CSGY 6634 Computer Vision

Project-1 Report

By Shreeraj Pawar

srp8095

Table of Contents

Introduction

Histogram Equalization is a technique used in image processing to adjust the contrast of the image. We perform this task by using the histogram of the image.

A histogram is a graph of pixel intensity value on the X-axis and the frequency of the pixel having a particular intensity value on the Y-axis. The intensity values for a grayscale image lie between 0-255 with 0 being a black pixel and 255 being a white pixel.

To perform histogram equalization, we need try to spread the frequency of dark and light pixels, so light images get darker and dark images get brighter. We do this by normalizing the cumulative density function of the histogram.

Steps for Histogram Equalization –

- Convert the image into “uint8” form so that all the pixel intensities are integers.
- Generate a histogram h where $h(i) = number of pixels with intensity i$
- Find the probability density function PDF of the histogram h. This can be done by dividing the histogram by the number of pixels N. $p(i)$ tells us what the probability is to find a pixel with intensity i.

$$p(i) = \frac{h(i)}{N}$$

- Find the cumulative density function of the PDF.

$$cdf(i) = \sum_{i=0}^{L} p(i) \qquad L = max\ intensity$$

- Normalize the CDF by multiplying each element by (L-1)
- Use the normalized CDF as a mapping function for to get the output image

$$S = T(r) \qquad r = input image T = mapping function$$
$$S = output\ image$$

Now, the question remains why does this work. To understand this we will consider the following CDF of input image –

Pixel Intensity i	CDF(i)
178	0.52
190	0.55
223	0.66
230	0.77

Now when we multipy by (L – 1) which is 255 and floor them we get the new pixel intensity values –

Pixel Intensity i	New Pixel Intensities
178	117
190	140
223	168
230	196

Assume the original image has the following frequencies for the above pixel intensities –

Pixel Intensity i	Frequency
178	238
190	272
223	328
230	427

Now if we map our new pixel intensities to these frequencies we will get –

Pixel Intensity i	Frequency
117	238
140	272
168	328
196	427

Thus you can see how high pixel intensity values very reduced in number using the normalized CDF and the input image as a map.

Results -

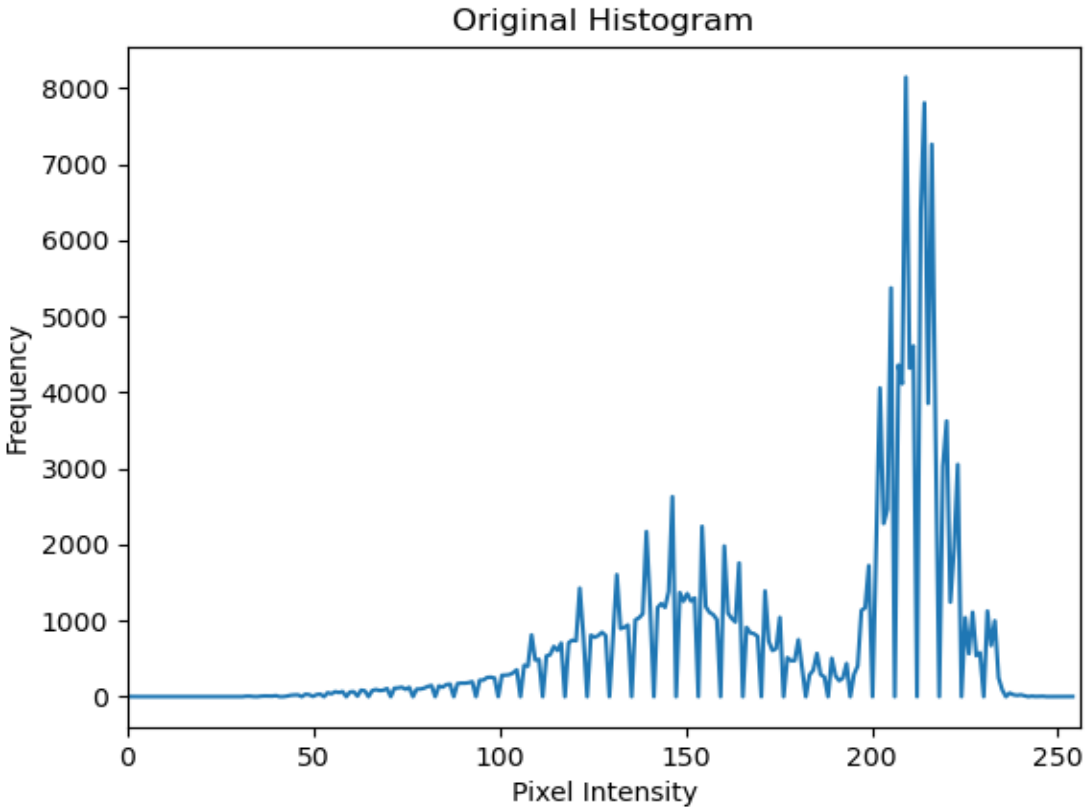
Image 1 -

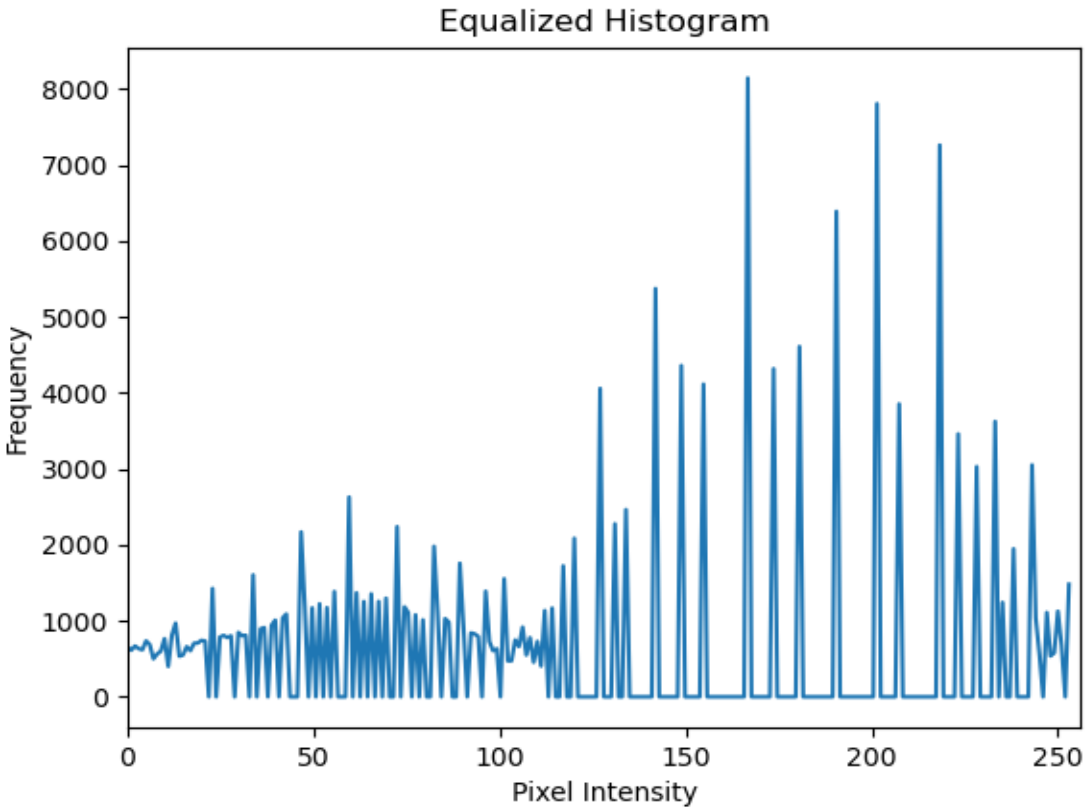
Figure 1: Original Image

Converting to uint8 form makes it image slightly darker as all the floating-point values are floored to integers.

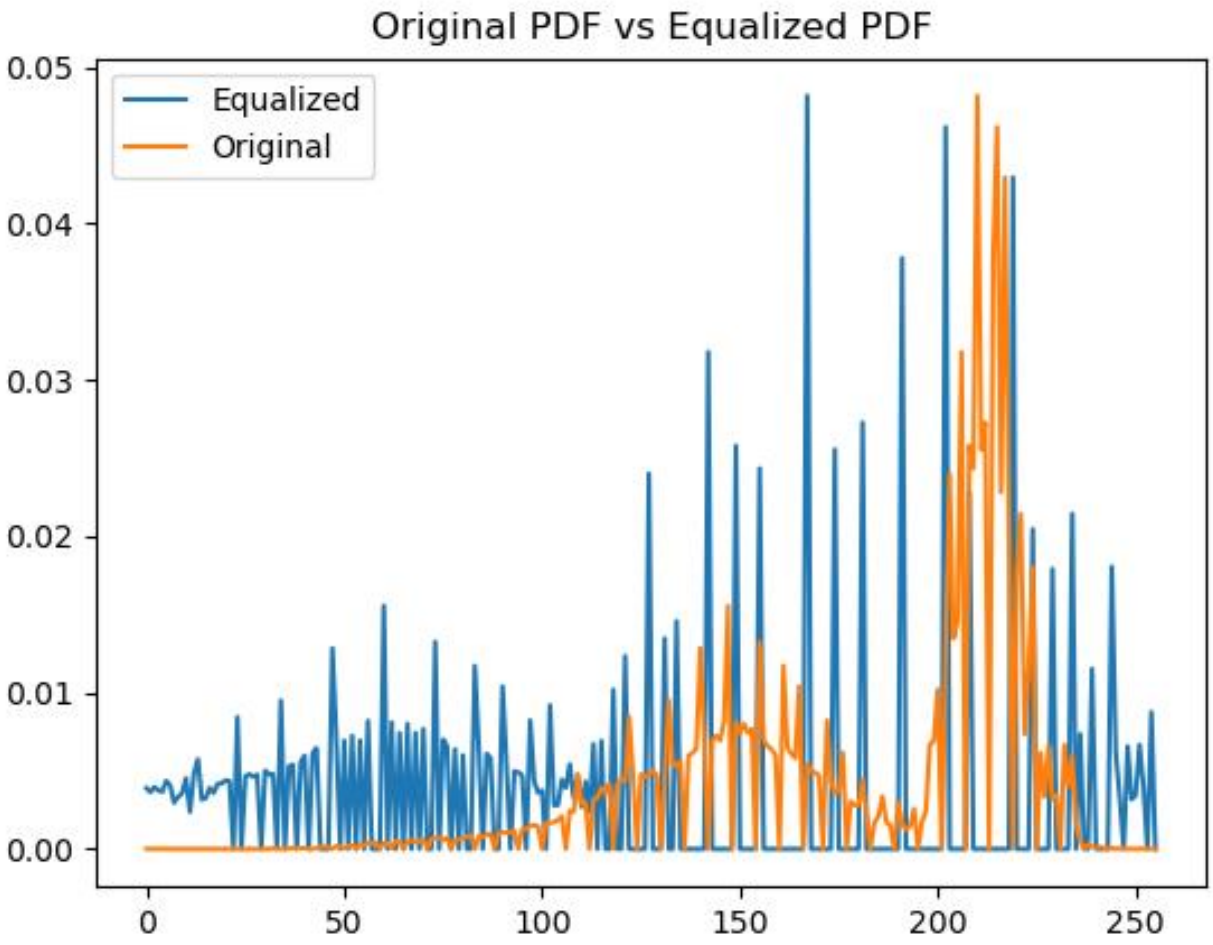
Figure 3: Original Image Converted to uint8

