**CO543: Image Processing**

**Lab 2**

Ranage R.D.P.R. - E/19/310

The following is the original image that was used to implement the functions in the lab tasks.



The following functions are used throughout the lab to plot each figure to show the results.

import cv2

import numpy as np

import matplotlib.pyplot as plt

def show\_2\_images(image1, image2, title1, title2):

    fig, ax = plt.subplots(1, 2, figsize=(10, 5))

    ax[0].imshow(image1, cmap='gray')

    ax[0].set\_title(title1)

    ax[1].imshow( image2, cmap='gray')

    ax[1].set\_title(title2)

    plt.show()

def show\_3\_images(image1, image2, image3, title1, title2, title3):

    fig, ax = plt.subplots(1, 3, figsize=(15, 5))

    ax[0].imshow(image1, cmap='gray')

    ax[0].set\_title(title1)

    ax[1].imshow( image2, cmap='gray')

    ax[1].set\_title(title2)

    ax[2].imshow( image3, cmap='gray')

    ax[2].set\_title(title3)

    plt.show()

**1. Image thresholding**

**Lab Task 01 : Write a function to perform image thresholding using point processing taking the image file and the threshold value from the user.**

# Function to to perform image thresholding using point processing

def image\_thresholding(image\_path, threshold\_value):

    # Read the image in grayscale mode

    image = cv2.imread(image\_path, 0)

    # Apply thresholding

    \_, thresholded\_image = cv2.threshold(image, threshold\_value, 255, cv2.THRESH\_BINARY)

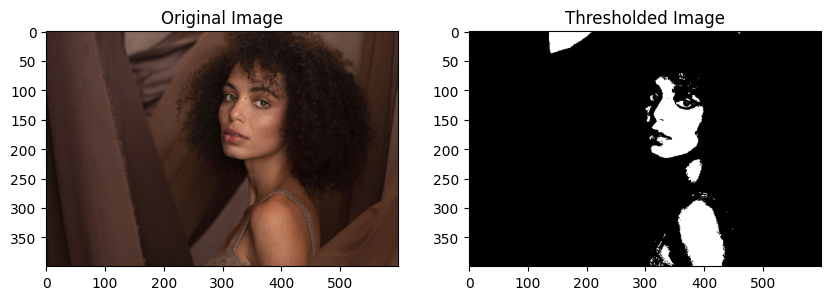
    return thresholded\_image

# Loading the original image

img = plt.imread('image.jpg')

thresh\_img = image\_thresholding("image.jpg", 100)

show\_2\_images(img,thresh\_img,'Original Image', 'Thresholded Image')

****

**2. Image arithmetic operations**

**Lab Task 02 : Read two images and perform addition and subtraction.**

**I=I1+I2; # Addition of two**

**I=I1-I2; # Subtraction of two images**

def image\_addition(image1, image2):

    # Ensure both images have the same dimensions

    image2\_resized = cv2.resize(image2, (image1.shape[1], image1.shape[0]))

    # Perform addition using OpenCV function

    add\_image = cv2.add(image1, image2\_resized)

    return add\_image

img1 = plt.imread('image2.jpg')

img2 = plt.imread('image.jpg')

add\_img = image\_addition(img1,img2)

show\_3\_images(img1,img2,add\_img,'Image 1', 'Image 2','Added Image')

****

def image\_subtraction(image1, image2):

    # Ensure both images have the same dimensions

    image2\_resized = cv2.resize(image2, (image1.shape[1], image1.shape[0]))

    # Perform addition using OpenCV function

    sub\_image = cv2.subtract(image1, image2\_resized)

    return sub\_image

sub\_img = image\_subtraction(img1,img2)

show\_3\_images(img1,img2,sub\_img,'Image 1', 'Image 2','Subtracted Image')

****

**3. Spatial Processing**

**i). Write simple programs to demonstrate the following. Show the original and resultant images in the same figure to compare them easily.**

# will use the gray scale image

gray\_img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

plt.imshow(gray\_img,cmap='gray')



**a. Log transformation**

# Log transformation function

def log\_transform(image):

    # Apply log transformation

    c = 255/(np.log(1 + np.max(img)))

    log\_transformed = c \* np.log(1 + img)

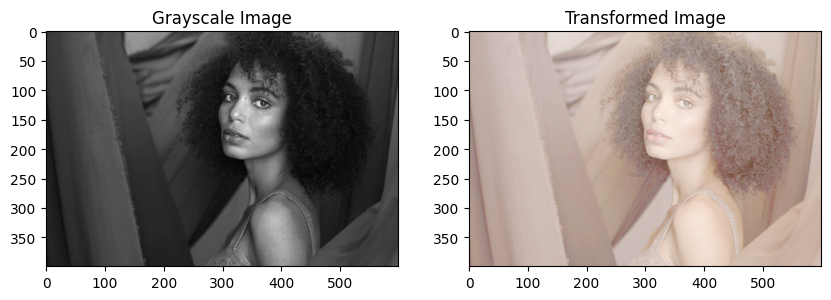
    # Specify the data type.

    log\_transformed = np.array(log\_transformed, dtype = np.uint8)

    return log\_transformed

log\_trans\_img = log\_transform(gray\_img)

show\_2\_images(gray\_img,log\_trans\_img,'Grayscale Image', 'Transformed Image')



**b. Power transformation**

# Power transformation function

def power\_transform(image, gamma=1.0):

