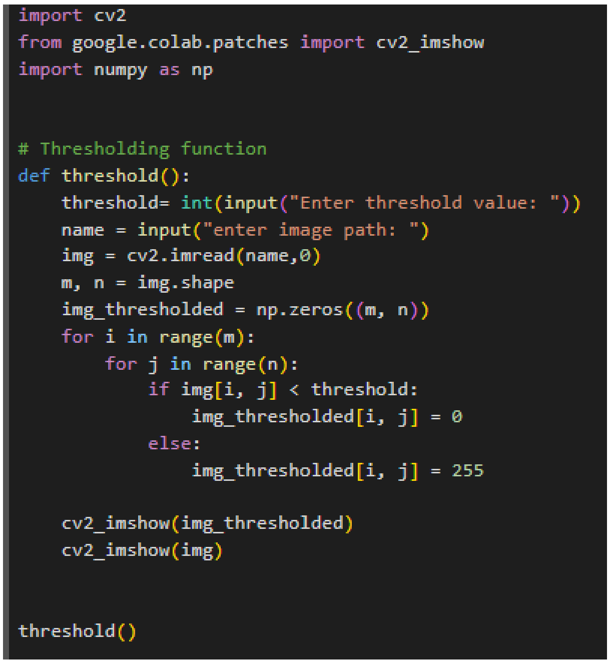
CO543 Image Processing E/19/167

Lab 02: Report Jayawardena H.D.N.S.

1. Image Thresholding

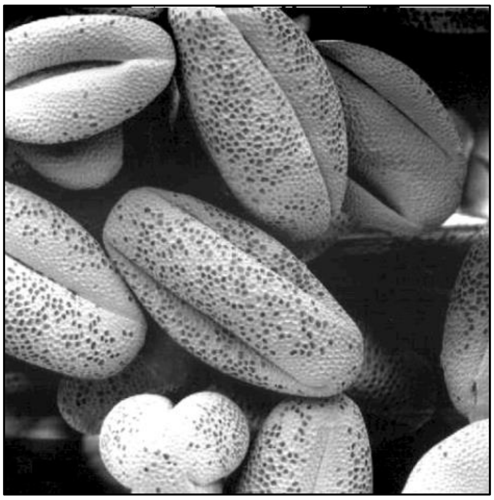
Task1

Code

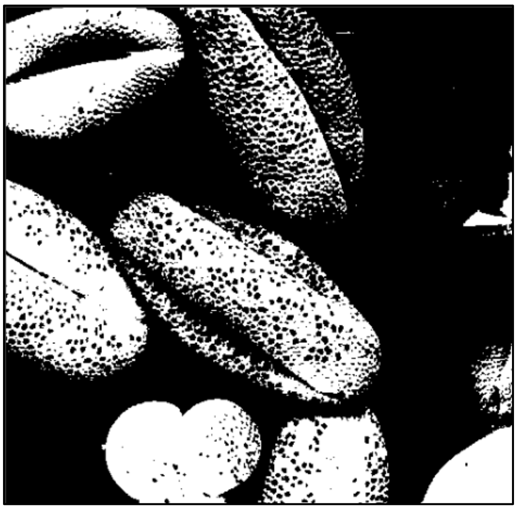


```
import cv2
from google.colab.patches import cv2_imshow
import numpy as np


# Thresholding function
def threshold():
    threshold= int(input("Enter threshold value: "))
    name = input("enter image path: ")
    img = cv2.imread(name,0)
    m, n = img.shape
    img_thresholded = np.zeros((m, n))
    for i in range(m):
        for j in range(n):
            if img[i, j] < threshold:
                img_thresholded[i, j] = 0
            else:
                img_thresholded[i, j] = 255

    cv2_imshow(img_thresholded)
    cv2_imshow(img)


threshold()
```

Input: high_contrast.tif



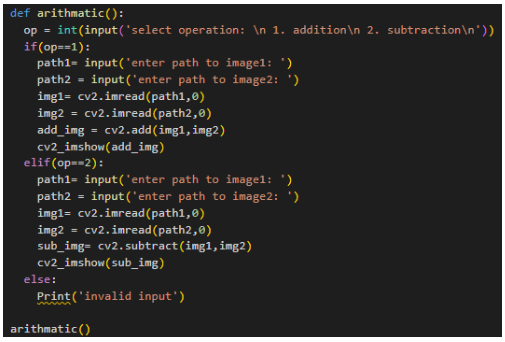
Output



2. Image Arithmetic Operation

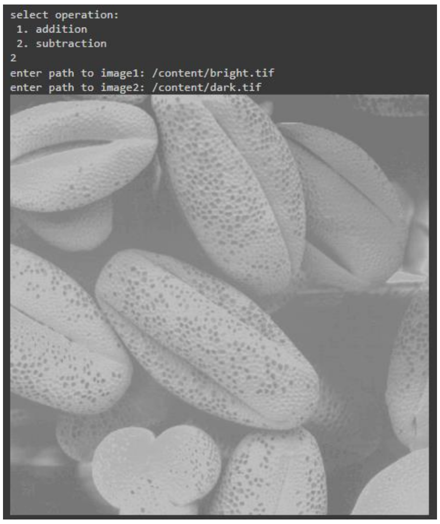
Lab task 2

Code



```
def arithmatic():
  op = int(input('select operation: \n 1. addition\n 2. subtraction\n'))
  if(op==1):
    path1= input('enter path to image1: ')
    path2 = input('enter path to image2: ')
    img1= cv2.imread(path1,0)
    img2 = cv2.imread(path2,0)
    add_img = cv2.add(img1,img2)
    cv2_imshow(add_img)
  elif(op==2):
    path1= input('enter path to image1: ')
    path2 = input('enter path to image2: ')
    img1= cv2.imread(path1,0)
    img2 = cv2.imread(path2,0)
    sub_img= cv2.subtract(img1,img2)
    cv2_imshow(sub_img)
  else:
    Print('invalid input')

arithmatic()
```

