Digital Image Processing
Assignment # 1

Submitted by

Ali Akram 260336
Minhaj Khokhar 265439

Submitted to Dr. Shahzor Ahmad

DEPARTMENT OF ELECTRICAL ENGINEERING

College of Electrical and Mechanical Engineering (CEME)

Task Distribution

Ali Akram

- Problem 1 and Problem 3

Minhaj Khokhar

- Problem 2

Problem # 1

Gamma Mapping of HDR Images

Part A

Grayscale removes all color information, leaving only the luminance of each pixel. Making edges or feature of image easy to identify based on luminance of pixel. Below we can see the original image with its grayscale.


image Gray 1


image 1


image 2

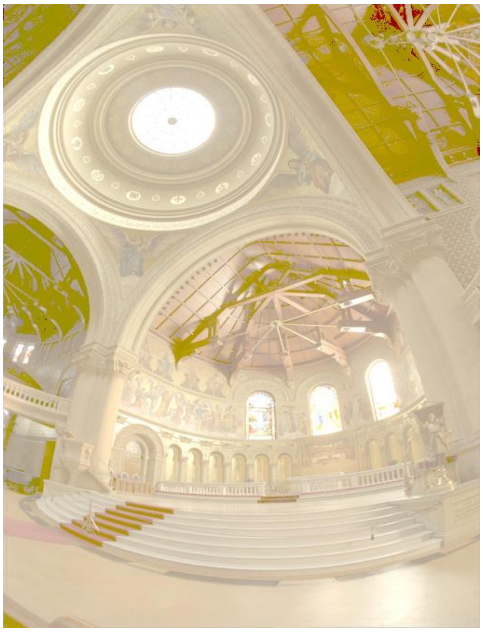

image Gray 2

Part B

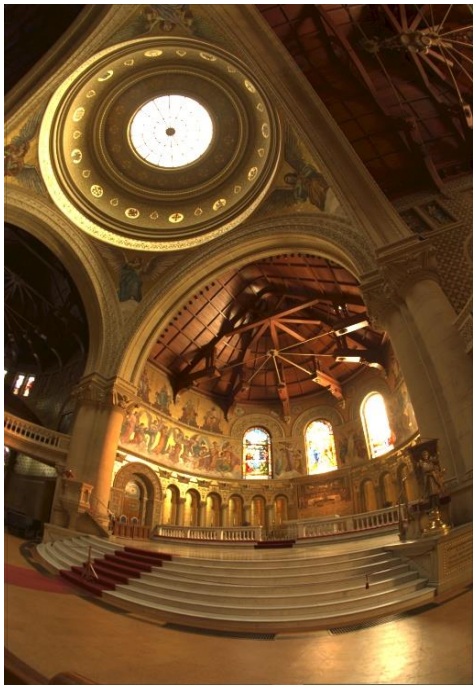
Image 1

Gamma = 0.1

Image 2



Gamma = 0.5



Gamma = 1.2

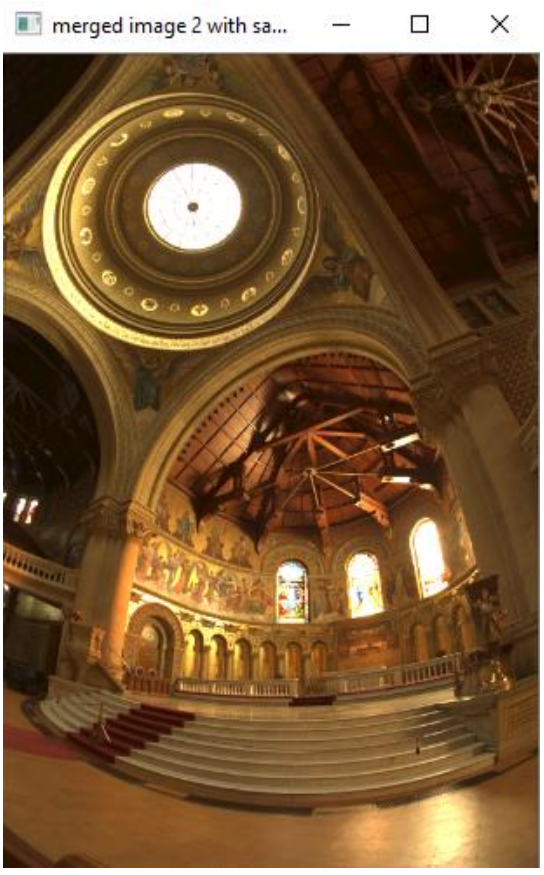
Gamma = 2.2

We saw that gamma value 0.5 give us best result. As in 1.2 and 2.2 value image become darker and for 0.1 image is washed out

```
gamma_corrected = np.array(255 * (img1 / 255) ** gamma, dtype='uint8')
cv2.imwrite('gamma_transformed' + str(gamma) + '.jpg', gamma_corrected)
```