    # Apply power transformation

    transformed\_image = np.uint8(255 \* (image / 255) \*\* gamma)

    return transformed\_image

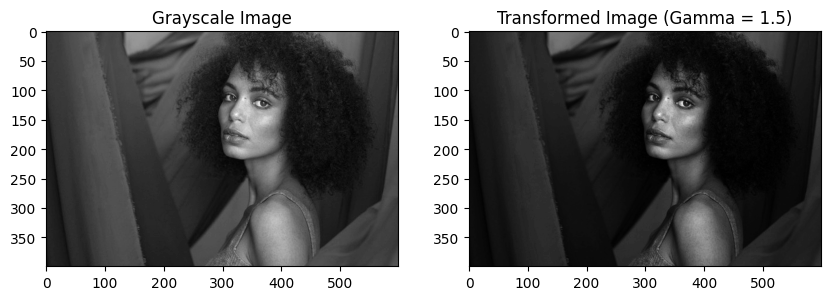
pow\_trans\_img = power\_transform(gray\_img,0.2)

show\_2\_images(gray\_img,pow\_trans\_img,'Grayscale Image','Transformed Image(Gamma = 0.2)')



pow\_trans\_img = power\_transform(gray\_img,1.5)

show\_2\_images(gray\_img,pow\_trans\_img,'Grayscale Image','Transformed Image (Gamma = 1.5)')



**c. Contrast Stretching**

# Contrast Stretching function

def piecewise\_linear\_transform(pix, r1, s1, r2, s2):

    if (0 <= pix and pix <= r1):

        return (s1 / r1)\*pix

    elif (r1 < pix and pix <= r2):

        return ((s2 - s1)/(r2 - r1)) \* (pix - r1) + s1

    else:

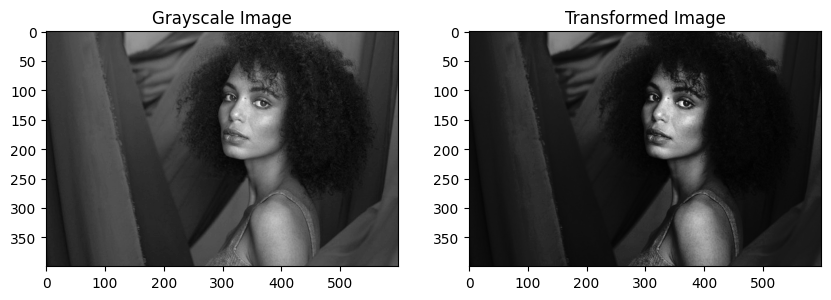
        return ((255 - s2)/(255 - r2)) \* (pix - r2) + s2

# Vectorize the function to apply it to each value in the Numpy array.

contrast\_stretching = np.vectorize(piecewise\_linear\_transform)

contrast\_stretched\_img = contrast\_stretching(gray\_img, 50,30,170,255)

show\_2\_images(gray\_img,contrast\_stretched\_img,'Grayscale Image', 'Transformed Image')



**d. Gray level slicing**

# Gray level slicing function

def gray\_level\_slicing(image, min\_intensity, max\_intensity):

    row, column= image.shape

    #  Create an zeros array to store the sliced image

    sliced\_image = np.zeros((row,column),dtype = 'uint8')

    #  Specify the min and max range

    min\_range = min\_intensity

    max\_range = max\_intensity

    # Loop over the input image and if pixel value lies in desired range set it to 255

    # otherwise set it to desired value

    for i in range(row):

        for j in range(column):

            if gray\_img[i,j]>min\_range and gray\_img[i,j]<max\_range:

                sliced\_image[i,j] = 255

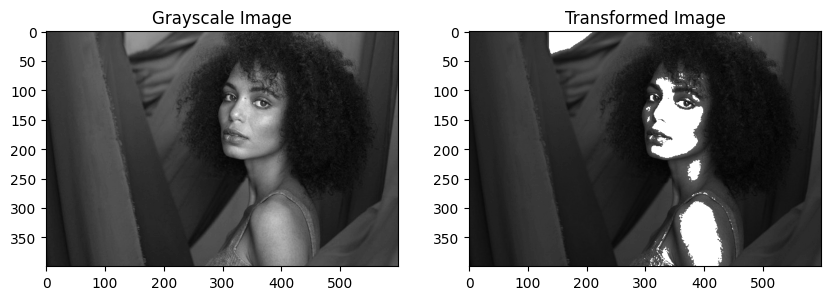
            else:

                sliced\_image[i,j] = gray\_img[i-1,j-1]

    return sliced\_image

gray\_sliced\_img = gray\_level\_slicing(gray\_img, 90, 220)

show\_2\_images(gray\_img,gray\_sliced\_img,'Grayscale Image', 'Transformed Image')



**e. Bit plane slicing**

def bit\_plane\_slicing(image):

    # Initialize list to store bit planes

    bit\_planes = []

    # Iterate over each bit plane from 0 to 7

    for bit in range(8):

        # Apply bit plane slicing

        bit\_plane = (image >> bit) & 1

        # Convert the bit plane to 8-bit unsigned integer

        bit\_plane = np.uint8(bit\_plane \* 255)

        # Append the bit plane to the list

        bit\_planes.append(bit\_plane)

    return bit\_planes

bit\_sliced\_imgs = bit\_plane\_slicing(gray\_img)

fig, ax = plt.subplots(3, 3, figsize=(15, 11))

ax[0, 0].imshow(gray\_img, cmap='gray')

ax[0, 0].set\_title('Original Image')

ax[0, 1].imshow(bit\_sliced\_imgs[0], cmap='gray')

ax[0, 1].set\_title('Bit Plane 1')

ax[0, 2].imshow(bit\_sliced\_imgs[1], cmap='gray')

ax[0, 2].set\_title('Bit Plane 2')

ax[1, 0].imshow(bit\_sliced\_imgs[2], cmap='gray')

ax[1, 0].set\_title('Bit Plane 3')

ax[1, 1].imshow(bit\_sliced\_imgs[3], cmap='gray')

ax[1, 1].set\_title('Bit Plane 4')

ax[1, 2].imshow(bit\_sliced\_imgs[4], cmap='gray')

ax[1, 2].set\_title('Bit Plane 5')

ax[2, 0].imshow(bit\_sliced\_imgs[5], cmap='gray')

ax[2, 0].set\_title('Bit Plane 6')

