**Local Histogram Analysis**

**Name:** Ahmad Amjad Mughal

**Reg no:** 121672

**Class:** BSCS-6C

**Q1: Consider the image given below. Perform Global histogram equalization to enhance this image. Save and show the original histogram and the processed histogram along with the input and resulting images.**

**Code:**

import cv2

import numpy as np

from matplotlib import pyplot as plt

# read a image using imread

img = cv2.imread('MyImage.png', 0)

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

plt.plot(hist)

plt.title(" Original Histogram")

plt.xlabel("Value")

plt.ylabel("Frequency")

plt.show()

# creating a Histograms Equalization

# of a image using cv2.equalizeHist()

equilised = cv2.equalizeHist(img)

# stacking images side-by-side

result = np.hstack((img, equilised))

# show image input vs output

hist2 = cv2.calcHist([equilised],[0],None,[256],[0,256])

plt.plot(hist2)

plt.title(" Original Histogram")

plt.xlabel("Value")

plt.ylabel("Frequency")

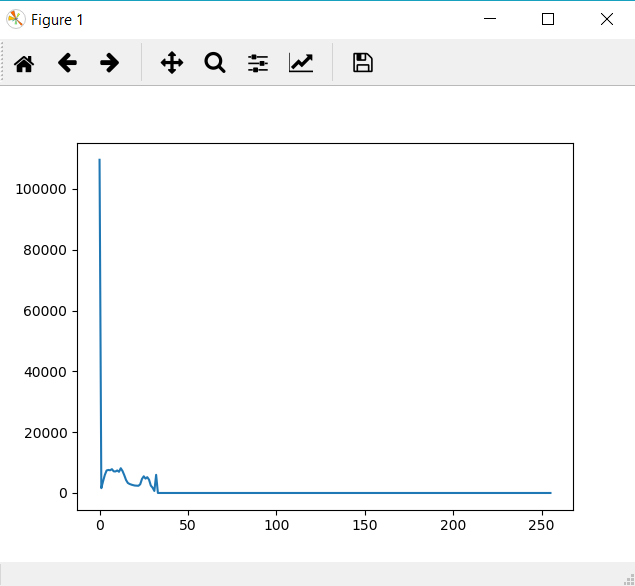
plt.show()

cv2.imshow('MyResultedImage.png', equilised)

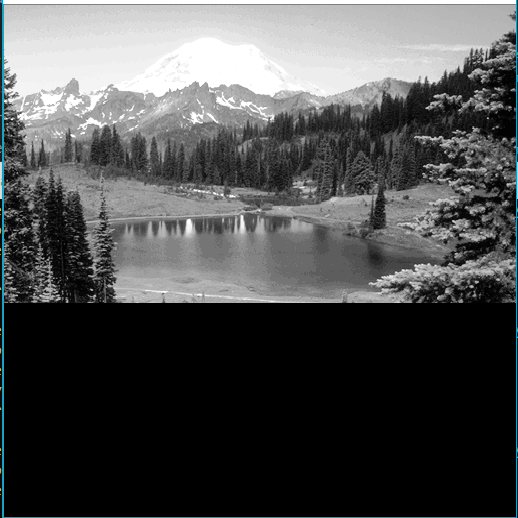
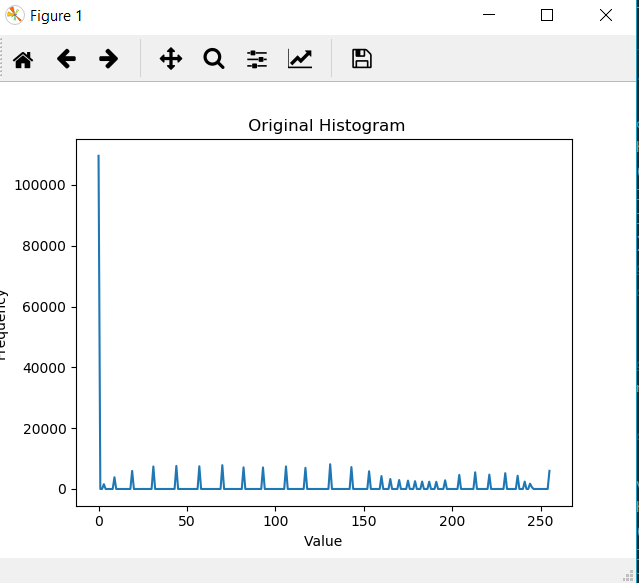
cv2.waitKey(0)

cv2.destroyAllWindows()