Output

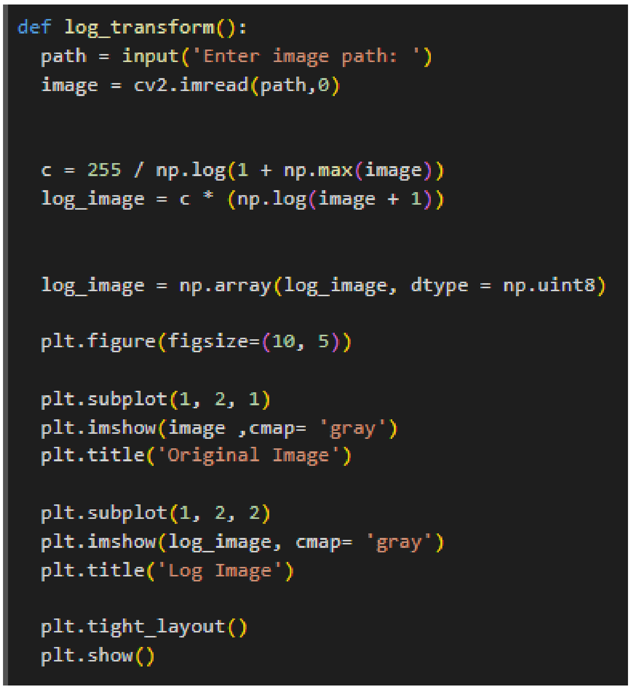


```
select operation:
 1. addition
 2. subtraction
2
enter path to image1: /content/bright.tif
enter path to image2: /content/dark.tif
```

3. Spatial Filtering

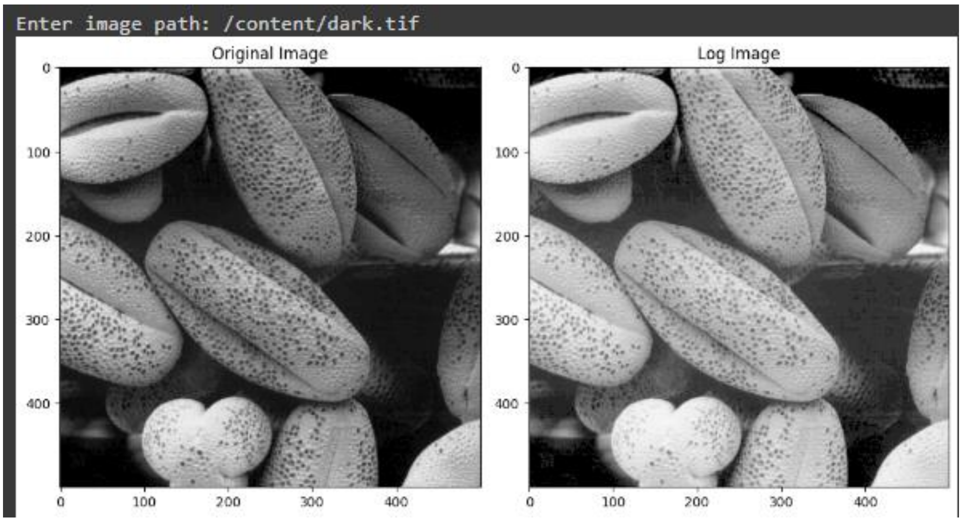
1. A) Log Transform

   Code



```
def log_transform():
  path = input('Enter image path: ')
  image = cv2.imread(path,0)


  c = 255 / np.log(1 + np.max(image))
  log_image = c * (np.log(image + 1))


  log_image = np.array(log_image, dtype = np.uint8)

  plt.figure(figsize=(10, 5))

  plt.subplot(1, 2, 1)
  plt.imshow(image ,cmap= 'gray')
  plt.title('Original Image')

  plt.subplot(1, 2, 2)
  plt.imshow(log_image, cmap= 'gray')
  plt.title('Log Image')

  plt.tight_layout()
  plt.show()
```

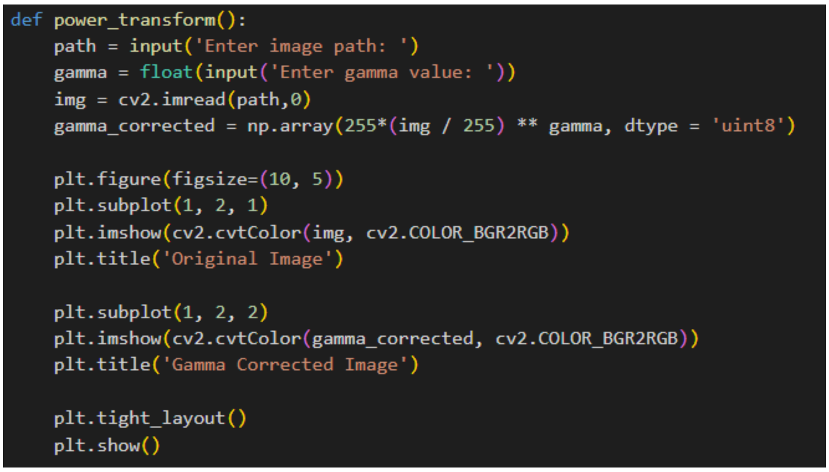
Output



```
Enter image path: /content/dark.tif
```