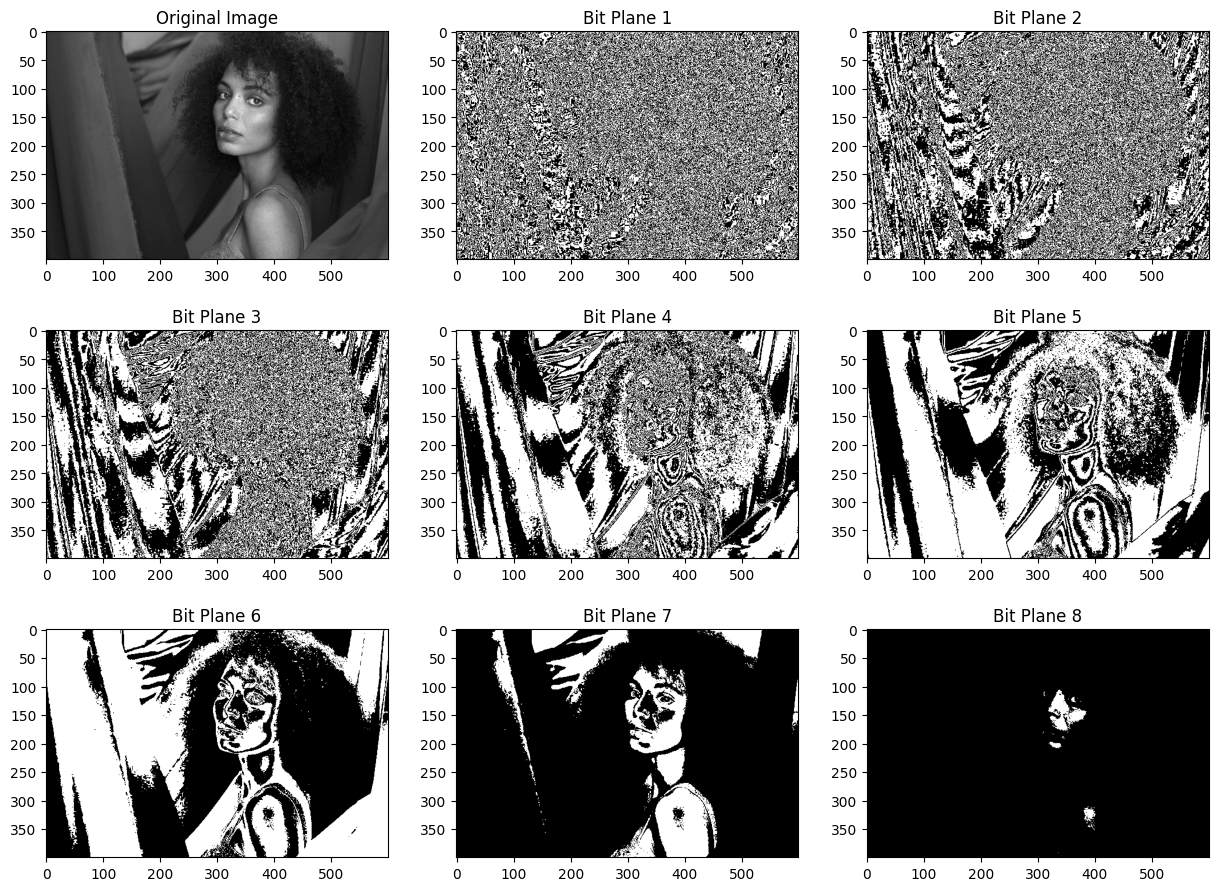
ax[2, 1].imshow(bit\_sliced\_imgs[6], cmap='gray')

ax[2, 1].set\_title('Bit Plane 7')

ax[2, 2].imshow(bit\_sliced\_imgs[7], cmap='gray')

ax[2, 2].set\_title('Bit Plane 8')

plt.show()



**ii). Consider the graph for a typical transformation function used for Contrast Stretching in the given figure and determine the behavior of the function with respect to given changes.**

**a. When r1 =s1 and r2=s2**

r1 = 40

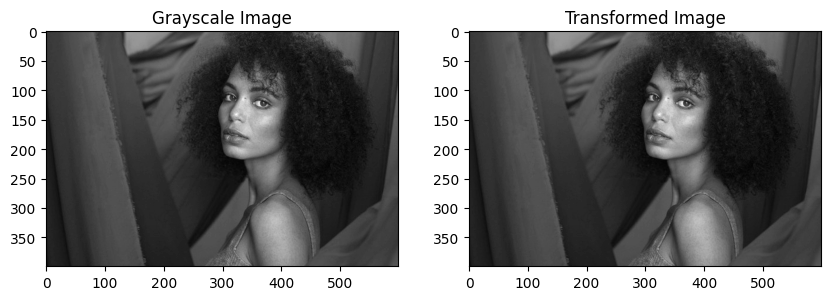
s1 = 40

r2 = 120

s2 = 120

contrast\_stretched\_img = contrast\_stretching(gray\_img, r1, s1, r2, s2)

show\_2\_images(gray\_img,contrast\_stretched\_img,'Grayscale Image', 'Transformed Image')



When r1=s1 and r2=s2, the points (r1, s1) and (r2, s2) lie on the line s = r, which is the identity line. This means that the output value s is equal to the input value r for all r, and thus the transformation function does not change the image contrast.

