2019702002
Nivedita Rufus

Digital Image Processing
Assignment 1

—

INTRODUCTION

In this assignment we implement a variety of basic image manipulation techniques such as:

- Chroma keying
- Contrast stretching
- Quantisation
- Intensity transformation
- Histogram equalization and matching

Q1)

1) **Write a function that takes a color image and finds the most frequently occurring color from the image**

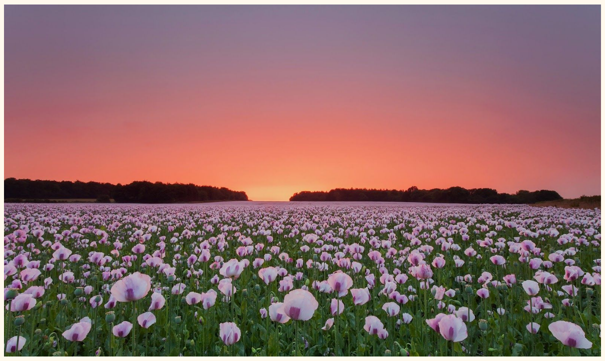
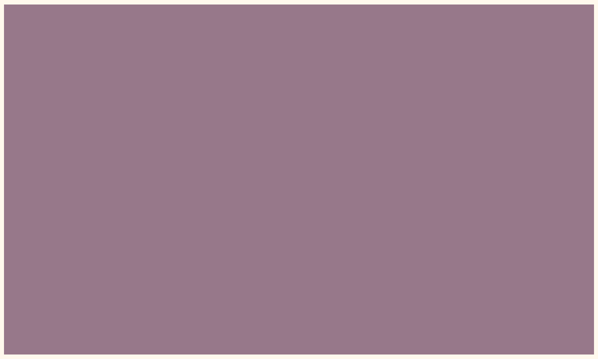
Approach:

The approach implemented to get the most frequent color of an image was the use of Kmeans clustering. The centroid of the largest cluster will be the most frequently occurring color in the image.

Steps:

- Compute clusters using Kmeans clustering with k=3 for the pixel intensities of the image.
- Assign the cluster index for each sample.
- Return the most common occurring cluster center i.e the centroid of the largest cluster.

```
def get_frequent_color(image, k=3):
    image = image.reshape((image.shape[0] * image.shape[1], 3))
    clt = KMeans(n_clusters = k)
    labels = clt.fit_predict(image)
    label_counts = Counter(labels)
    dominant_color=clt.cluster_centers_[label_counts.most_common(1)[0][0]]
    return list(dominant_color)
```

Input:	Output: Most frequent color
	

Q1)

2) Write a function mergeImage which takes two images fg and bg that extracts the foreground object and places it in the background and returns the resultant image.

Approach:

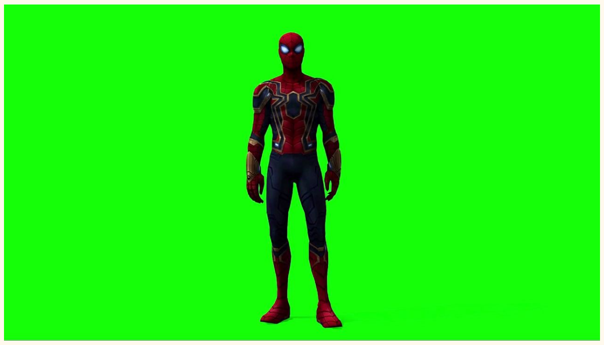
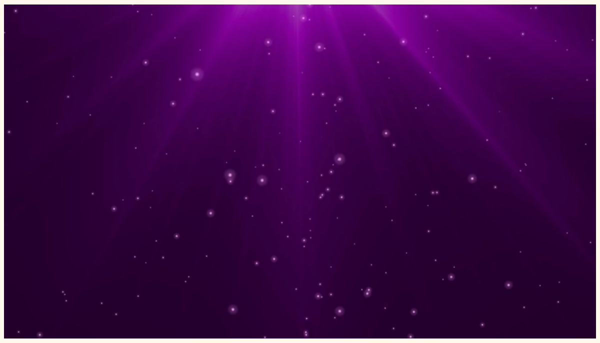
The approach implemented was to generate the alpha matrix for the given image(foreground) using Vlahos matting:

“ The Vlahos formula for αo, abstracted from the claims of his earliest electronic patent [18] and converted to our notation, is αo = 1 – a1(Bf – a2Gf), clamped at its extremes to 0 and 1, where the ai are tuning adjustment constants.”

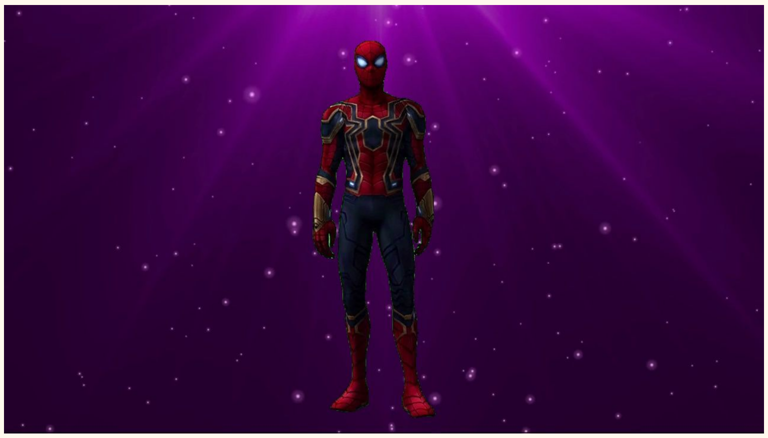
Steps:

- Compute the alpha matrix for the foreground image using Vlahos Matting formula.
- Multiply the foreground image with the computed alpha matrix.
- Assign background image image values for pixels, where αo = 0 respectively.

```
def mergeImage(f_image,b_image):
      height, width, channels = f_image.shape
      a1 = .01
      a2 = 2
      alpha = numpy.ones((height, width))
      for x in range(0,height):
            for y in range(0,width):
                  pixel = f_image[x,y]

                  B = pixel[0]
                  G = pixel[1]

                  alpha[x,y] = 1-(a1*(G-(a2*B)))

                  if alpha[x,y]<0:
                        alpha[x,y]=0
                  if alpha[x,y]>1:
                        alpha[x,y]=1

                  f_image[x,y] = alpha[x,y]*f_image[x,y]

                  if alpha[x,y]==0:
                        f_image[x,y] = b_image[x,y]
      return f_image
```

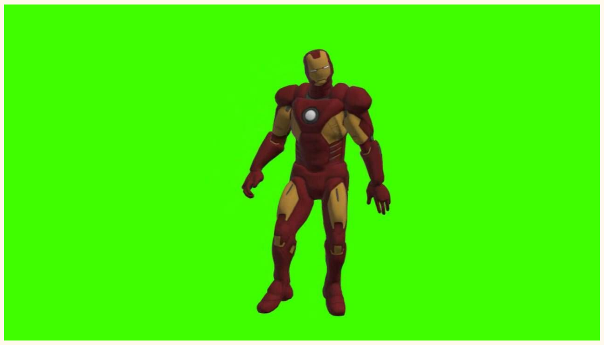
Foreground:	**Background:**
	

Merged image:



Q1)

3) **Try out with different foreground and background images of your choice and show the results.**

Foreground:	Background:
	

Merged Image:

Q2)

1) **Write a function linContrastStretching which takes a grayscale image im, a and b that enhances the contrast such that the resulting intensity range is [a, b]**

Approach:

The approach implemented to increase the contrast of the given image to the desired range [a,b] was:

r = (r - alow)*(b-a)/(ahigh - alow)

Where, r = value of the pixel intensity

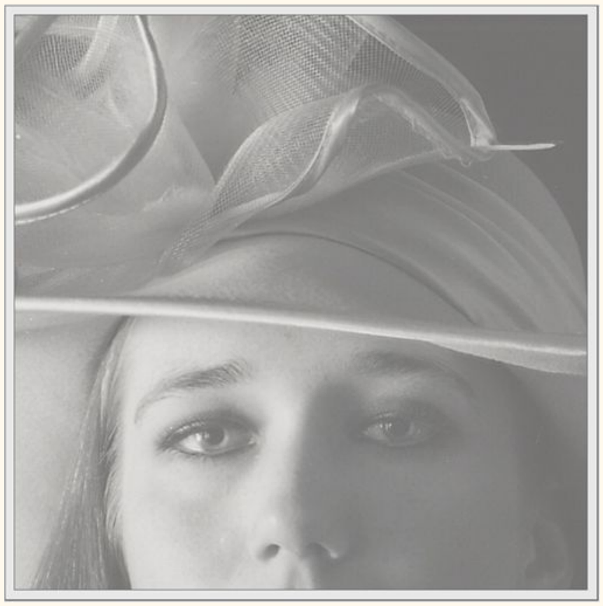
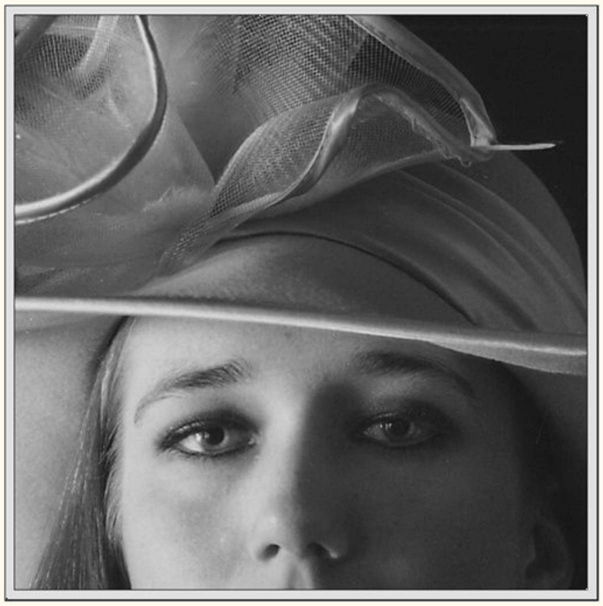
alow = lowest pixel value in image

ahigh = highestt pixel value in image

Steps:

- Compute alow and ahigh from the image
- Assign each pixel in the image with the value generated based on the formula given above