**Original Image**https://faculty.cs.byu.edu/~farrell/cs450/mystery.png **Graphs**



**Global Equalized Image**



**Q2: Perform the local histogram equalization of the same image by using**

1. **Tiling approach: at least four tiles. Save and show the original histograms and the processed histogram from each tile, along with the final complete image.**

**Code:**

**import cv2**

**import numpy as np**

**from matplotlib import pyplot as plt**

**from PIL import Image**

**img = cv2.imread('MyImage.png',0)**

**a = np.asarray(img)**

**#Blocks of Tiles are used in shape of Arrays**

**tile1= np.zeros((256,256),dtype=np.int8)**

**tile2= np.zeros((256,256),dtype=np.int8)**

**tile3= np.zeros((256,256),dtype=np.int8)**

**tile4= np.zeros((256,256),dtype=np.int8)**

**#Function Histogram that takes current block and perform Histogram equalization on that entire tile.**

**def histogram(x):**

**img1 = Image.fromarray(x, 'L')**

**plt.hist(x.ravel(),256,[0,256]); plt.show()**

**equ = cv2.equalizeHist(x.astype(np.uint8))**

**plt.hist(equ.ravel(),256,[0,256]); plt.show()**

**res = np.hstack((img1,equ))**

**return equ**

**#Dividing the whole image into number of blocks of same sizes**

**for row in range(len(a)):**

**for pixel in range(len(a[0])):**

**if (row<256 and pixel < 256):**

**tile1[row][pixel]= a[row][pixel]**

**elif (row <256 and pixel < 512):**

**tile2[row][pixel-256]=a[row][pixel]**

**elif((row<512 and row>255)and (pixel<256)):**

**tile3[row-256][pixel]=a[row][pixel]**

**else:**

**tile4[row-256][pixel-256]=a[row][pixel]**

**#Function calls for each block of tiles**

**t1=histogram(tile1)**

**t2=histogram(tile2)**

**t3=histogram(tile3)**

**t4=histogram(tile4)**

**#After applying Histogram Equalization on each tile we finally merged them into one resultant image**

**for row in range(len(a)):**

**for pixel in range(len(a[0])):**

**if (row<256 and pixel < 256):**

**a[row][pixel]=t1[row][pixel]**

**elif (row <256 and pixel < 512):**

**a[row][pixel]=t2[row][pixel-256]**

**elif((row<512 and row>255)and (pixel<256)):**

**a[row][pixel]= t3[row-256][pixel]**

**else:**

**a[row][pixel]=t4[row-256][pixel-256]**

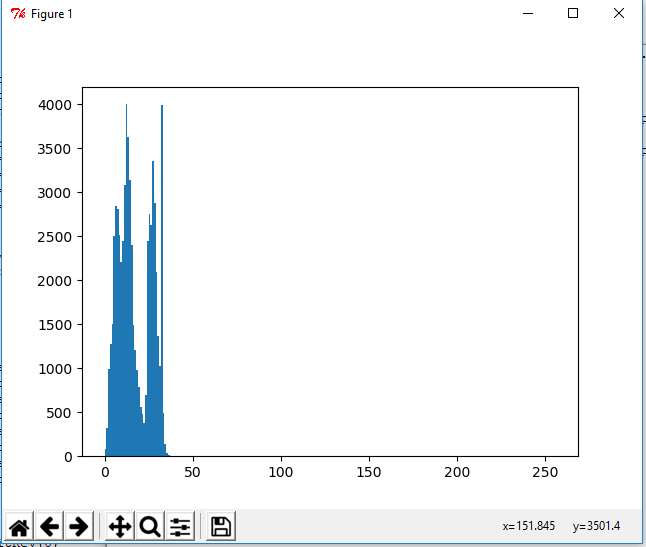
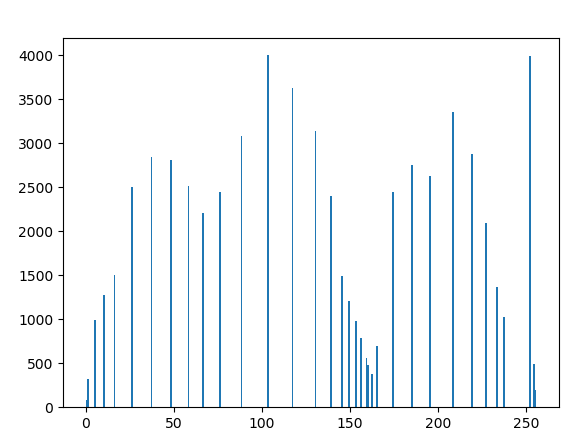
**cv2.imshow('res1',a)**

