PROJECT 1 – REPORT

A1: HISTOGRAM EQUALIZATION

Introduction and Description:

Wikipedia Definition: Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. The terms used in the definition can be explained as follows:

The frequency distribution of a collection of data reveals how frequently each unique value appears. The most typical graph for displaying frequency distributions is a histogram.

The range of brightness in an image, from lightest to darkest, is known as contrast. Very bright highlights and very dark shadows characterize high-contrast photographs. Images with low contrast feature a limited palette of colors.

Images having low contrast will have a narrow range of intensity values and that of high contrast will have a wide range of intensity values.

This technique typically improves many photos' overall contrast, especially when the image is represented by a constrained range of intensity values. By making this adjustment, the intensities on the histogram can be more uniformly dispersed, employing the entire range of intensities. This makes it possible for regions with less local contrast to acquire more contrast. This is achieved via histogram equalization's efficient spreading out of the densely populated intensity values that are employed to reduce visual contrast.

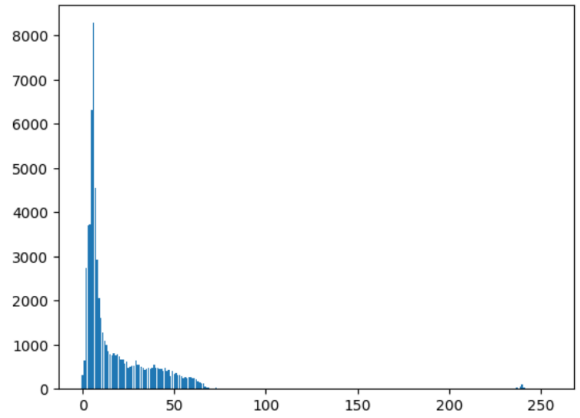
Following are the steps to perform Histogram Equalization:

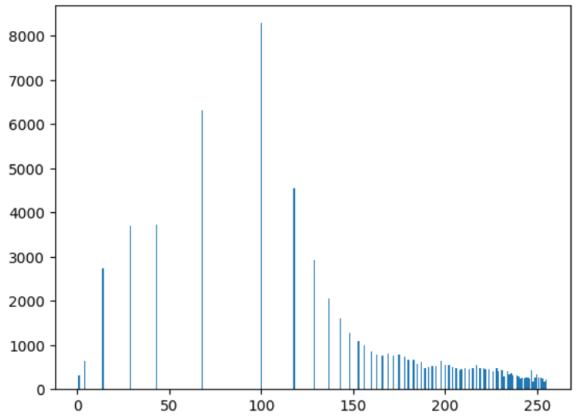
- Compute the Probability Distribution Function for the frequencies for given low contrast intensity values. This is done in order to Normalize the frequencies.
- Compute Cumulative Distribution Function for the respective probability distributions. These are useful in order to calculate the new intensity values for a given low contrast intensities.
- Use the Transfer Function to compute Equalized(Adjusted) intensity levels for the high-contrast image.

Methodology:

As part of A1, we were given the Input image file ‘indoors.png’. Shown below are the low-contrast input image(left) and respective high-contrast output image(right) derived using a Histogram Equalization code in Python as shown in the Appendix.

Looking at the above 2 images, the intensity range on the first image is much lower than that of the second. Moreover, the darker-intensity pixels have a higher frequency in the input image whereas the frequency of intensities seems well distributed in the output image. If this is not obvious looking at the images directly kindly refer to the below intensity-frequency plots for the above 2 images. In the below plots the x-axis refers to the intensity and the y-axis refers to the frequency that is the number of pixels of that given intensity in the entirety of the image.



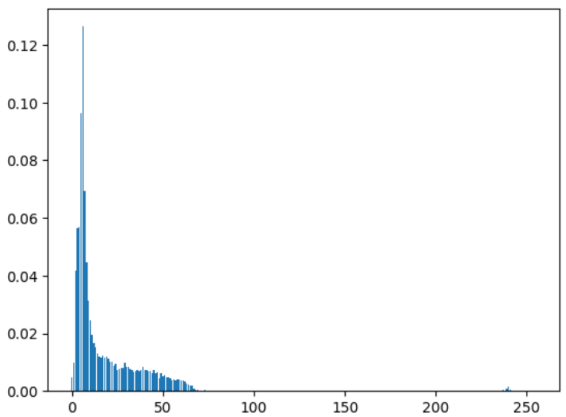


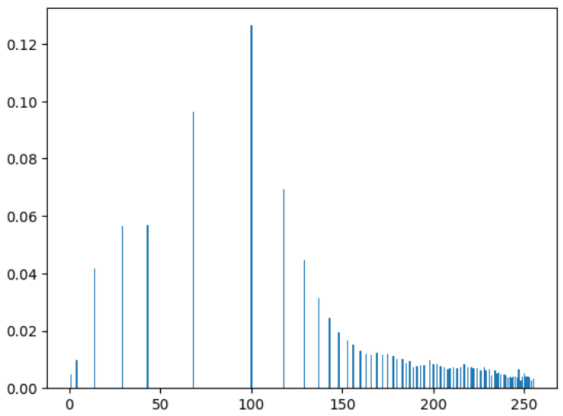
The input histogram has been stretched out uniformly for the entire range using Histogram Equalization. Intensity with higher frequencies have been stretched whereas those with lower frequencies have been compressed. Although this causes memory loss (loss of image features/details) which is not ideal, it still gives good contrast-adjusted results.

The frequencies for a low-contrast image need to be normalized using the probability distribution function as the first step of Histogram Equalization. The probability distribution function is the probability of a given occurrence of intensity. It can be computed as the ratio of the frequency of the given intensity level to the total number of pixels in the image as shown in the formula below:

$$\mathbf{P(r_i) = n_i / N}$$

Here P refers to the probability function, r_i refers to the i^{th} input intensity, n_i refers to the frequency of the i^{th} input intensity, and N is the total number of pixels in the input image. The plot of the normalized intensities is as follows, the x-axis represents intensity and the y-axis represents the respective probability of occurrence of intensity. The plot on the left represents the normalized histogram of the input image while that on the right represents the normalized histogram of the output image.



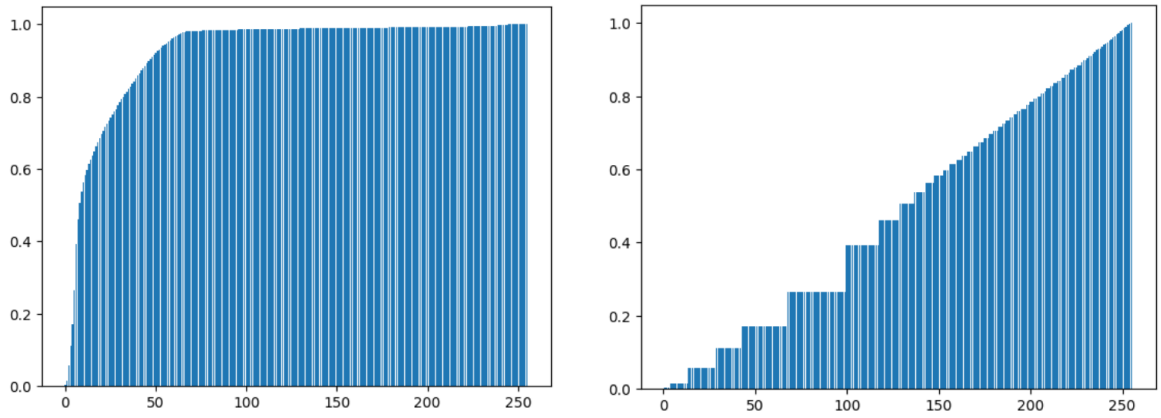


The normalized graphs look exactly like the unnormalized ones since we are only dividing the intensity values by a constant. These graphs represent the probability of a certain pixel existing in the image.

The next step of histogram equalization requires us to calculate and plot the cumulative distribution functions. The cumulative distribution function of a certain intensity represents the summation of probabilities of intensities up to the current intensity level as shown in the formula below:

$$\mathbf{C(r_i) = \sum_{0\ to\ i} (P(r_i))}$$

Here C refers to the cumulative probability, r_i refers to the i^{th} input intensity and P is the probability distribution value calculated in the previous step. The plot of the cumulative probability distribution for the intensity range is as follows, the x-axis represents intensity and the y-axis represents the respective cumulative probability of intensity. The plot on the left represents the CDF histogram of the input image while that on the right represents the CDF histogram of the output image.



The first observation from the above graphs is that the graphs are increasing in nature as probabilities are getting added with the rise in the intensity values. The second observation is that the CDF value for the last intensity i.e. 255 is equal to 1, indicating that the entire range of values has been cumulated. From the graphs, it can be inferred that the order of frequency bins for intensities will remain the same as the CDF is monotonically increasing. Lastly, the CDF for the output image is more uniformly distributed as it seems to be gradually increasing whereas that of the input image has a steep increase in the beginning and then flattens at 1. It can be concluded that the more uniformly increasing the CDF graph is the better contrast an image has.

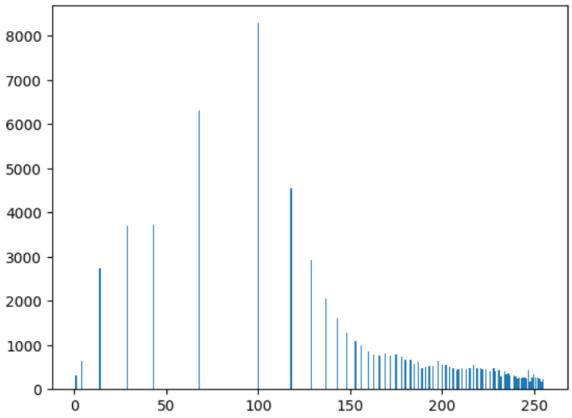
The final step of the histogram equalization requires us to apply the transfer function in order to recompute the new intensity values of the high-contrast pixels in the image. Here, since we are using uniform equalization, we multiply the CDF values of the intensity to (L-1) where L refers to the intensity range. The formula is as follows.