Part C

Applying same gamma value 0.5 to each channel since we saw above it gave us best result.



Applying different values to different channel

By applying gamma value 2 to R channel 1.5 to green and 0.5 to blue channel we saw that image show more blue color

By applying gamma value 0.5 to R channel 1.5 to green and 2 to blue channel we saw that image show more Red color

Using different values of gamma for different channel give us image with some color channel more prominent while same gamma value for R, G, B channel gave us better image with all three channels mapped same.

Splitting

```
(B, G, R) = cv2.split(img1)
# show each channel individually
cv2.imshow("Red", R)
cv2.imshow("Green", G)
cv2.imshow("Blue", B)
```

Same gamma value to all channel

```
gamma_corrected1 = np.array(255 * (B / 255) ** 0.5, dtype='uint8')
gamma_corrected2 = np.array(255 * (G / 255) ** 0.5, dtype='uint8')
gamma_corrected3 = np.array(255 * (R / 255) ** 0.5, dtype='uint8')
cv2.imwrite('gamma_transformed1_same' + str(0.5) + '.jpg', gamma_corrected1)
cv2.imwrite('gamma_transformed2_same' + str(0.5) + '.jpg', gamma_corrected2)
cv2.imwrite('gamma_transformed3_same' + str(0.5) + '.jpg', gamma_corrected3)
```

Different gamma value for each channel

```
gamma_corrected1 = np.array(255 * (B / 255) ** 0.5, dtype='uint8')
gamma_corrected2 = np.array(255 * (G / 255) ** 1.5, dtype='uint8')
gamma_corrected3 = np.array(255 * (R / 255) ** 2, dtype='uint8')
merged = cv2.merge([gamma_corrected1, gamma_corrected2, gamma_corrected3])
cv2.imshow('(2) merged image with different gamma value ', merged)
```

Problem # 2

Image Subtraction for Tamper Detection

Mean Squared Error between the two images is the sum of the squared difference between the two images

```
def mse(imageA, imageB):
    err = np.sum((imageA.astype("float") - imageB.astype("float")) ** 2)
    err /= float(imageA.shape[0] * imageA.shape[1])
    return err
```

Defining the function to shift images

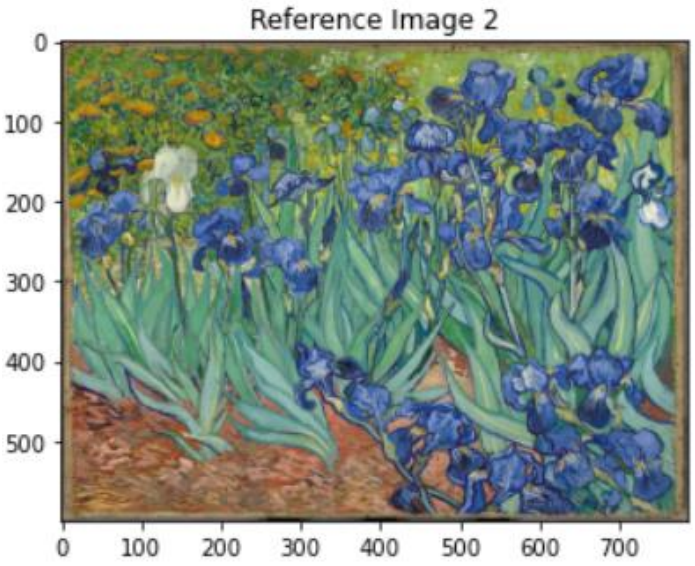
```
def shift(im_temp, im_real, i, typ):
    if typ == 'hr':
        img_s = np.zeros_like(im_temp)
        img_s[:, i + 1:] = im_temp[:, :-(i + 1)]
    if typ == 'hl':
        img_s = np.zeros_like(im_temp)
        img_s[:, :-(i + 1)] = im_temp[:, i + 1:]
    if typ == 'vu':...
    if typ == 'vd':...
    if typ == 'ul':...
    if typ == 'ur':...
    if typ == 'll':...
    if typ == 'lr':
        img_s = np.zeros_like(im_temp)
        img_s[:-(i + 1), :-(i + 1)] = im_temp[i + 1:, i + 1:]

    return mse(im_real, img_s), img_s
```

