 1)/log(L)));
exp_image = uint8((exp(double(image)) .^ (log(L) / (L-1))) - 1);
imshow(exp_image)
```

2.2.2 Results

Figure 4: **Left:origin image,Right: log image, Bottom:anti log image**

2.2.3 Observation

In log transformation as shown in Figure 4 (Right side), we see this transformation maps a narrow range of low intensity values in the input into a wider range of output levels. For example, note how input levels in the range [0, L/4] map to output levels to the range [0, 3L/4]. Conversely, higher values of input levels are mapped to a narrower range in the output. We use a transformation of this type to expand the values of dark pixels in an image, while compressing the higher-level values.

In anti log as shown in Figure 4 (Bottom), this transformation maps the wider range of high intensity value to narrower range of output level. the input levels in the range [0, 3L/4] map to output levels to the range [0, L/4]. This is basically the inverse of log transformation.

2.3 Gamma correction image

2.3.1 Code

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

# fucntion to give gamma corrected image output
def gamma_corrected(img,gamma,image_name):
    # Define newimg
    # final image with all values zeroes
```

```
    gamma_image = np.zeros(img.shape,dtype=np.uint8)
    # mapping old image to newimage
    max_value = np.max(img)
    # filling the new image with the gamma corrected intensity value
    for j in range(gamma_image.shape[1]):
        for i in range(gamma_image.shape[0]):
            gamma_image[i][j] = int(255*((img[i][j]/255)**(gamma)))
    #plot the corrected gamma image
    plt.imshow(gamma_image,cmap='gray', vmin=0, vmax=255)
    plt.show(block=False)


#load the image
image_name='sample1.jpg'
img = cv2.imread(image_name, 0)

values_list=[0.4, 2.5, 10, 25, 100]
for values in values_list:
    gamma_corrected(img,values,image_name)
```

2.3.2 Results

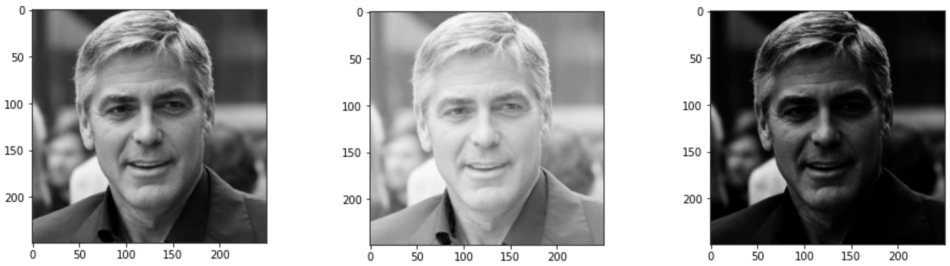

Figure 5: **original image, image with gamma=0.4, image with gamma=2.5**

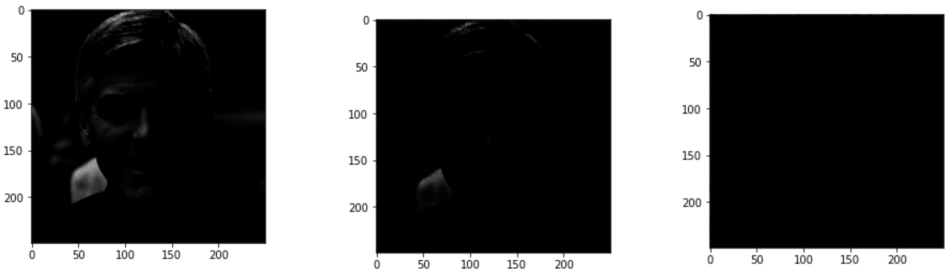

Figure 6: **image with gamma=10, image with gamma=25, image with gamma=100**

2.3.3 Observation

In the gamma transformation , the equation is s = cr(gamma) where c is taken 1 for the experiment. We can see that gamma with fractional values (gamma ¡ 1) map a narrow range of dark input values into a wider range of output values, with the opposite being true for higher values of input levels as shown in Figure 5. For gamma ¿ 1 , it tend tend to produce darker images as shown in Figure 5 and 6