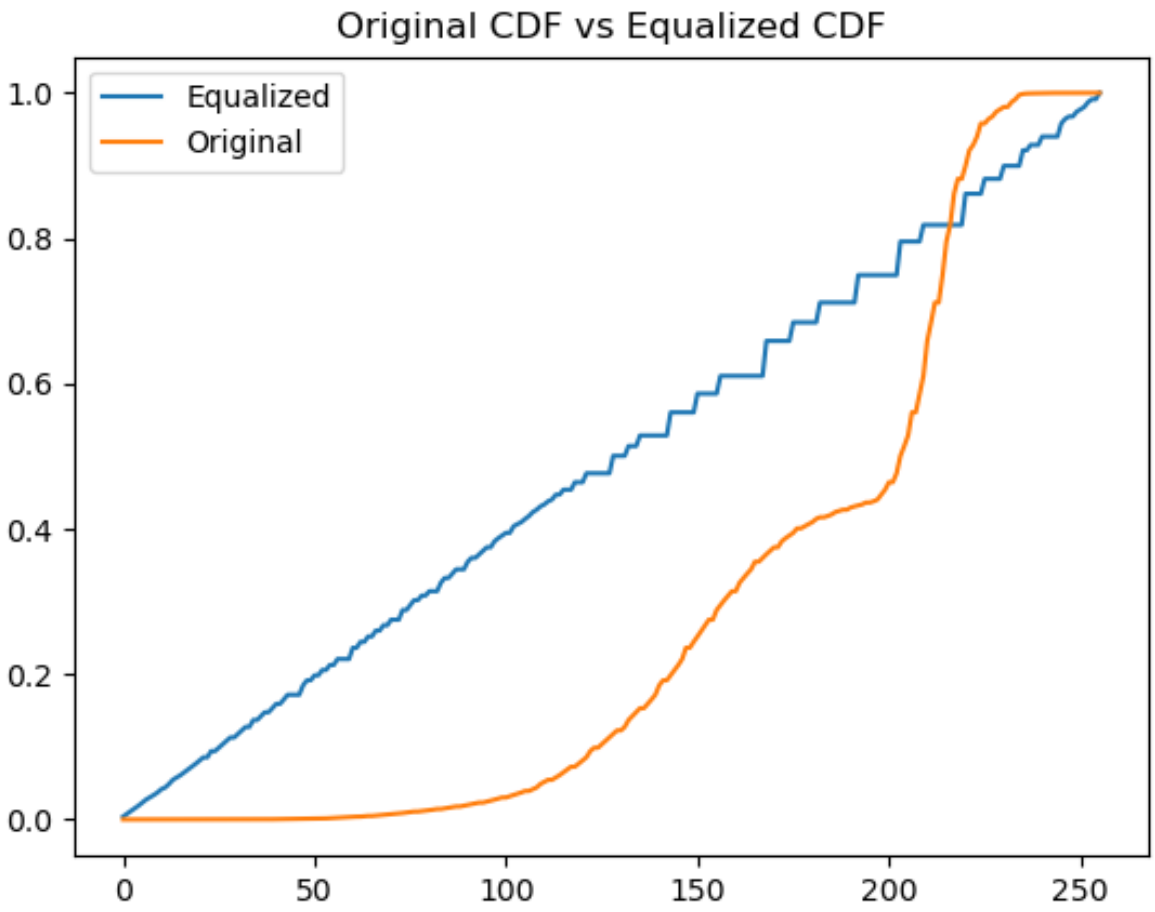
Figure 2: Equalized Image





Equalized Histogram has more pixels in the low intensity range (0-150) leading to a darker image.


Original PDF vs Equalized PDF
Equalized
Original
0.05
0.04
0.03
0.02
0.01
0.00
0
50
100
150
200
250



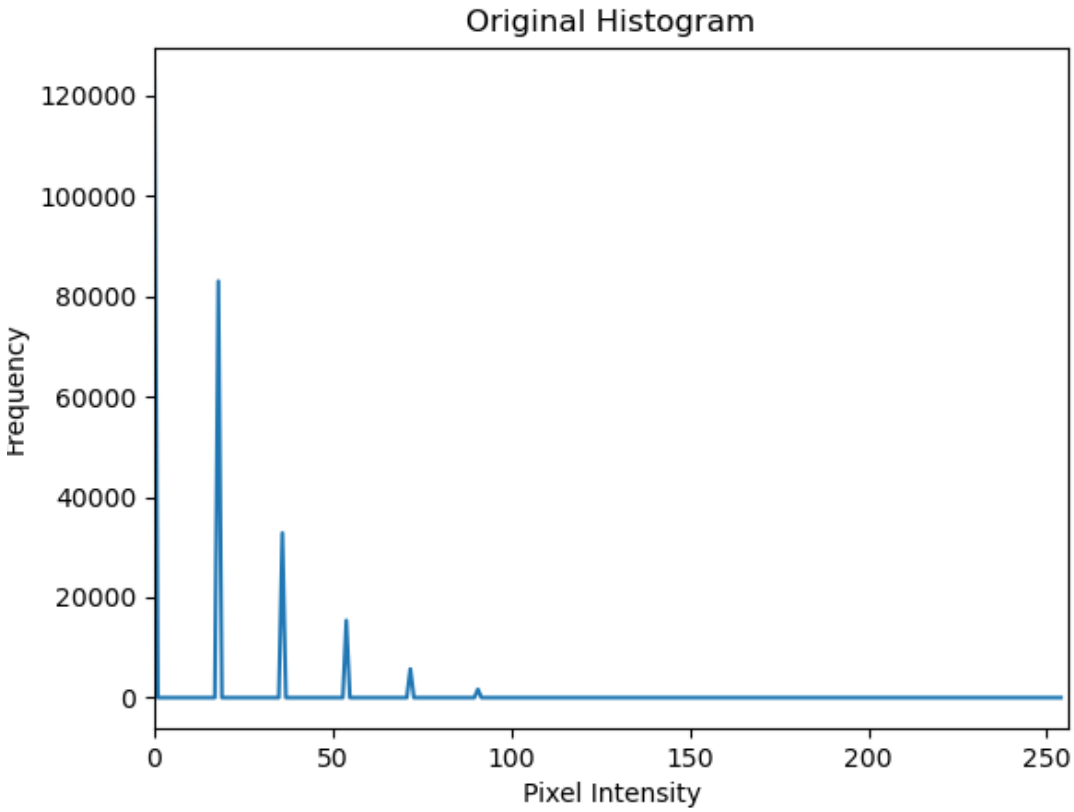
Blue line indicates the CDF of the Histogram equalized image. The orange line shows the CDF of the original image. The linear nature of the equalized histogram tells us that the probability of each histogram value increases gradually and has sort of linear nature.

This means no pixel intensity dominates the other (like in our original image). This is evident by the equalized histogram.