**b. When r1=r2, s1=0 and s2=L-1**

r1 = 40

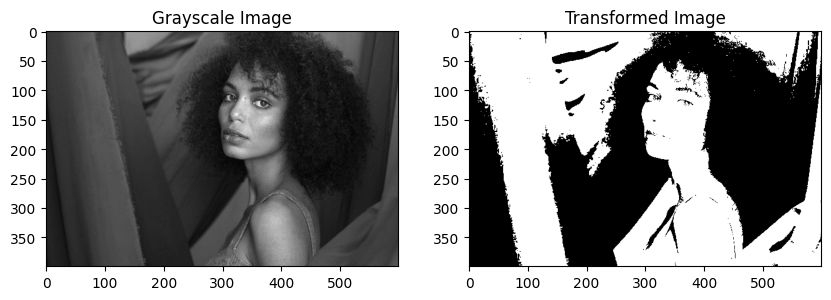
s1 = 0

r2 = 40

s2 = 255

contrast\_stretched\_img = contrast\_stretching(gray\_img, r1, s1, r2, s2)

show\_2\_images(gray\_img,contrast\_stretched\_img,'Grayscale Image', 'Transformed Image')



When r1=r2, the transformation function becomes a binary function. For all values of r less than r1, the output s is 0, and for all values of r greater than r1, the output s is L-1. This creates a high-contrast image where all pixel values below a certain threshold are set to 0 (black) and all pixel values above the threshold are set to L-1 (white).

**4. Masking**

**Lab Task 04 : Write a program to read any image, resize it to 256x256. Apply the masks shown in following figures so that only the middle part of the image is visible.**

def apply\_mask(image, margin, output\_size=256):

    # Resize the image to the specified output size

    image\_resized = cv2.resize(image, (output\_size, output\_size))

    # Create a mask to keep only the middle part of the image visible

    mask = np.zeros\_like(image\_resized)

    center = output\_size // 2

    mask[margin: 255 - margin, margin: 255 - margin] = 255

    # Apply the mask to the resized image

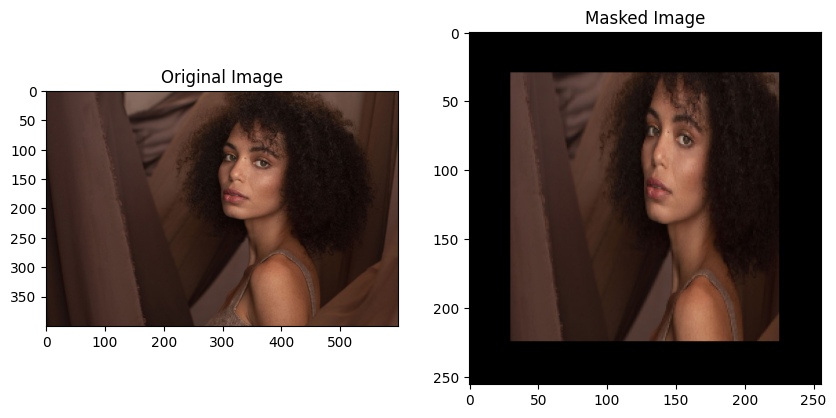
    masked\_image = cv2.bitwise\_and(image\_resized, mask)

    return masked\_image

img = plt.imread('image.jpg')

masked\_image = apply\_mask(img, 30, 256)

show\_2\_images(img,masked\_image,'Original Image', 'Masked Image')

****

def apply\_circular\_mask(image, margin, output\_size=256):

    # Resize the image to the specified output size

    image\_resized = cv2.resize(image, (output\_size, output\_size))

    # Create a circular mask to keep only the middle part of the image visible

    mask = np.zeros\_like(image\_resized)

    center = output\_size // 2

    radius = output\_size // 2

    cv2.circle(mask, (center, center), radius - margin, (255, 255, 255), -1, cv2.LINE\_AA)

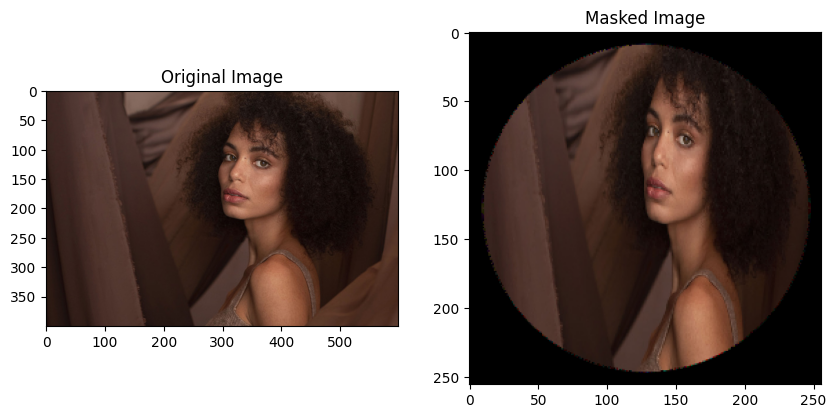
    # Apply the mask to the resized image

    masked\_image = cv2.bitwise\_and(image\_resized, mask)

    return masked\_image

masked\_image = apply\_circular\_mask(img, 10, 256)

show\_2\_images(img,masked\_image,'Original Image', 'Masked Image')