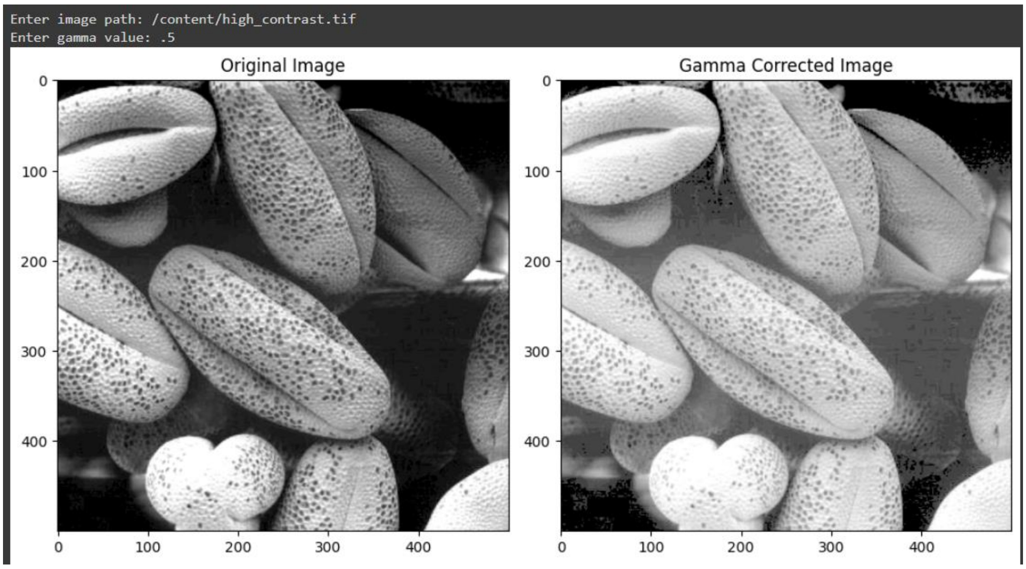
B) Power Transform

Code



```
def power_transform():
    path = input('Enter image path: ')
    gamma = float(input('Enter gamma value: '))
    img = cv2.imread(path,0)
    gamma_corrected = np.array(255*(img / 255) ** gamma, dtype = 'uint8')

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(gamma_corrected, cv2.COLOR_BGR2RGB))
    plt.title('Gamma Corrected Image')

    plt.tight_layout()
    plt.show()
```

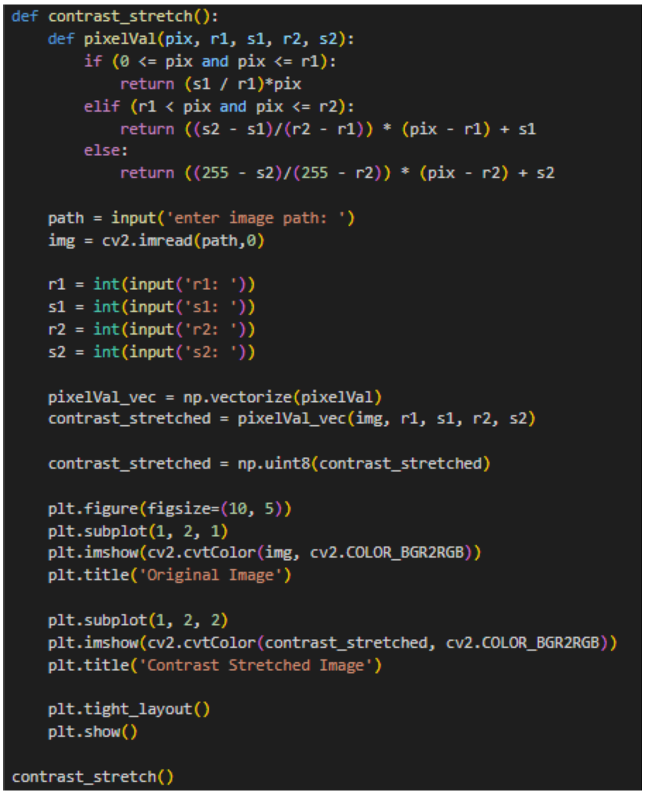
Output



```
Enter image path: /content/high_contrast.tif
Enter gamma value: .5
```

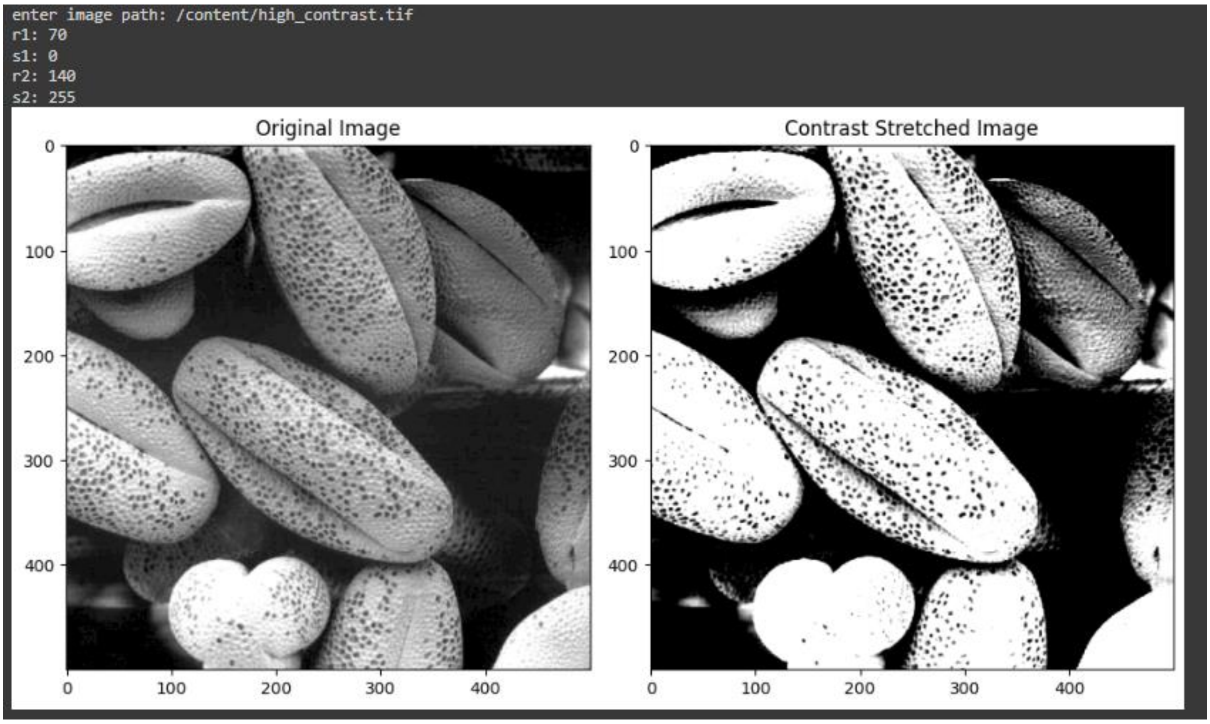
C) Contrast Streching

Code



```
def contrast_stretch():
    def pixelVal(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

    path = input('enter image path: ')
    img = cv2.imread(path,0)

    r1 = int(input('r1: '))
    s1 = int(input('s1: '))
    r2 = int(input('r2: '))
    s2 = int(input('s2: '))

    pixelVal_vec = np.vectorize(pixelVal)
    contrast_stretched = pixelVal_vec(img, r1, s1, r2, s2)

    contrast_stretched = np.uint8(contrast_stretched)

    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(contrast_stretched, cv2.COLOR_BGR2RGB))
    plt.title('Contrast Stretched Image')

    plt.tight_layout()
    plt.show()

contrast_stretch()
```