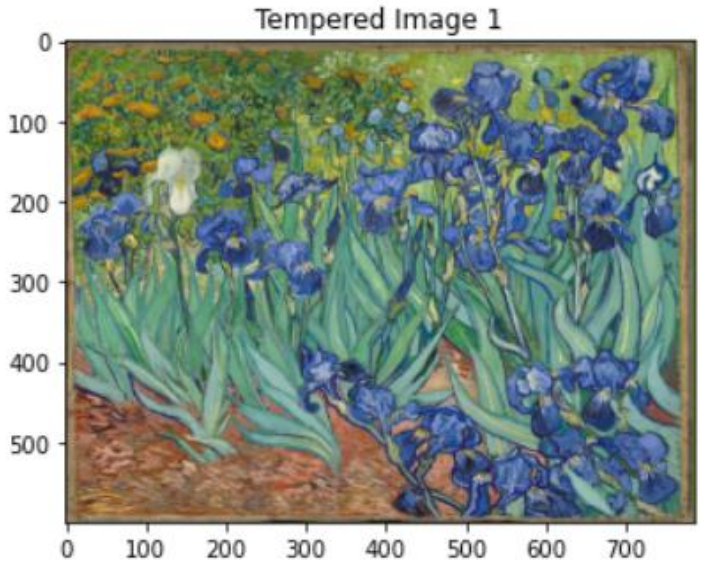
Defining function for finding the minimum error index and the direction of shift
Creating dictionary to store error

```
def MSD(im_temp, im_real):
    error = {'hr': [], 'hl': [], 'vu': [], 'vd': [], 'ul': [], 'ur': [], 'll': [], 'lr': []}
```