**5. Brightness**

**Lab Task 05 : Write your own Python OpenCV function addbrightness() and use it to increase brightness of a given image.(Hint: Use Image arithmetic operations)**

def add\_brightness(image, brightness):

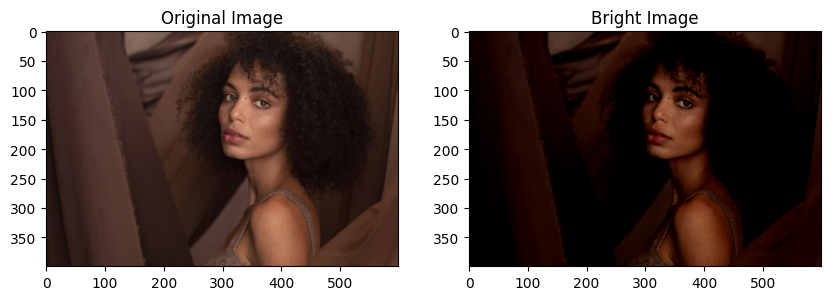
    # Add brightness to the image

    brightened\_image = cv2.add(image, brightness)

    return brightened\_image

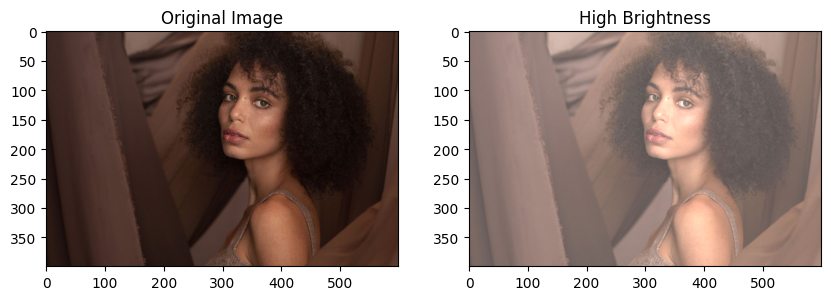
bright\_img = add\_brightness(img, -50)

show\_2\_images(img,bright\_img,'Original Image', 'Low Brightness')

****

bright\_img = add\_brightness(img, 80)

show\_2\_images(img,bright\_img,'Original Image', 'High Brightness')

****

**6. Histogram Processing**

**a. Histogram Calculation in OpenCV**

**Use inbuilt OpenCV cv2.calcHist() function to display the histogram of a given image.**

from io import BytesIO

def display\_cv2\_histogram(image):

    # find frequency of pixels in range 0-255

    hist = cv2.calcHist([image],[0],None,[256],[0,256])

    plt.hist(image.ravel(),256,[0,256])

    plt.title('Histogram (OpenCV)')

    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.show()

display\_cv2\_histogram(gray\_img)

****

**b. Histogram Calculation in Numpy**

**Use inbuilt numpy np.histogram() function to display the histogram of a given image.**

def display\_numpy\_histogram(image):

    # Calculate histogram

    hist,bins = np.histogram(image,256,[0,256])

    # Plot histogram

    plt.hist(image.ravel(),256,[0,256])

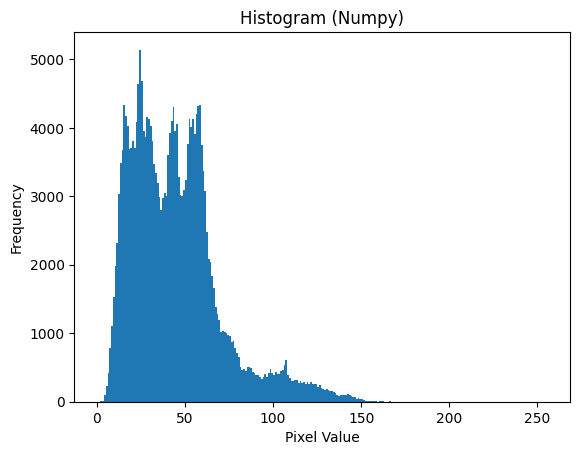
    plt.title('Histogram (Numpy)')

    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.show()

display\_numpy\_histogram(gray\_img)

****

**c. Then write your own histogram functions for the following scenarios**

**i. Show a histogram plot for a grayscale image.**

**ii. Show three histograms for a given RGB image.**

def grayscale\_histogram(image):

    # Calculate histogram

    # hist, bins = np.histogram(image.flatten(), 256, [0, 256])

    # Plot histogram

    plt.hist(image.ravel(),256,[0,256])

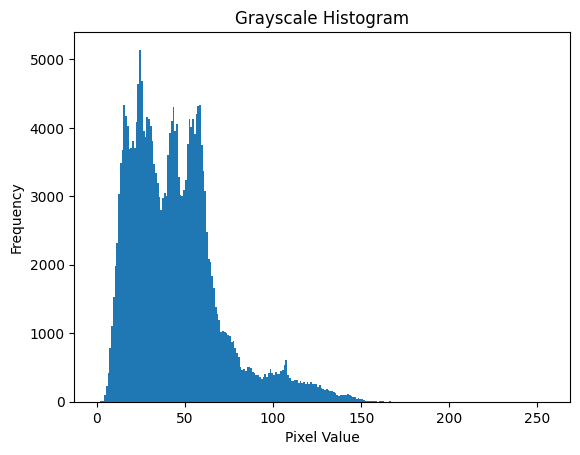
    plt.title('Grayscale Histogram')

    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.show()

grayscale\_histogram(gray\_img)



def rgb\_histogram(image):