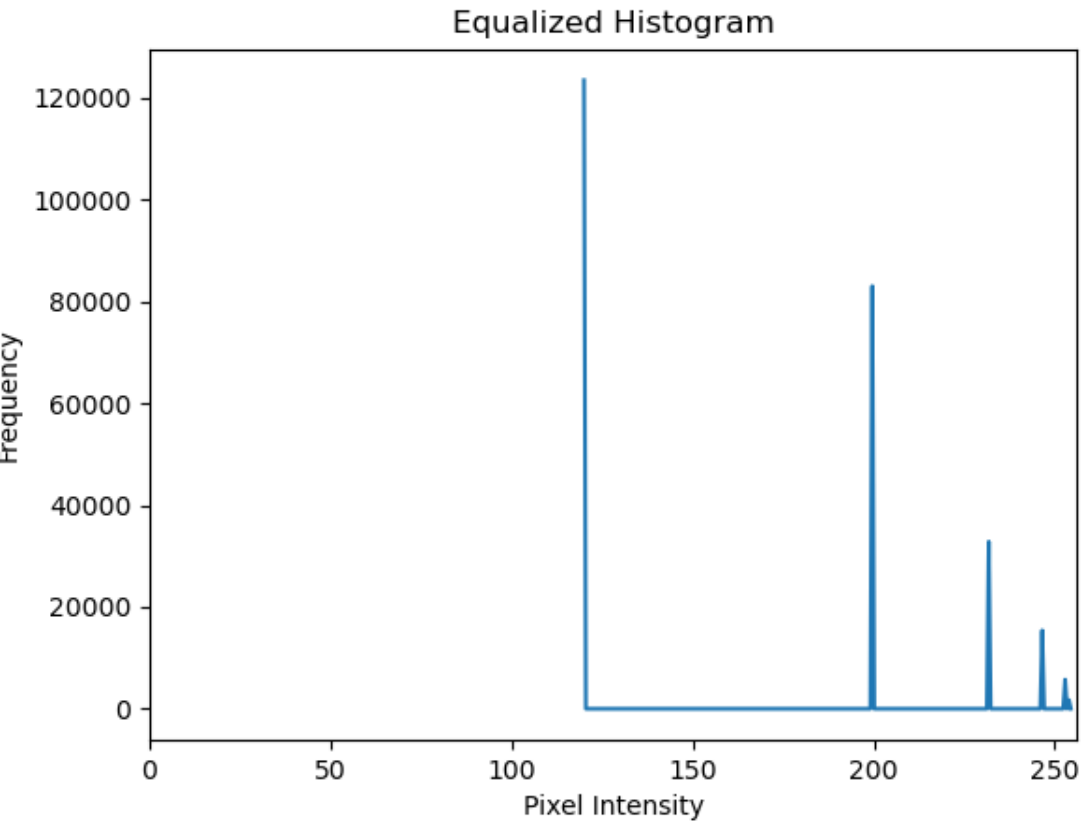
Image a1_b :-

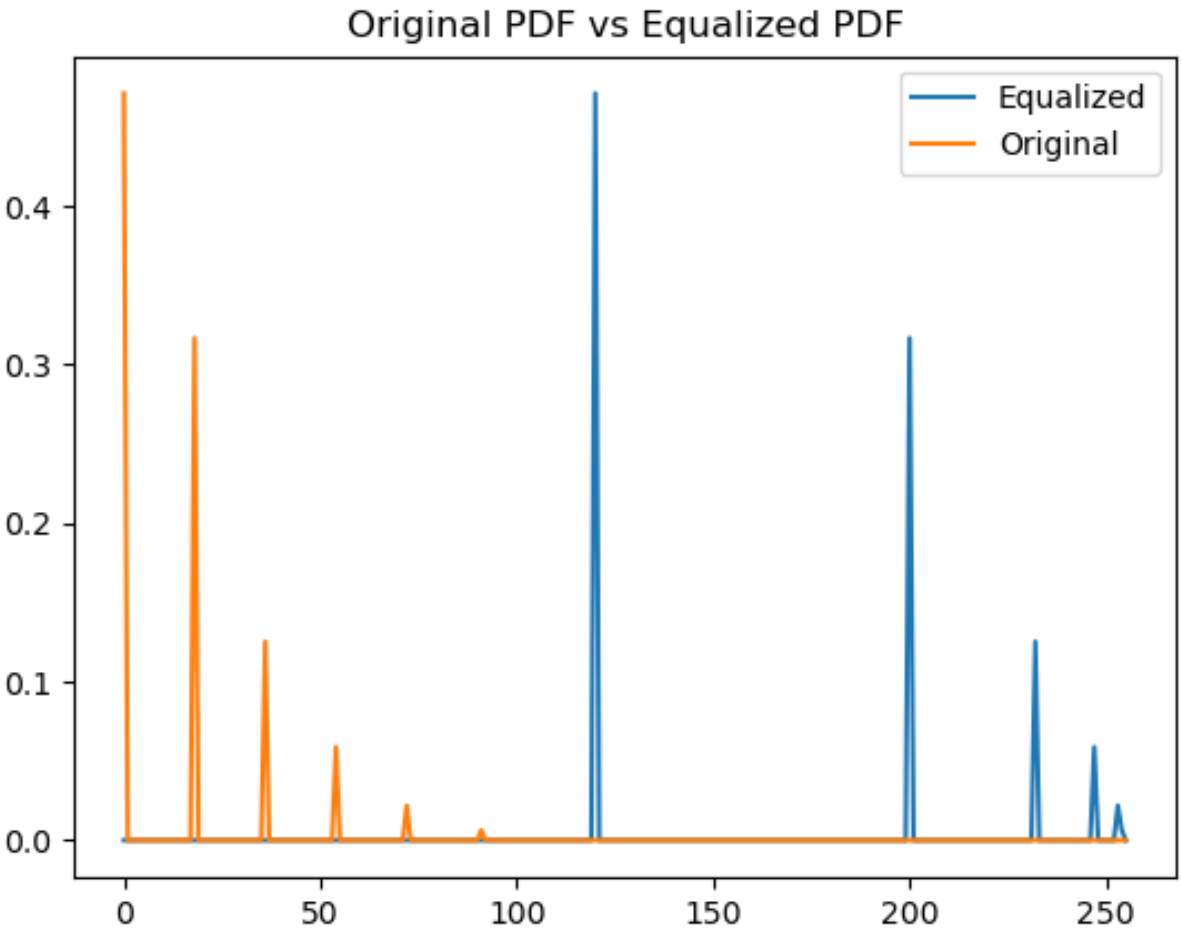
Figure 5: Original Image converted to uint8

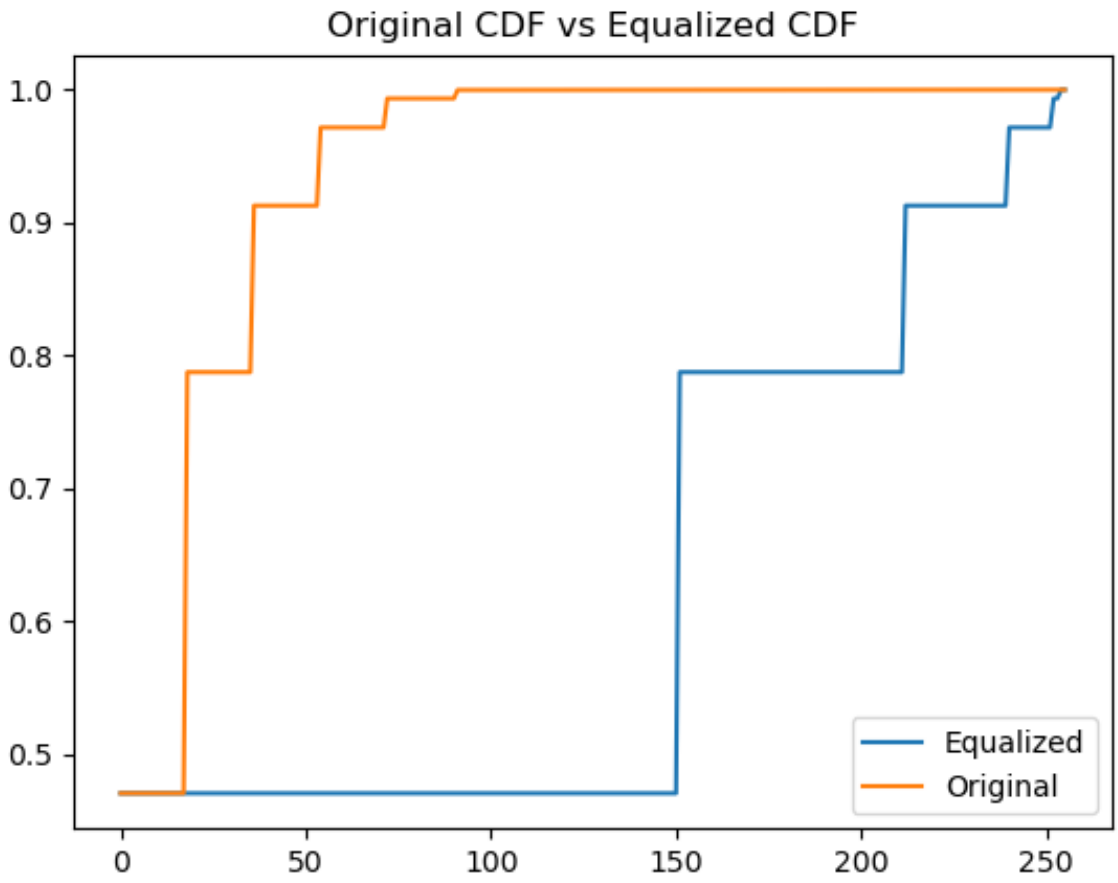
Figure 4: Equalized Image



Equalized Histogram has all the pixels in the range 130-255 leading to an image with high noise and losing all the blacks




Original PDF vs Equalized PDF
Equalized
Original
0.4
0.3
0.2
0.1
0.0
0
50
100
150
200
250



The CDF for original image is increases rapidly for low intensities and then reaches max value. This means the there are large number of pixels in low having low intensities. This is true as the original image (Figure 4) is so dark that barely any information can be infered from it.

On contrast the graph of equalized image CDF doesn't increase till pixel intensity 150 and then increases rapidly. The equalized image has lost nearly all its black pixels leading to noisy image.

Although the equalized image gave us plenty of information it also gave us a lot of noise and a grainy image. There was also loss information as the color of the guys shirt in Figure 5 lost its original color.

Performing histogram equalization again -

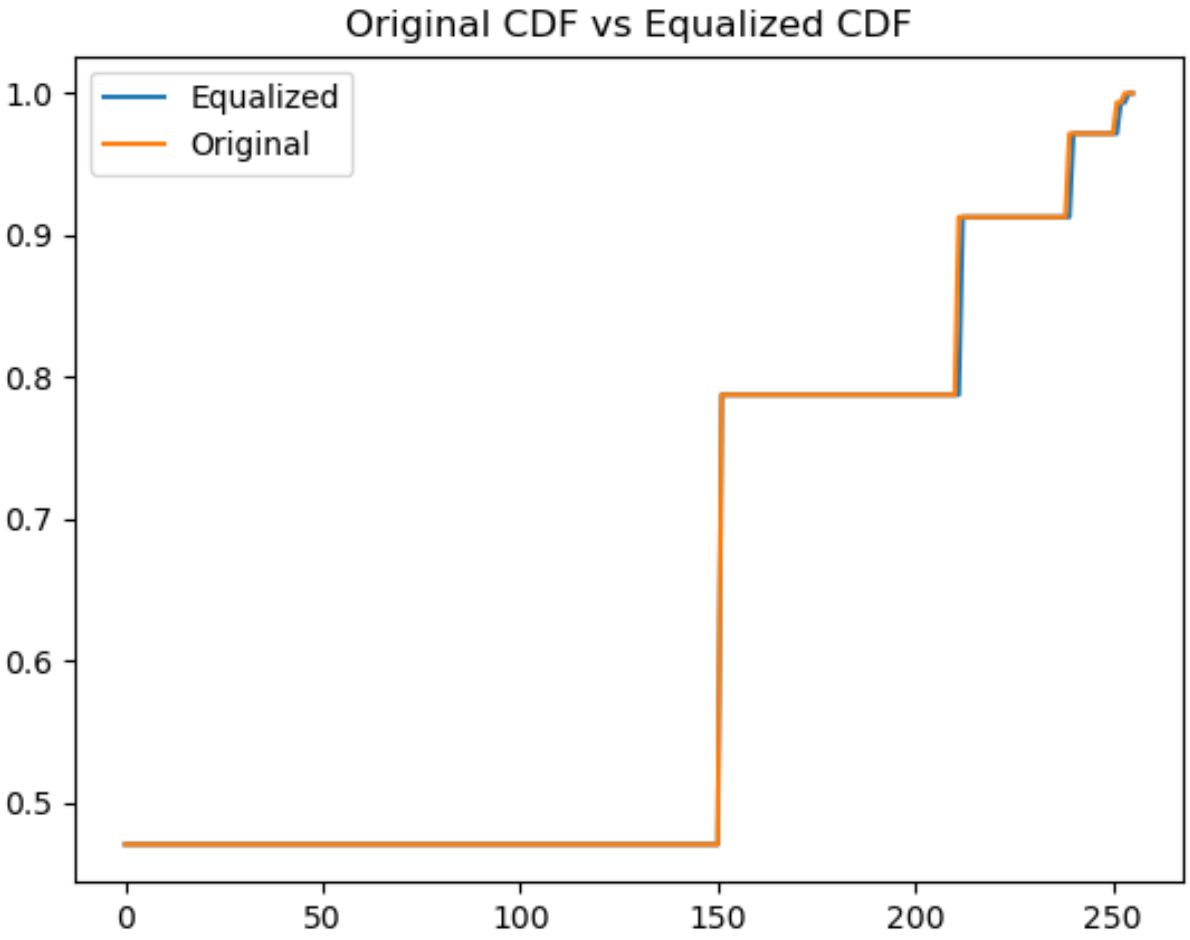
We will use *Figure 4* for this purpose. Since the image was already of the uint8 format we do not convert it again.