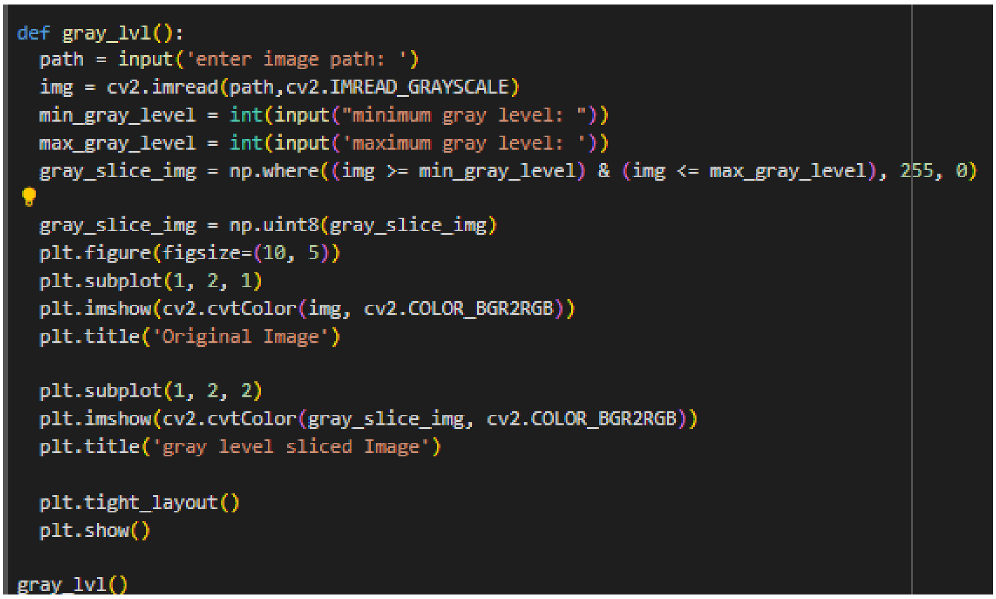
Output



```
enter image path: /content/high_contrast.tif
r1: 70
s1: 0
r2: 140
s2: 255
```

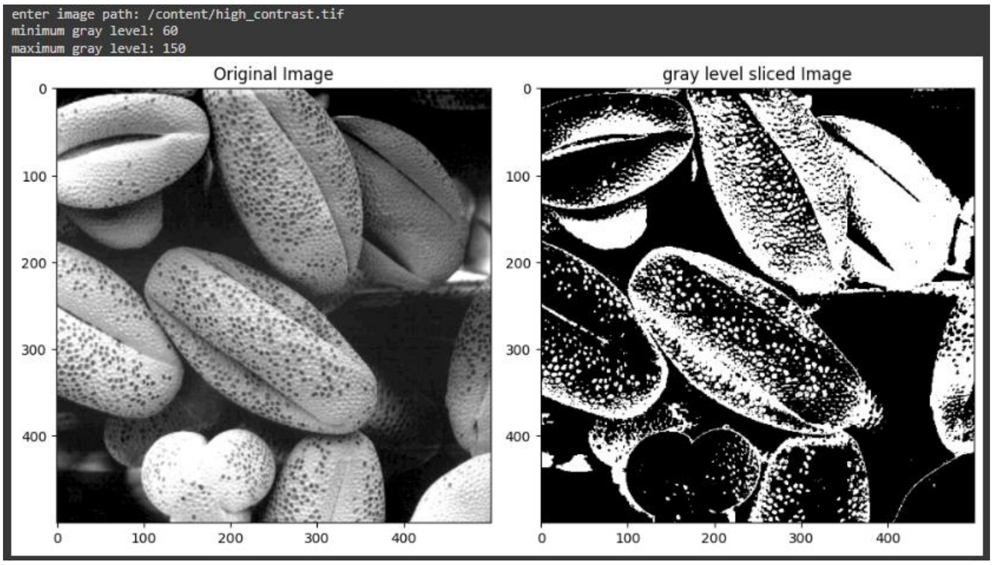
D) Gray level slicing

Code



```
def gray_lvl():
  path = input('enter image path: ')
  img = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
  min_gray_level = int(input("minimum gray level: "))
  max_gray_level = int(input('maximum gray level: '))
  gray_slice_img = np.where((img >= min_gray_level) & (img <= max_gray_level), 255, 0)

  gray_slice_img = np.uint8(gray_slice_img)
  plt.figure(figsize=(10, 5))
  plt.subplot(1, 2, 1)
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  plt.title('Original Image')

  plt.subplot(1, 2, 2)
  plt.imshow(cv2.cvtColor(gray_slice_img, cv2.COLOR_BGR2RGB))
  plt.title('gray level sliced Image')

  plt.tight_layout()
  plt.show()

gray_lvl()
```

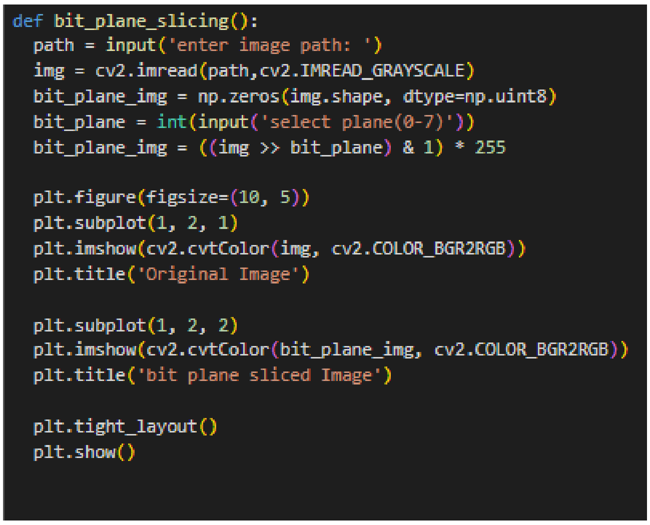
Output



```
enter image path: /content/high_contrast.tif
minimum gray level: 60
maximum gray level: 150
```