    # Separate color channels

    red\_channel = image[:,:,0]

    green\_channel = image[:,:,1]

    blue\_channel = image[:,:,2]

    # Plot histograms

    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)

    # plt.plot(hist\_red, color='red')

    plt.hist(red\_channel.ravel(),256,[0,256], color='red')

    plt.title('Red Channel Histogram')

    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.subplot(1, 3, 2)

    plt.hist(green\_channel.ravel(),256,[0,256], color='green')

    plt.title('Green Channel Histogram')

    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.subplot(1, 3, 3)

    plt.hist(blue\_channel.ravel(),256,[0,256], color='blue')

    plt.title('Blue Channel Histogram')

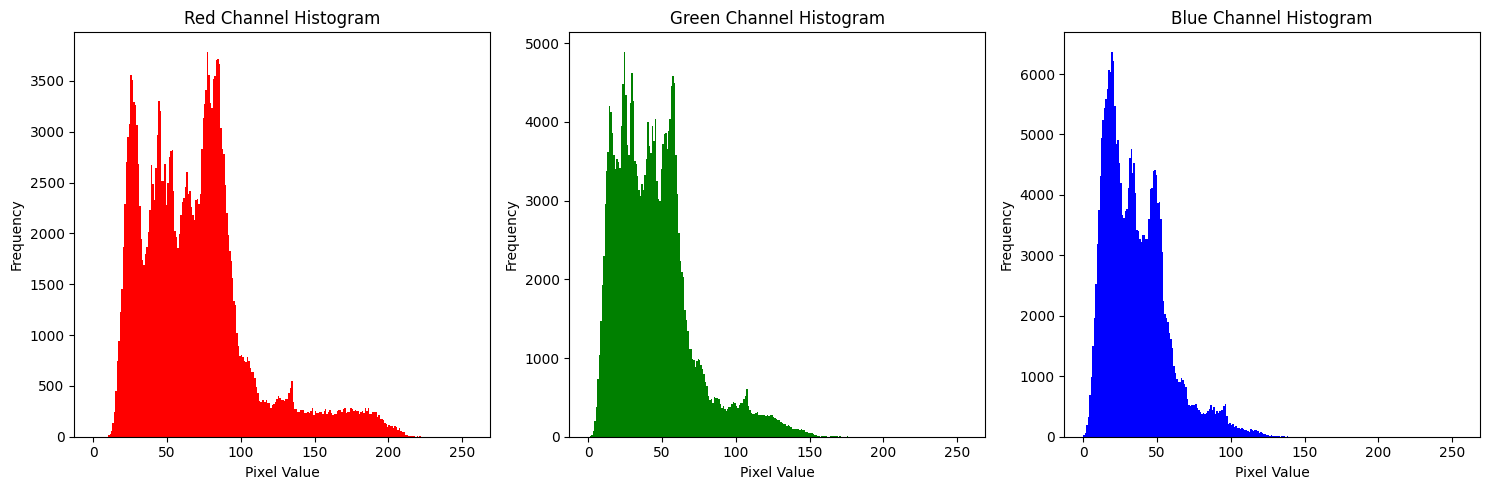
    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.tight\_layout()

    plt.show()

rgb\_histogram(img)



**d. Consider the four images given in the resources folder. Plot the histogram for each image. Perform Histogram Equalization on each image and plot the histograms of the resultant images. Comment on the results you have obtained.**

def plot\_histogram(image, title, main\_title):

     # Calculate histogram

    hist = cv2.calcHist([image], [0], None, [256], [0, 256])

    # Plot histogram

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)

    plt.suptitle(main\_title)

    plt.imshow(image, cmap='gray')

    plt.title("Original Image")

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.hist(image.ravel(),256,[0,256], color='black')

    plt.title(title)

    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.xlim([0, 256])

    plt.tight\_layout()

    plt.show()

# Loading the images

ref1 = plt.imread('bright.jpg')

ref2 = plt.imread('dark.jpg')

ref3 = plt.imread('high\_contrast.jpg')

ref4 = plt.imread('low\_contrast.jpg')

# Plotting image and their histogram before Histogram Equalization

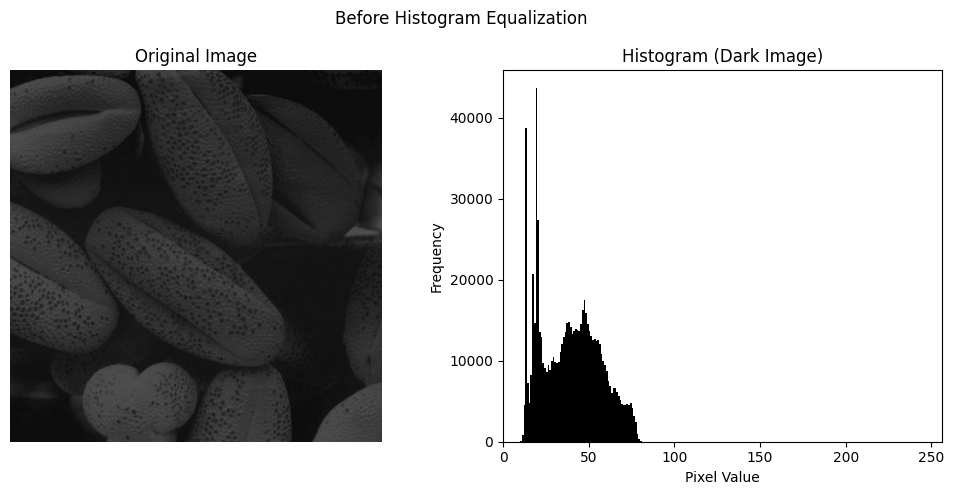
plot\_histogram(ref1, 'Histogram (Bright Image)', 'Before Histogram Equalization')