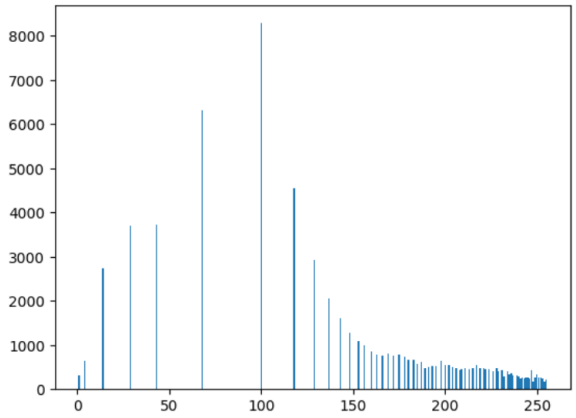
$$\mathbf{s = T(r) = CDF(r)\ X\ (L\text{-}1)}$$

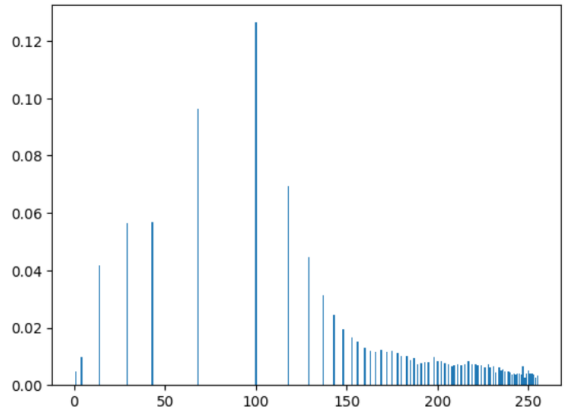
Here s refers to the output intensity value, T is the transfer function, r is the input intensity value, and as stated above L is the intensity range which in this case is 256 (0-255).

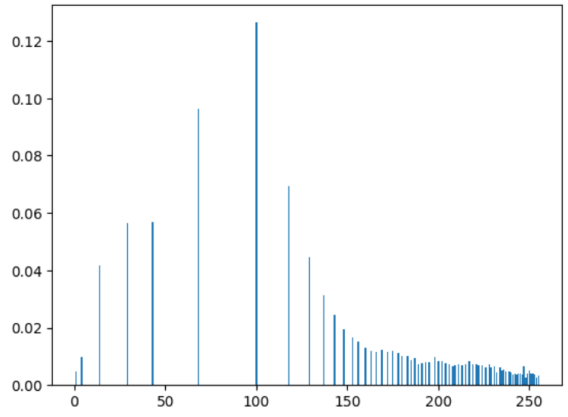
The s value for each intensity is calculated and then rounded off to the nearest whole number. The value then replaces the original intensity value in the image. Once all the values of the input image are replaced by the new intensity values, the high-contrast image is obtained.

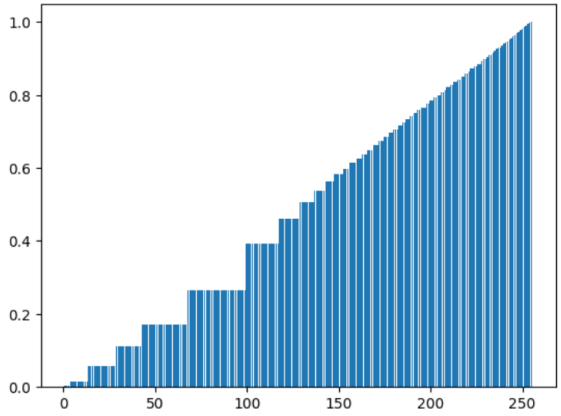
Let us reapply the method on the corrected high-contrast image. Now the input on the left is the high-contrast image we just created and the output on the right is the image obtained after performing histogram equalization again. The graphs below the images correspond to intensity vs frequency, intensity vs PDF and intensity vs CDF respectively.

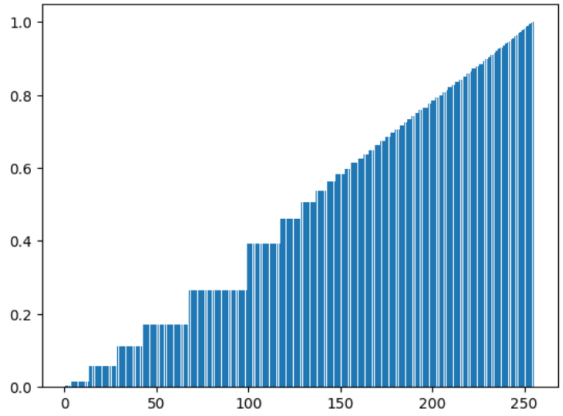

8000
7000
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250


8000
7000
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250


0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250


0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
50
100
150
200
250

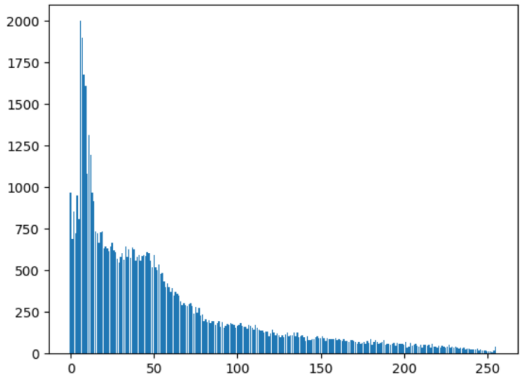

1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250

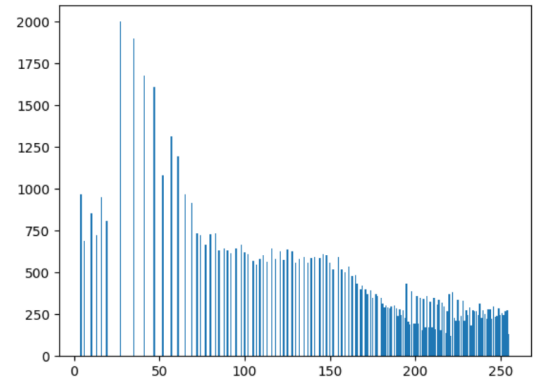

1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
250

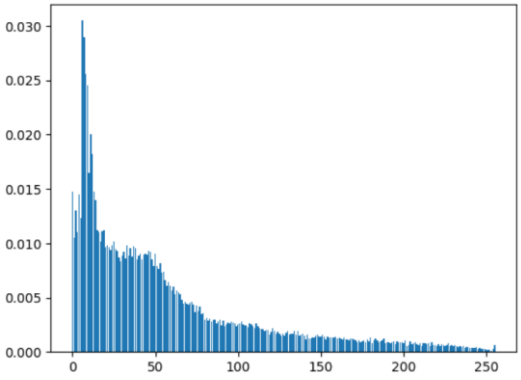
It can be observed that the image contrast remains more or less the same. Thus it can be concluded that histogram equalization is an effective method for low-contrast images only.

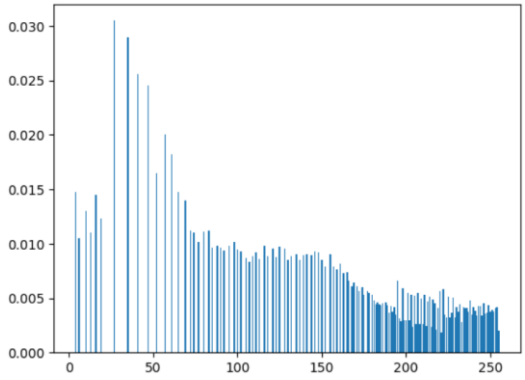
Let us try to apply histogram equalization to another low-contrast image. Now the input on the left is the low-contrast image and the output on the right is the high-contrast image obtained after performing histogram equalization again. The graphs below the images correspond to intensity vs frequency, intensity vs PDF and intensity vs CDF respectively.

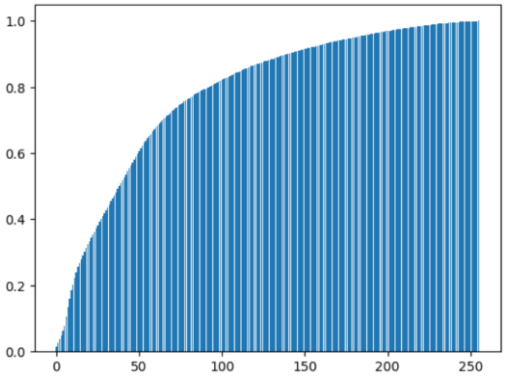


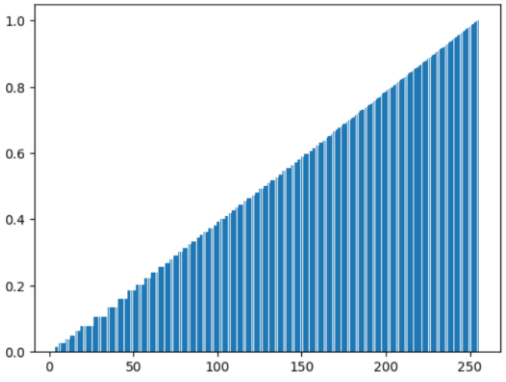












It can be observed that histogram equalization has really enhanced the contrast of this image. In the low-contrast image, the text was barely visible but in the high-contrast image, it is clearly legible. Other features of the image also can be understood well. The frequency and PDF distribution is more uniformly spread over the intensity range and the CDF graph is uniformly increasing.