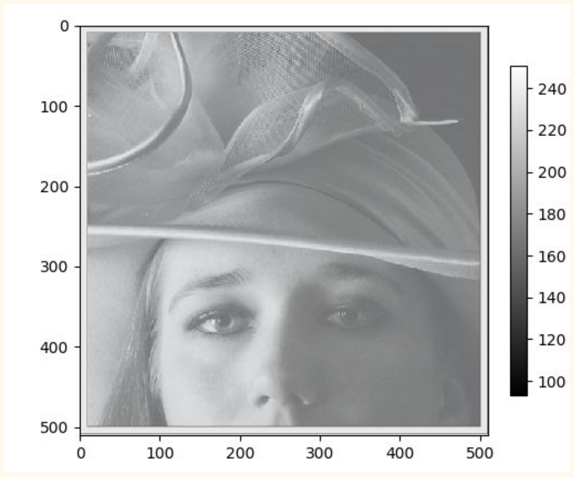
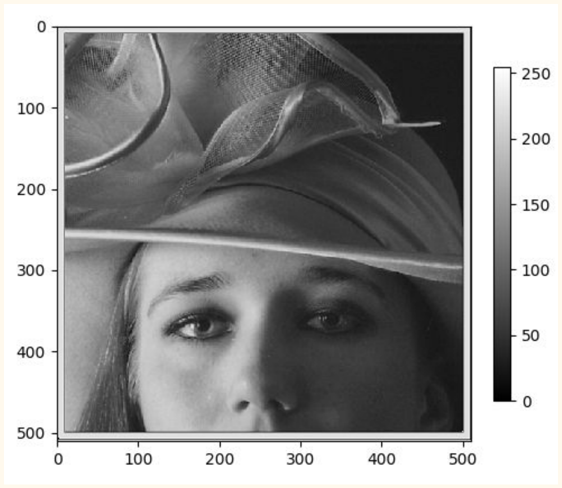
```python
def linContrastStretching(im, a=0, b=255,ahigh,alow):
      for x in range(im.shape[0]):
            for y in range(im.shape[1]):
                  im[x][y] =  ((im[x][y] - alow)*(b-a)/(ahigh - alow))
      return im
```

Input:	Output:
	

Q2)

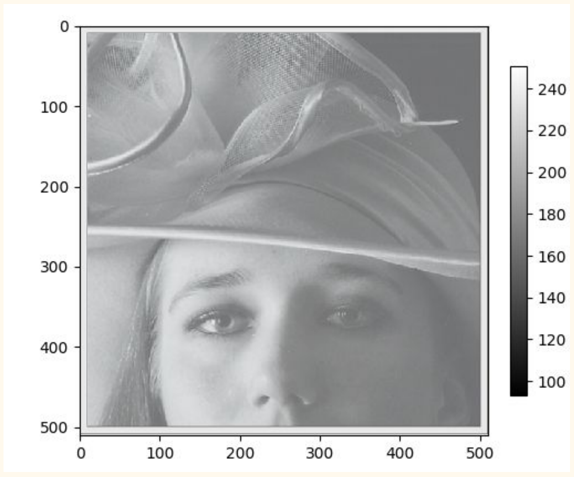
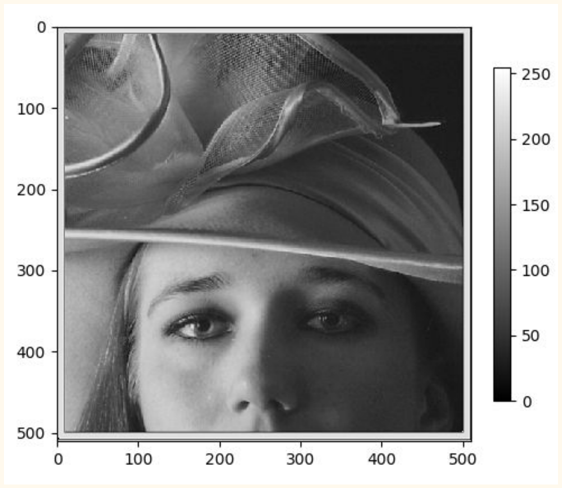
2) **Display the input image and the resultant image side-by-side along with their colorbars (a strip containing k most frequently occurring colors). Give suitable explanation for the resulting colorbars.**

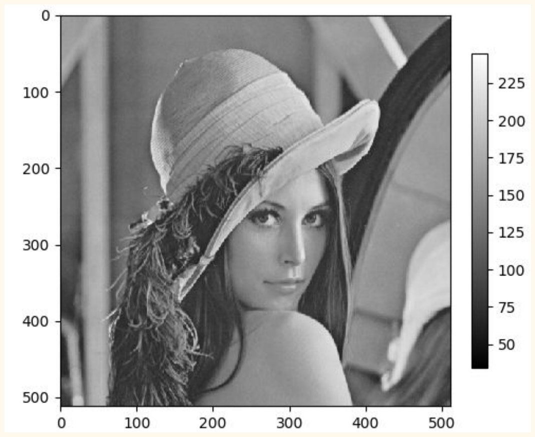
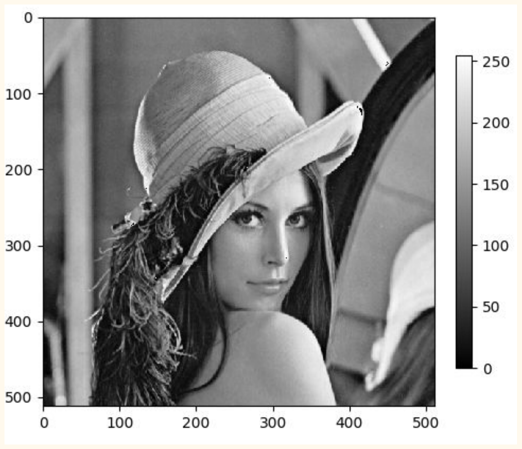
The colorbars of the images represent the range of grey values present in them. For the input image it can be noticed that the range of intensities present in the color bar is quite narrow i.e. (100-240 approx.) but after subjecting this image to linear contrast stretching the colorbar of the resultant image covers a whole wide range all the grey levels i.e (0-255). The contrast has improved which can also be visualised from the images.

Input:	Output:
	

Q2)

3) **Use your function on multiple images and argue why the effect is more on some images while it is not that apparent on the others.**

Input:	Output:
	

 0 100 200 300 400 500 0 100 200 300 400 500 225 200 175 150 125 100 75 50	 0 100 200 300 400 500 0 100 200 300 400 500 250 200 150 100 50 0

From the two given inputs, it is very apparent that the effect of the linear contrast stretching is more pronounced in input(1) than in input(2)(*lena*). The reason for this can be inferred from their respective colorbars. For the input(1) the colorbar has a very narrow range i.e (100-240) wherein for input(2) the range though doesn't cover all 256 grey levels is much more of a broad spectrum than input(1) i.e (50-230); Hence, the effect is more pronounced for images with a very narrow spectrum of intensity levels.

Q3)

1) **Write a function BitQuantizeImage which takes an 8-bit image im and k, the number of bits to which the image needs to be quantized to and returns the k-bit quantized image. Display results for the image quantize.jpg**

Approach:

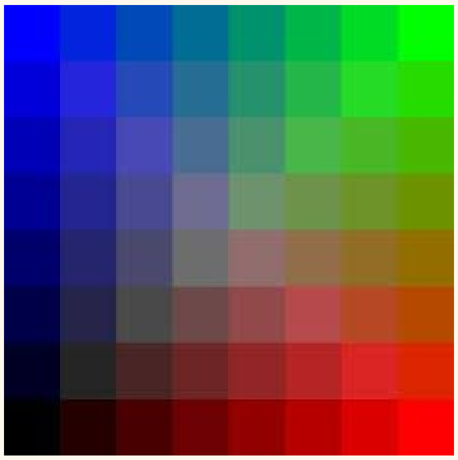
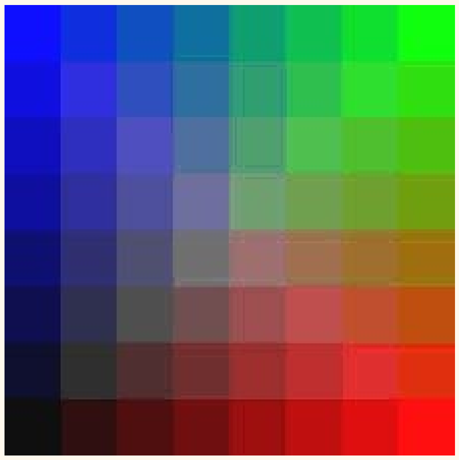
The approach used in the quantisation of the given image was to first compute the value of the number of grey values present in the quantised image and then map all intensities to these levels respectively

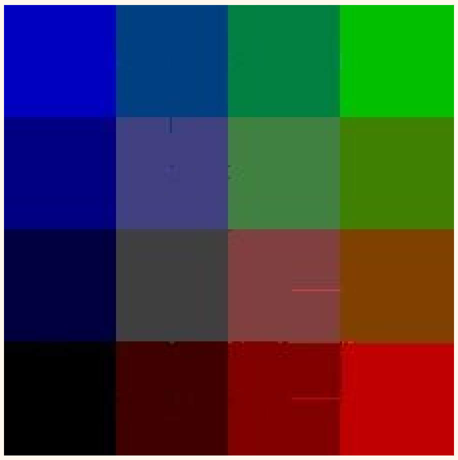
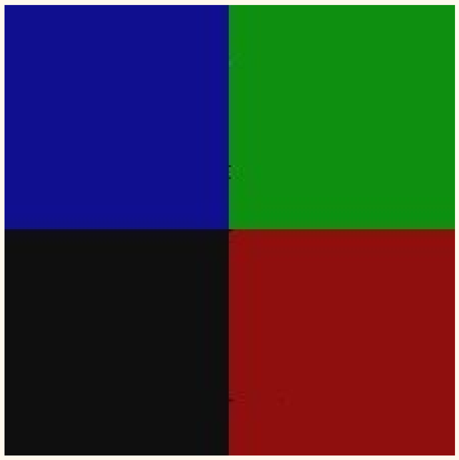
Steps:

- Compute no. of gray levels to be present in the quantised image after getting no. of bits to which image needs to quantised.
- Divide each intensity by the value computed above and round it off to the nearest integer.

