Project 1: Histogram Equalization and Otsu Thresholding

Computer Vision

Simardeep Singh Mehta

N12540493

New York University

Tandon School Of Engineering

Professor Guido Gerig

Index and Contents

A1) Histogram Equalization

Histogram equalization is an image processing technique that is used to increase the contrast of an image by redistributing the pixel values of the image in such a way that it creates a more uniform histogram. Before going into detail about how histogram equalization works, let's first understand what a histogram of an image is.

A1(a) Histogram of an image

An image's histogram shows the frequency distribution of the pixel values in the image graphically. Pixel intensity values range from 0 to 255 on the X-axis, and the number of times each pixel value appears in the image is represented on the Y-axis.

A picture's histogram assists in our understanding and visualization of a certain characteristics. The histogram will be skewed to the left, for instance, if the image being studied is dark, because there will be more values in lower intensity.

On the same lines, a low contrast image will be represented by a narrow histogram. The purpose of histogram equalization is to make the original image's histogram more evenly distributed so that it produces an output image of high contrast.

How Histogram Equalization works?

So, histogram equalization basically requires creation of Probability distribution function and cumulative distribution function for applying the process of equalization which is the task at hand in the first question

The Histogram Equalization consists of the following steps.

1. ***Histogram Calculation or Probability Density Function (PDF:*** We compute the image's histogram by iterating through the image's pixel values and count the number of times each pixel value appears. After that, we normalize the output and bring the sum of the frequencies to one by dividing the frequencies by the total number of intensity values. This is known as the image's Probability Density Function.
2. ***Cumulative Distribution Function (CDF)***: The image's cumulative distribution function is calculated in the following stage. The cumulative sum of PDF values up to a given intensity value defines the CDF at that level. To obtain the final equalized image, we then use the Cumulative Distribution Function as the mapping function.
3. ***Equalization***: We apply the mapping function to every pixel in the original image in this step. In essence, we obtain the specific pixel's CDF value and utilize it in the finished picture. The final image will be an enhanced contrast version of the original image that has been histogram equalized.

Histogram Equalization Algorithm

1. **Compute the Histogram:** Calculate the histogram of the image, which gives the number of pixels at each intensity level.

2. **Calculate the Cumulative Distribution Function (CDF):** The CDF is the cumulative sum of the histogram. It maps the intensity levels in the input image to new intensity levels in the output image.

3. **Normalize the CDF:** The CDF is normalized to span the full intensity range of the image. This is done by subtracting the minimum histogram value from each entry in the CDF and then multiplying the result by the ratio of the intensity range to the number of pixels in the image.

4. **Create the Equalized Histogram:** Use the normalized CDF to map the intensity levels of the input image to new levels which result in the equalized histogram. Each pixel in the input image is replaced with its corresponding value in the normalized CDF.

5. **Generate the Equalized Image:** Replace each pixel in the input image with its new intensity value to get the histogram-equalized image.

A1 (b) Histogram Equalization in Action

Applying histogram equalization algorithm to a low contrast image, we can clearly view the people.png before applying histogram equalization as follows:

Figure 1: Original image on which histogram equalization is applied

We see that the above image is extremely dark with majority of pixel values that are close to 0(zero).

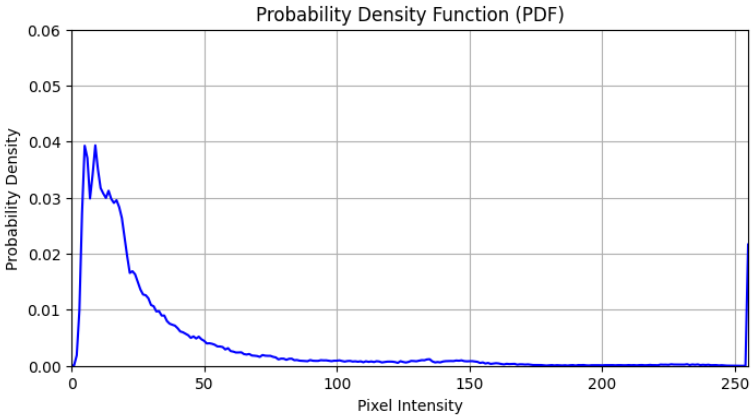


Figure 2: The PDF of the original image

As we clearly observe from the above given PDF, we have a peak at the left side of the spectrum which basically indicates that the frequency of pixel values in the darker side of the spectrum is high.