Figure 6: Input Image

Figure 7: 2nd time Equalized image

After applying histogram equalization, we get the same image again. This is evident as the CDF of the input image is already normalized and applying the same operations won't change the output.

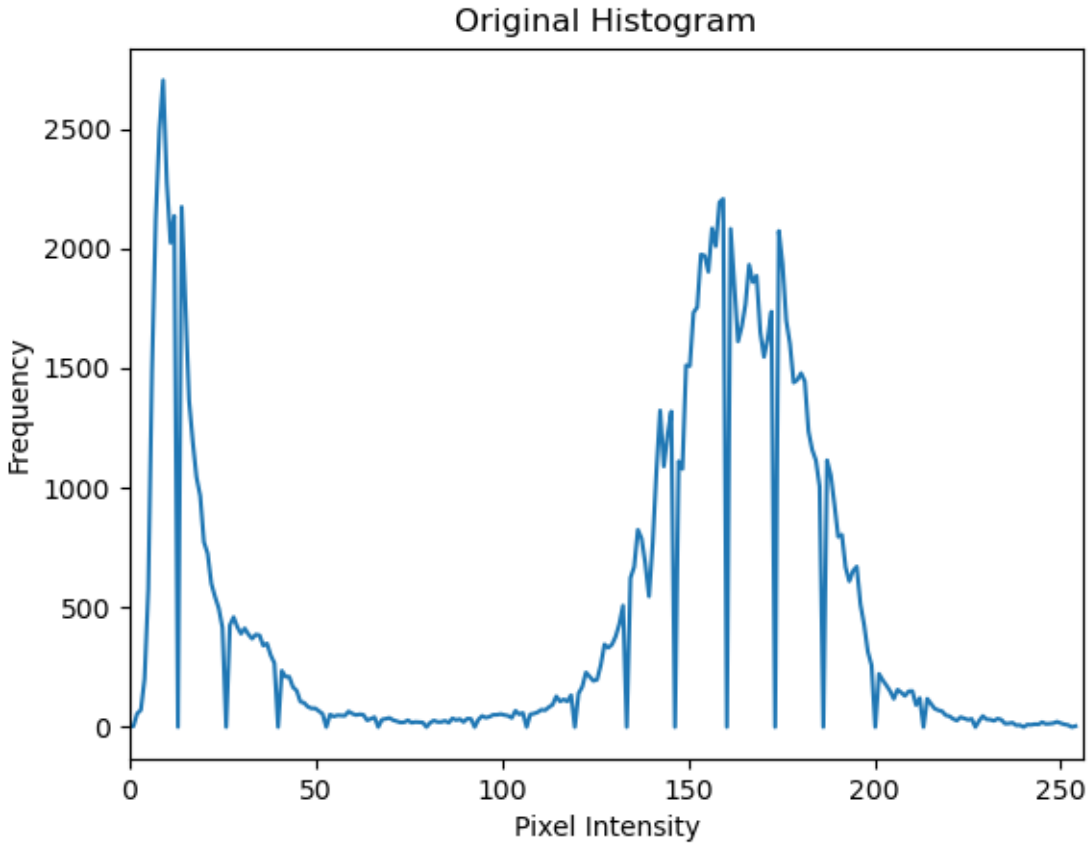
We can see that the CDF of both the images Figure 8 and Figure 9 are overlapping. This is because the initial CDF was normalized and applying normalization again won’t change the CDF. Also, we use the histogram equalized image for mapping so final image will be the same.