**cv2.waitKey(0)**

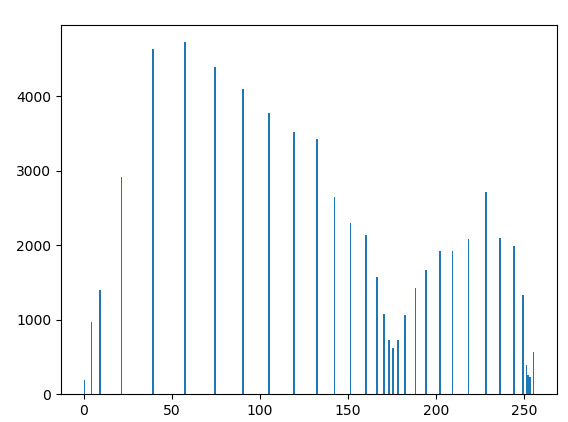
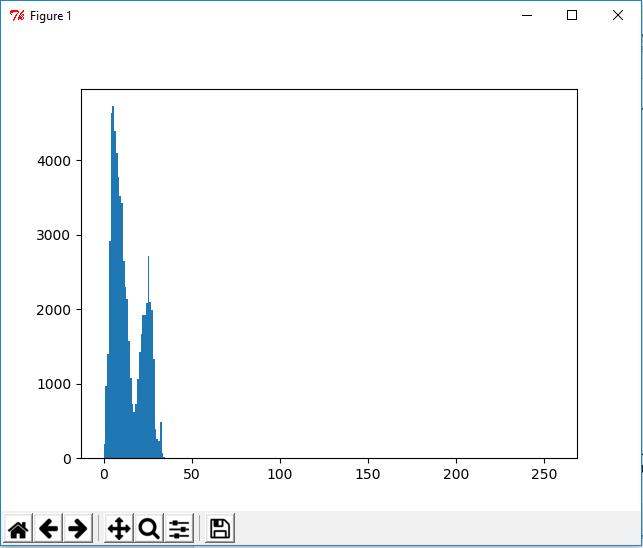
**cv2.destroyAllWindows()**

**Screenshot**

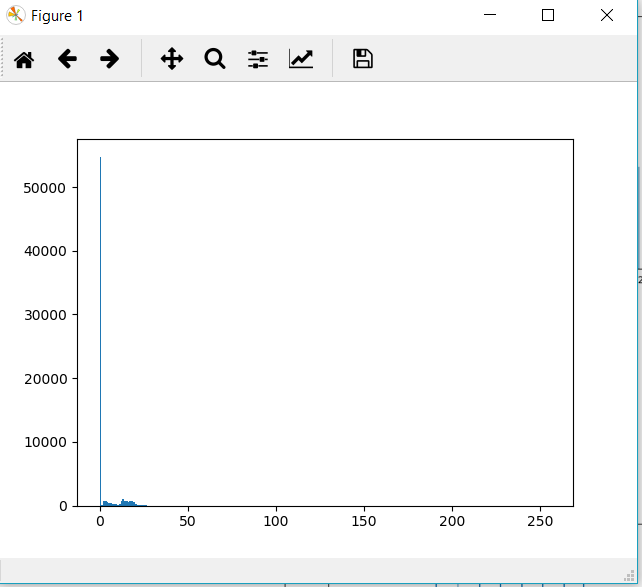
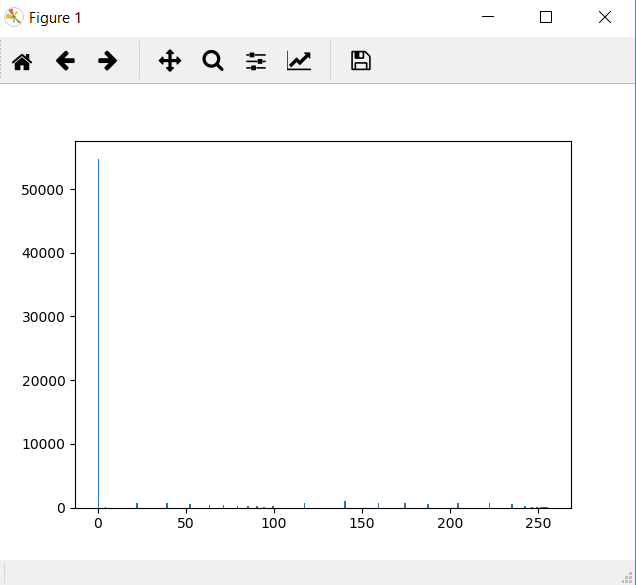
**Tile 1 before and After Histogram Equalization**