- Finally multiply the value generated for each pixel in step 2 with the value computed from step1 and map it to that pixel respectively.

```python
def BitQuantiseImage(image,bits=4):
      k= int(256/(math.pow(2,bits)))
      for x in range(image.shape[0]):
            for y in range(image.shape[1]):
                  image[x][y][0] = int(k*int(int(image[x][y][0])/k))
                  image[x][y][1] = int(k*int(int(image[x][y][1])/k))
                  image[x][y][2] = int(k*int(int(image[x][y][2])/k))

      return image
```

Input:	**Output: 4 bits**
	

Output: 2 bits	Output: 1 bit
	

Q3)

2) **Write a code to display different bit planes of an 8-bit gray-scale image. Display results for the image cameraman.png**

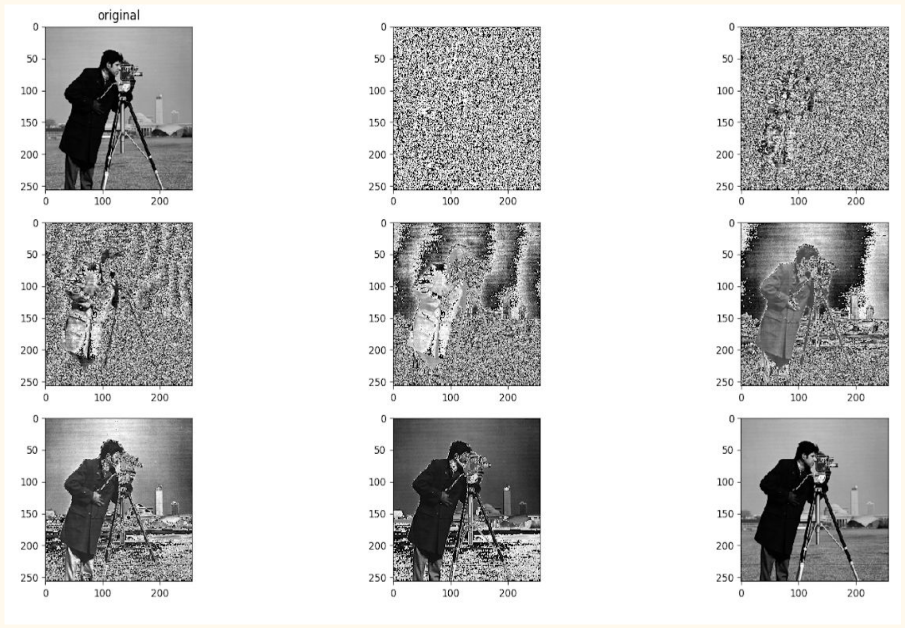
Approach:

The approach used was to divide the image with powers of two to obtain each bit plane of the image.

Steps:

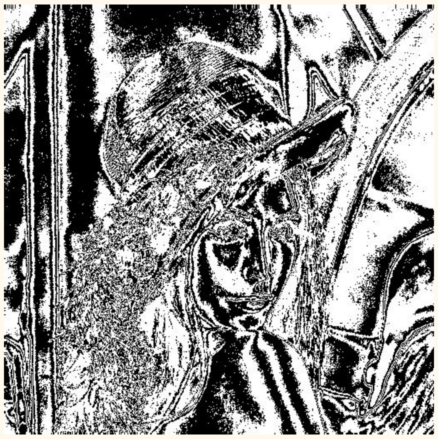
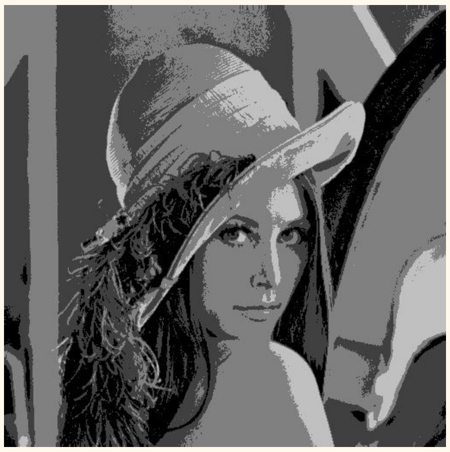
- Compute the value of each intensity by iteratively storing the respective values
- Replace the intensity in the image by each of the computed values for that respective bit plane.

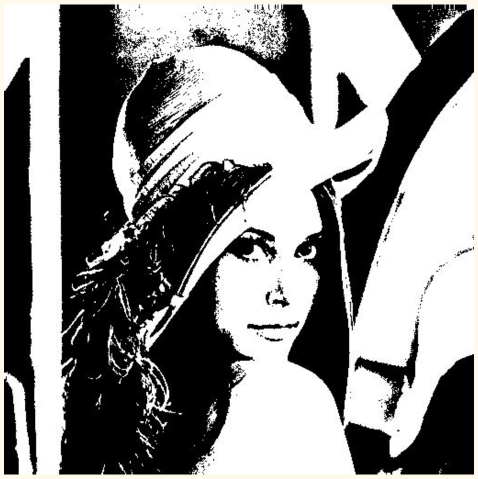
```
def bit_plane(image,bits=8):
      plane = np.empty([image.shape[0],image.shape[1],bits])
      for x in range(bits):
            plane[:,:,x] = (image/math.pow(2,x))%2
      return plane
```

Input:	Output:
	

Q3)

3) **Given original image lena.jpg identify the operations applied on the images lena1.jpg, lena2.jpg and lena3.jpg**

Input:	**Output:**
	**The image is subjected to contrast stretching for a very narrow range of intensities followed by thresholding to 0 and 255** **Reason:** the final image has only black and white values, so clearly thresholding is done on the processed image which has been subjected to contrast stretching to have the image final intensity range to be narrow and lower.
	**The image has been subjected to quantisation.** **Reason:** False contouring is observed

	**The image is subjected to thresholding, i.e all values less than 127 is mapped to 0 and others to 255** **Reason:** only two intensity values are observed

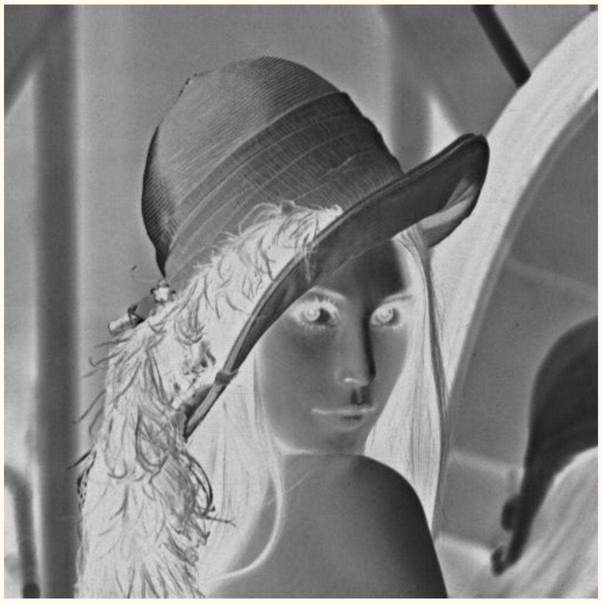
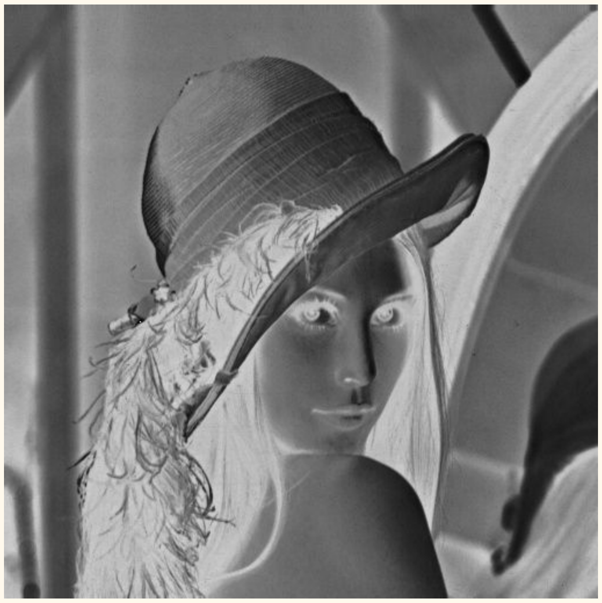
Q4)

1) **Write a function to create the negative of an image. The function should take the image and maximum intensity as arguments. Produce the transformed output for first 8 k-bit quantized forms of lena.jpg.**

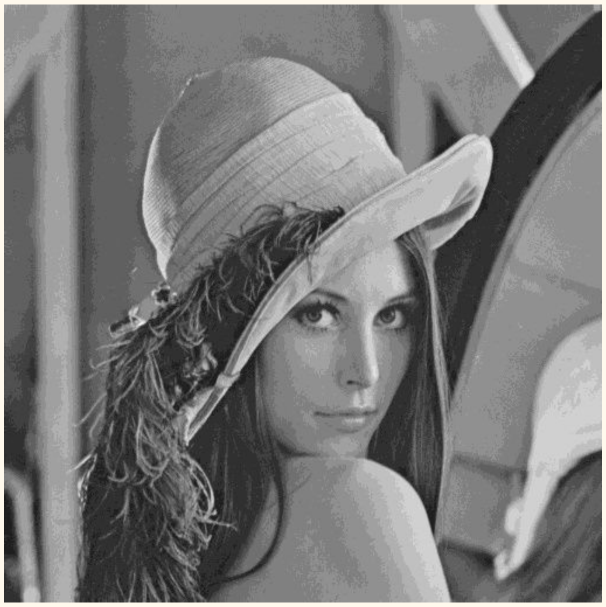
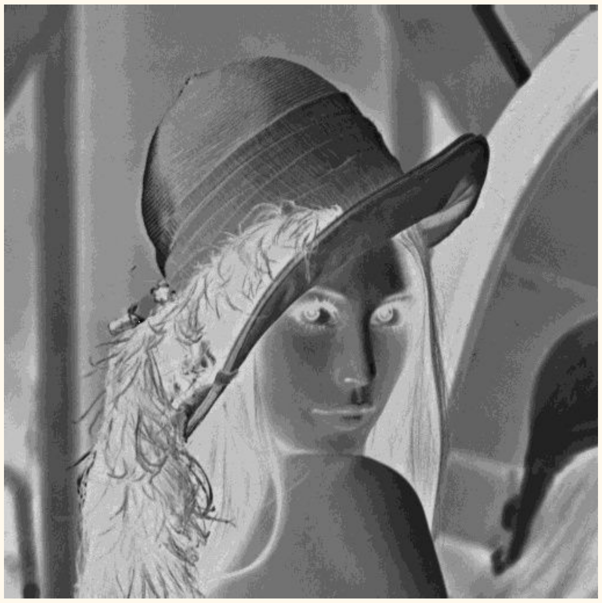
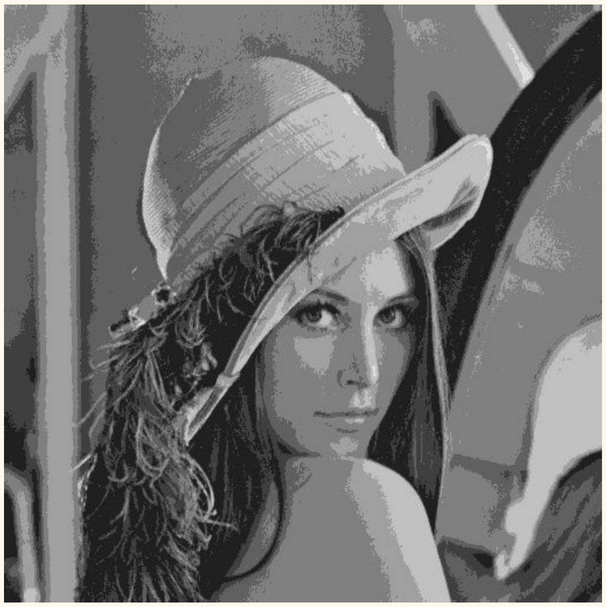
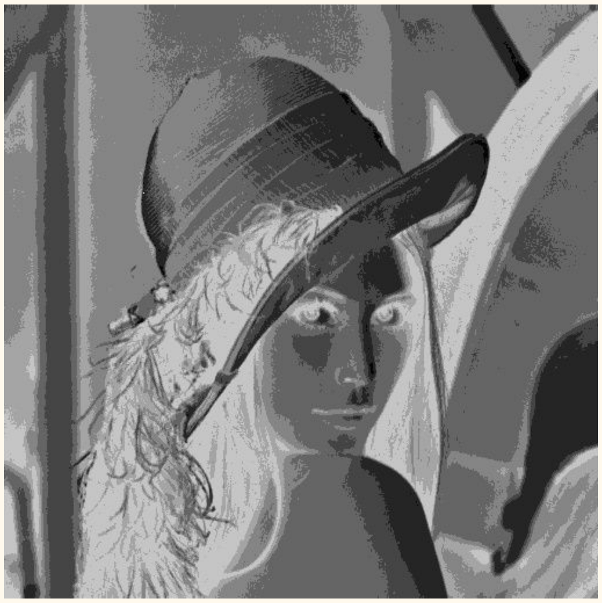
Approach:

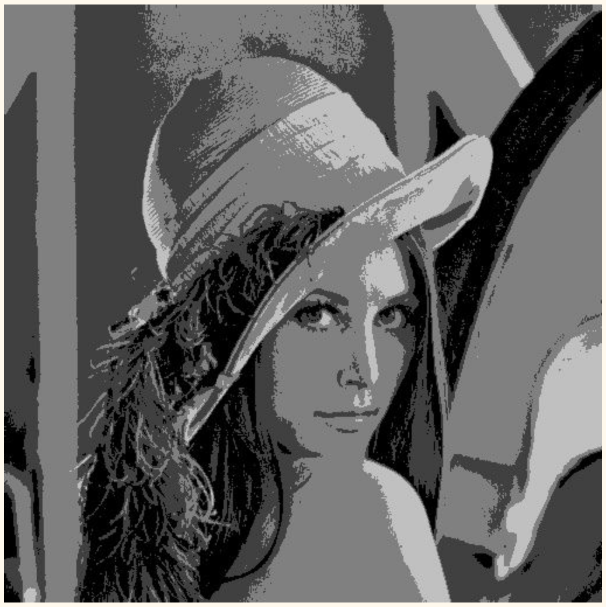
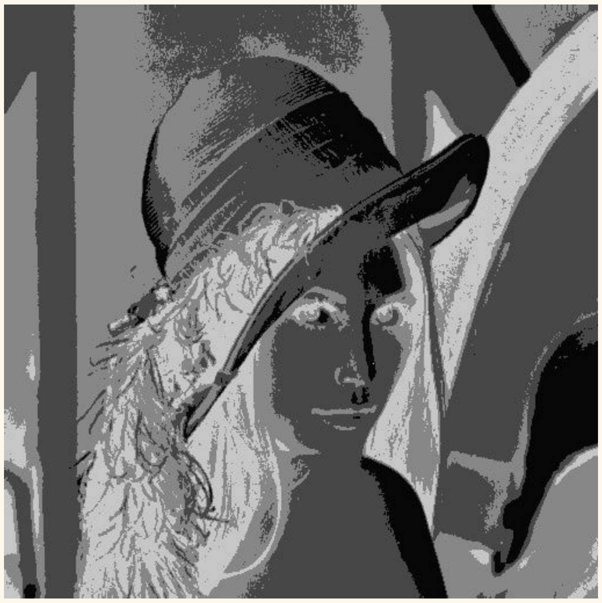
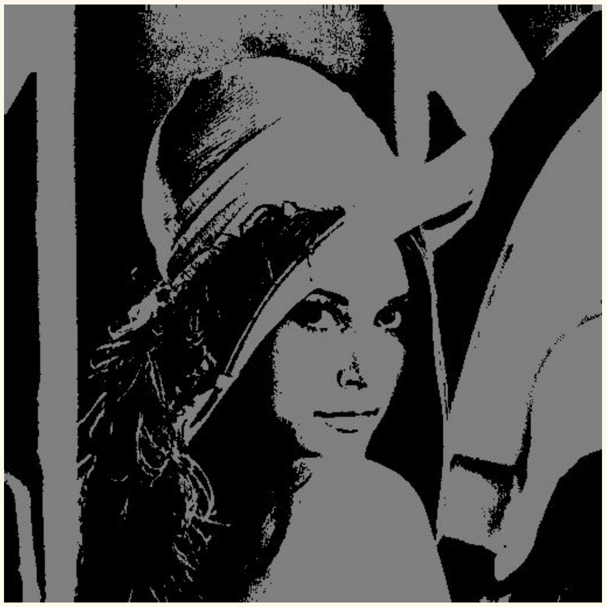
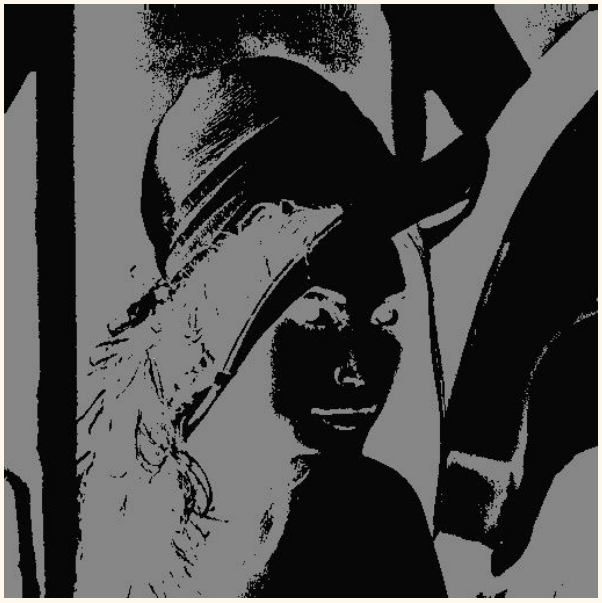
The approach used was to replace every pixel intensity ‘r’ with its negative i.e. ‘L - r’ where L = 255 for all the quantised forms of lena.jpg

```
def negativeImage(image, maxintensity = 255):
      for x in range(image.shape[0]):
            for y in range(image.shape[1]):
                  image[x][y][0] = maxintensity - int(image[x][y][0])
                  image[x][y][1] = maxintensity - int(image[x][y][1])
                  image[x][y][2] = maxintensity - int(image[x][y][2])

      return image
```

Input:	Output:
8-bits	
7-bits	

6-bits	
5-bits	

4-bits 	
3-bits 	

2-bits 	
1-bit 	

Q4)

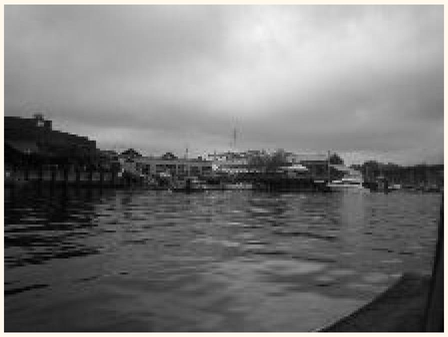
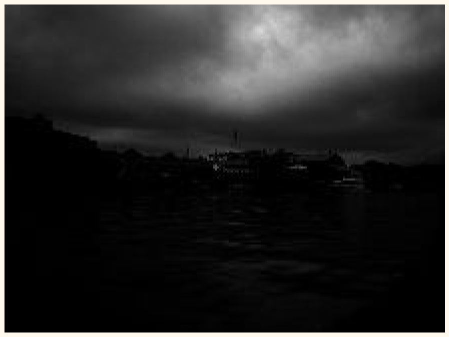
2) **Using the image gamma-corr.png, apply the Gamma Transform s = r γ and vary γ. Report your observations.**

Approach:

The approach used was to replace every pixel intensity '***r***' with '***s***', where

$$s = r^{\gamma}$$

```python
def gamma_corr(img,gamma_value=1):
      image = np.array(255*(img/255)**gamma_value,dtype='uint8')

      return image
```

Input:	**Output: gamma = 0.1**
	
Output: gamma = 2.2	**Output: gamma = 4**

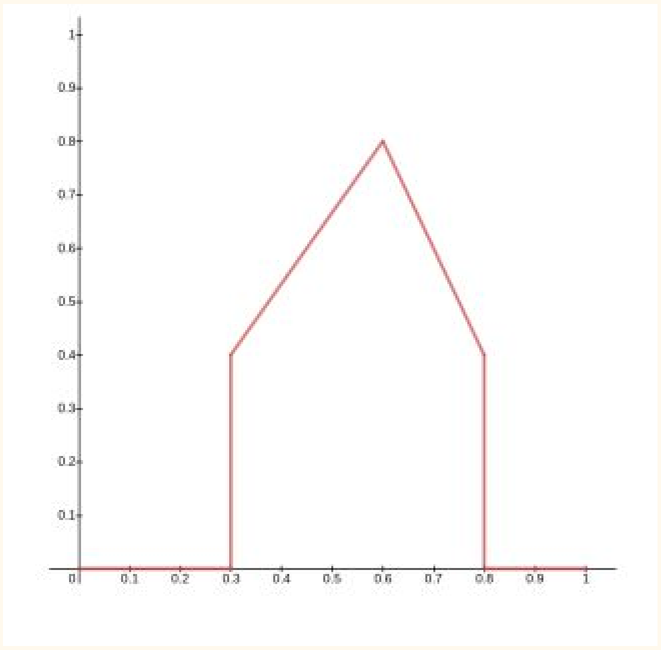
Observations:

Gamma was varied in increasing order from gamm = 0.1 to gamma = 4. It was observed that as the value of Gamma increases the images tends to become more and more brighter.