A2: OTSU THRESHOLDING

As per Wikipedia, thresholding is the simplest technique for segmenting images in digital image processing. Thresholding can be used to produce binary pictures from a grayscale image. The most basic thresholding techniques convert each pixel in an image to a black or white pixel depending on whether it exceeds or falls below a predetermined value known as the threshold T.

While the user can choose the threshold T manually in some circumstances, there are many situations when the user prefers that the threshold be chosen by an algorithm automatically. The threshold in those circumstances ought to be the "best" threshold possible, meaning that the division of the pixels above and below the threshold ought to be as similar as possible.

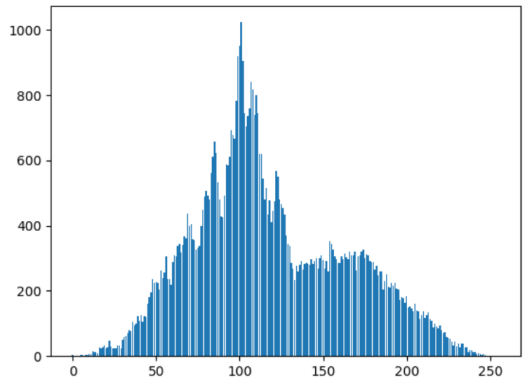
According to Open Source Computer Vision Python documentation, The function cv.threshold is used to apply the thresholding. The first argument is the source image, which should be a grayscale image. The second argument is the threshold value which is used to classify the pixel values. The third argument is the maximum value which is assigned to pixel values exceeding the threshold. OpenCV provides different types of thresholding which is given by the fourth parameter of the function. The types of thresholding are as follows:

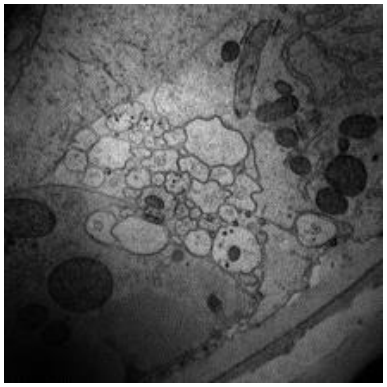
THRESH_BINARY Python: cv.THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC Python: cv.THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO Python: cv.THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$

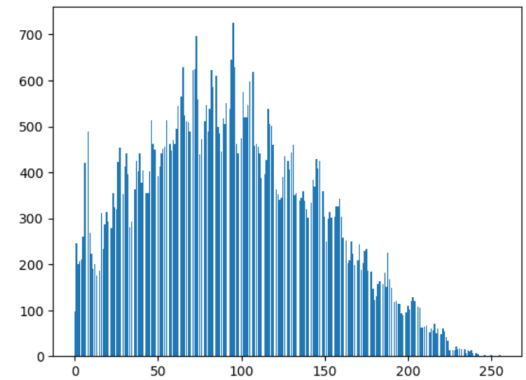
THRESH_BINARY is the most basic form of threshold that can be applied to an image. It implies that a pixel is set to the maximum value in argument 3 (which is usually 255 – white) if its intensity is above user given threshold, else it is set to 0 (Black). Similarly, one can interpret the other threshold types. This can also be done by iterating over the entire image matrix using for loops and checking if pixel intensity is above or below the user-entered threshold using the if statement and then reassigning pixel intensity. However since these inbuilt functions have been provided, we can use them for testing manual thresholding.

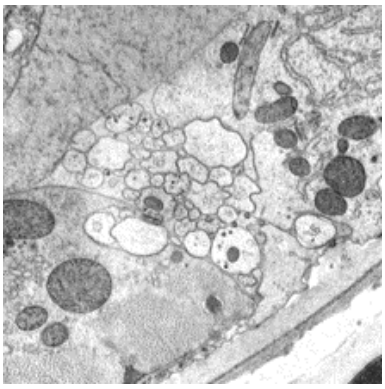
One can also directly use the Otsu thresholding as a threshold type by having cv.THRESH_BINARY + cv.THRESH_OTSU as the threshold type as demonstrated in the code in the Appendix. However, in this case, the threshold value entered by the user manually has no meaning as the Otsu algorithm will automatically choose an appropriate threshold. I have used this inbuilt Otsu Threshold type in the threshold function to visually compare the results to the results obtained from my Otsu Threshold code.

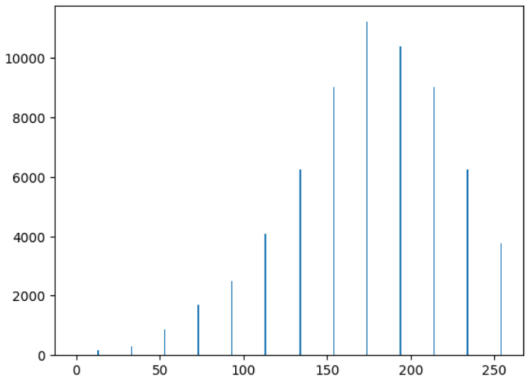
As part of assignment 2, we were given 3 images to perform threshold on – 'b2_a.png', 'b2_b.png', and 'b2_c.png'. They have been displayed below along with their intensity vs frequency histograms.



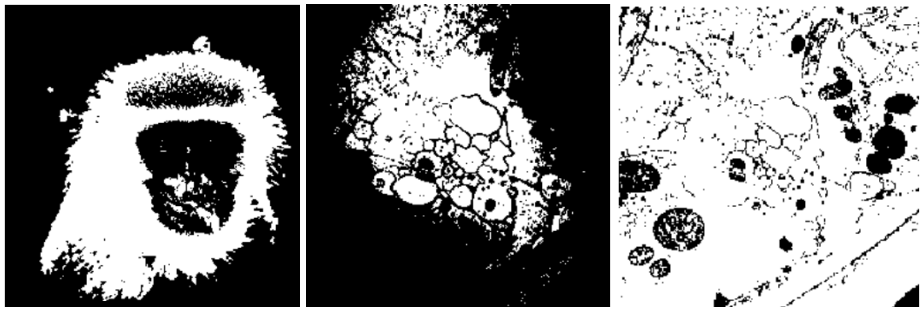




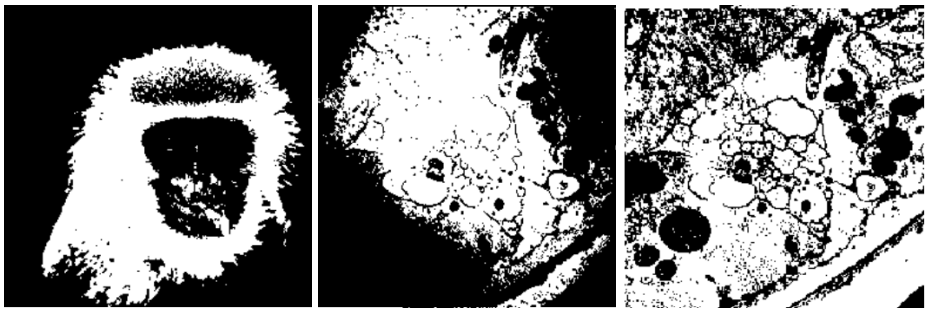




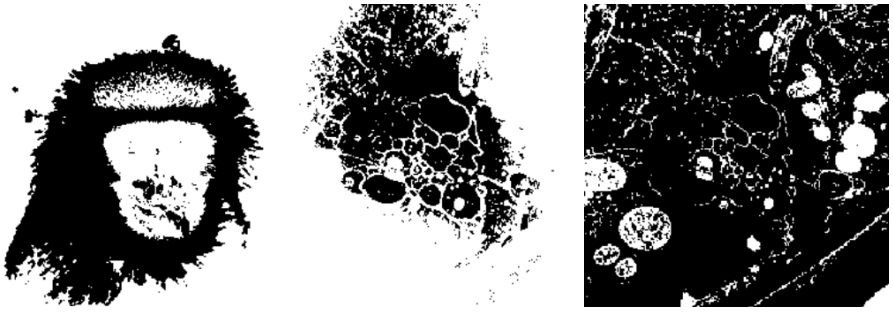
On displaying the images it seems that the third image is the high-contrast version of the second image. In the manual thresholding part of the assignment we have applied the cv.threshold function to the input image with all the possible threshold types and a user-entered threshold value of 127. This value has been chosen at random. It is also midway the intensity range. Following are the results obtained for the different threshold types with and without using the inbuilt cv.THRESH_OTSU type.