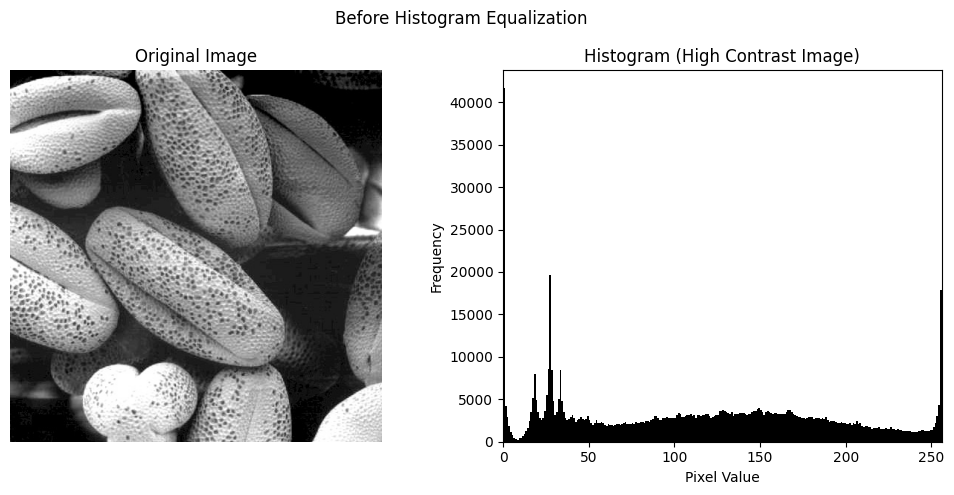
plot\_histogram(ref2, 'Histogram (Dark Image)', 'Before Histogram Equalization')

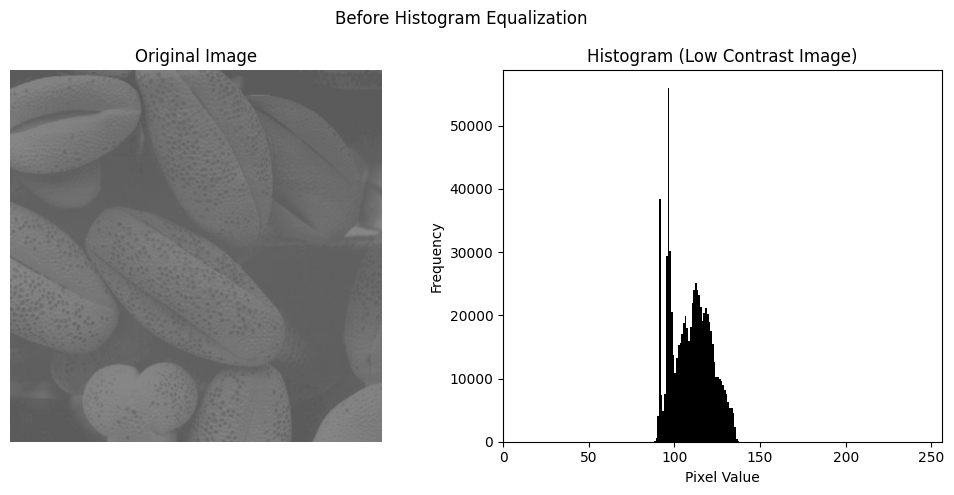
plot\_histogram(ref3, 'Histogram (High Contrast Image)', 'Before Histogram Equalization')

plot\_histogram(ref4, 'Histogram (Low Contrast Image)', 'Before Histogram Equalization')

****

****

****

****

def histogram\_equalization(image, title, main\_title):

    # Convert image to grayscale

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    # Perform histogram equalization

    equalized = cv2.equalizeHist(gray)

    # Plot histogram

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)

    plt.suptitle(main\_title)

    plt.imshow(equalized, cmap='gray')

    plt.title("Resultant Image")

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.hist(equalized.ravel(),256,[0,256], color='black')

    plt.title(title)

    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.xlim([0, 256])

    plt.tight\_layout()

    plt.show()

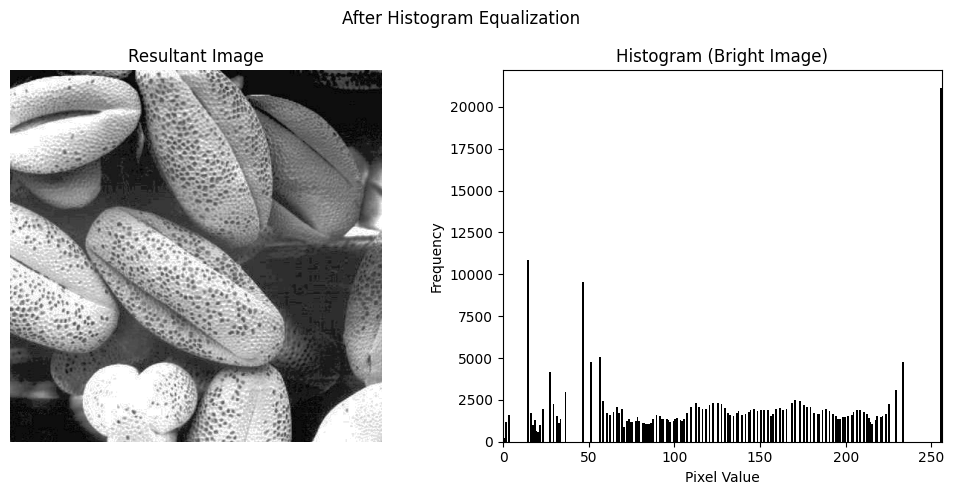
# Plotting image and their histogram after Histogram Equalization