2.4 Power of image

2.4.1 Python code

```
import cv2
```

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

# function to perform the power of an image
def power(img,power_value,image_name):
    # new image with zeroes value
    power_image = np.zeros(img.shape,dtype=np.uint8)

    max_value = np.max(img)
    #mapping the new value in the new image by using the power equation
    for j in range(power_image.shape[1]):
        for i in range(power_image.shape[0]):
            power_image[i][j] = int(255*((img[i][j]/255)**(power_value)))
    #ploting the graph
    plt.imshow(power_image,cmap='gray', vmin=0, vmax=255)
    plt.show(block=False)



image_name='sample1.jpg'
img = cv2.imread(image_name, 0)
plt.imshow(img,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)
values_list=[2,3,4]
for values in values_list:
    power(img,values,image_name)
```

2.4.2 Results

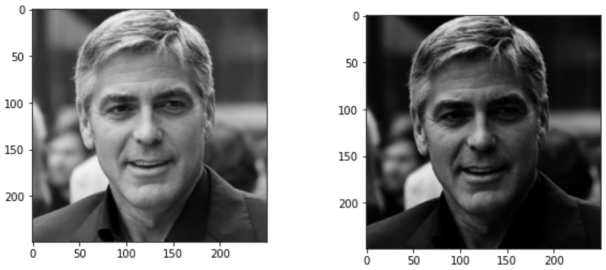

Figure 7: **Left:original image, Right: power image with power 2**

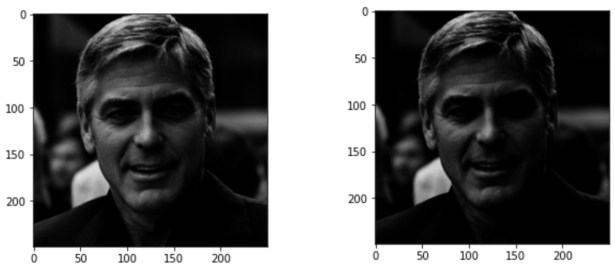

Figure 8: **Left:power image with power value 3 , Right: power image with power value 4**

2.4.3 Observation

The power of image , tend to produce darker image as it maps the broad range of higher intensity into the narrow range of lower intensity. This is also shown in Figure 7 and 8.

2.5 Plot bit plane

2.5.1 Matlab Code

```
% reading image's pixel in c
image = rgb2gray(imread('sample1.jpg'));
double_image = double(image);
% from bit1 to bit8
bit1 = mod(double_image, 2);
bit2 = mod(floor(double_image/2), 2);
bit3 = mod(floor(double_image/4), 2);
bit4 = mod(floor(double_image/8), 2);
bit5 = mod(floor(double_image/16), 2);
bit6 = mod(floor(double_image/32), 2);
bit7 = mod(floor(double_image/64), 2);
bit8 = mod(floor(double_image/128), 2);
% combining image again to form equivalent to original grayscale image
final_image = (2 * (2 * (2 * (2 * (2 * (2 * (2 * bit8 + bit7) + bit6) + bit5) +
    bit4) + bit3) + bit2) + bit1);
% plotting original image in first subplot
subplot(2, 5, 1);
imshow(image);
title('Original Image');
%plot all images
subplot(2, 5, 2);
imshow(bit1);
title('Bit  1');
subplot(2, 5, 3);
imshow(bit2);
title('Bit  2');
subplot(2, 5, 4);
imshow(bit3);
title('Bit  3');
subplot(2, 5, 5);
imshow(bit4);
title('Bit  4');
subplot(2, 5, 6);
imshow(bit5);
title('Bit  5');
subplot(2, 5, 7);
imshow(bit6);
title('Bit  6');
subplot(2, 5, 8);
imshow(bit7);
title('Bit  7');
subplot(2, 5, 9);
imshow(bit8);
title('Bit  8');
% plot the reconstructed image
subplot(2, 5, 10);
imshow(uint8(final_image));
title('Recombined Image');
```

2.5.2 Results

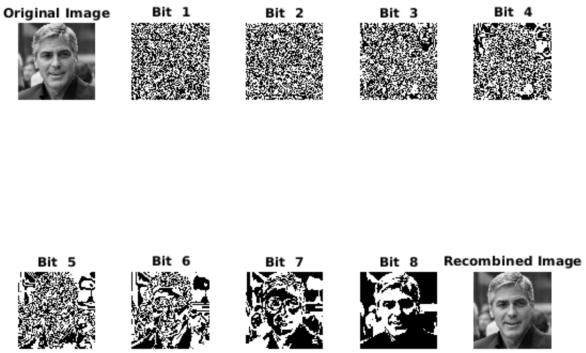

Figure 9: **origin image, log image, anti log image**

2.5.3 Observation

In bit plane slicing as shown in FIGURE 9 , it highlight the contribution made to total image appearance by specific bits. the four higher-order bit planes contain a significant amount of the visually-significant data as shown in bottom part of figure 9 . The lower-order planes (bit 1 to 4) contribute to more subtle intensity details in the image as shown in upper part of Figure 9.

2.6 Histogram and histogram equalizer

2.6.1 Code