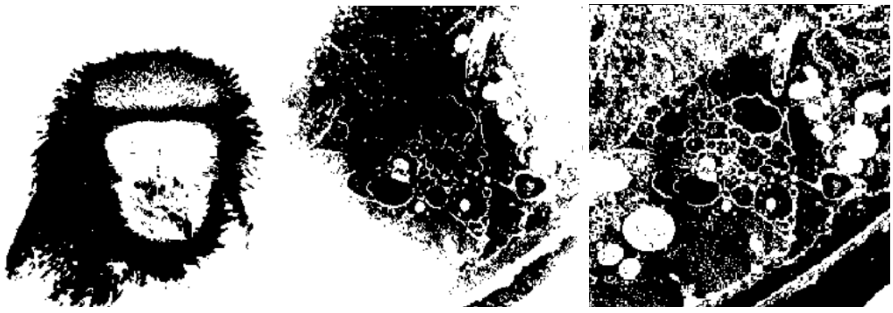
Binary



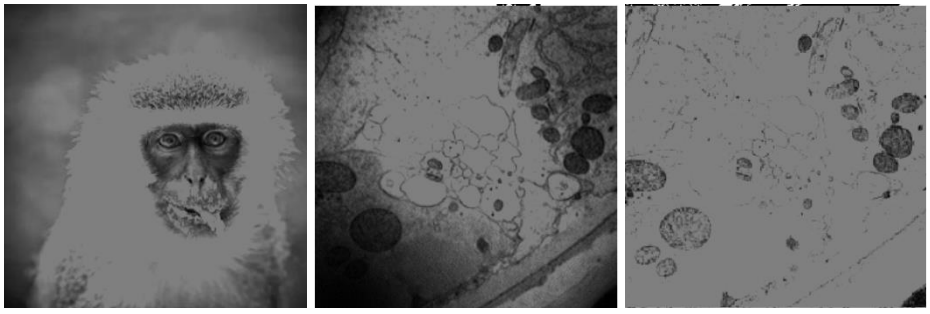
Binary Otsu



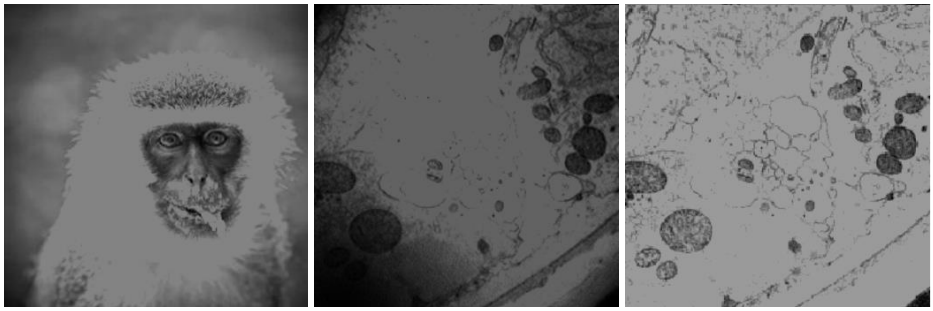
Binary Inverse

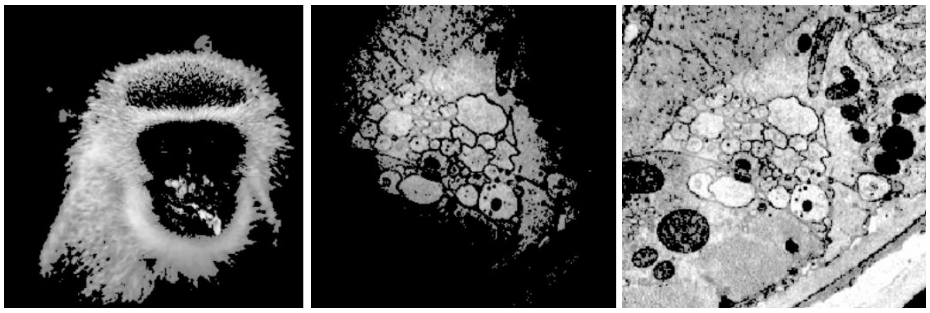


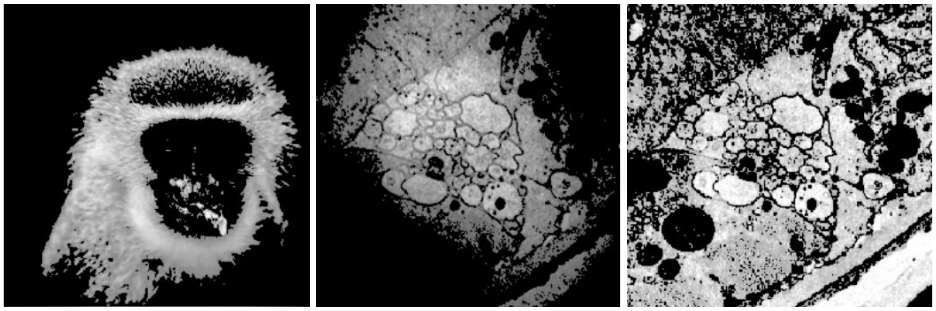
Binary Inverse Otsu

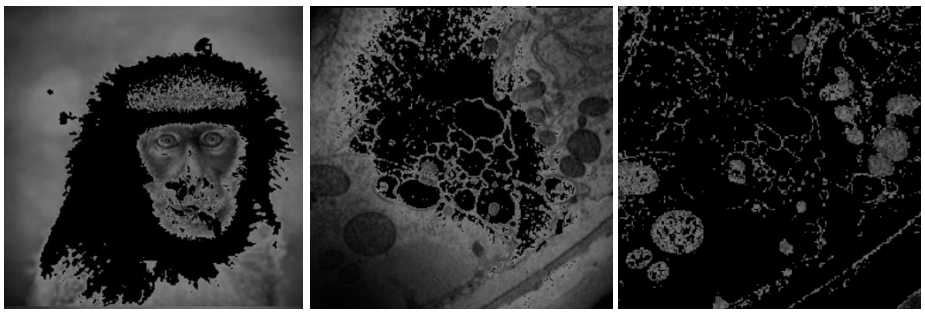


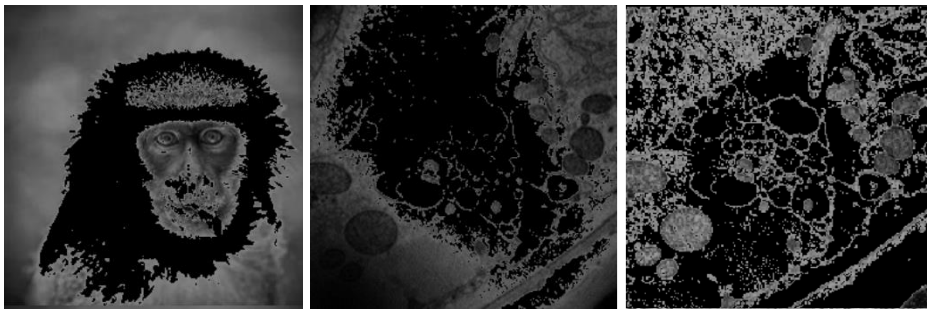
Trunc


Trunc Otsu


Tozero


Tozero Otsu


Tozero Inverse


Tozero Inverse Otsu

In the first image, the inbuilt Otsu threshold type is slightly increasing the quality of the threshold image. Unnecessary pixels labeled wrong in the image have been corrected after adding the Otsu parameter to the threshold type. In the second image, the inbuilt Otsu threshold type is slightly decreasing the quality of the threshold image. Pixels around the objects/shapes are being relabelled

causing the shapes in images to look more blurry. Lastly, In the third image, the inbuilt Otsu threshold type drastically increases the quality of the threshold image. Now, more features/shapes in the image are definitely visible. Since the third image is the high-contrast version of the second it can be inferred that Otsu works better on high-contrast images than on low-contrast ones.

Otsu thresholding is an automatic thresholding method that finds the intensity that has the minimum intra-class variance or the maximum inter-class variance. This intensity value is set as the optimum threshold for Otsu thresholding. Its equation can be described as follows:

$$(Var_w)^2(t) = W_0(t)\,(Var_0)^2(t) + W_1(t)\,(Var_1)^2(t)$$

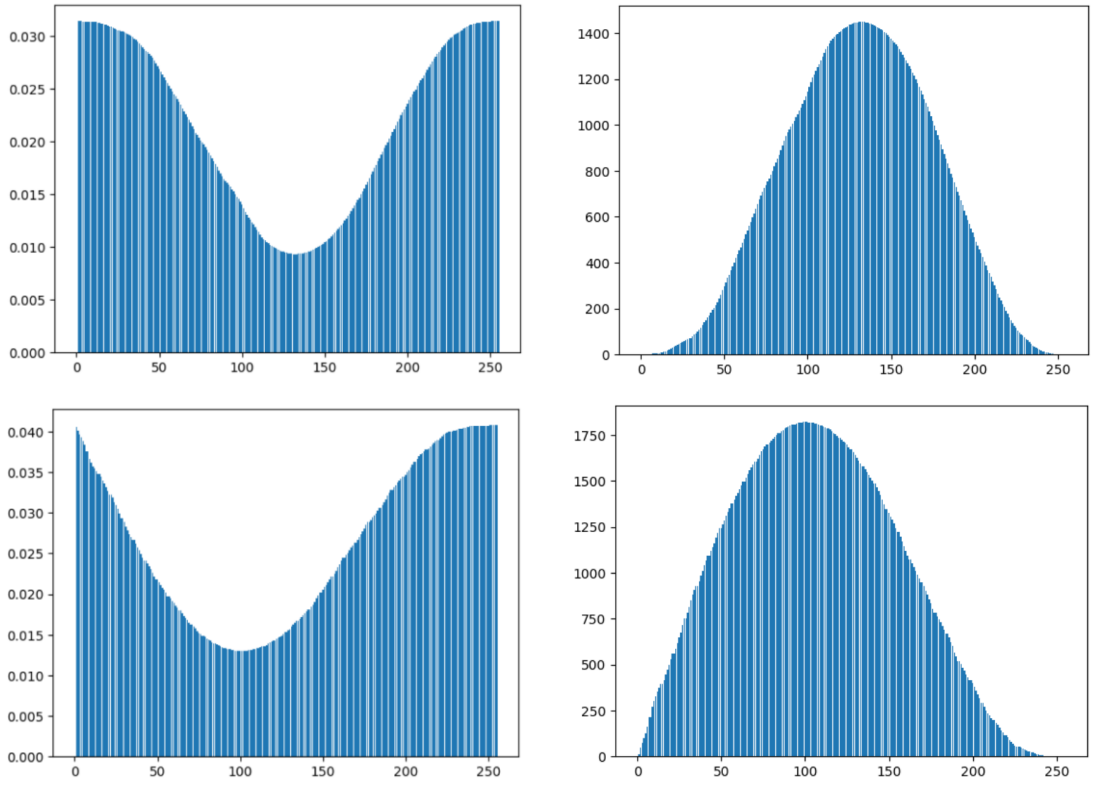
Here, Var_w refers to the intra-class variance at threshold t, W_0 refers to the number of pixels less than t, Var_0 refers to the variance of pixels less than t, W_1 refers to the number of pixels greater than or equal to t, and Var_1 refers to the variance of pixels greater than or equal to t. For each threshold, Var_w is calculated and the threshold having minimum Var_w is chosen as the Otsu threshold. Here, Var_0 and Var_1 are calculated by first finding the mean values as shown in the code in the Appendix. The background pixels are represented by 0 and the foreground pixels are represented by 1.