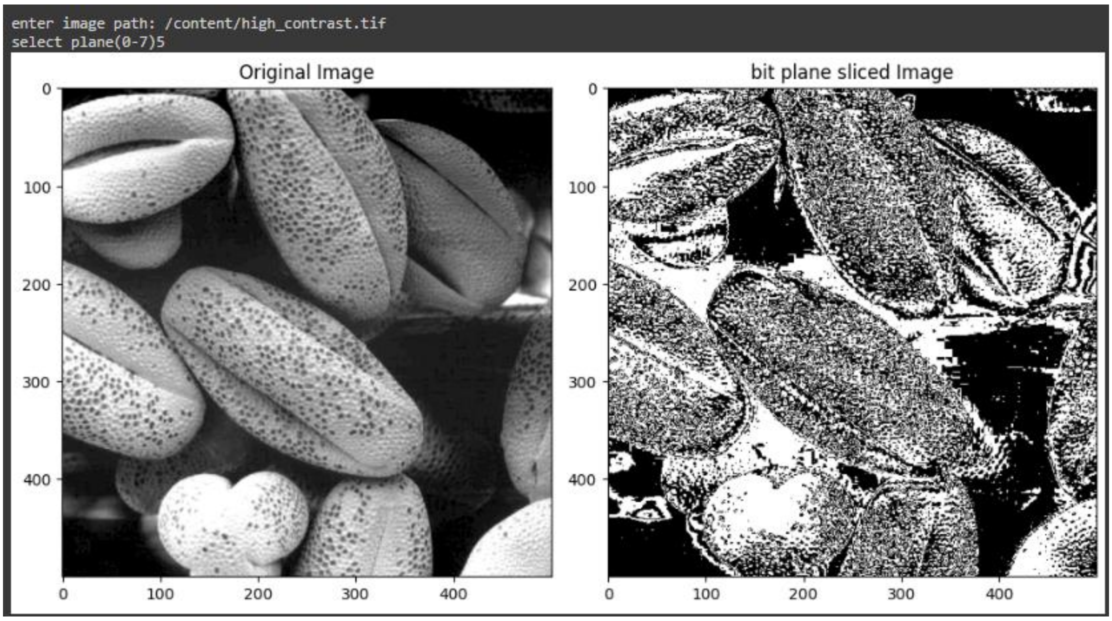
E) Bit Plane slicing

Code



```
def bit_plane_slicing():
  path = input('enter image path: ')
  img = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
  bit_plane_img = np.zeros(img.shape, dtype=np.uint8)
  bit_plane = int(input('select plane(0-7)'))
  bit_plane_img = ((img >> bit_plane) & 1) * 255

  plt.figure(figsize=(10, 5))
  plt.subplot(1, 2, 1)
  plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  plt.title('Original Image')

  plt.subplot(1, 2, 2)
  plt.imshow(cv2.cvtColor(bit_plane_img, cv2.COLOR_BGR2RGB))
  plt.title('bit plane sliced Image')

  plt.tight_layout()
  plt.show()
```

Output



```
enter image path: /content/high_contrast.tif
select plane(0-7)5
```

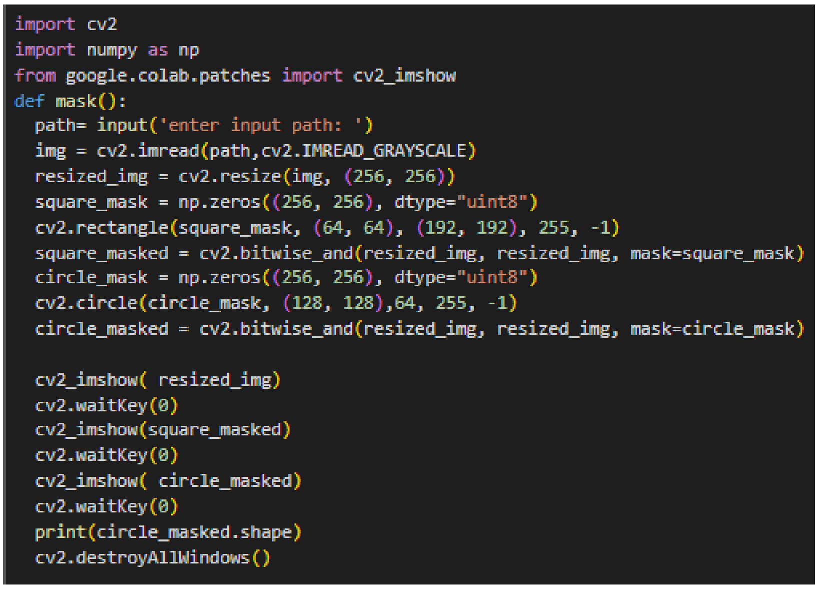
2. a) When r1 = s1 and r2 = s2, the contrast stretching function becomes a linear function that doesn't change the image. This is because the pixel values remain the same, i.e., the output pixel value is the same as the input pixel value. In other words, no contrast stretching is performed.

   b) When r1 = r2, s1 = 0 and s2 = L-1 (where L is the maximum pixel value, typically 255 for an 8-bit image), the contrast stretching function becomes undefined because mapping all pixel values to either 0 or L-1 is not possible. This scenario doesn't make sense in the context of contrast stretching because it doesn't provide a valid range of pixel values to stretch the contrast