```
from PIL import Image
import math
import numpy as np

import matplotlib.pyplot as plt

def histogram_equalization(img,image_name,CDF):
    map_array= [0]*256
    #finding mapping by the formulae s=(L-1)*CDF[r] where L is 256 in this case
    for i in range(256):
        map_array[i] = round(255*CDF[i])
    # define newimg
    equal_img = np.zeros(img.shape,dtype=np.uint8)
    #mapping old image to newimage
    for j in range(equal_img.shape[1]):
        for i in range(equal_img.shape[0]):
            equal_img[i][j] = map_array[int(img[i][j])]

    return equal_img



# function to construct histogram
def calculate_histogram(img):
    # intensity ranges from [0, 255]
    hist = [0]*256
```

```
    for j in range(img.shape[1]):
        for i in range(img.shape[0]):
            # count no of pixels with intensity img[i][j]
            hist[int(img[i][j])] += 1
    return hist


# function to plot bar graph with x, y values
def plot_diagram(y_values):
    B = range(256)
    plt.bar(B,y_values, color='g')
    plt.xlabel('Intensity value')
    plt.ylabel('Frequency')
    plt.title('Histogram')



import cv2
image_name='sample1.jpg'
img = cv2.imread(image_name, 0)

plt.imshow(img,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)

histogram = calculate_histogram(img)

plot_bar(histogram)
plt.show(block=False)

equalized = cv2.equalizeHist(img)

plt.imshow(equalized,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)

histogram = calculate_histogram(equalized)