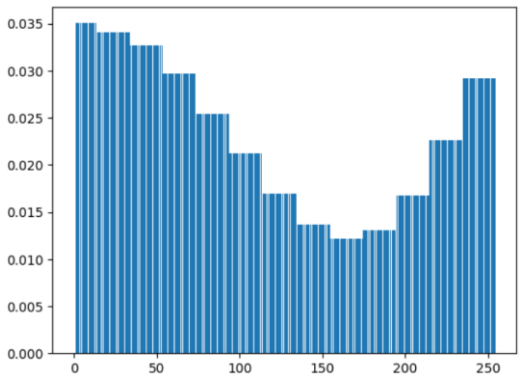
The alternative equation for finding the Otsu threshold is by maximizing the inter-class variance which is the same as minimizing the intra-class variance. The equation can be described as follows:

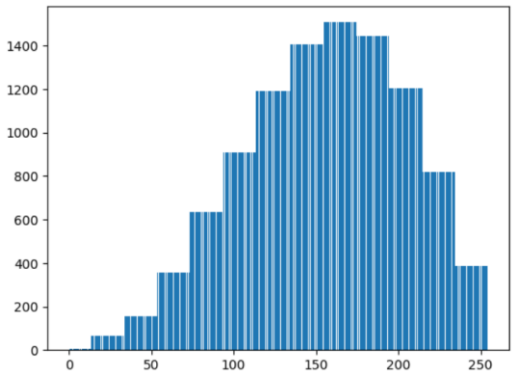
$$(Var_b)^2(t) = W_0(t)\,W_1(t)\,(Mean_0(t) - Mean_1(t))^2$$

Here Var_b is the inter-class variance. It is calculated for each threshold value in the range and the one with the maximum inter-class variance is chosen as the Otsu threshold. In the Appendix, there is code for finding the Otsu threshold by both methods.

Given below are the plots for the intra (left) and inter (right) class variances of the intensity range for the 3 given input images respectively.







It can be observed that the intra-class variance initially decreases until the minimum threshold is achieved and then increases again whereas the inter-class variance gradually increases until the maximum threshold is achieved and then decreases again. Also, it can be observed that these two graphs are opposite of each other in shape and the minimum threshold in the first graph is the same as the maximum threshold in the second in every case. Thus, it can be inferred that the calculations done in the code by both methods match the same Otsu threshold and thus are mathematically correct.

Now we can also check the visual results and compare them to the manual threshold results with Otsu added to the threshold type. In this case, we will call the cv.threshold functions without adding cv.THRESH_OTSU in the threshold type parameter. We will pass the value of threshold which is the index number of the minimum intra-class variance or the index number of the maximum inter-class variance which ideally should be the same index.

Given below is the Otsu threshold, minimum intra-class variance, and the maximum inter-class variance for the given input images.

First Image: b2_a.png

```
Otsu Threshold:  131
Minimum Intra-Class Variance:  0.009354531525074216
Maximum Inter-Class Variance:  1448.0136082304264
```

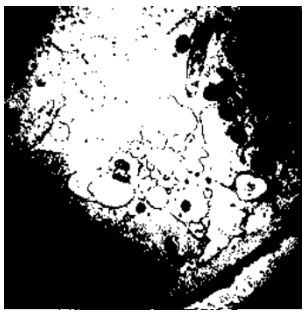
Second Image: b2_b.png

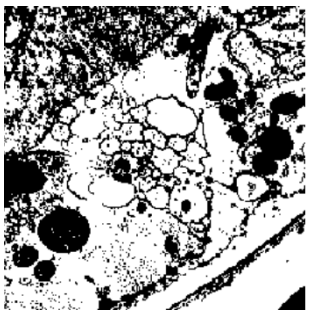
```
Otsu Threshold:  98
Minimum Intra-Class Variance:  0.01299592143231117
Maximum Inter-Class Variance:  1820.0308771863783
```

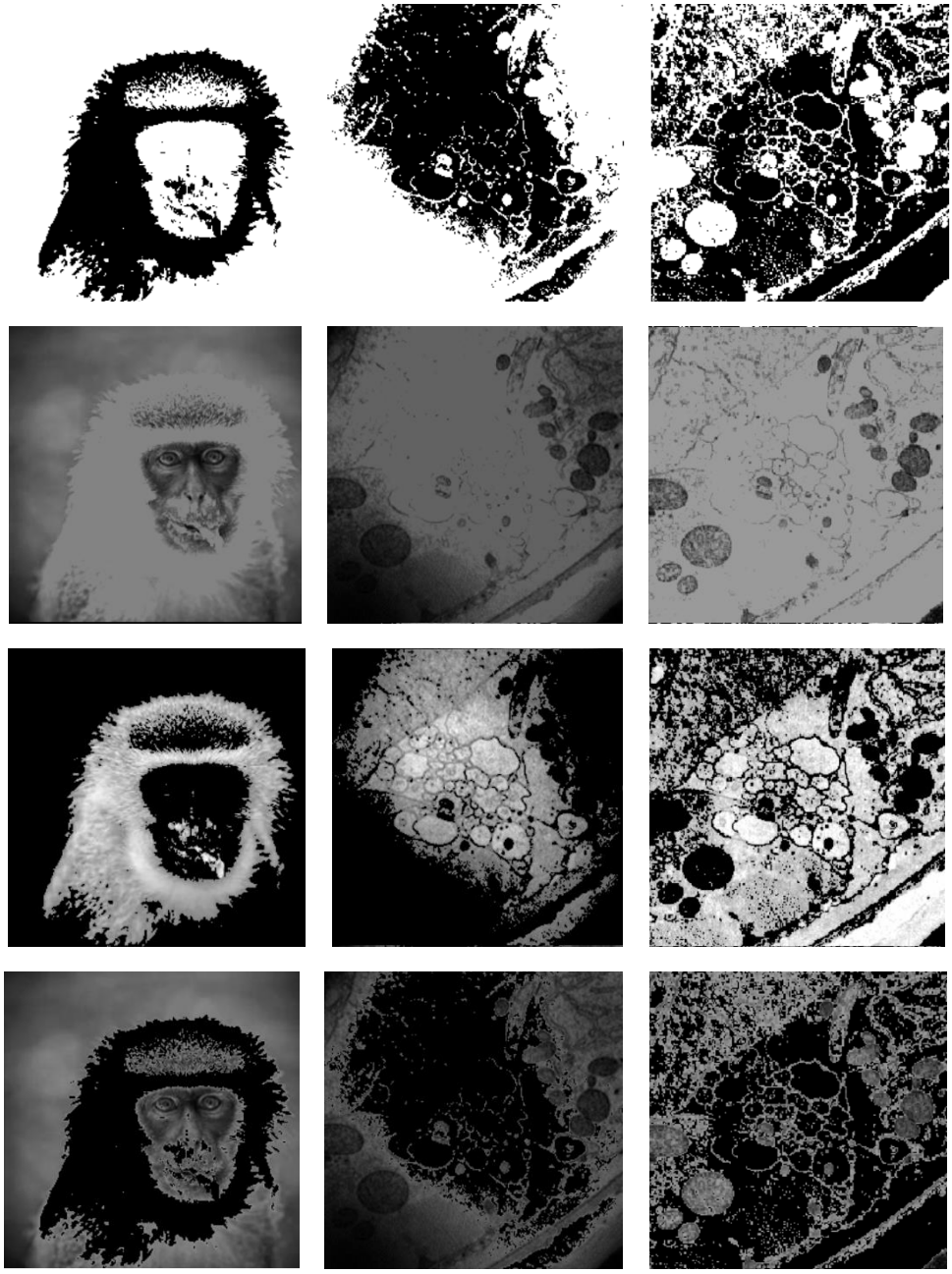
Third Image: b2_c.png

```
Otsu Threshold:  154
Minimum Intra-Class Variance:  0.012123562073845916
Maximum Inter-Class Variance:  1506.470214939216
```

Given below are the visual outputs:







It can be observed that the visual results of images obtained after Otsu thresholding are the same as those obtained when using the inbuilt Otsu Threshold type function. This shows that the experiment was carried out accurately.

The automatic threshold produces good results for images with higher contrast. If we have low-contrast images, we should ideally perform histogram equalization first and then go for automatic thresholding. In the first and third images, the pixels denoting the monkey and pixels denoting the shapes in the rug were separated out and one can tell these features apart from the background in the threshold image.

A3: HISTOGRAM MATCHING

According to Wikipedia, Histogram matching, also known as histogram specification, is the act of transforming an image's histogram to match a predetermined histogram. A specific scenario in which the provided histogram has a uniform distribution is the well-known histogram equalization method.