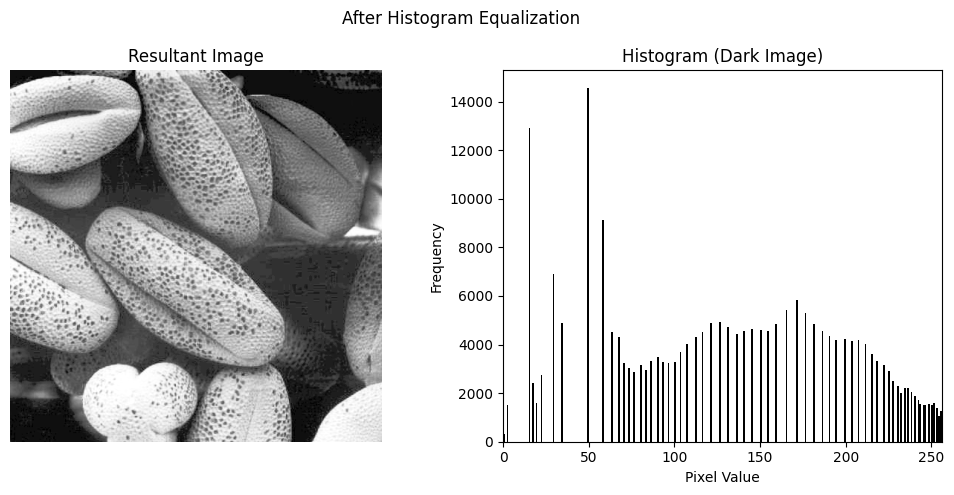
histogram\_equalization(ref1,'Histogram (Bright Image)','After Histogram Equalization')

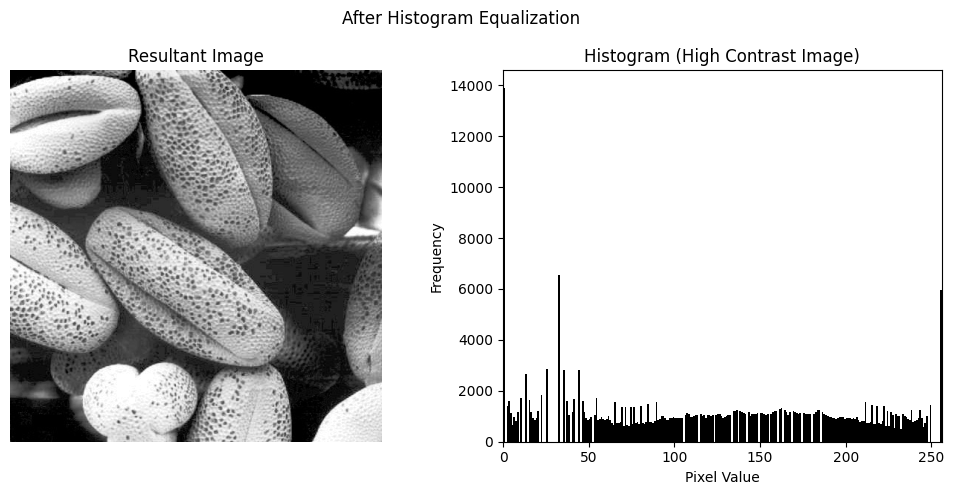
histogram\_equalization(ref2,'Histogram (Dark Image)','After Histogram Equalization')

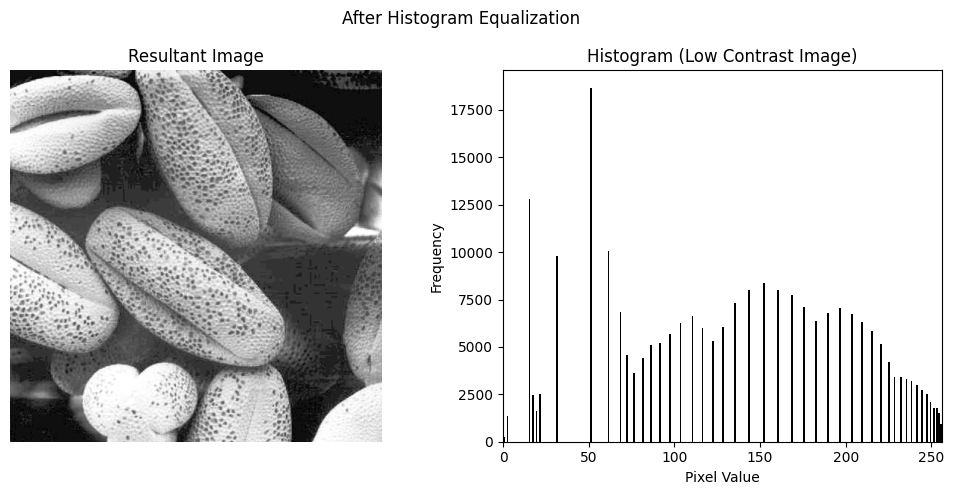
histogram\_equalization(ref3'Histogram(High Contrast Image)','After Histogram Equalization')

histogram\_equalization(ref4,'Histogram(Low Contrast Image)','After Histogram Equalization')









Histogram equalization enhances the contrast of images by spreading out the intensity values. The equalized images tend to have better contrast and detail compared to the original images, as seen in the histograms. However, sometimes histogram equalization can lead to unnatural-looking images, especially when there are already high contrasts in the original images. It's essential to evaluate the results based on the specific characteristics of each image.