plot_bar(histogram)
plt.show(block=False)
```

2.6.2 Results

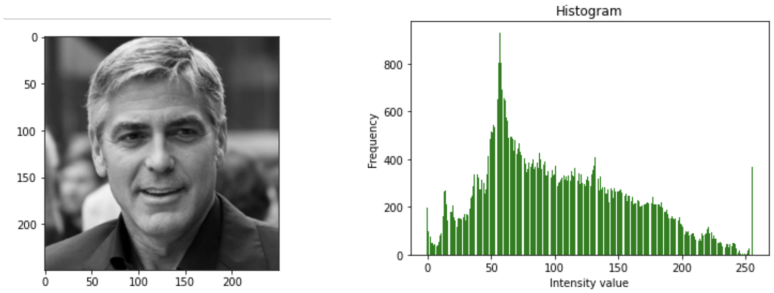

Figure 10: **origin image and histogram**

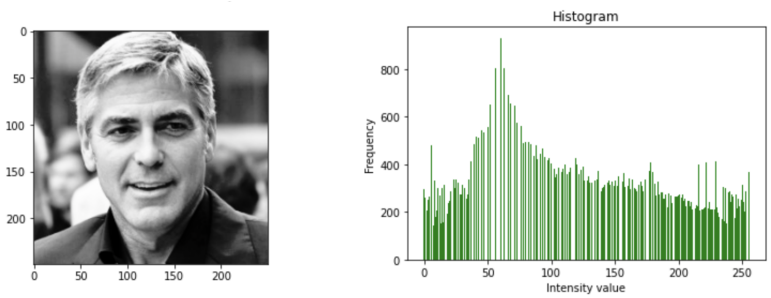

Figure 11: **Histogram equalized image and original image**

2.6.3 Observation

Histogram equalization makes contrast adjustment using the image's histogram. In figure 11 , we can see in histogram , the intensity values are better distributed.the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

2.7 Highlight image where intensity is [120,200] and same elsewhere

2.7.1 Code

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

#function to highlight the image where the values less than 120 and greater than
    . 200 are mapped to same value and the in between are mapped to 255 . SP that
     it is highlighted as white. This phenomenonis called intensity level slicing
    .
def highlight(x, minval, maxval):

    if(x <minval):
        return x
    elif(x>=minval and x<=maxval):
        return 255
    else:

        return x


def highlight_image(img):
    minval = 120
    maxval = 200

    #define new image with all zeores
    newimg = np.zeros(img.shape,dtype=np.uint8)

    # perform the highlighting operation and make the new image
    for j in range(newimg.shape[1]):
        for i in range(newimg.shape[0]):
            newimg[i][j] = int(highlight(img[i][j],minval,maxval))

    #plot the original image
    plt.imshow(newimg,cmap='gray', vmin=0, vmax=255)
    plt.show(block=False)
    #plot the highlighted image
    plt.imshow(img,cmap='gray', vmin=0, vmax=255)
```

```
#load the image in gray scale
image_name='sample1.jpg'
image1 = cv2.imread(image_name, 0)

highlight_image(image1)
```

2.7.2 Results

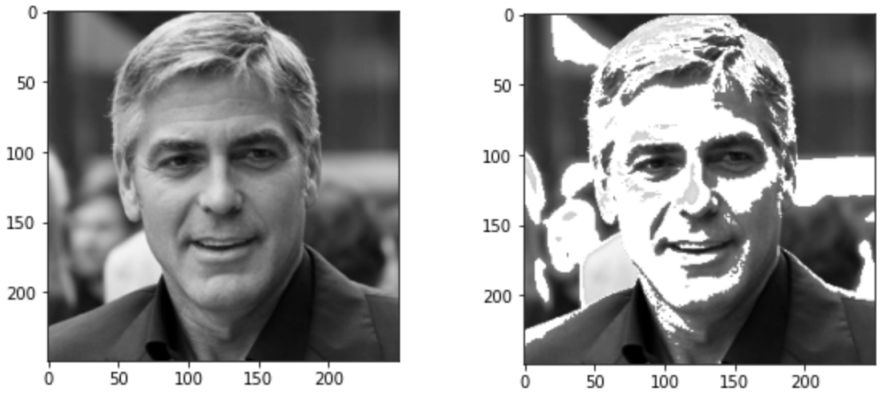

Figure 12: **Left:original image ,Right: highlighted image**

2.7.3 Observation

In figure 12 , we highlight the intensity level in range 120 to 200 to new intensity value i.e. 255 and rest value remain unchanged. This method is called intensity level slicing where we brighten the desired range of intensities, but leaves all other intensity levels in the image unchanged.