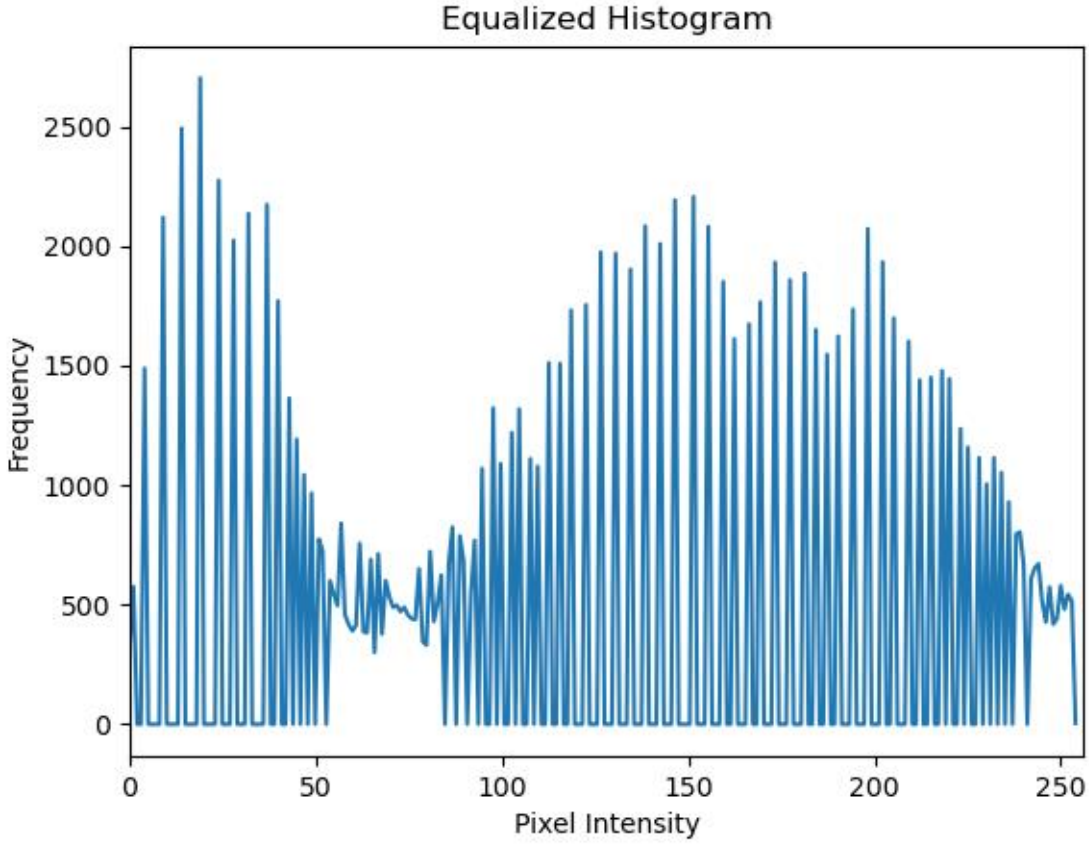


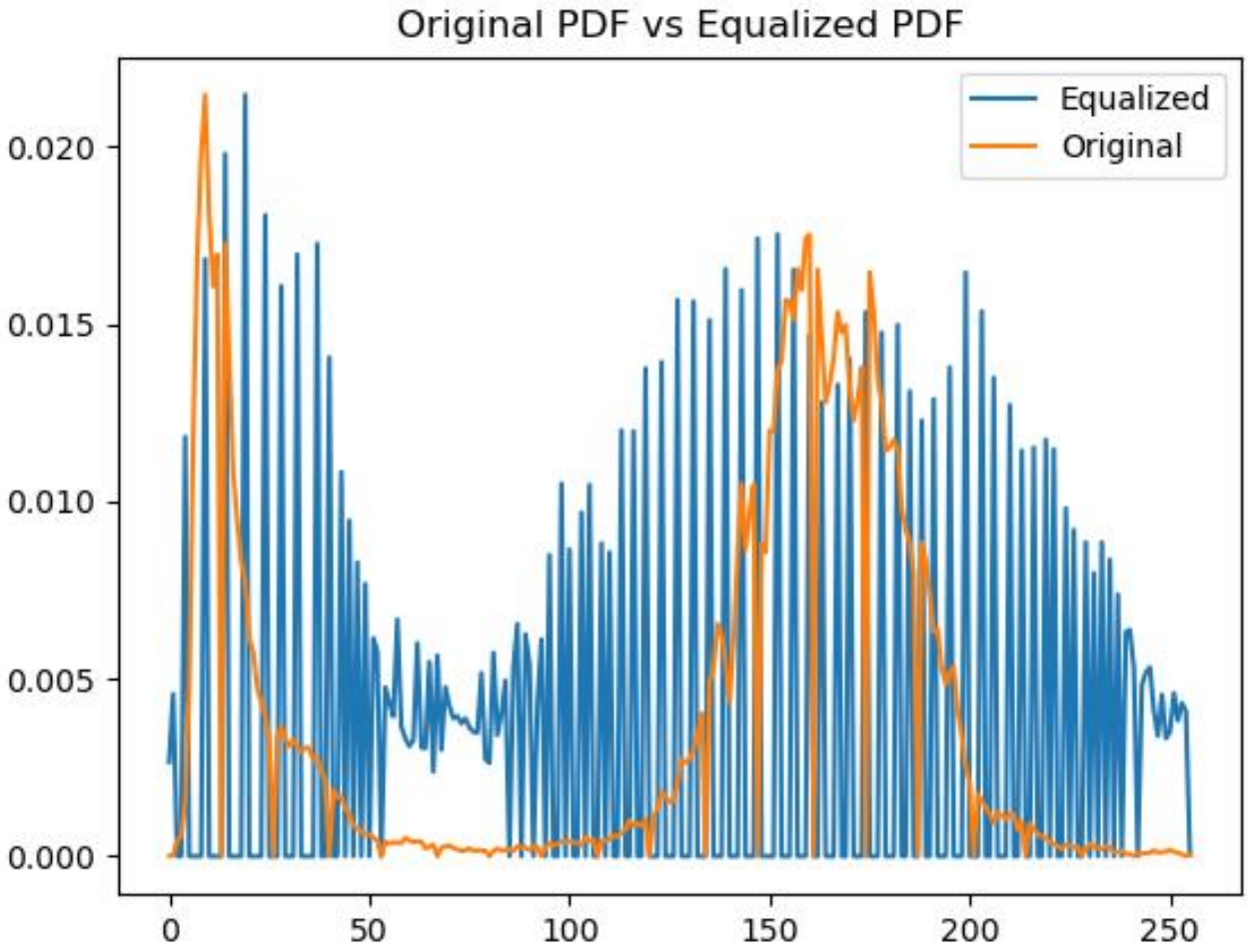
New Image –

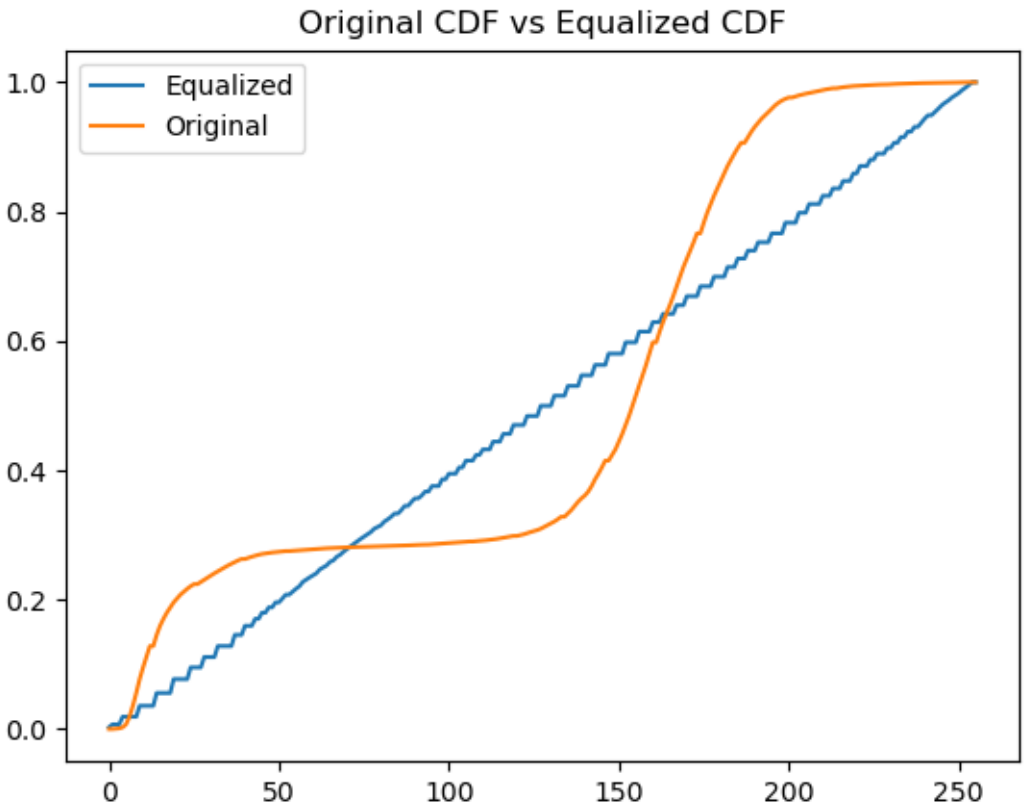
Figure 8: Input Image

Figure 9: Histogram Equalized


Original Histogram
Frequency
Pixel Intensity
0
500
1000
1500
2000
2500
0
50
100
150
200
250


Equalized Histogram
Frequency
Pixel Intensity
0
500
1000
1500
2000
2500
0
50
100
150
200
250


Original PDF vs Equalized PDF
Equalized
Original
0.020
0.015
0.010
0.005
0.000
0
50
100
150
200
250



The Original histogram shows there are a greater number of pixels in the low intensity levels and in high intensity levels. It has low number of pixels in the mid intensity levels. It is evident as the image has dark building like structure and clouds.

After Equalization the frequency of the mid intensity level pixels rise giving us a more detailed image.

The CDF of the original image rises abruptly then slows down and again rises abruptly till it reaches its max value. This is because of the high number of low and high intensity pixels and very less mid intensity pixels (range 50 – 130). The straight-line nature of the equalized CDF shows the intensity levels are spread out evident by the output image.

Conclusion

Histogram equalization is a global solution, modifying contrast with respect to the CDF of the entire image. Now consider a case where we split the input image into smaller regions and apply histogram equalization to each region.

Let the image be of size 500x500. Assume we split the image into 50x50 pixel squares. Thus we have a total of 100 square regions numbered as (i,j) i<=10, j<=10

Let box (1,1) have intensity values less than 5 (totally black for naked eye) so histogram equalization will try to spread the intensity range and in turn make the image too bright adding noise in the process.

Now applying histogram equalization on square (1,2), this square may or may not be similar to (1,1). If it has high number of low and high intensity pixels the equalization will give mid range values.

Doing this for all squares will result in all of them looking different. Giving a checkered effect in the final image.

As histogram equalization takes into count of the all the pixels during its calculations, constricting the image and performing localized calculations will lead to localized equalization without considering the other regions.

This localized approach may be useful if the image has repeating patterns maybe like the scales of a fish or a chess board.

Image Thresholding

Introduction -

Image thresholding is a technique of image segmentation, segmenting the foreground and the background. Thresholding can be used to create binary images given a greyscale image.

In simple terms thresholding is done if a pixel having intensity, I is greater than a specified threshold value T then that pixel would be white or for that pixel I=255. Else the value I = 0

Otsu's method, used to perform automatic image thresholding. In the simplest form, the algorithm returns a single intensity threshold that separate pixels into two classes, foreground, and background. The threshold is determined by minimizing the value of interclass variance of pixel intensities.

Results -

Manual Image Thresholding –

Setting the threshold value to t = 169

Figure 10: Input Image a2_a

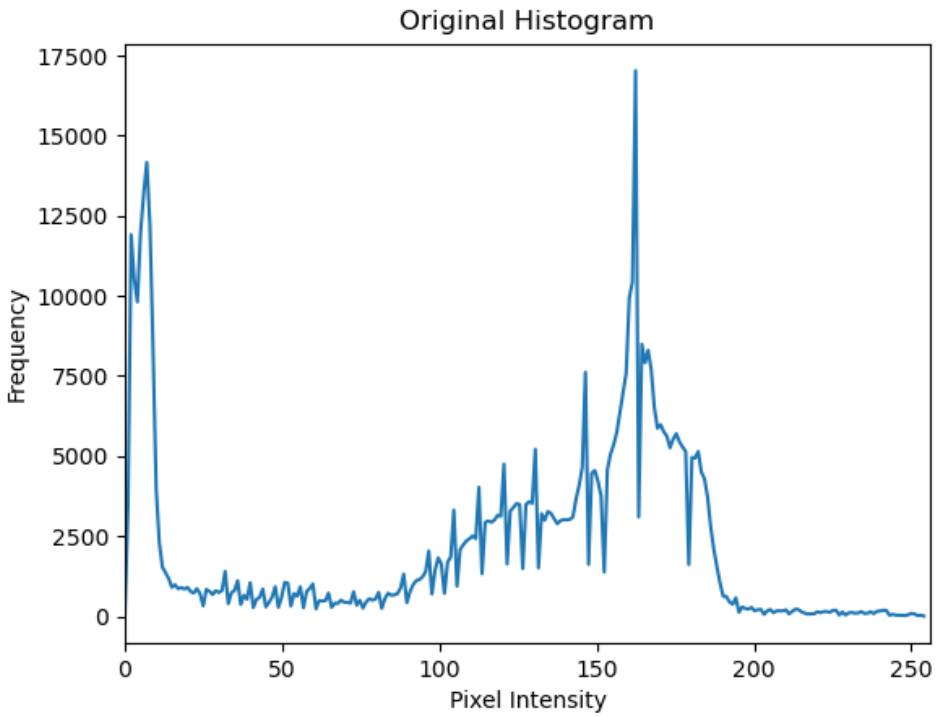
Figure 11: Manual Thresholding Output

The output of manual thresholding is heavily distorted, and we cannot discern between the foreground and the background.

Using Otsu’s Method for Thresholding –

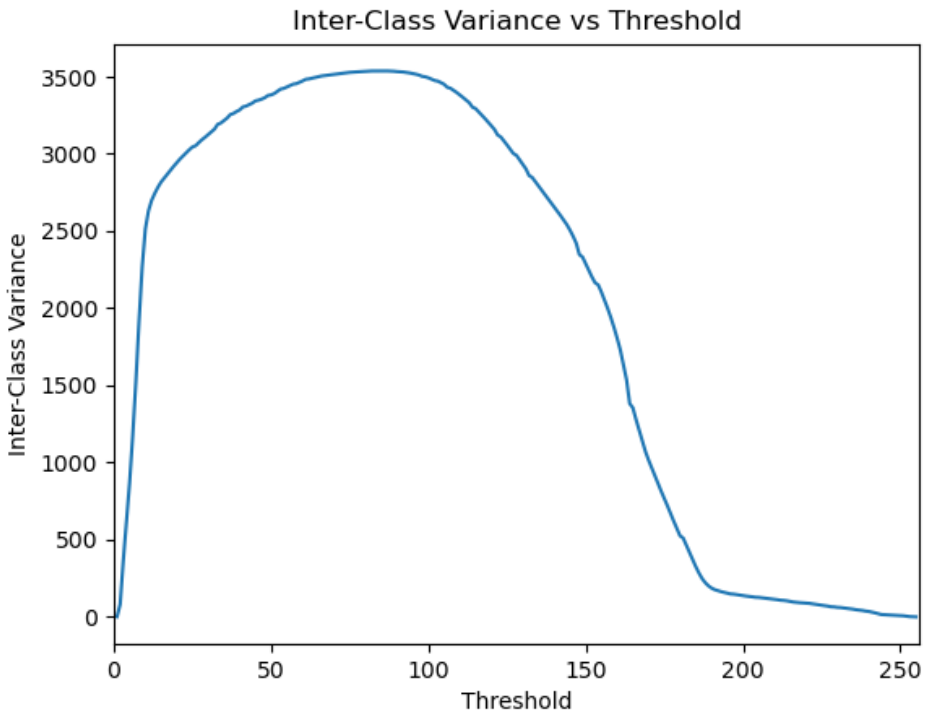
Figure 12: Thresholding using Otsu's Method

Corresponding Histogram -



It can be observed that the foreground lie in range 0 – 50 as the person is wearing black color clothes. The background is light so there are large number of pixels in the high intensity range.

Maximum Variance would be somewhere between 50 – 100.



The maximum value of interclass variance was found at threshold value 85.