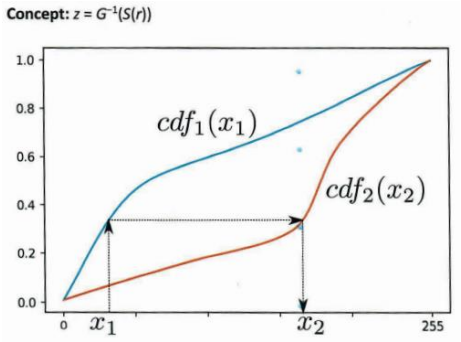
As a relative detector calibration method, histogram matching can be used to balance detector responses. When two photographs were taken at the same time under the same local illumination (such as shadows), but with different sensors, atmospheric conditions, or global illumination, it can be used to normalize the images.

According to notes provided in class, the goal of histogram matching is to map the intensities of the source image so that the new histogram matches the goal histogram. Thus, this functionality takes 2 inputs a source image (ideally of lower contrast) and a target image (ideally of higher contrast). The goal of this is also to increase the contrast of the source image to that of the target image. If we are using RGB images histogram matching is also used to incorporate the color scheme of the target image in the source image.

The process followed to perform histogram matching is as follows. First, we find the probability distribution function of both the source and target histogram. Then we calculate the cumulative distribution function. These processes can be done exactly like in A1: HISTOGRAM EQUALIZATION. Lastly, we map the cumulative probability value of the source image intensity level to the nearest cumulative probability value of the target image intensity level. This target image intensity level swaps out the source image intensity level in the matched image. Mathematically it can be represented by the following equations:

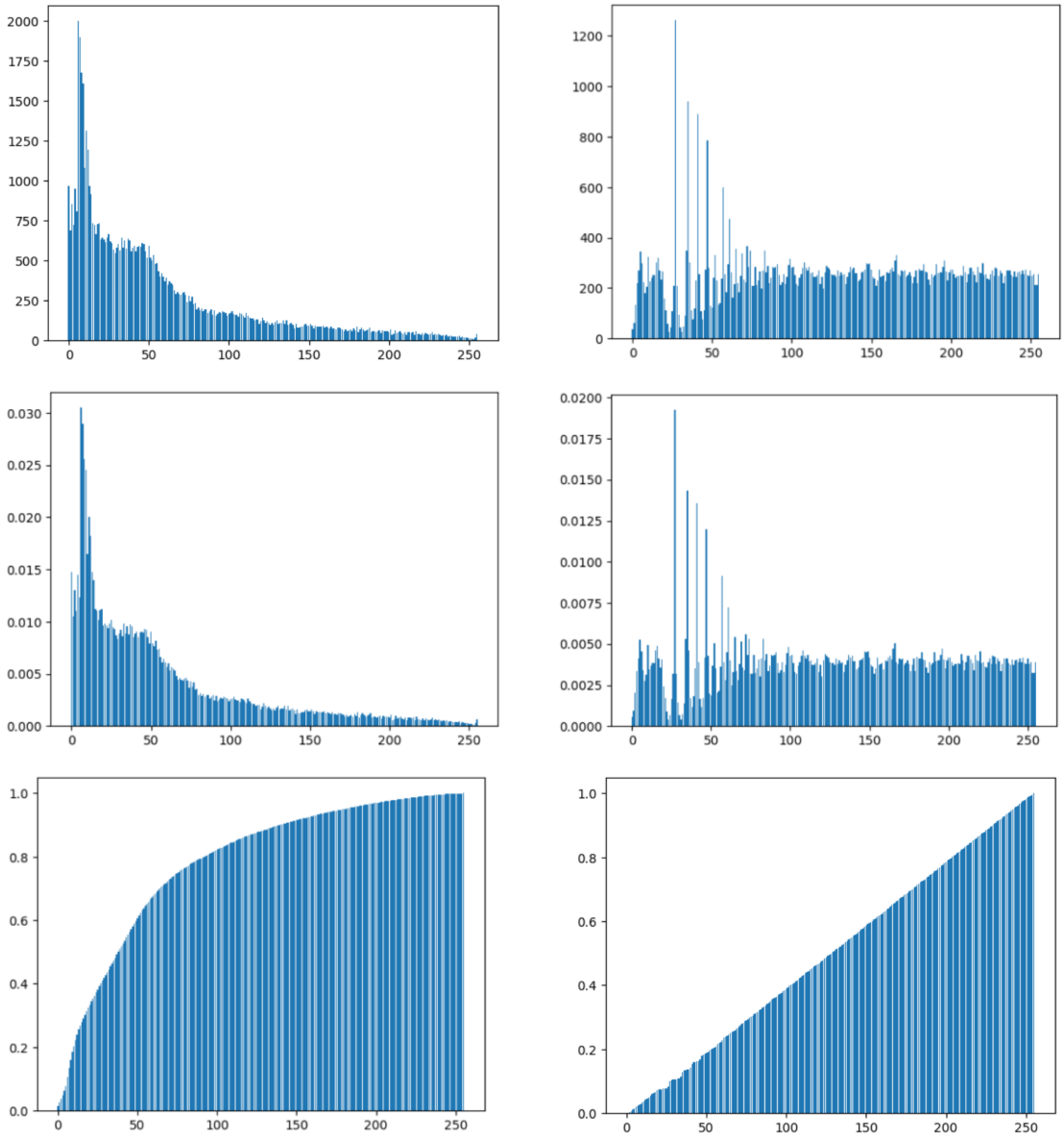
$$\mathbf{s = T(r) \text{ and } s = G(z) \text{ therefore } T(r) = G(z) \text{ and } z = G^{-1}(s) = G^{-1}(T(r))}$$

Here, r refers to the source image intensity value, T(r) refers to the cumulative distribution function for r, z refers to the target image intensity value, G(z) refers to the cumulative distribution function for z, and s refers to the output matched cumulative distribution value of both the images.

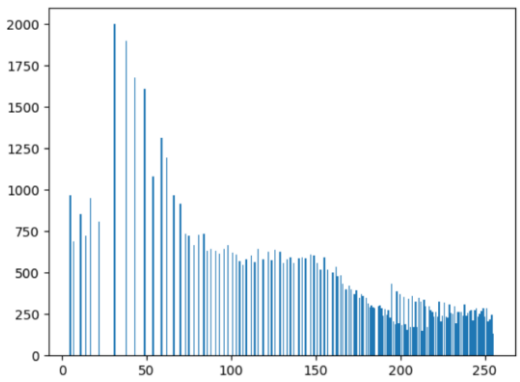


Experiment 1: Let the source image be a low-contrast image and the target image be a high-contrast image of the same scene. Perform histogram matching and discuss results.

The left side represents the source image and the right side represents the target image. Given below are the intensity vs frequency, intensity vs PDF, and intensity vs CDF graphs for the images.



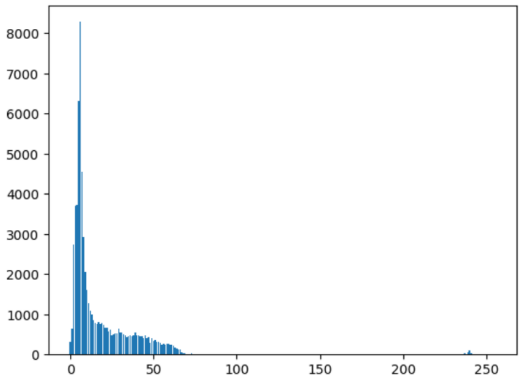
It is obvious looking at the CDF graphs of the images that if we map the CDF of each intensity value of the source image to the closest CDF intensity value of the target image then a lot of pixel values will get much brighter as the target image has a uniform distribution. The results of the experiment created a matched image as follows.

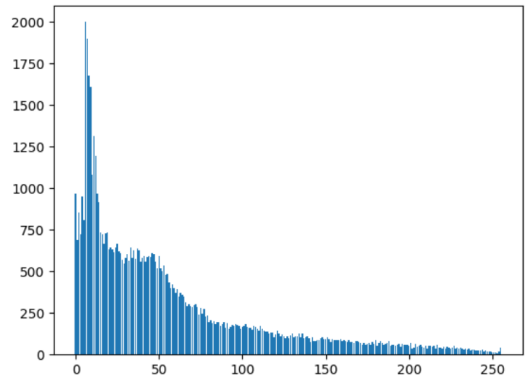


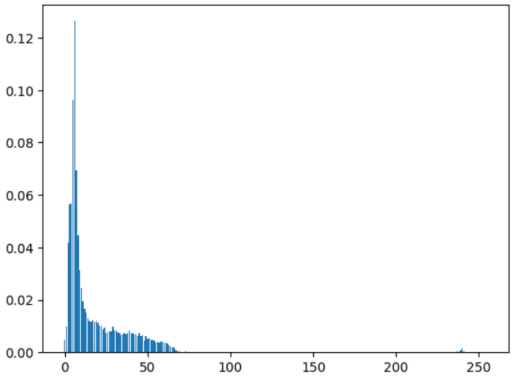
The matched image has more uniformly distributed frequencies. Its features have brightened and are better visible compared to the original source image and its contrast has increased. Thus, this experiment has given desired results and displays an effective use case of histogram matching.