3 Question 3

3.1 Question 3.a - histogram equilizer without inbuilt function

3.1.1 Python Code

```
from PIL import Image
import math
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt
import matplotlib.image as mpimg


import matplotlib.pyplot as plt

def histogram_equalization(img,image_name,CDF):
    map_array= [0]*256
    #finding mapping by the formulae s=(L-1)*CDF[r] where L is 256 in this case
    for i in range(256):
        map_array[i] = round(255*CDF[i])
    # define newimg
    equal_img = np.zeros(img.shape,dtype=np.uint8)
    #mapping old image to newimage
```

```
    for j in range(equal_img.shape[1]):
        for i in range(equal_img.shape[0]):
            equal_img[i][j] = map_array[int(img[i][j])]

    return equal_img



# function to construct histogram
def calculate_histogram(img):
    # intensity ranges from [0, 255]
    hist = [0]*256
    for j in range(img.shape[1]):
        for i in range(img.shape[0]):
            # count no of pixels with intensity img[i][j]
            hist[int(img[i][j])] += 1
    return hist

# function to find the cummulative distribution function of image
def claculate_CDF(hist, gray_scale_array):
    # total number of pixels in grayscale image
    no_of_pixel = gray_scale_array.shape[0]*gray_scale_array.shape[1]
    hist = [i*1.0/no_of_pixel for i in histogram]
    CDF = [0]*256
    temp = 0
    for i in range(256):
        temp += hist[i]
        CDF[i] = temp
    return CDF

# function to plot bar graph with x, y values
def plot_diagram(y_values):
    B = range(256)
    plt.bar(B,y_values, color='g')
    plt.xlabel('Intensity value')
    plt.ylabel('Frequency')
    plt.title('Histogram')


images = loadmat('testimage.mat')
gray_scale_array = images['im']


plt.imshow(gray_scale_array,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)


histogram = calculate_histogram(gray_scale_array)

plot_diagram(histogram)
plt.show(block=False)



CDF = claculate_CDF(histogram,gray_scale_array)

equal_img = histogram_equalization(gray_scale_array,image_name,CDF)

plt.imshow(equal_img,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)
```

```
#Getting PIl object from np array
histogram2 = calculate_histogram(equal_img)
plot_diagram(histogram2)
plt.show(block=False)
```

3.1.2 Results

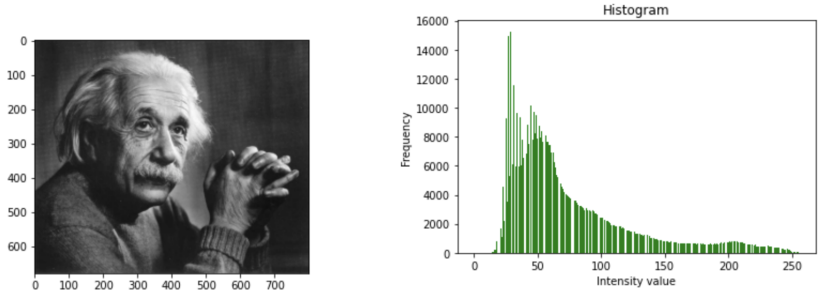

Figure 13: **Left:original image , Right: histogram**

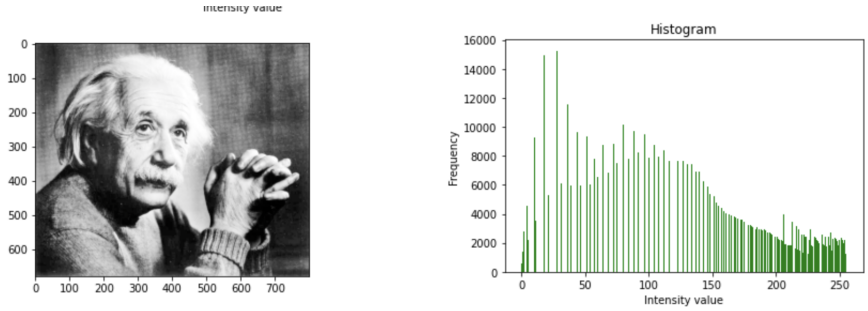

Figure 14: **Left:Equalized image, Right: histogram**

3.1.3 Observation

In Figure 14 , we see the equalized image where the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast as shown in Figure 14 .