Max value of Inter-class variance = 3537.97

Image a2_b –

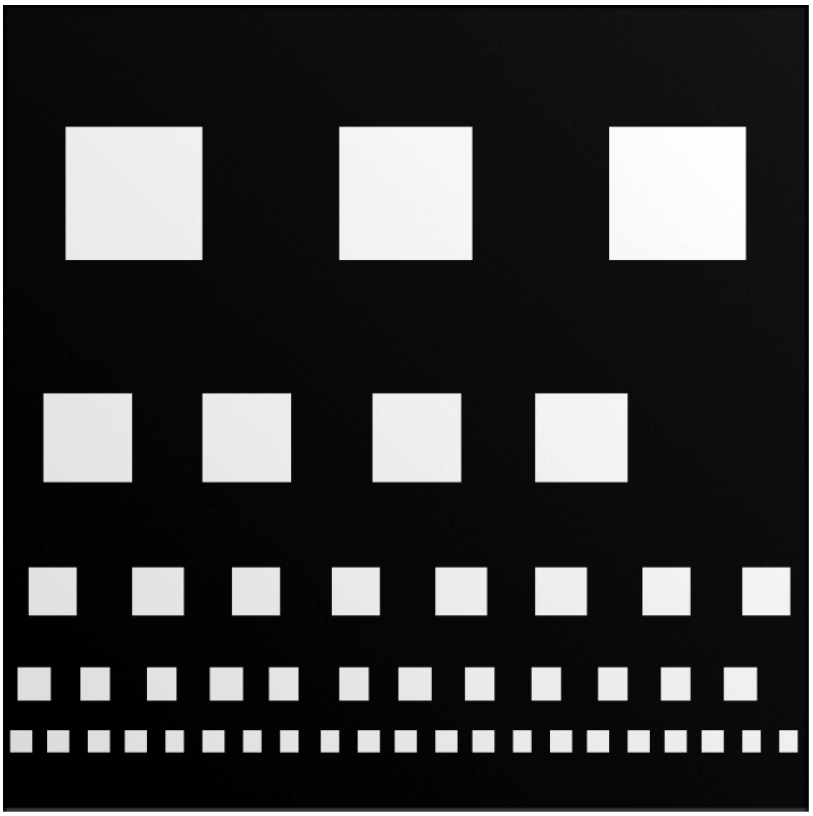


Figure 13: Input Image a2_b

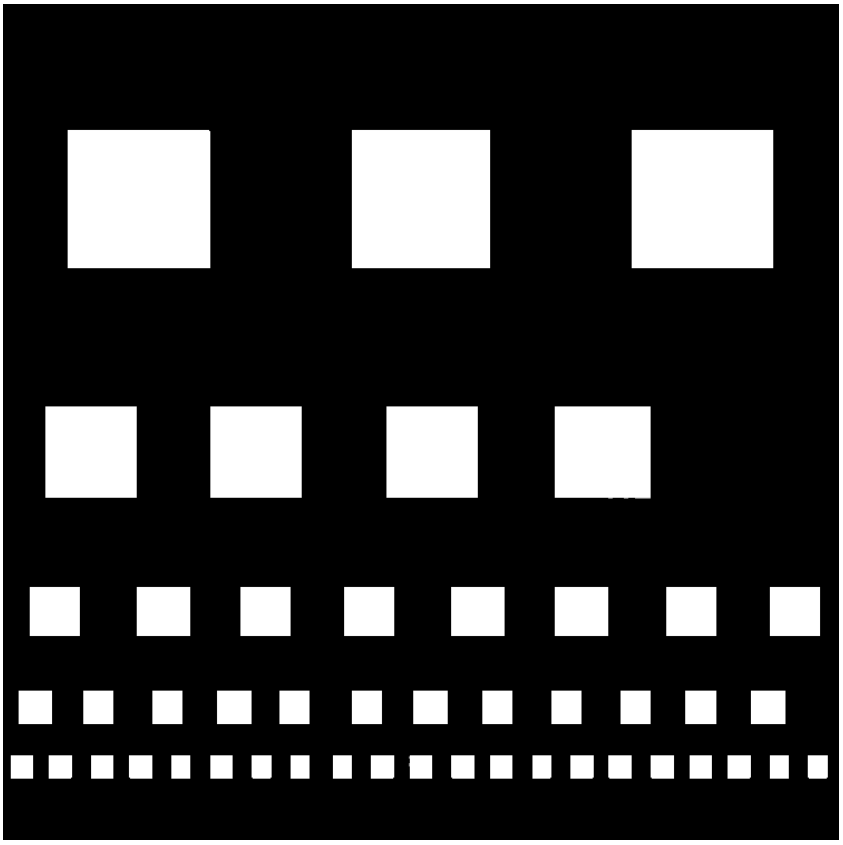
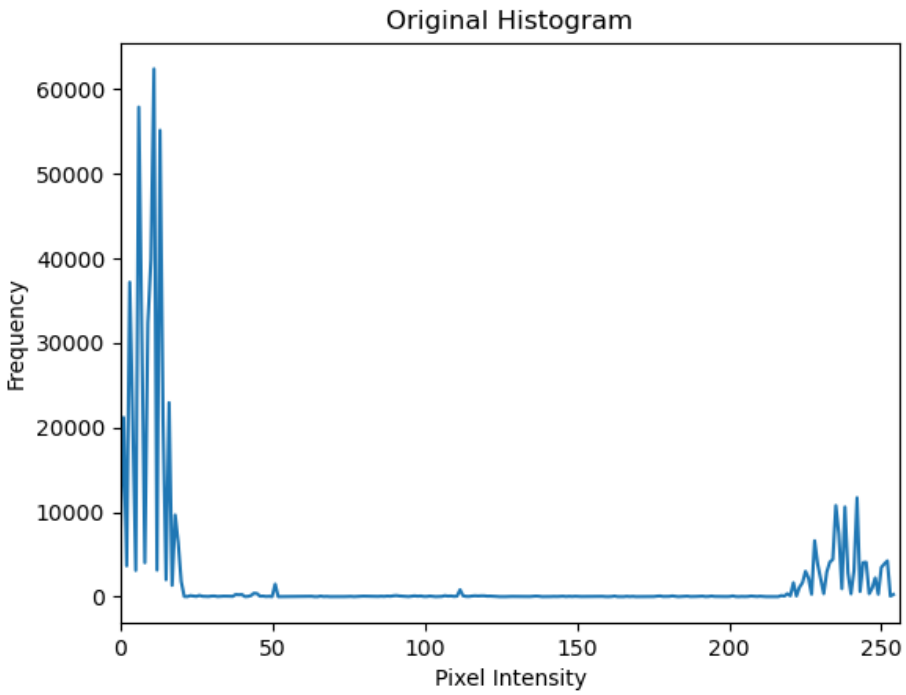


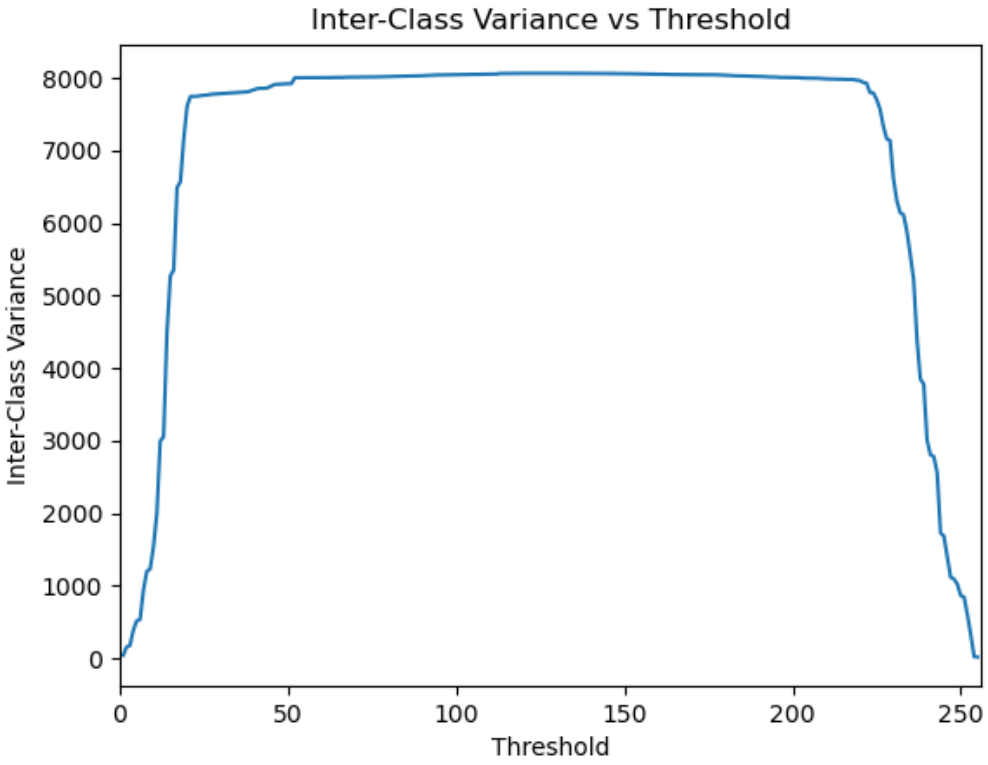
Figure 14: Otsu Output

Here the output does not change for both manual thresholding at t=169 and Otsu thresholding as the input image has a clear distinction between the foreground and background. Plotting the histogram will tell us why

Corresponding Histogram -



The threshold value can be anything between 50 – 200 as there are barely any pixels in that intensity range.



The maximum value of interclass variance was found at threshold value 124.