4. Masking

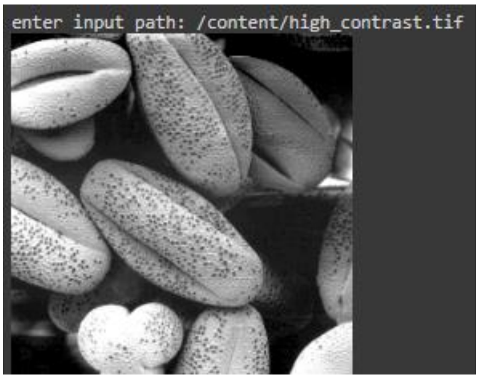
Lab Task 04

Code



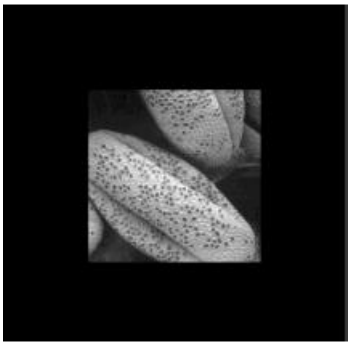
```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
def mask():
  path= input('enter input path: ')
  img = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
  resized_img = cv2.resize(img, (256, 256))
  square_mask = np.zeros((256, 256), dtype="uint8")
  cv2.rectangle(square_mask, (64, 64), (192, 192), 255, -1)
  square_masked = cv2.bitwise_and(resized_img, resized_img, mask=square_mask)
  circle_mask = np.zeros((256, 256), dtype="uint8")
  cv2.circle(circle_mask, (128, 128),64, 255, -1)
  circle_masked = cv2.bitwise_and(resized_img, resized_img, mask=circle_mask)

  cv2_imshow( resized_img)
  cv2.waitKey(0)
  cv2_imshow(square_masked)
  cv2.waitKey(0)
  cv2_imshow( circle_masked)
  cv2.waitKey(0)
  print(circle_masked.shape)
  cv2.destroyAllWindows()
```

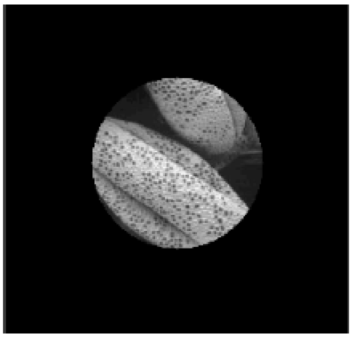
Input



```
enter input path: /content/high_contrast.tif
```

Output

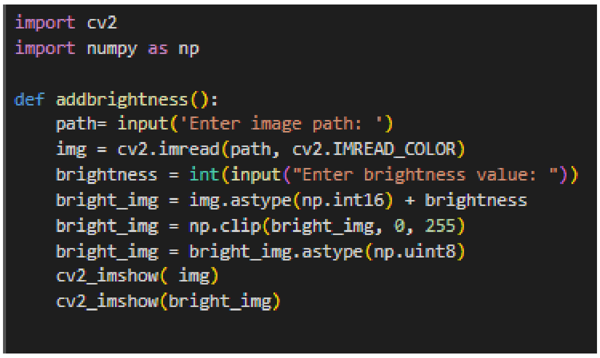




5. Brightness

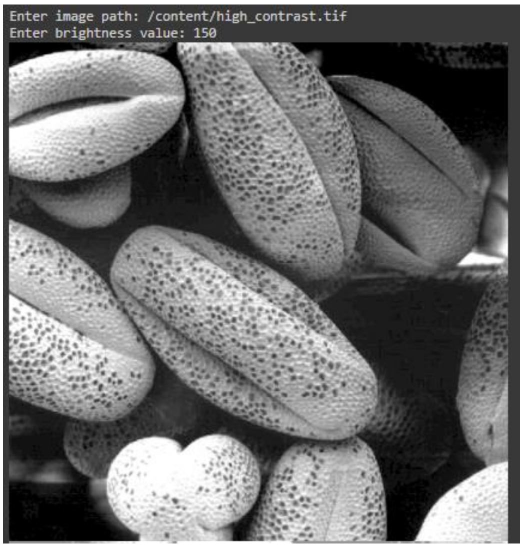
Lab Task 05

Code



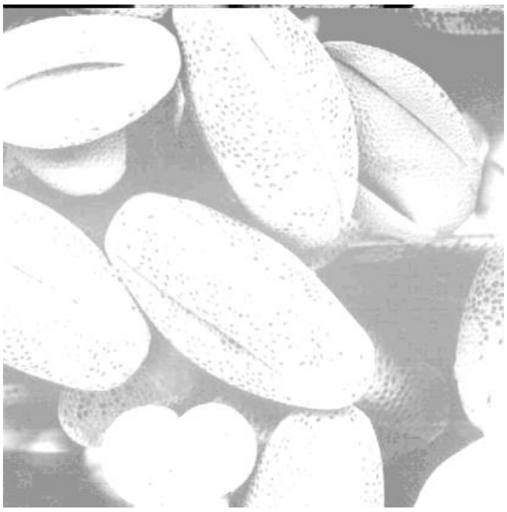
```
import cv2
import numpy as np

def addbrightness():
    path= input('Enter image path: ')
    img = cv2.imread(path, cv2.IMREAD_COLOR)
    brightness = int(input("Enter brightness value: "))
    bright_img = img.astype(np.int16) + brightness
    bright_img = np.clip(bright_img, 0, 255)
    bright_img = bright_img.astype(np.uint8)
    cv2_imshow( img)
    cv2_imshow(bright_img)
```

Input



```
Enter image path: /content/high_contrast.tif
Enter brightness value: 150
```

Output

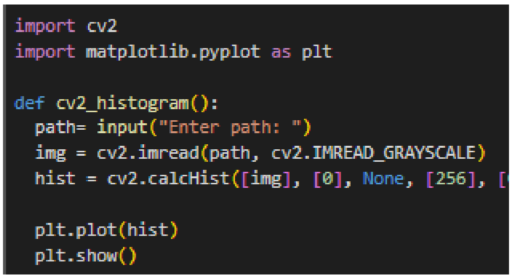


6. Histogram Processing

Lab Task 06

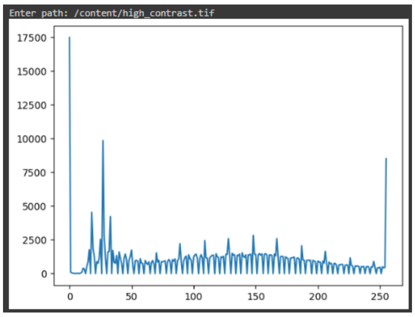
1. Histogram Calculation in OpenCV

Code



```
import cv2
import matplotlib.pyplot as plt

def cv2_histogram():
  path= input("Enter path: ")
  img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
  hist = cv2.calcHist([img], [0], None, [256], [

  plt.plot(hist)
  plt.show()
```