3.2 Question3.b - In built function for histogram equilizewr and histogram

3.2.1 Code

```
import cv2
import matplotlib.pyplot as plt

#inbuilt function of histogram equalizer
equalized_inbuilt = cv2.equalizeHist(gray_scale_array)

#plot the equalized image
plt.imshow(equalized_inbuilt ,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)

#calculate the histogram of equalized image
histogram = calculate_histogram(equalized_inbuilt)

plot_bar(histogram)
plt.show(block=False)
```

3.2.2 Results

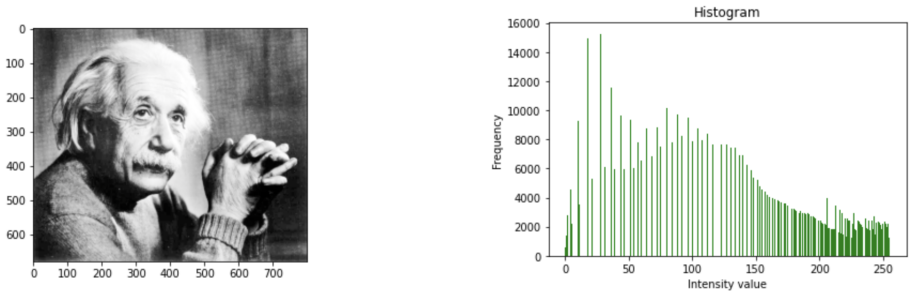

Figure 15: **Left: Equalized image with inbuilt function , Right: Histogram**

3.2.3 Observation

Figure 15 shows the equalized image using the inbuilt function of opencv library. We can see that the intensities are better distributed inthe histogram.

3.3 question3.c - Adaptive histogram equalizer

3.3.1 Python code

```
# clahe inbuilt finction is used
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
equalized_clahe = clahe.apply(gray_scale_array)

#plot the equalized output image
plt.imshow(equalized_clahe ,cmap='gray', vmin=0, vmax=255)
plt.show(block=False)

#calculate the histogram of equalized output image
histogram = calculate_histogram(equalized_clahe)

plot_bar(histogram)
plt.show(block=False)
```

3.3.2 Results

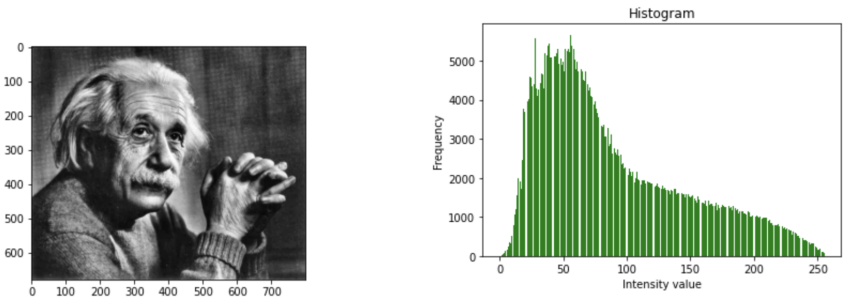

Figure 16: **Left: Adaptive Equalized image with clahe function , Right: histogram**

3.3.3 Observation

In Figure 16, we see that the adaptive equalized method improve the local contrast and enhancing the definitions of edges in each region of an image. The adaptive method computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image.

3.4 Mean square erroor

3.4.1 Python code

```
# mean square value is sum of squared differences between two image
def mse(imageA, imageB):

    err = np.sum((imageA - imageB) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])

    return err

mse(equalized_inbuilt,equal_img)
mse(equalized_inbuilt,equalized_clahe)
mse(equal_img,equalized_clahe)
```

3.4.2 Results and Observation

- Means square error between the equalized histogram image with inbuilt function and non inbuilt function is 0
- Means square error between the equalized histogram image with inbuilt function and adaptive equalized image is 105.37117452135493
- Means square error between the equalized histogram image with non inbuilt function and adaptive equalized image is 105.37117452135493