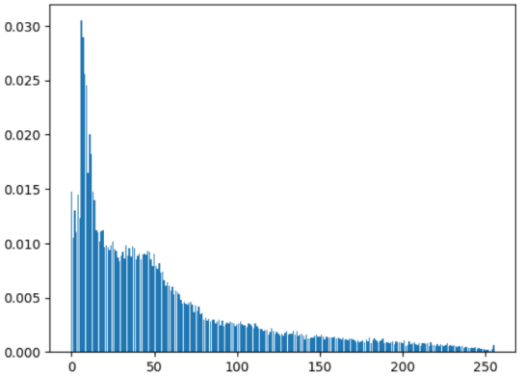
Experiment 2: Let the source image be a low-contrast image and the target image be a low-contrast image of a different scene. Perform histogram matching and discuss results.

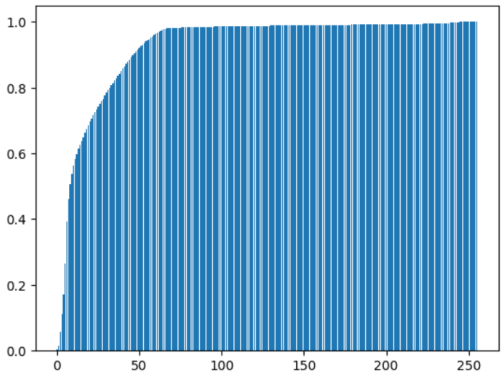
The left side represents the source image and the right side represents the target image. Given below are the intensity vs frequency, intensity vs PDF, and intensity vs CDF graphs for the images.

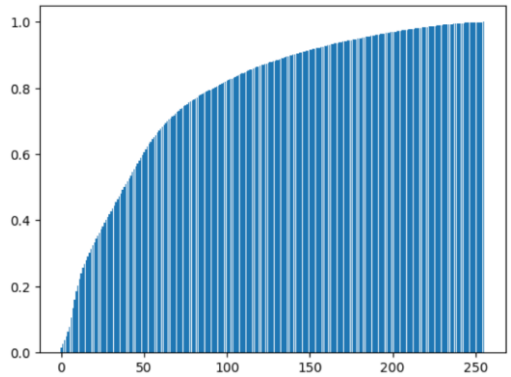




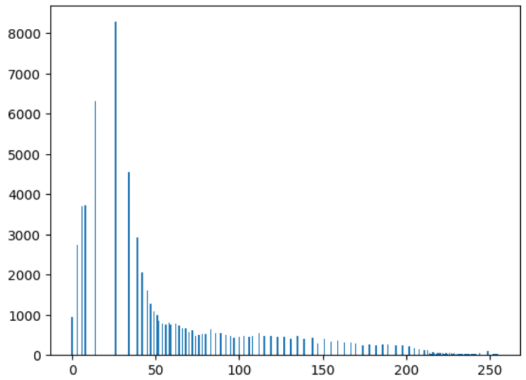








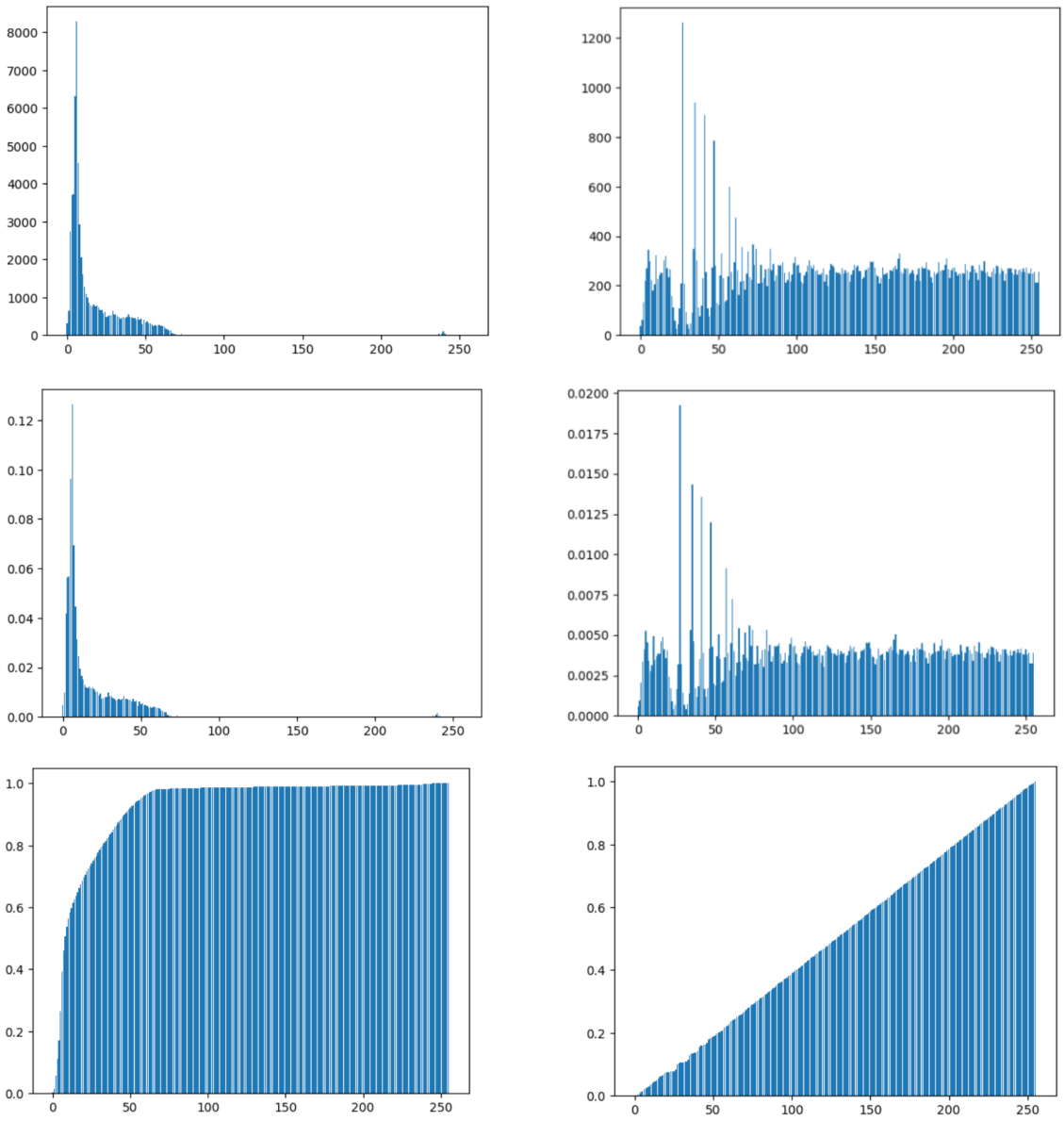
The results of the experiment created a matched image as follows.



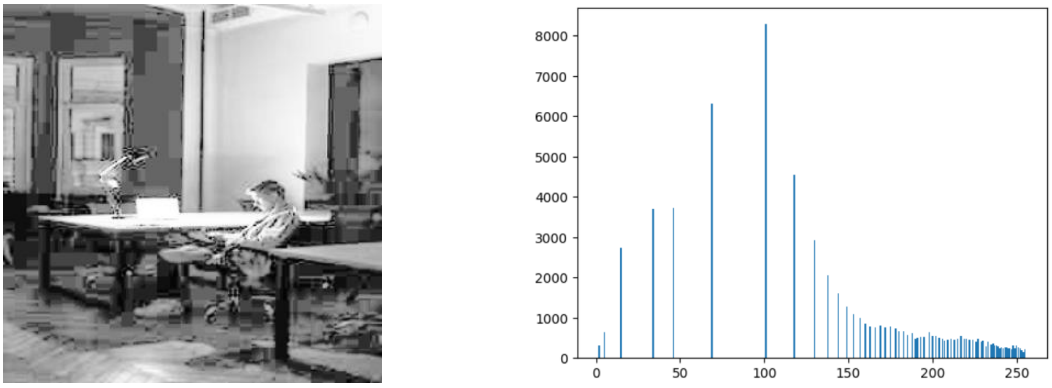
The matched image has more uniformly distributed frequencies. Its features have brightened and are better visible compared to the original source image and its contrast has increased. Thus, this experiment has given desired results and displays an effective use case of histogram matching. Even though both images were of low contrast, a different scene has a different pixel distribution and thus can be used to bring out features of the source image as seen in the result above.

Experiment 3: Let the source image be a low-contrast image and the target image be a high-contrast image of a different scene. Perform histogram matching and discuss results.

The left side represents the source image and the right side represents the target image. Given below are the intensity vs frequency, intensity vs PDF, and intensity vs CDF graphs for the images.



The results of the experiment created a matched image as follows.



The matched image has more uniformly distributed frequencies. Its features have brightened and are better visible compared to the original source image and its contrast has increased. Thus, this experiment has given desired results and displays an effective use case of histogram matching. A different scene has a different pixel distribution and thus can be used to bring out features of the

source image as seen in the result above. The result is similar to the result we got on performing histogram equalization on 'indoors.png'.