Min_e will store min value of error in all direction when we shift

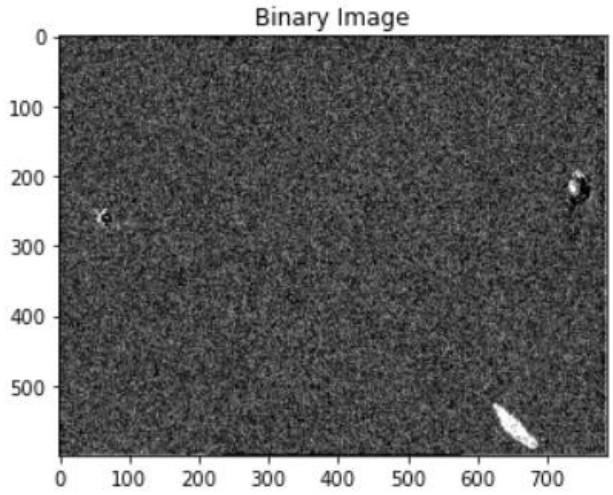
Processing image 1



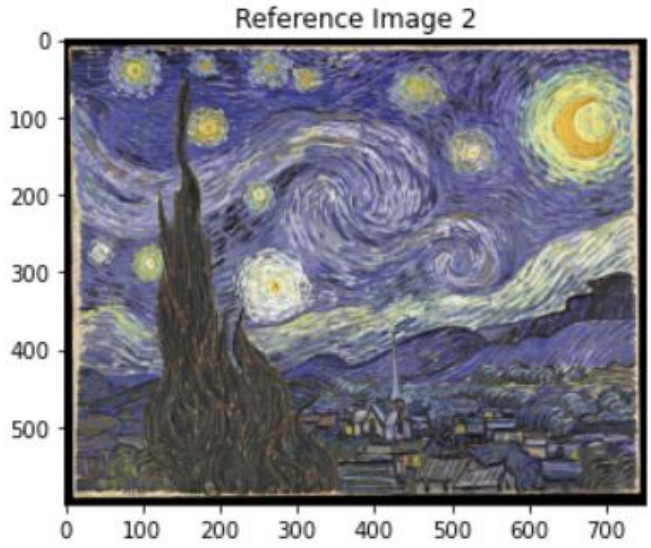


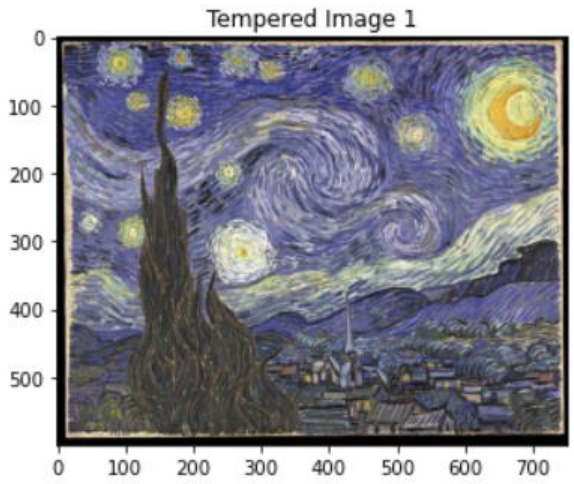

```
shift_index, typ = MSD(im_temp1, im_real1)

print("Value of pixel shift:", shift_index)
print("Type of shift:", typ)

_, img_s1 = shift(im_temp1, im_real1, shift_index, typ)

plt.imshow(img_s1)
```

```
subtracted_img1 = im_real1 - img_s1

gray = cv2.cvtColor(subtracted_img1, cv2.COLOR_BGR2GRAY)

ret, bw_img = cv2.threshold(gray,127,255,cv2.THRESH_BINARY)
plt.imshow(bw_img, cmap = 'gray')
plt.title('Binary Image')
```



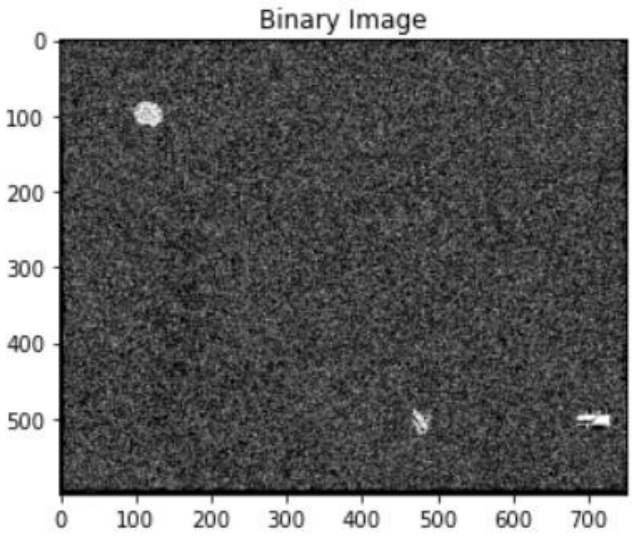
Processing image 2






```
shift_index, typ = MSD(im_temp2, im_real2)

print("Value of pixel shift:", shift_index)
print("Type of shift:", typ)

_, img_s2 = shift(im_temp2, im_real2, shift_index, typ)

plt.imshow(img_s2)
```

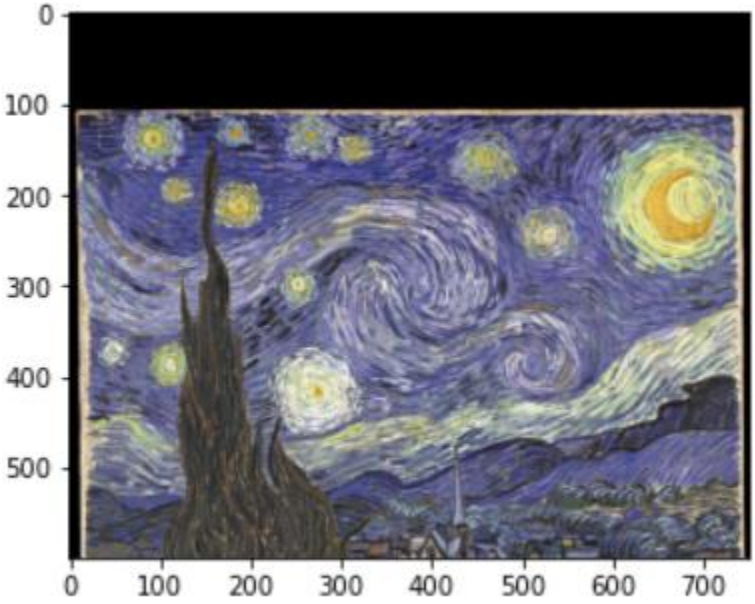
```
subtracted_img2 = im_real2 - img_s2

gray = cv2.cvtColor(subtracted_img2, cv2.COLOR_BGR2GRAY)

ret, bw_img = cv2.threshold(gray,127,255,cv2.THRESH_BINARY)
plt.imshow(bw_img, cmap = 'gray')
plt.title('Binary Image')
```