Q4)

3) **Write a function to implement a piecewise linear transform for interval (a,b) for each linear segment and produces the transformed output image. Produced transformed outputs of lena.jpg for the following functions:**

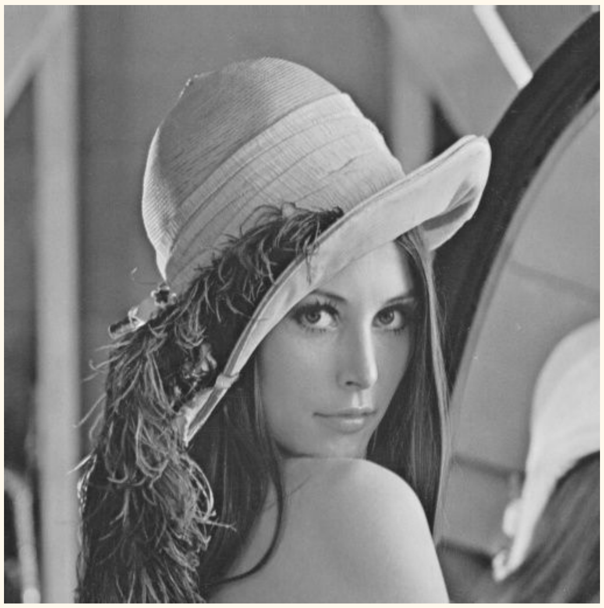
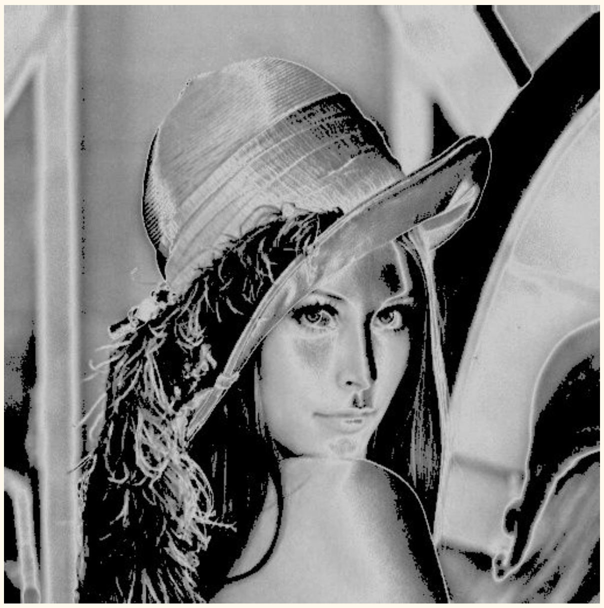
a)



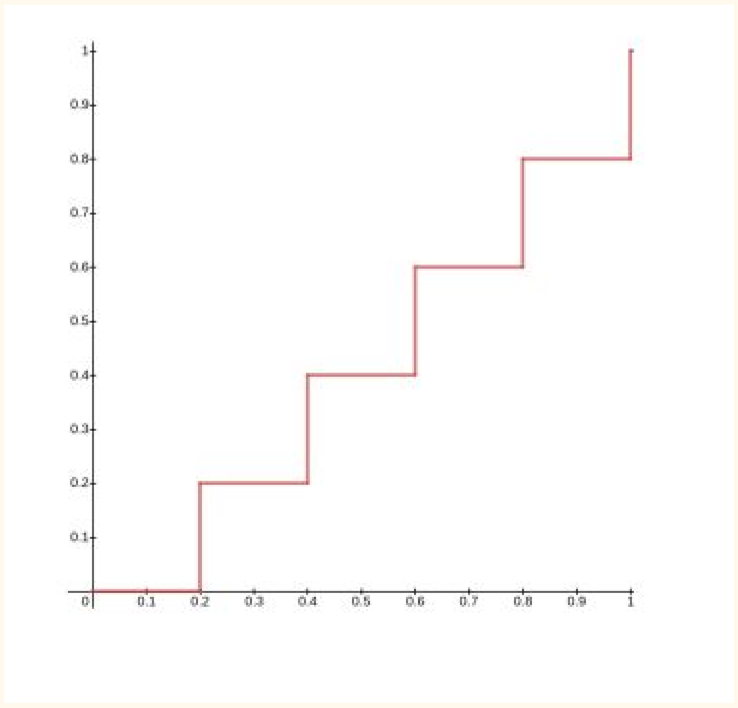
Approach:

The approach used was to mathematically transform every pixel value with intensity ***'r'*** with the given transformation function.

```
def piecewise_tranform(image):
      for x in range(image.shape[0]):
            for y in range(image.shape[1]):
                  k=(image[x][y])/255

                  if k>=0.3 and k<=0.6:
                        image[x][y] = (((k-0.3)*4/3) + 0.4)*255

                  elif k>0.6 and k<=0.8:
                        image[x][y] = (((k-0.8)*(-2)) + 0.4)*255
                  else:
                        image[x][y]=0
      return image
```

Input:	**Output:**
	

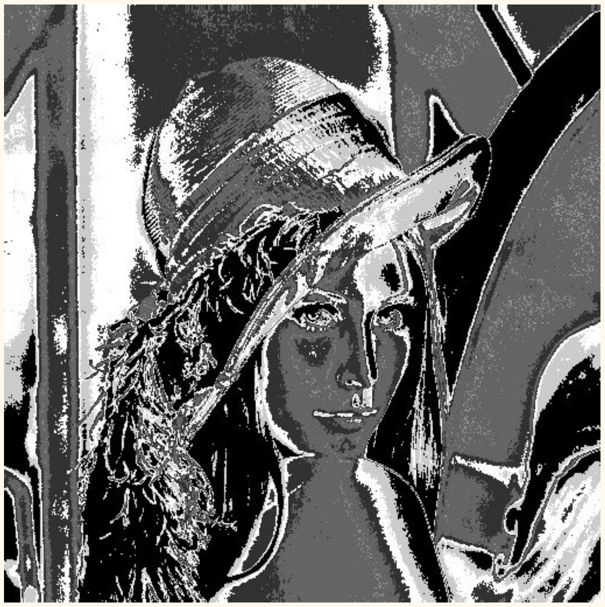
b)



Approach:

The approach used was to mathematically transform every pixel value with intensity ***'r'*** with the given transformation function.

```
def step_tranform(image):
      for x in range(image.shape[0]):
            for y in range(image.shape[1]):
                  k = int((image[x][y]*10)/255)

                  image[x][y] = 255*(k*0.2)
      return image
```

Input:	Output:
	

Q5)

1) **What can you say about the histogram of a resulting image if we keep the MSB bits in the bitplane to 0?**

   Answer: Most of the information is contained in the MSB of the image. If it is set to 0 we will lose a lot of information. The image becomes darker and histogram shits to the left.

2) **What can you say about the histogram of a resulting image if we keep the LSB bits in the bitplane to 0?**

   Answer: the histogram becomes more sparse and all intensities corresponding to odd values get reduced to even values, i.e heights of some peaks increases.

3) **Transmission is usually achieved through packets containing a start bit, a byte of information and a stop bit. Baud rate is a common measure for digital data transmission and is defined as the number of bits transmitted per second. How much time would it take to transmit 512 x 512 grayscale image with intensity 0-255 over a 56K baud link? Similarly, calculate the time required to transmit the same image over a 3000K baud link.**

Answer: image size = 512 x 512

No. of pixels = 262144

No. of bits per pixel = 10= 8+2

Bits transmitted = 10 x 262144 = 2621440

Bytes transmitted = 327680

Time for 56K baud link = 327680/56000 = **5.8s**

Time for 3000K baud link = 327680/3000,000 = **0.19s**

Q6)

1) **Write a function histEqualization which takes a grayscale image im, and applies histogram equalization on the entire image.**

Approach:

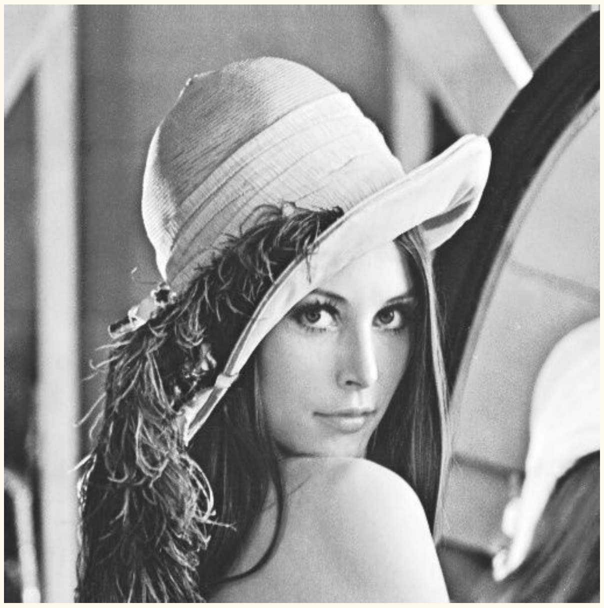
The approach used was to mathematically transform every pixel value with intensity ***'r'*** with the given transformation function i.e :

$$s = T(r) = (L - 1)^* \sum_{i=0} (p_i(r))$$

Steps:

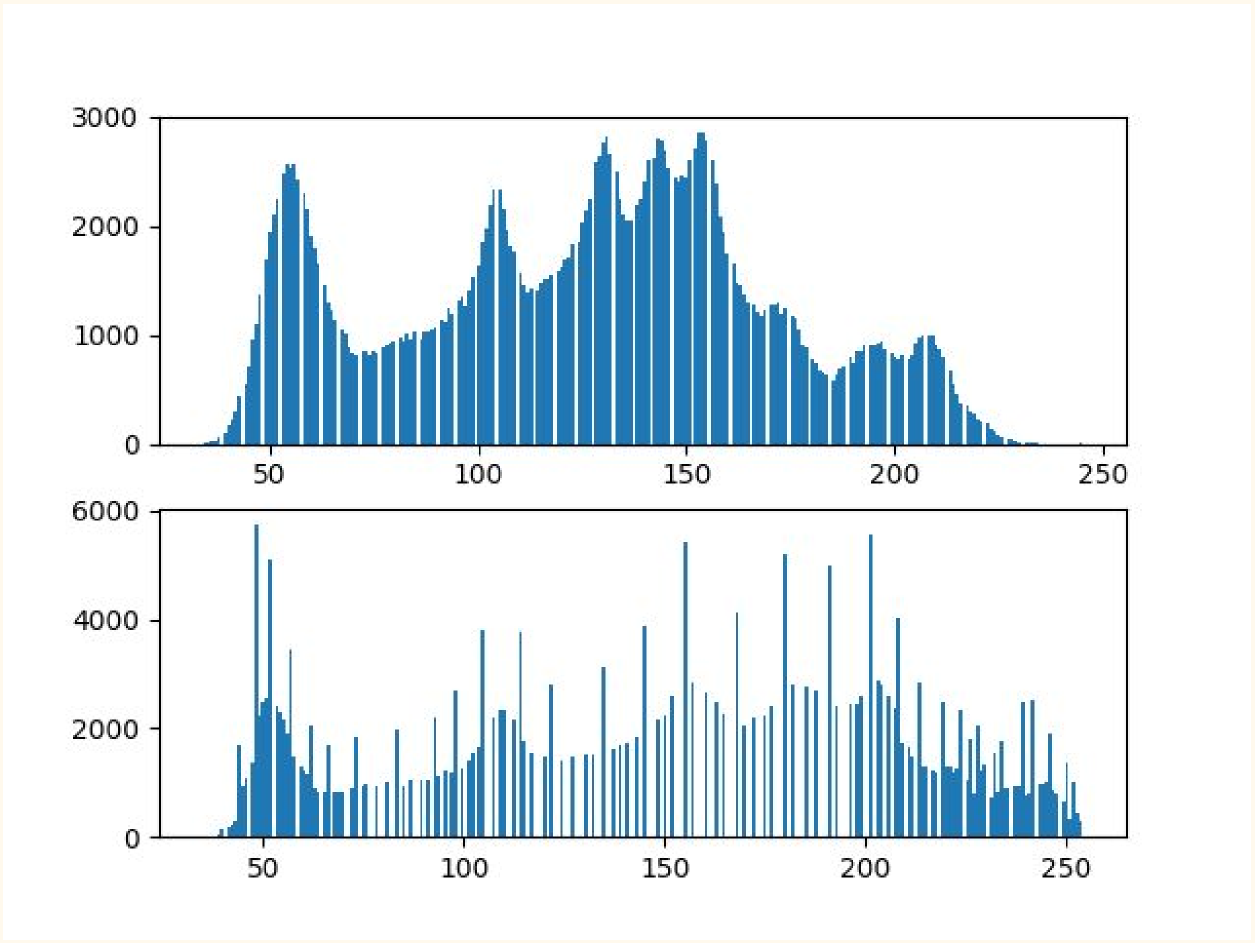
- Compute the number of pixels for each intensity
- Compute the probability of occurrence for each pixel value.
- Compute the cumulative probability for each value ***'r',*** corresponding to T(r)
- Replace the value with T(r)

```
def get_histogram(image,i):
      plt.subplot(2,1,i)
      count, bins, patches = plt.hist(image.ravel(), bins=256)
      return count

def equalise(image,a):
      for x in range(image.shape[0]):
            for y in range(image.shape[1]):
                   image[x][y] = a[image[x][y]]
       return image



im = cv2.imread('lena.png')

count = get_histogram(im,1)
print(len(count))
a = count/im.size
a = np.cumsum(a)
a = a*255
im = equalise(im,a)
get_histogram(im,2)
cv2.imwrite("test.jpg",im)
plt.show()
```

Input:	**Output:**
	

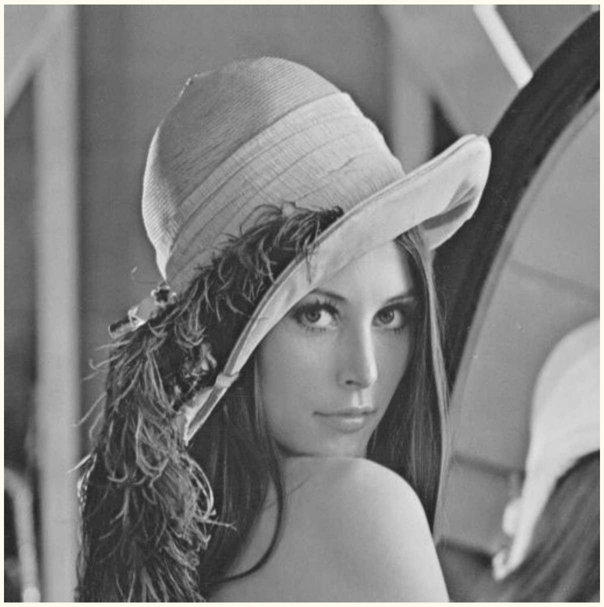
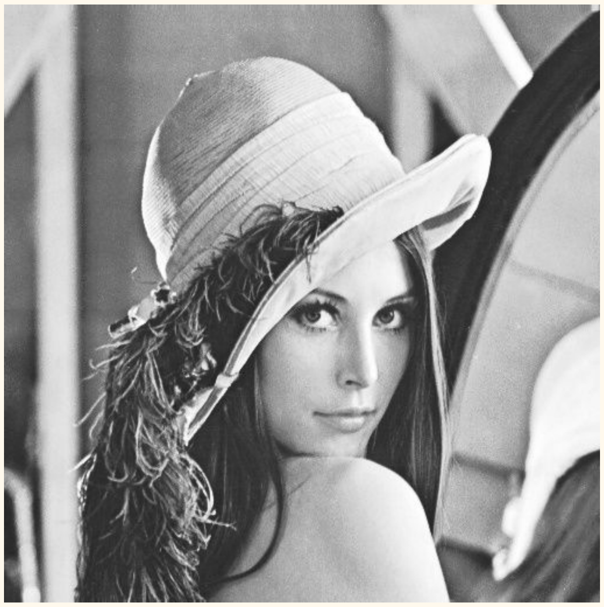
Comparison of the Histograms

NOTE : TOP image corresponds to input, BOTTOM image to output



Q6)

2) **Display the input image and the resultant image side-by-side and provide suitable explanation for the changes you observe for multiple input images**

Input:	**Output:**
	

Observations:

It can be observed that the histogram equalised image seems to be more enhanced than the input image. The brighter pixels appear more bright and the darker pixel appears more dark. On comparing the histograms the equalised image has a much broader histogram than the input.

Q6)

3) **Write a function histMatching which takes an input image and a reference image and applies histogram Matching on the input image by matching the histogram with that of the reference image. Use eye.png and eyeref.png (converted to grayscale) as the input and reference images respectively.**

Approach:

The approach used was to mathematically obtain for every pixel value in the input image and reference image with intensity ***'r' and 'r_{ref}'*** , respectively with the given transformation function i.e :

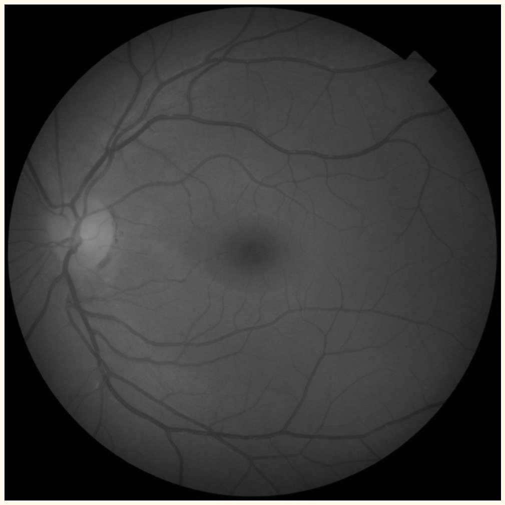
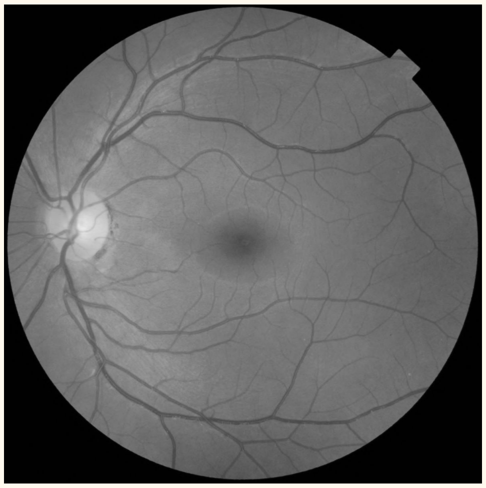
$$s = T(r) = (L - 1)* \sum_{i=0} (p_i(r))$$

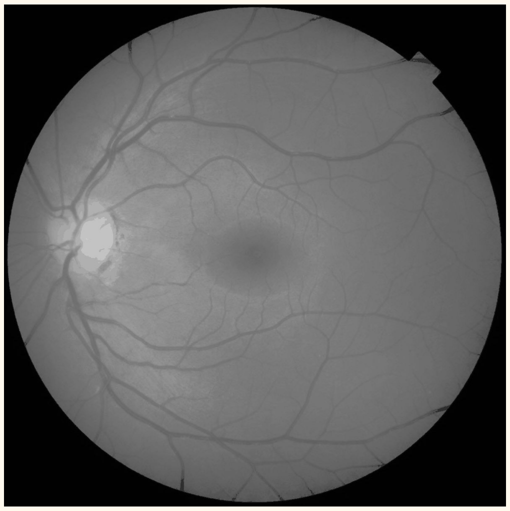
And then perform corresponding mapping between T(r) and T(r_{ref}) such that 'r' gets mapped to T(r_{ref}).

Steps:

- Compute the histogram equalised value for both images.
- Create the look-up table
- Pick equalised values from original image, find it in that of the specified image and hash down the index
- If the value doesn't exist then find the index of its nearest one.

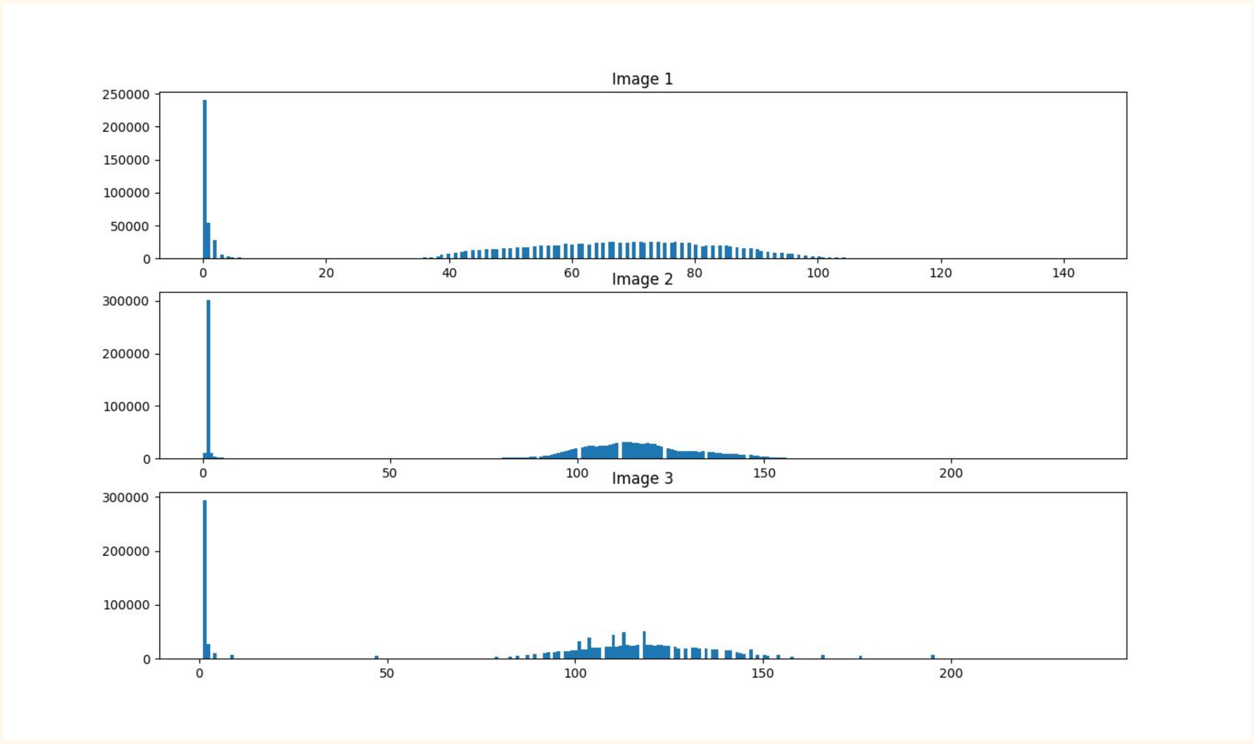
```python
def hist_equalize(org, spec):
    oldshape = org.shape
    org = org.reshape(-1)
    spec = spec.reshape(-1)
    rand1, bin_idx, s_counts = np.unique(org,
return_inverse=True,return_counts=True)
    rand2, t_counts = np.unique(spec, return_counts=True)
    s_quant = np.cumsum(s_counts)
    s_quant = s_quant/s_quant[-1]
    t_quant = np.cumsum(t_counts)
    t_quant = t_quant/t_quant[-1]
    our = s_quant*255
    our = our.astype(int)
    tmp = t_quant*255
    tmp = tmp.astype(int)
    b = []
    for data in our[:]:
        diff = tmp - data
        mask = np.ma.less_equal(diff, -1)
        if np.all(mask):
            c = np.abs(diff).argmin()
        masked_diff = np.ma.masked_array(diff, mask)
        b.append(masked_diff.argmin())
    b = np.array(b,dtype='uint8')
    return b[bin_idx].reshape(oldshape)
```