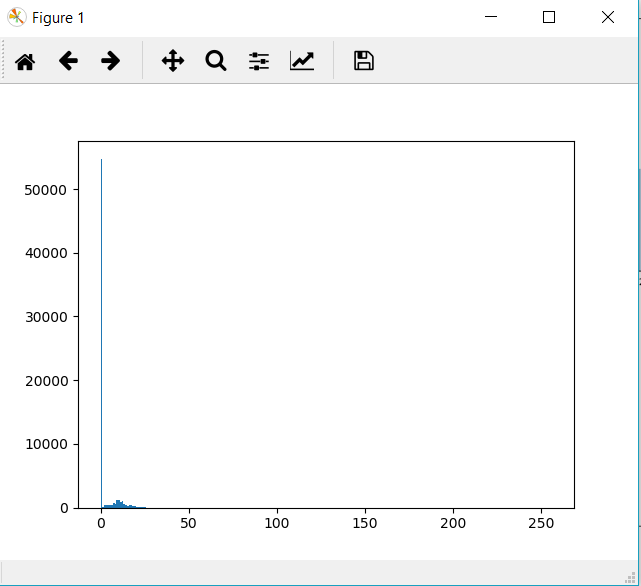
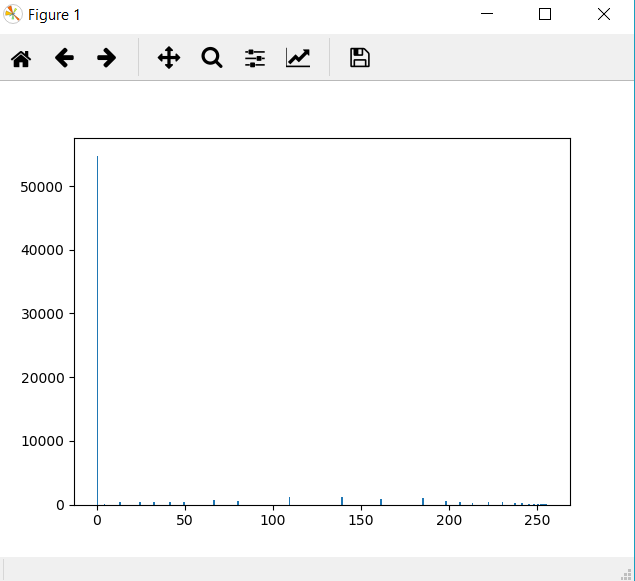
**Tile 2 before and After Histogram Equalization**

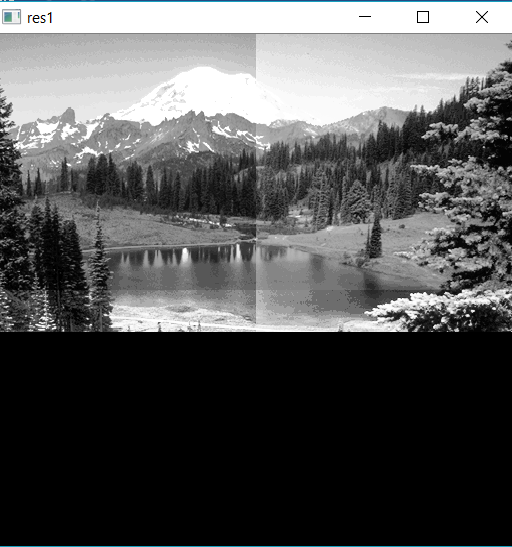


**Tile 3 before and after Histogram Equalization**

****

**Tile 4 before and after Histogram Equalization**

****

****

1. **By sliding window approach: displacement by one pixel size in each iteration. Save and show the original histograms and the processed histograms, along with the final complete image.**