Just to demonstrate the shifting of images

```
img2 = np.zeros_like(im_temp2)
img2[:-50,:-50]= im_temp[50:,50:]
plt.imshow(img2)
```

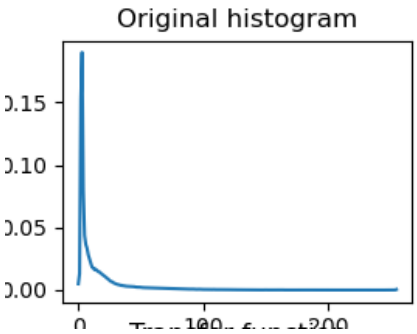

```
img3 = np.zeros_like(im_temp2)
img3[100:,:] = im_temp[:-100,:]
plt.imshow(img3)
```



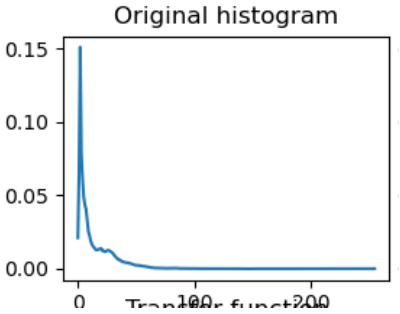
Problem # 3
Nighttime Road Contrast Enhancement

Part A

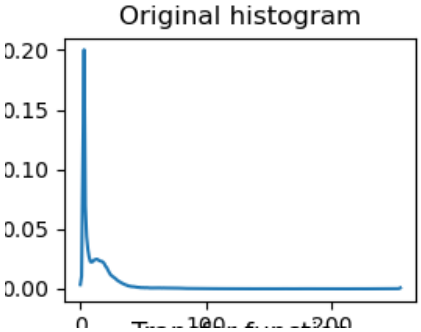
hw1_dark_road_1.jpg



hw1_dark_road_2.jpg



hw1_dark_road_3.jpg

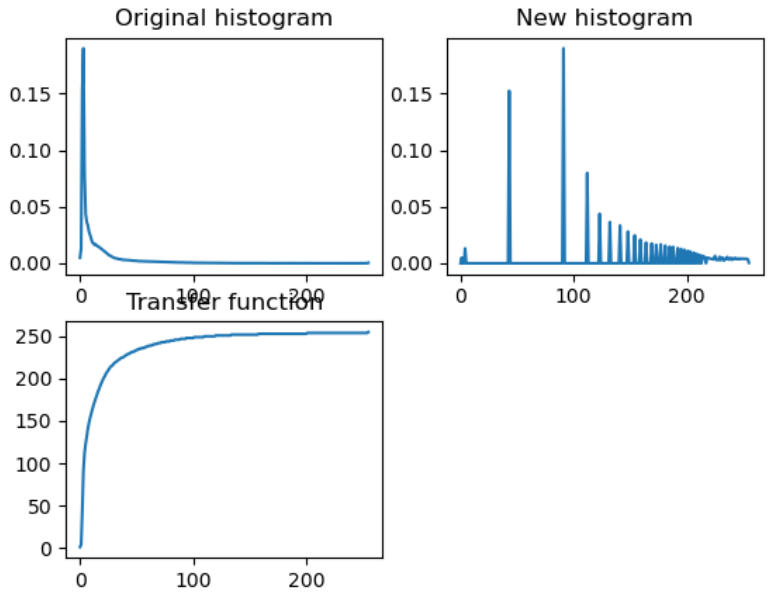


We can see that components of histogram of three images are concentrated on the low side of the gray scale so its a dark image