Input:(gray scale)	Reference :(gray scale)
	

Output:	
	

Comparison of the Histograms

NOTE : TOP image corresponds to input, MIDDLE to reference, BOTTOM image to output



Q6)

4) **You are provided with 4 images (part1.png, part2.png, part3.png, part4.png) with different contrast levels which correspond to four quadrants of canyon.png as shown in Figure 5. Retrieve the original image(converted to grayscale) using these four images(converted to grayscale) as closely as possible.**

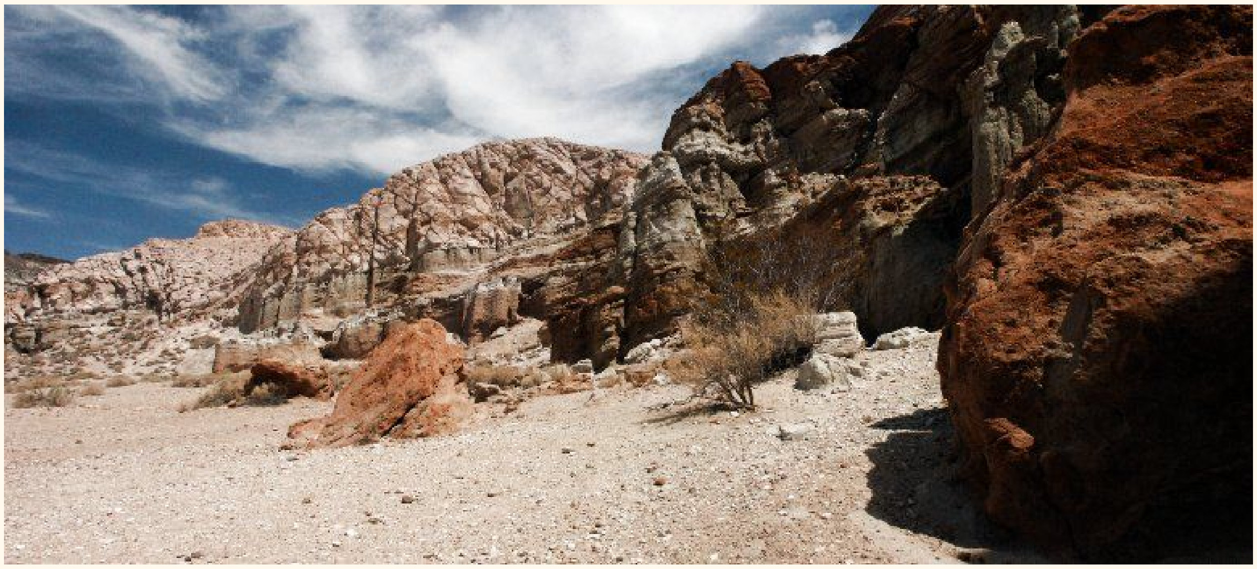
Approach:

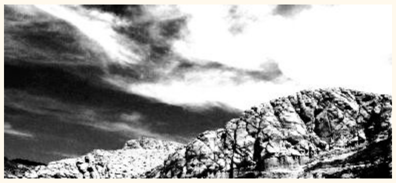
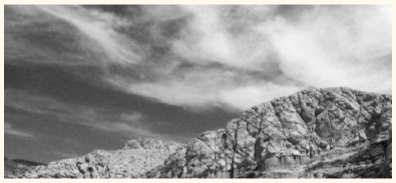
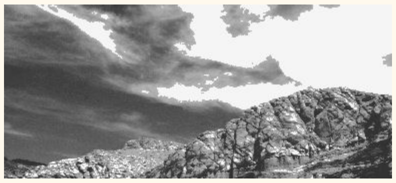
The approach was to match the histograms of each of the images to the corresponding the quadrants of the original image.

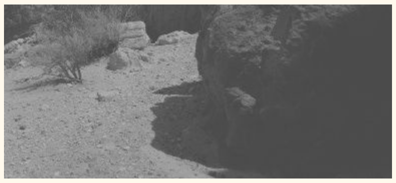
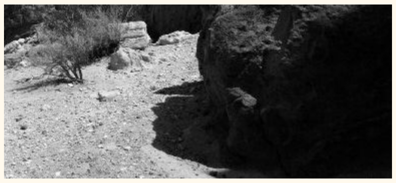
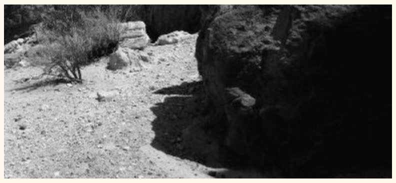
Steps :

- Divide the original image into four quadrants
- Use the function of histogram matching on each quadrant.

Input :



Input (gray-scale)	Reference(gray- scale)	Output
Quadrant 1		
		
Quadrant 2		

Quadrant 3		
Quadrant 4		
		

Q7)

1) **Choose an image of your choice and apply histogram equalization to it. Apply histogram equalization to the resulting image and compare the two images. What are your observations?**

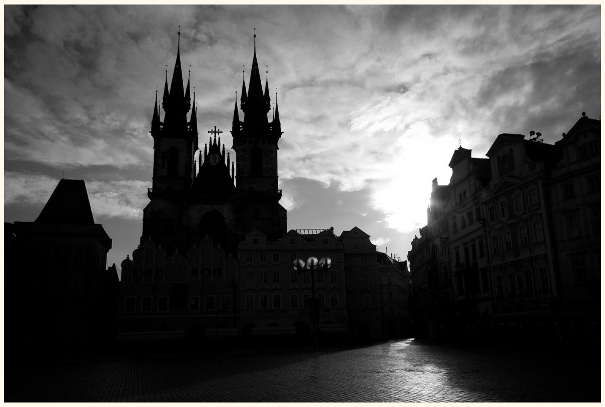
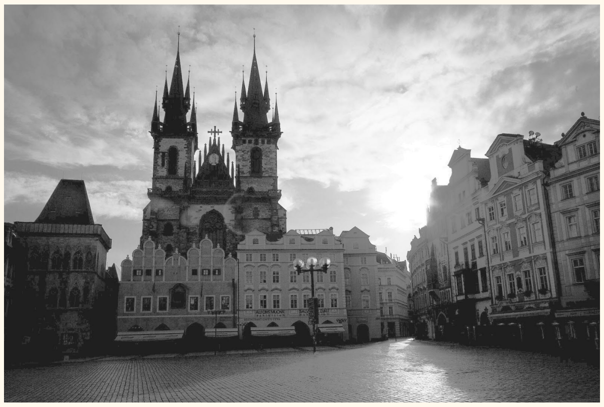
Approach:

The approach used was to mathematically transform every pixel value with intensity ***'r'*** with the given transformation function i.e :

$$s = T(r) = (L - 1)^{*} \sum_{i=0} (p_i(r))$$

Steps:

- Compute the number of pixels for each intensity
- Compute the probability of occurrence for each pixel value.
- Compute the cumulative probability for each value ***'r',*** corresponding to T(r)
- Replace the value with T(r)

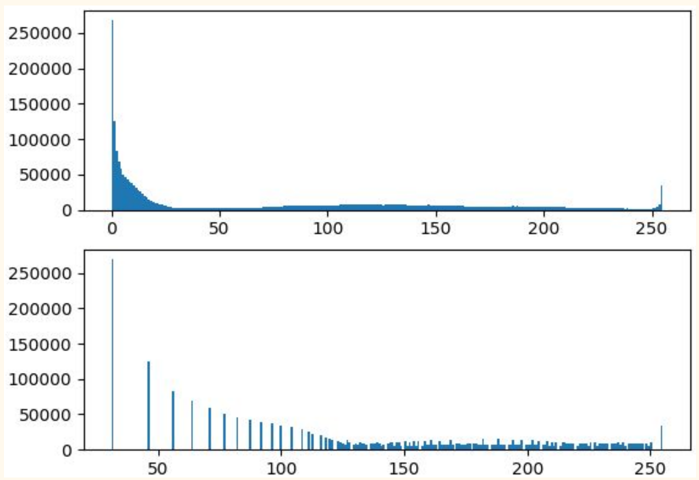
Input:	**Output:**
	

Observations:

The histogram equalisation reveals a lot of details which were unable to be visualised in the input image. The image was primarily very dark i.e had high histogram peaks on the lower intensities. On equalisation the histogram gets more broadly distributed.