**Code**

**import cv2**

**import matplotlib.pyplot as plt**

**import numpy as np**

**img = cv2.imread("MyImage.png",0)**

**equ = cv2.equalizeHist(img)**

**res = np.hstack((img,equ)) #stacking images side-by-side**

**img\_array = np.asarray(img)**

**newimage = np.zeros(img.shape, dtype=np.uint8)**

**tile1 = np.zeros((256,256), dtype=np.uint8)**

**for row in range(len(img\_array)-256):**

**for column in range(len(img\_array[0])-256):**

**for tile\_row in range(256):**

**for tile\_col in range(256):**

**tile1[row][column] = img\_array[row+tile\_row][column+tile\_col]**

**newimage[row][column] = tile1[row][column]**

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

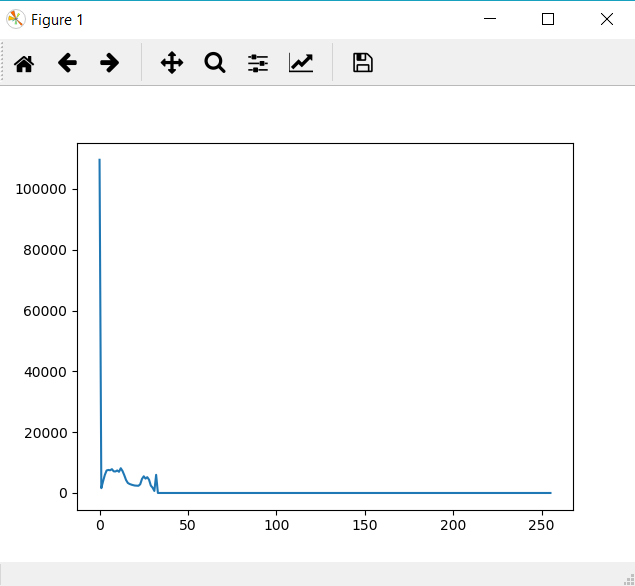
**plt.plot(hist)**

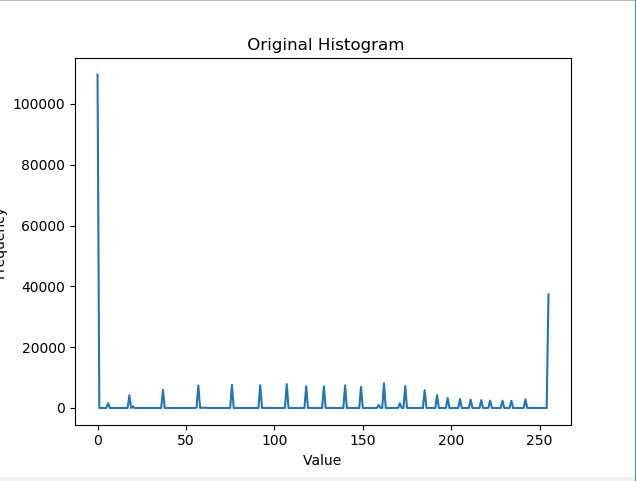
**plt.show()**

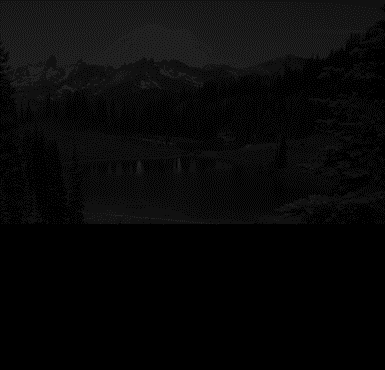
**cv2.imshow('resultantImage.png', newimage)**

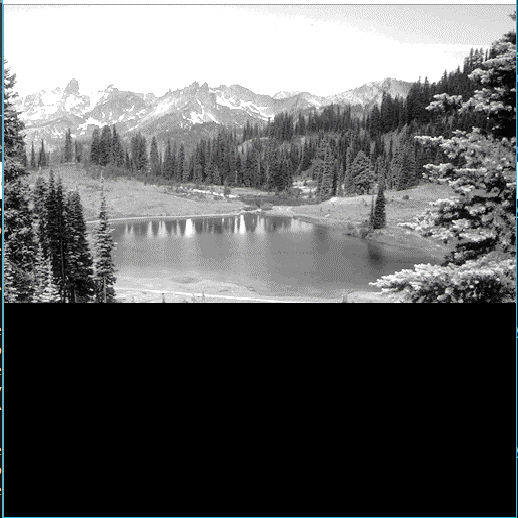
**cv2.waitKey(0)**

**cv2.destroyAllWindows()**





****



**Q3: What artifacts/ effects are observed when applying equalization at different levels in an image?**

Ans: By Applying Tiling approach, each block or tile is enhanced and pixels occupy maximum grey levels and much detail can be extracted at specific location. While in Slicing window, we have a fixed window of 256\*256 as I used this in my program and neighboring window operations are performed. This local equalization takes much time as compared to tiling approach. No doubt, contrast is much improved in both cases as greater grey values are obtained or used but tiling method is much better than Slicing method.

**Conclusion**

We performed global as well as local equalization to enhance image. Both techniques are used to improve the contrast as well as brightness of image. We also do comparison of these local equalizations and reach to a point that Tiles and Slicing have certain advantages and disadvantages.