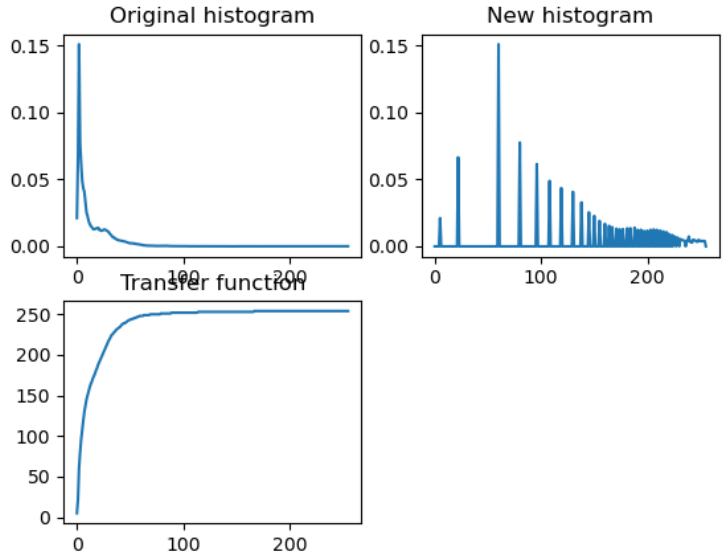
Part B
Histogram Equalization

Histogram equalization is to re-assigns the intensity values of pixels in the input image such that the output image contains a near uniform distribution of intensities

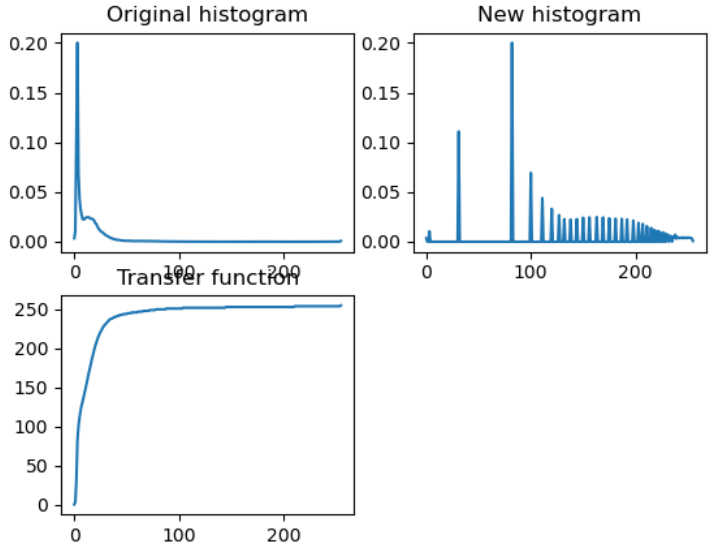
hw1_dark_road_1.jpg



hw1_dark_road_2.jpg



hw1_dark_road_3.jpg

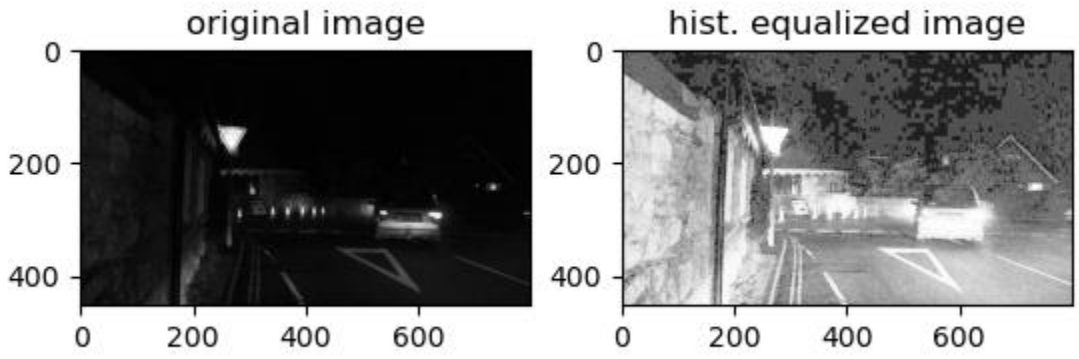


Histogram Equalization

```
h = imhist(im)
cdf = np.array(cumsum(h))
sk = np.uint8(255 * cdf)
s1, s2 = im.shape
Y = np.zeros_like(im)

for i in range(0, s1):
    for j in range(0, s2):
        Y[i, j] = sk[im[i, j]]
H = imhist(Y)
```