APPENDIX: PYTHON CODE

To run the code open link: https://colab.research.google.com/drive/10ukiVyyHdCbuCJdQy-ZwJIqBeVl477yy?usp=sharing

```
from PIL import Image
import numpy as np
import collections
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
import cv2 as cv

plotno = 1

def preprocess_img(im_in):
  original_image = Image.open(im_in)
  resized_image = original_image.resize((256, 256))
  im = np.array(resized_image.convert('L'))
  im_uint8 = ((im - np.min(im)) * (1/(np.max(im) - np.min(im)) *
255)).astype('uint8')
  return im_uint8

def find_missing(lst):
  return [i for x, y in zip(lst, lst[1:]) for i in range(x + 1, y) if y
- x > 1]

def create_pdf(im_in):
  cv2_imshow(im_in)
  freq = {}
  for i in im_in:
    for j in i:
      if j not in freq:
        freq[j] = 1
      else:
        freq[j] += 1
  freq = collections.OrderedDict(sorted(freq.items()))
  missing = find_missing(list(freq.keys()))
  for i in missing:
    freq[i] = 0
  freq = collections.OrderedDict(sorted(freq.items()))
  intensities = list(freq.keys())
  frequencies = list(freq.values())
  pdf = []
  total = sum(frequencies)
  for i in frequencies:
    pdf.append(i/total)
```

```
  return pdf, intensities, frequencies


def create_cdf(pdf):
  total = 0
  cdf = []
  for i in pdf:
    total += i
    cdf.append(total)
  return cdf

def plot(l1, l2, plotno):
  plt.bar(l1, l2)
  plt.show()
  plt.savefig('plot' + str(plotno) + '.png')
  return plotno + 1

def histogram_equalization(im_in, plotno):
  pdf, intensities, frequencies = create_pdf(im_in)
  cdf = create_cdf(pdf)
  L = len(intensities)
  op_intensities = []
  for i in cdf:
    op_intensities.append(round(i*(L-1)))
  equalized_im = im_in
  for i in range(len(im_in)):
    for j in range(len(im_in[i])):
      x = im_in[i][j]
      if x+1 < len(op_intensities):
        equalized_im[i][j] = op_intensities[x]
  cv2_imshow(equalized_im)
  plotno = plot(intensities, frequencies, plotno)
  plotno = plot(intensities, pdf, plotno)
  plotno = plot(intensities, cdf, plotno)
  return equalized_im, plotno

im_in = preprocess_img('indoors.png')
equalized_im, plotno = histogram_equalization(im_in, plotno)
cv.imwrite('indoors_op.png', equalized_im)

equalized_im2, plotno = histogram_equalization(equalized_im, plotno)
cv.imwrite('indoors_op2.png', equalized_im2)

im_in = preprocess_img('test.jpg')
equalized_im, plotno = histogram_equalization(im_in, plotno)
cv.imwrite('test_op.jpg', equalized_im)

equalized_im2, plotno = histogram_equalization(equalized_im, plotno)
```

```
cv.imwrite('test_op2.jpg', equalized_im2)


def manual_threshold(im_in, threshold):
  ret, thresh1 = cv.threshold(im_in, threshold, 255, cv.THRESH_BINARY)
  cv2_imshow(thresh1)
  ret, thresh1 = cv.threshold(im_in, 0, 255, cv.THRESH_BINARY +
cv.THRESH_OTSU)
  cv2_imshow(thresh1)

  ret, thresh2 = cv.threshold(im_in, threshold, 255,
cv.THRESH_BINARY_INV)
  cv2_imshow(thresh2)
  ret, thresh2 = cv.threshold(im_in, 0, 255, cv.THRESH_BINARY_INV +
cv.THRESH_OTSU)
  cv2_imshow(thresh2)

  ret, thresh3 = cv.threshold(im_in, threshold, 255, cv.THRESH_TRUNC)
  cv2_imshow(thresh3)
  ret, thresh3 = cv.threshold(im_in, 0, 255, cv.THRESH_TRUNC +
cv.THRESH_OTSU)
  cv2_imshow(thresh3)

  ret, thresh4 = cv.threshold(im_in, threshold, 255, cv.THRESH_TOZERO)
  cv2_imshow(thresh4)
  ret, thresh4 = cv.threshold(im_in, 0, 255, cv.THRESH_TOZERO +
cv.THRESH_OTSU)
  cv2_imshow(thresh4)

  ret, thresh5 = cv.threshold(im_in, threshold, 255,
cv.THRESH_TOZERO_INV)
  cv2_imshow(thresh5)
  ret, thresh5 = cv.threshold(im_in, 0, 255, cv.THRESH_TOZERO_INV +
cv.THRESH_OTSU)
  cv2_imshow(thresh5)

  manual_thresh_img = thresh1
  return manual_thresh_img

def otsu_helper(threshold, pdf):
  w0, w1 = 0, 0
  for i in range(threshold):
    w0 += pdf[i]
  for i in range(threshold, len(pdf)):
    w1 += pdf[i]
  u0, u1 = 0, 0
  for i in range(threshold):
    u0 += (i * pdf[i])
```

```
  for i in range(threshold, len(pdf)):
    u1 += (i * pdf[i])
  if w0 == 0 or w1 == 0:
    return 0
  u0 /= w0
  u1 /= w1
  inter_class_var = w0 * w1 * ((u0**2) + (u1**2) - (2 * u0 * u1))
  return inter_class_var

def otsu_helper2(im_in, threshold, pdf):
  N = len(im_in) * len(im_in[0])
  w0, w1, u0, u1 = 0, 0, 0, 0
  for i in range(threshold):
    w0 += pdf[i]
    u0 += (i * pdf[i])
  for i in range(threshold, len(pdf)):
    w1 += pdf[i]
    u1 += (i * pdf[i])
  if w0 == 0 or w1 == 0:
    return 0
  u0 /= w0
  u1 /= w1
  var0, var1 = 0, 0
  for i in range(threshold):
    var0 += ((i**2) + (u0**2) - (2 * i * u0)) * pdf[i]
  for i in range(threshold, len(pdf)):
    var1 += ((i**2) + (u1**2) - (2 * i * u1)) * pdf[i]
  var0 /= w0
  var1 /= w1
  w0 /= N
  w1 /= N
  var = (w0 * var0) + (w1 * var1)
  return var

def otsu_threshold(im_in, plotno):
  pdf, intensities, frequencies = create_pdf(im_in)
  plotno = plot(intensities, frequencies, plotno)
  plotno = plot(intensities, pdf, plotno)
  all_intra_class_variance = [0]
  all_inter_class_variance = [0]
  for i in range(1, len(intensities)):
    all_intra_class_variance.append(otsu_helper2(im_in, i, pdf))
    all_inter_class_variance.append(otsu_helper(i, pdf))

  plotno = plot(intensities, all_intra_class_variance, plotno)
  plotno = plot(intensities, all_inter_class_variance, plotno)

  all_intra_class_variance = all_intra_class_variance[1:]
```

```
  all_inter_class_variance = all_inter_class_variance[1:]

  min_val = min(all_intra_class_variance)
  min_idx = all_intra_class_variance.index(min_val)

  max_val = max(all_inter_class_variance)
  max_idx = all_inter_class_variance.index(max_val)

  if min_idx == max_idx:
    print('Otsu Threshold: ', min_idx)
    print('Minimum Intra-Class Variance: ', min_val)
    print('Maximum Inter-Class Variance: ', max_val)
  else:
    print('ERROR')

  ret, thresh1 = cv.threshold(im_in, min_idx, 255, cv.THRESH_BINARY)
  cv2_imshow(thresh1)
  ret, thresh2 = cv.threshold(im_in, min_idx, 255,
cv.THRESH_BINARY_INV)
  cv2_imshow(thresh2)
  ret, thresh3 = cv.threshold(im_in, min_idx, 255, cv.THRESH_TRUNC)
  cv2_imshow(thresh3)
  ret, thresh4 = cv.threshold(im_in, min_idx, 255, cv.THRESH_TOZERO)
  cv2_imshow(thresh4)
  ret, thresh5 = cv.threshold(im_in, min_idx, 255,
cv.THRESH_TOZERO_INV)
  cv2_imshow(thresh5)

  otsu_thresh_img = thresh1
  return otsu_thresh_img, plotno

im_in = preprocess_img('b2_a.png')
manual_thresh_img = manual_threshold(im_in, 127)
otsu_thresh_img, plotno = otsu_threshold(im_in, plotno)

im_in = preprocess_img('b2_b.png')
manual_thresh_img = manual_threshold(im_in, 127)
otsu_thresh_img, plotno = otsu_threshold(im_in, plotno)

im_in = preprocess_img('b2_c.png')
manual_thresh_img = manual_threshold(im_in, 127)
otsu_thresh_img, plotno = otsu_threshold(im_in, plotno)

def histogram_matching(src_im, trg_im, plotno):
  pdf1, intensities1, frequencies1 = create_pdf(src_im)
  cdf1 = create_cdf(pdf1)
  plotno = plot(intensities1, frequencies1, plotno)
  plotno = plot(intensities1, pdf1, plotno)
```

```
  plotno = plot(intensities1, cdf1, plotno)
  pdf2, intensities2, frequencies2 = create_pdf(trg_im)
  cdf2 = create_cdf(pdf2)
  plotno = plot(intensities2, frequencies2, plotno)
  plotno = plot(intensities2, pdf2, plotno)
  plotno = plot(intensities2, cdf2, plotno)
  matched_im = src_im
  for i in range(len(src_im)):
    for j in range(len(src_im[i])):
      diff = []
      x = src_im[i][j]
      cdf1x = cdf1[x]
      for k in cdf2:
        diff.append(abs(k - cdf1x))
      index = diff.index(min(diff))
      matched_im[i][j] = index
  pdf3, intensities3, frequencies3 = create_pdf(matched_im)
  cdf3 = create_cdf(pdf3)
  plotno = plot(intensities3, frequencies3, plotno)
  return matched_im, plotno

src_im = preprocess_img('indoors.png')
trg_im = preprocess_img('test.jpg')
matched_im, plotno = histogram_matching(src_im, trg_im, plotno)

src_im = preprocess_img('test.jpg')
trg_im = preprocess_img('test_op.jpg')
matched_im, plotno = histogram_matching(src_im, trg_im, plotno)

src_im = preprocess_img('indoors.png')
trg_im = preprocess_img('test_op.jpg')
matched_im, plotno = histogram_matching(src_im, trg_im, plotno)
```