Max value of Inter-class variance = 8064.28

Image a2_c –

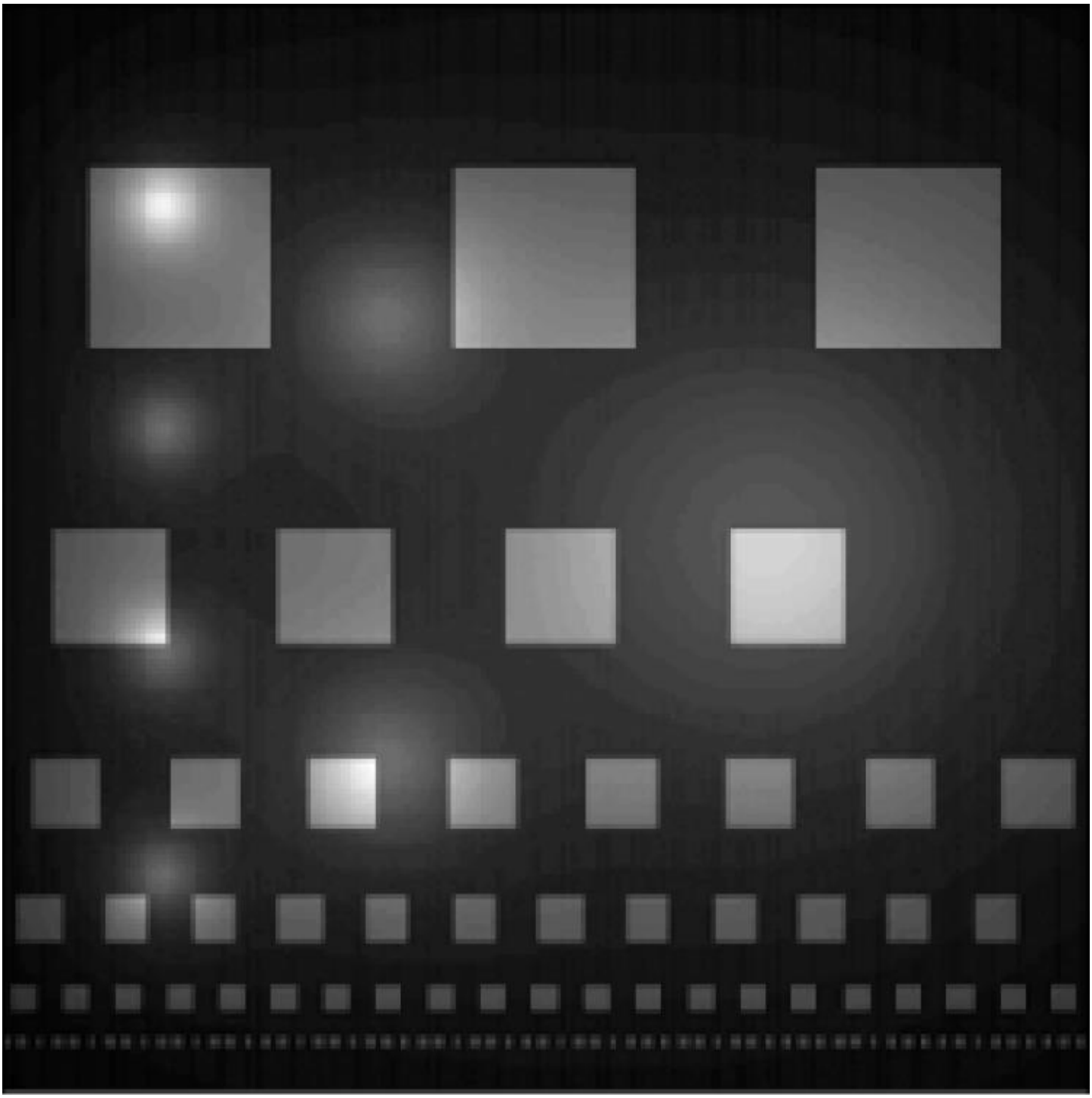


Figure 15: Input Image a2_c

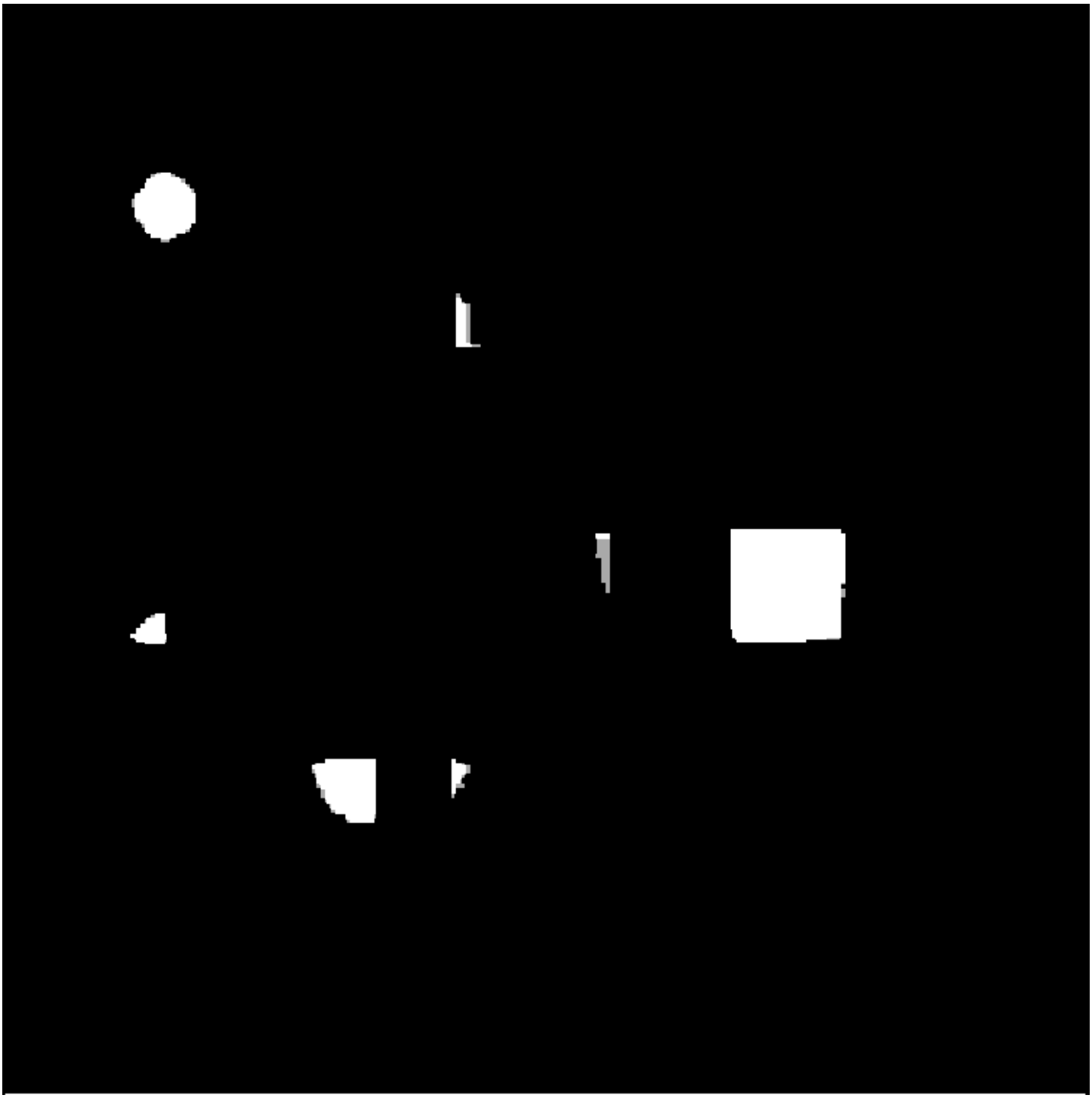


Figure 16: Manual thresholding

Manual Thresholding was performed at t=169

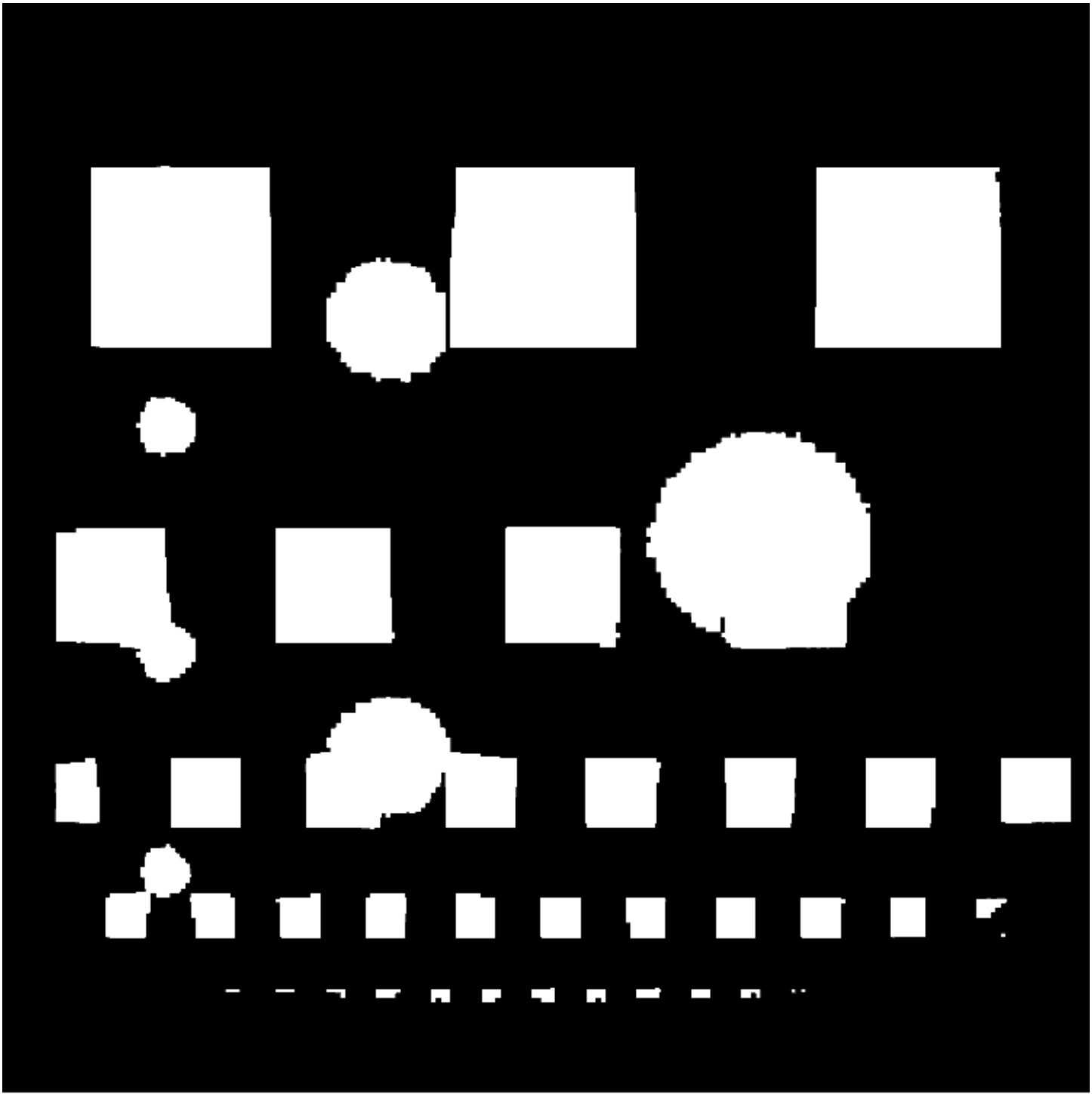
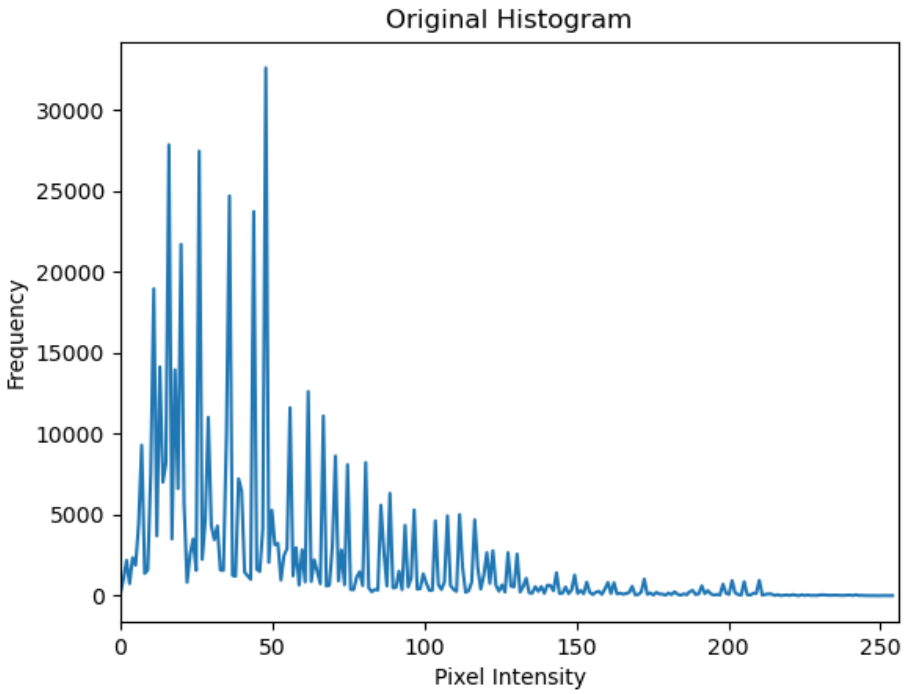
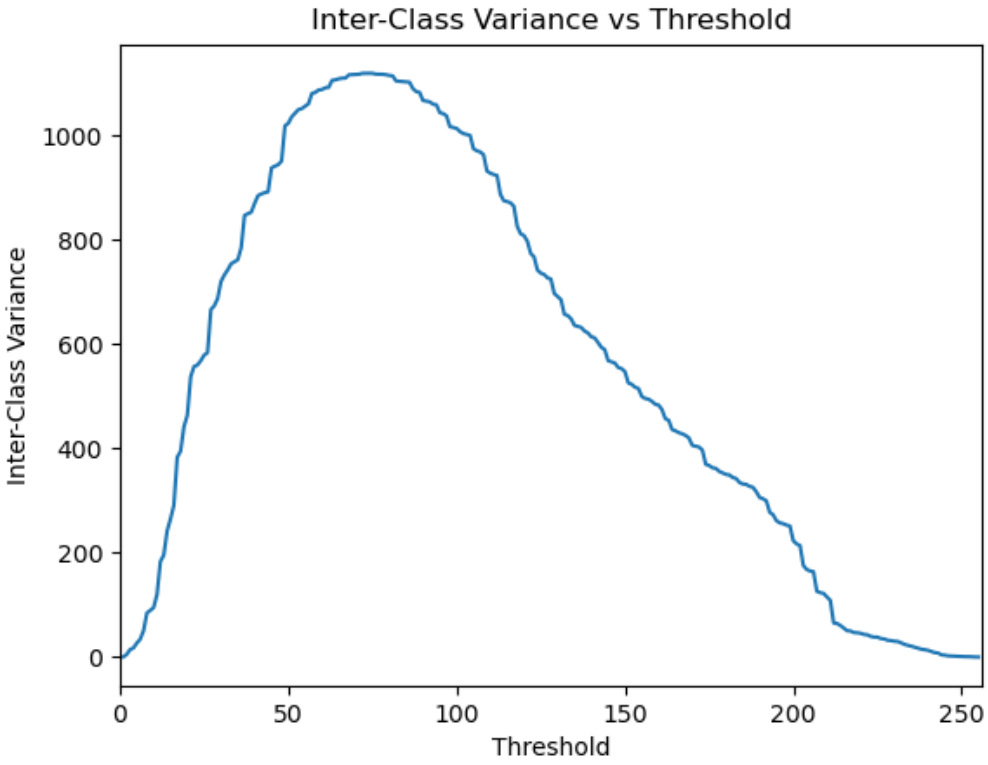