Transverse Function

```
return [sum(h[:i+1]) for i in range(len(h))]
```



Part C

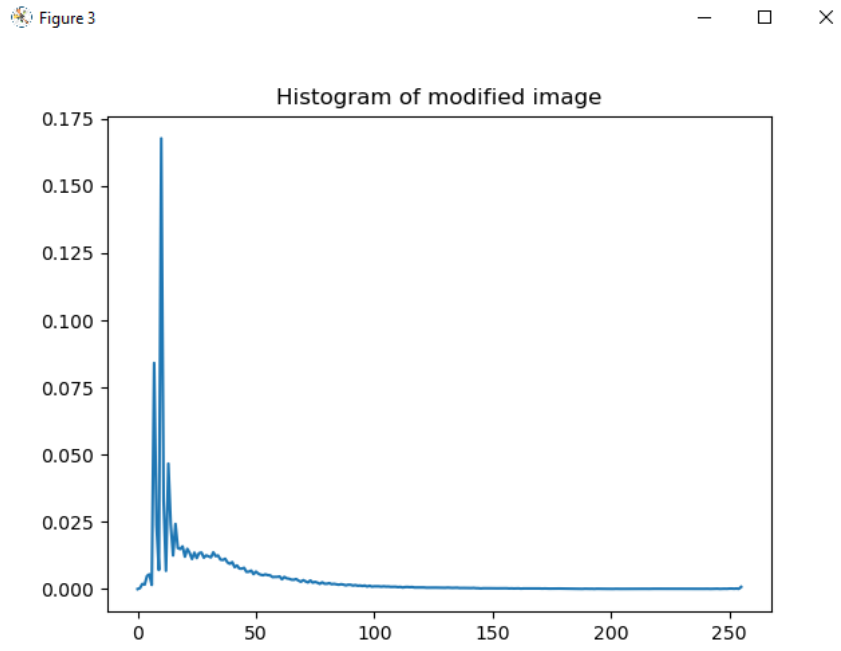
Adaptive Histogram Equalization

Adaptive histogram equalization is a technique used to improve contrast in images. It differs from ordinary histogram equalization in the respect that the adaptive method computes several histograms to a distinct section of the image.

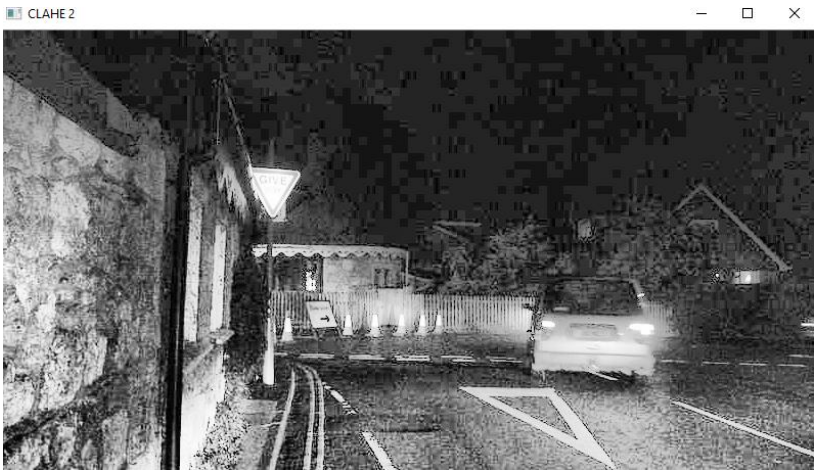
Clipping limit is threshold for contrast limiting. By experimenting with different values we saw that if we increase its value it will, in turn increase its noise.

Tile is grid size for CLAHE. It is dividing our input image into tile x tile cells and then applying histogram equalization to each cell

Here we used clipLimit = 3.0, tileGridSize = (8, 8) for best result



Below is a case of increase in noise when we increase cliplimit value to 15



So by experiment the best value is 3-5 if we increase it value further the noise also increases