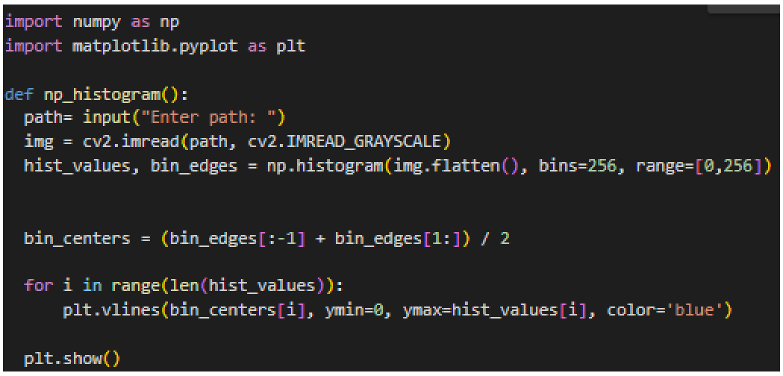
Output



```
Enter path: /content/high_contrast.tif
```

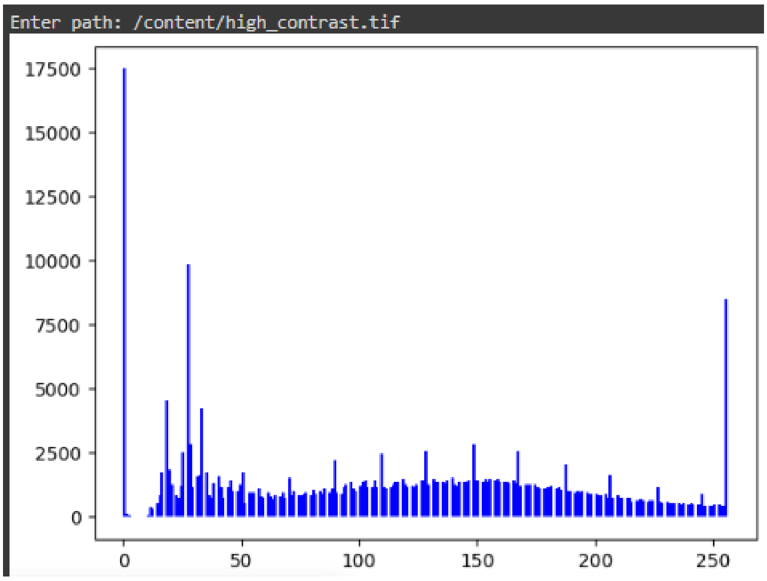
2. Histogram Calculation in NumPy

Code



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

def np_histogram():
  path= input("Enter path: ")
  img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
  hist_values, bin_edges = np.histogram(img.flatten(), bins=256, range=[0,256])


  bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2

  for i in range(len(hist_values)):
      plt.vlines(bin_centers[i], ymin=0, ymax=hist_values[i], color='blue')

  plt.show()
```

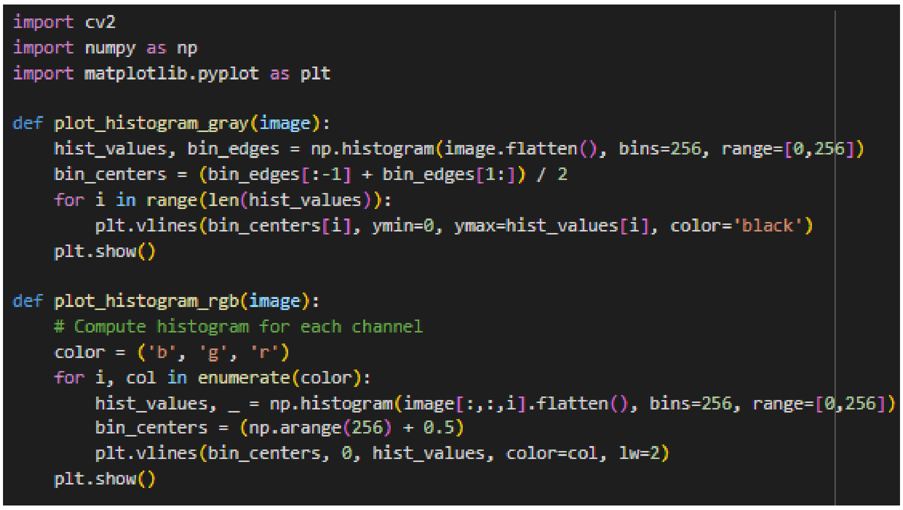
Output



```
Enter path: /content/high_contrast.tif
```

3.

Code

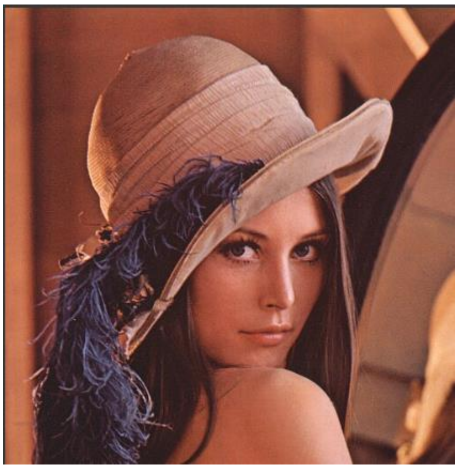


```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def plot_histogram_gray(image):
    hist_values, bin_edges = np.histogram(image.flatten(), bins=256, range=[0,256])
    bin_centers = (bin_edges[:-1] + bin_edges[1:]) / 2
    for i in range(len(hist_values)):
        plt.vlines(bin_centers[i], ymin=0, ymax=hist_values[i], color='black')
    plt.show()