Comparison of the Histograms

NOTE : TOP image corresponds to input, BOTTOM image to output

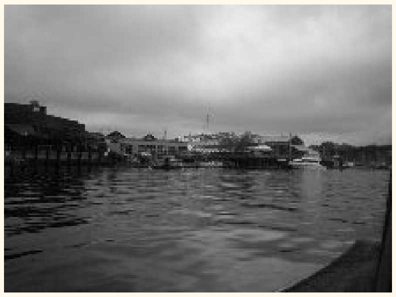


Q7)

2) **Pick the following combination of images and perform histogram transformation onthem.(20 points)**

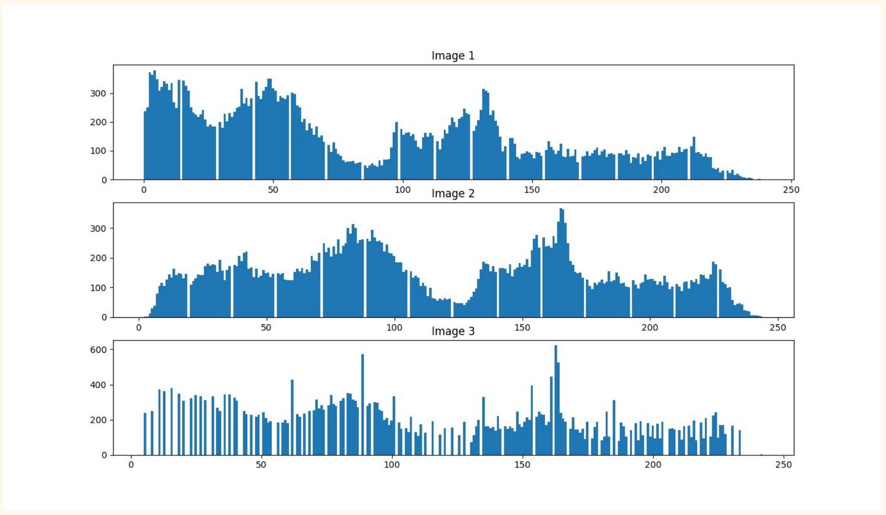
- **Similar Histograms**
- **Dark →Light Image**
- **Light →Dark Image**

a) **Similar Histograms**

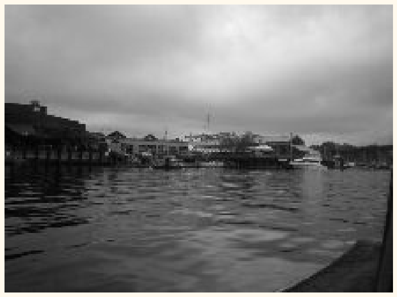
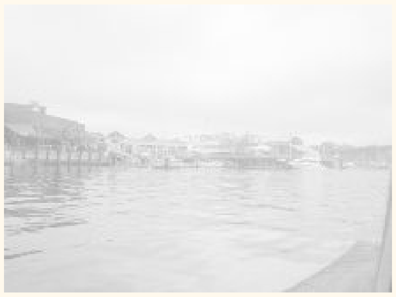
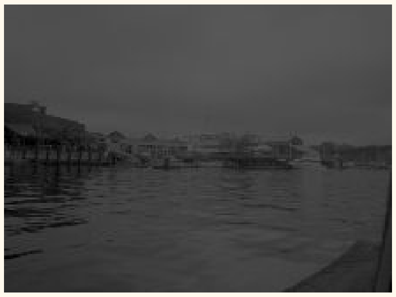
Input (gray-scale)	Reference(gray- scale)	Output
		

Comparison of the Histograms

NOTE : TOP image corresponds to input, MIDDLE to reference, BOTTOM image to output

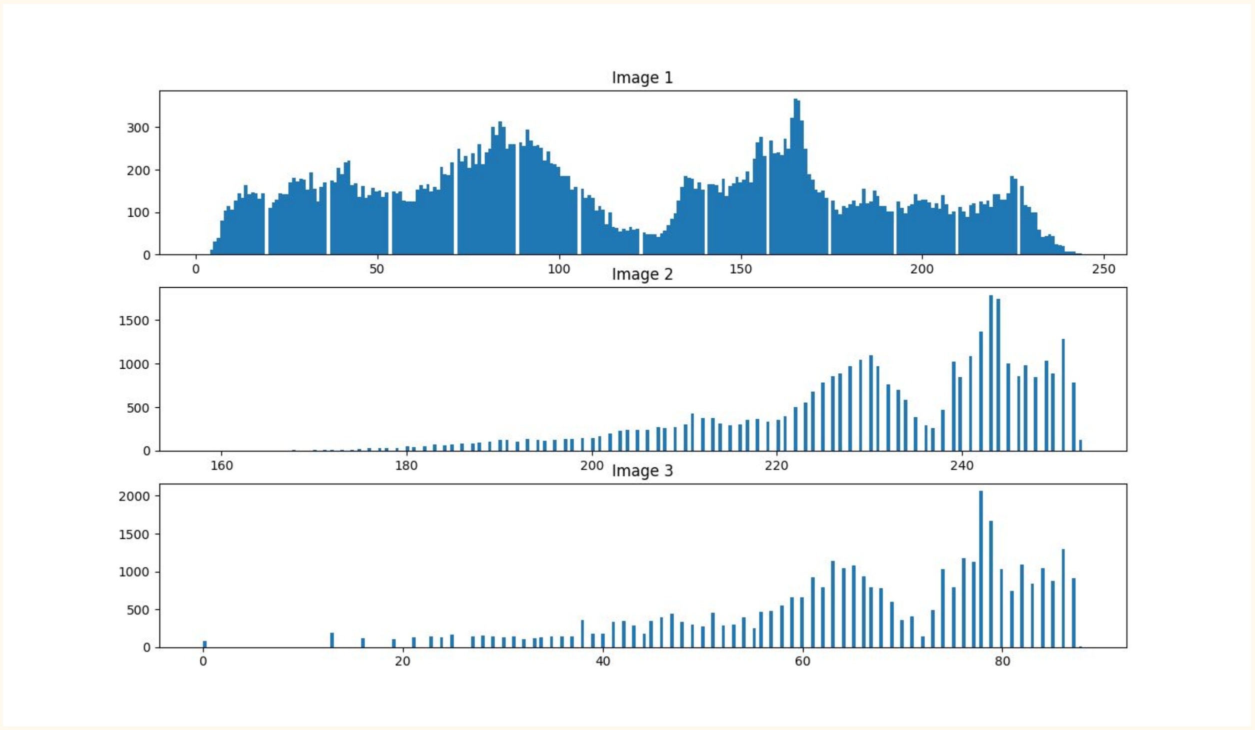


b) Dark →Light Image

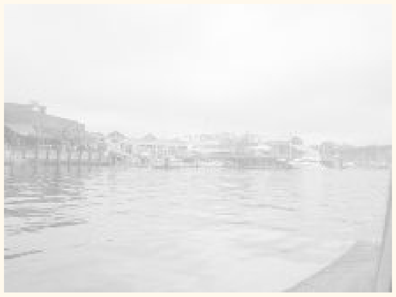
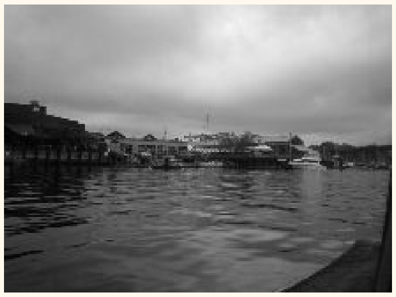
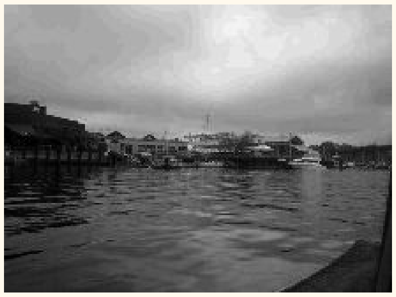
Input (gray-scale)	Reference(gray- scale)	Output
		

Comparison of the Histograms

NOTE : TOP image corresponds to input, MIDDLE to reference, BOTTOM image to output



c) Light →Dark Image

Input (gray-scale)	Reference(gray- scale)	Output
		

Comparison of the Histograms

NOTE : TOP image corresponds to input, MIDDLE to reference, BOTTOM image to output

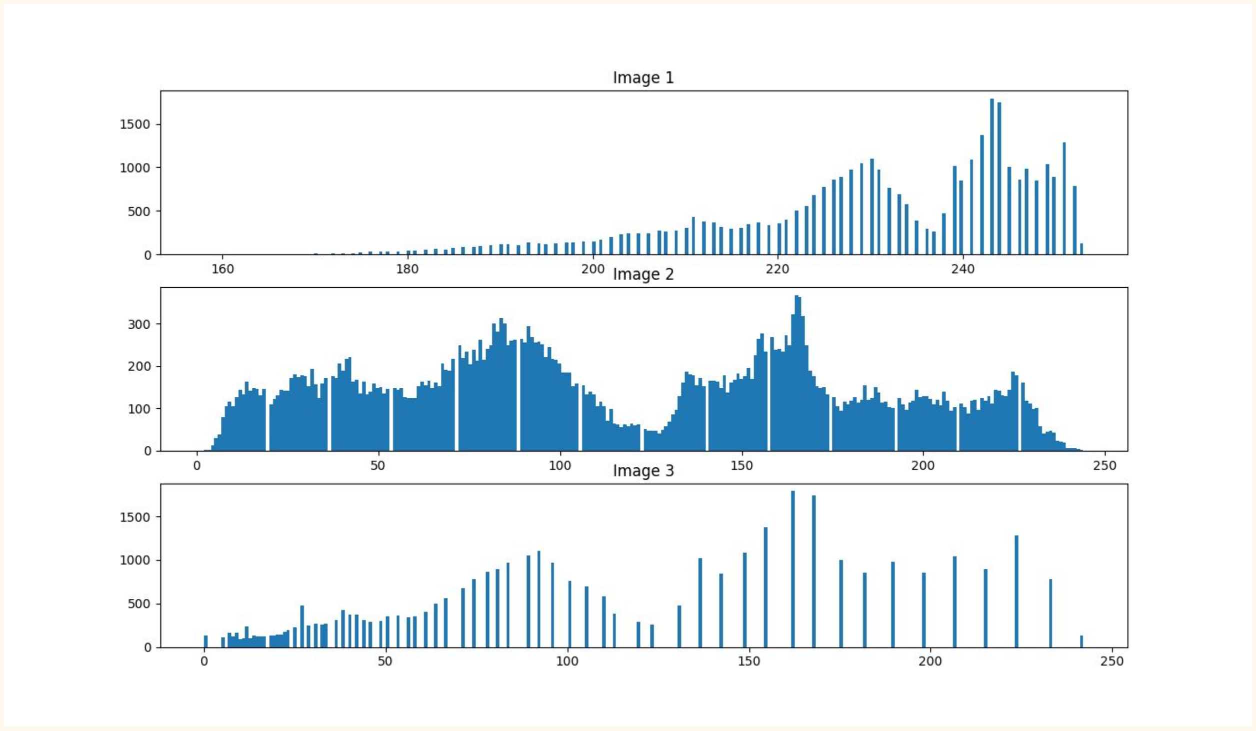


Image 1
Image 2
Image 3