Figure 17: Otsu Output

Although this image is similar to Figure 6 the otsu output is different because of the presence of those circular regions of increasing intensity towards their center.

Corresponding Histogram -





The maximum value of interclass variance was found at threshold value 74. Max value of Inter-class variance = 1119.74

Conclusion –

You can see from the results, Otsu's methods does a decent job in separating the foreground and the background.

It performs well when the histogram has a bimodal distribution (histogram of Figure 4) with a deep and wide gap between the two peaks.

The same does not hold true for other cases where the histogram does not show bimodal distribution. (Image a2_1

Appendix-

Histogram Equalization

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


def generate_hist(img, bins):  # to make a histogram (count distribution
frequency)
    h,w = img.shape[:2]
    val = [0]*bins
    for i in range(h):
        for j in range(w):
            val[img[i,j]]+=1
    return np.asarray(val) , np.arange(bins)


def create_pdf(hist, N):
    #divide histogram by number of pixels
    pdf = hist / N
    return pdf


def create_cdf(hist ,N):
    pdf = create_pdf(hist ,N)
    cdf = np.zeros((pdf.shape))
    cdf[0] = pdf[0]

    #get cummulative sum
    for i in range(1,len(pdf)):
        cdf[i] = cdf[i-1] + pdf[i]

    return cdf


def histogram_equalization(img, hist,N):
    cdf = create_cdf(hist ,N)
```

```
    #pixel mapping
    normalized_cdf =  np.floor(255 * cdf).astype("uint8")

    img_to_list = list(img.flatten())
    equalized_img = [ normalized_cdf[i] for i in img_to_list]

    #need to reshape since we used flatted image during mapping
    equalized_img = np.reshape(np.asarray(equalized_img) , img.shape)

    return equalized_img



if __name__ == '__main__':
    img = cv2.imread('a1_c.png',0)
    h,w = img.shape[:2]


    #convert to uint8
    img_uint8 = ( (img - np.min(img)) * 1/(np.max(img) - np.min(img))  *
255  ).astype('uint8')


    #genetate histogram
    hist_og , bins = generate_hist(img_uint8, bins = 256 )

    #equalized Histogram
    img_equalized = histogram_equalization(img_uint8, hist_og , h*w)
    H,W= img_equalized.shape[:2]



    #show output
    cv2.imshow("Original Image", img_uint8)
    cv2.imshow("Eqaulized Image", img_equalized)
    cv2.imwrite("a1_uint8.png", img_uint8)
    cv2.imwrite("a1_equalized.png", img_equalized)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
    #plot original histogram and Equalized
    plt.figure(1)
    plt.title("Original Histogram")
    plt.xlabel("Pixel Intensity")
    plt.xlim([0,256])
    plt.ylabel("Frequency")
    #bins is an array of size 257 so we ignore the last bin
    plt.plot(bins, hist_og )

    hist_equalized , bins = generate_hist(img_equalized, bins = 256 )
    plt.figure(2)
    plt.title("Equalized Histogram")
    plt.xlabel("Pixel Intensity")
    plt.xlim([0,256])
    plt.ylabel("Frequency")
    plt.plot(bins, hist_equalized)


    #plot original vs equalized pdf
    pdf = create_pdf(hist_og , h*w)
    pdf_e = create_pdf(hist_equalized, H*W)
    plt.figure(3)
    plt.title("Original PDF vs Equalized PDF")
    plt.plot(pdf_e, label = "Equalized")
    plt.plot(pdf , label = "Original")
    plt.legend()

    #plot original vs equalized cdf
    cdf = create_cdf(hist_og, h*w)
    cdf_e = create_cdf(hist_equalized, img_equalized.shape[-1] *
img_equalized.shape[0])
    plt.figure(4)
    plt.title("Original CDF vs Equalized CDF")
    plt.plot(cdf_e, label = "Equalized")
    plt.plot(cdf , label = "Original")
    plt.legend()
    plt.show()
```

Otsu's Thresholding

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


def generate_hist(img, bins):  # to make a histogram (count distribution
frequency)
    h,w = img.shape[:2]
    val = [0]*bins
    for i in range(h):
        for j in range(w):
            val[img[i,j]]+=1
    return np.asarray(val) , np.arange(bins)



def manual_threshold(im_in, threshold, OTSU):
# Threshold image with the threshold of your choice
    final_image = im_in.copy()
    final_image[ im_in > threshold] = 255
    final_image[im_in<threshold] = 0
    if OTSU:
        cv2.imshow("Output from Otsu Thresholding", final_image)
        cv2.imwrite("Otsu_out.png", final_image)
        cv2.waitKey(0)
        return

    cv2.imshow("Output from Manual Thresholding", final_image)
    cv2.waitKey(0)
    cv2.imwrite("Manual_out.png", final_image)


    return None


def otsu_threshold(im_in):
```

```
    # Create Otsu thresholded image
    h,w = img.shape[:2]
    mean_weight = 1/ (h*w)
    hist , bins = generate_hist(im_in, bins = 256)

    print(type(hist))
    print(type(bins))
    print(bins.shape)

    optimal_threshold = -1000
    inter_class_variance = -1000

    #to plot the variance
    plot_variance = []

    pixel_intensity = np.asarray(range(0,256))
    for i in range(len(bins[0:-1])):
        probability_0 = np.sum(hist[:i])
        probability_1 = np.sum(hist[i:])
        w0 = probability_0 * mean_weight
        w1 = probability_1 * mean_weight

        mean_u0 = np.sum( pixel_intensity[:i] * hist[:i] ) /
probability_0
        mean_u1 = np.sum( pixel_intensity[i:] * hist[i:] ) /
float(probability_1)

        temp = w0 * w1 * ((mean_u0 - mean_u1) **2)
        plot_variance.append(temp)

        if temp>inter_class_variance:
            optimal_threshold = i
            inter_class_variance = temp


    print("\nOptimal value of Thresholding is ",optimal_threshold)
    print("\nMaximum Inter-class Varinace is ",
"{:.2f}".format(inter_class_variance))
```

```
    manual_threshold(im_in, optimal_threshold, True)

    #ploting histograms
    plt.figure(1)
    plt.title("Original Histogram")
    plt.xlabel("Pixel Intensity")
    plt.xlim([0,256])
    plt.ylabel("Frequency")
    #bins is an array of size 257 so we ignore the last bin
    plt.plot(bins, hist )

    #plotting inter-class variance vs threshold
    plt.figure(2)
    plt.title("Inter-Class Variance vs Threshold")
    plt.xlabel("Threshold")
    plt.xlim([0,255])
    plt.ylabel("Inter-Class Variance")
    plt.plot(range(0,255), plot_variance)



    plt.show()


    return None



if __name__ == '__main__':
    img = cv2.imread('a2_c.png',0)

    cv2.imshow("Original Image" , img)


    #manual thresholding at value 169
    manual_threshold(img, 169, False)

    otsu_threshold(img)
```