def plot_histogram_rgb(image):
    # Compute histogram for each channel
    color = ('b', 'g', 'r')
    for i, col in enumerate(color):
        hist_values, _ = np.histogram(image[:,:,i].flatten(), bins=256, range=[0,256])
        bin_centers = (np.arange(256) + 0.5)
        plt.vlines(bin_centers, 0, hist_values, color=col, lw=2)
    plt.show()
```

Inputs

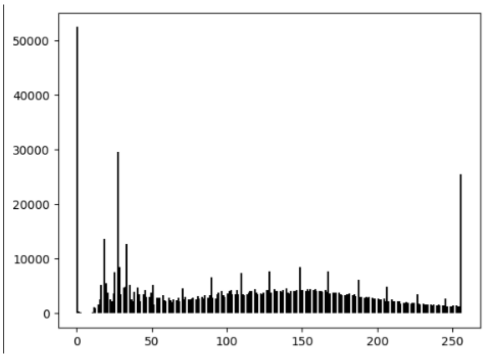
Grayscale

RGB

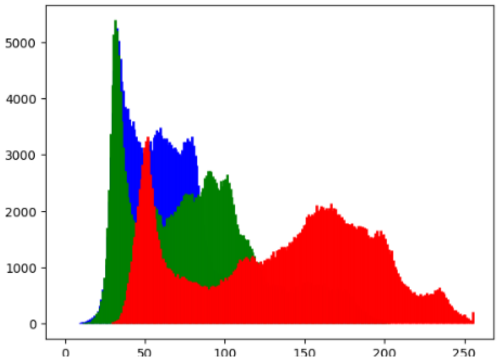


Histograms

Grayscale

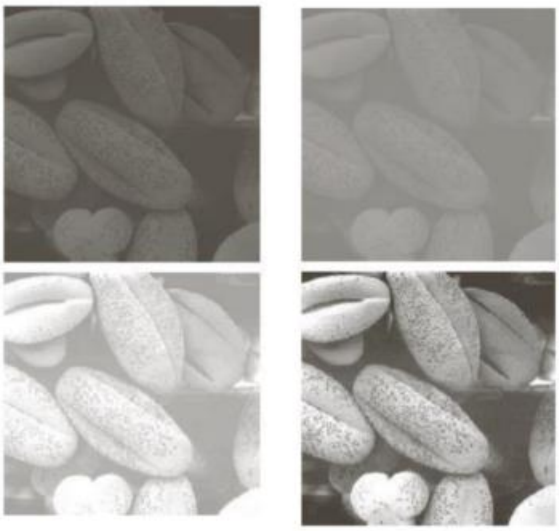


RGB

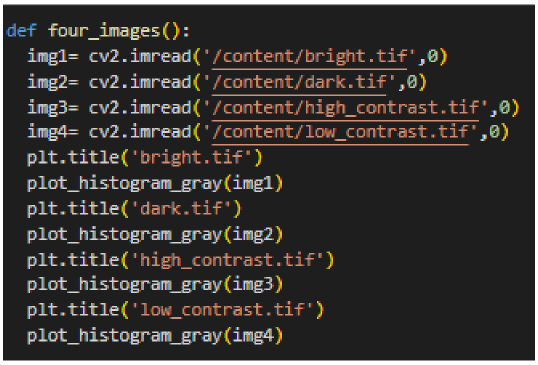


4.

Inputs

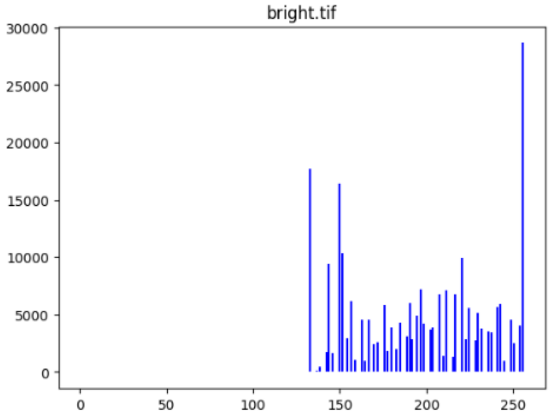


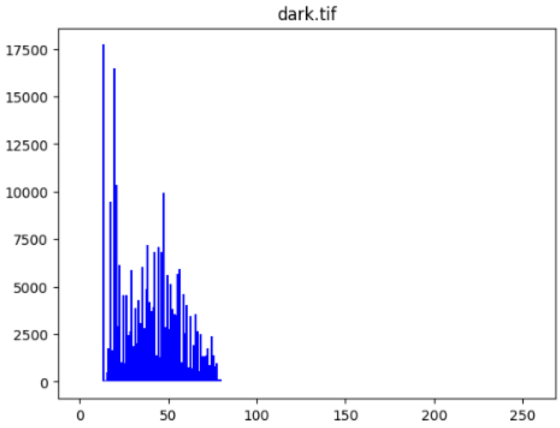
Code

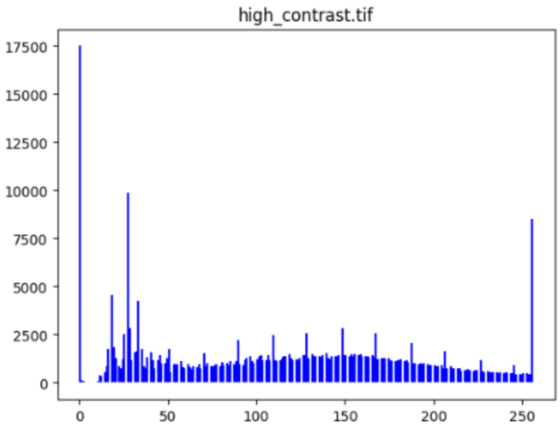


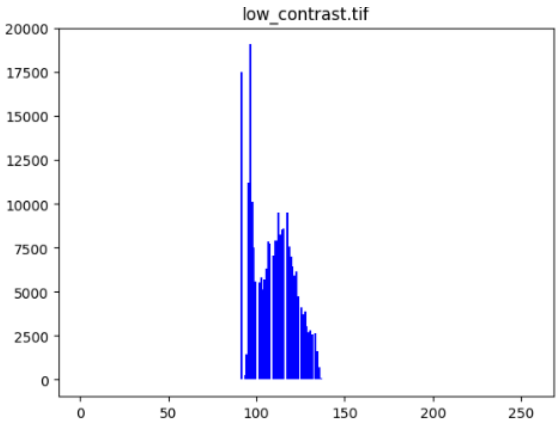
```
def four_images():
  img1= cv2.imread('/content/bright.tif',0)
  img2= cv2.imread('/content/dark.tif',0)
  img3= cv2.imread('/content/high_contrast.tif',0)
  img4= cv2.imread('/content/low_contrast.tif',0)
  plt.title('bright.tif')
  plot_histogram_gray(img1)
  plt.title('dark.tif')
  plot_histogram_gray(img2)
  plt.title('high_contrast.tif')
  plot_histogram_gray(img3)
  plt.title('low_contrast.tif')
  plot_histogram_gray(img4)
```

Outputs


bright.tif


dark.tif


high_contrast.tif


low_contrast.tif