To perform histogram equalization on the image, we compute the Cumulative Distribution Function of the image. The CDF is nothing but the cumulative sum of the PDF values up to the current intensity level.

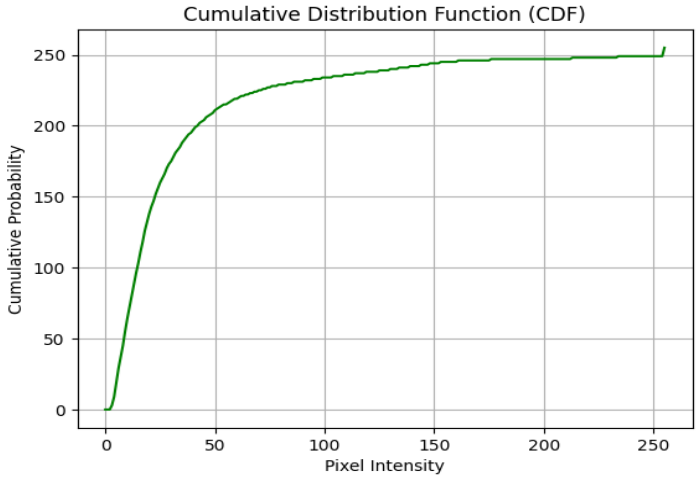


Figure 3: The CDF of the original image

Now we can use the CDF function of the image as a mapping function to get the histogram equalized image. We get the following image as output:

Figure 4: Histogram equalized output image

And the corresponding histogram of the output image is as follows:

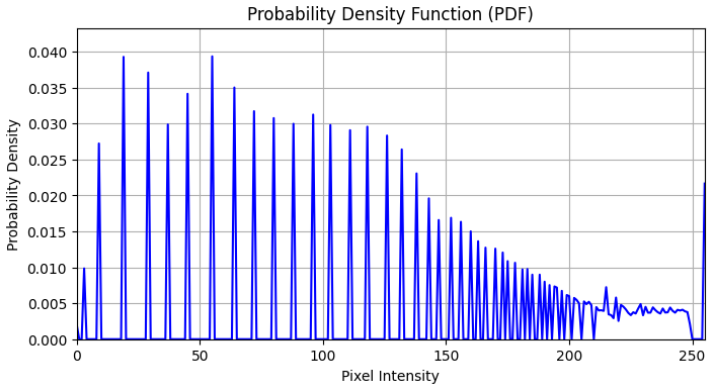


Figure 5: PDF of the histogram equalized output image

From the above histogram, the histogram equalization algorithm took the left skewed, dark image as input and created an output image with a histogram that is more spread out across all the available intensity values.

A1(c) Discussion of the above given experiment.

Changes in the Image and Histogram:

I. Before Histogram Equalization:

The original image is dominated by dark intensities, which is reflected in the concentration of pixel values at the lower end of the intensity spectrum. This results in an image that lacks distinction in the details, especially in the darker regions.

The PDF is sharply peaked at the lower intensities, which confirms our deduction that the image is more towards the darker side. This concentration at the lower end can obscure details and textures in the image.

II. **After Histogram Equalization:**

Post-equalization, the image displays a significantly enhanced contrast. The previously hidden details in the dark areas are now more apparent. The image has a balanced representation of shadows, mid-tones, and highlights, making the image clearer in contrast to input image.

The PDF of the equalized image is completely different from the initial one, rather than clustering at the lower end of the spectrum, it is spread across a whole range of possible intensities. This spread of pixel intensity values is indicative of an increase in contrast and visibility across the image.

A1 (d) CDF Explanation:

Now, let us compare the CDF of the original image and the histogram equalized image.

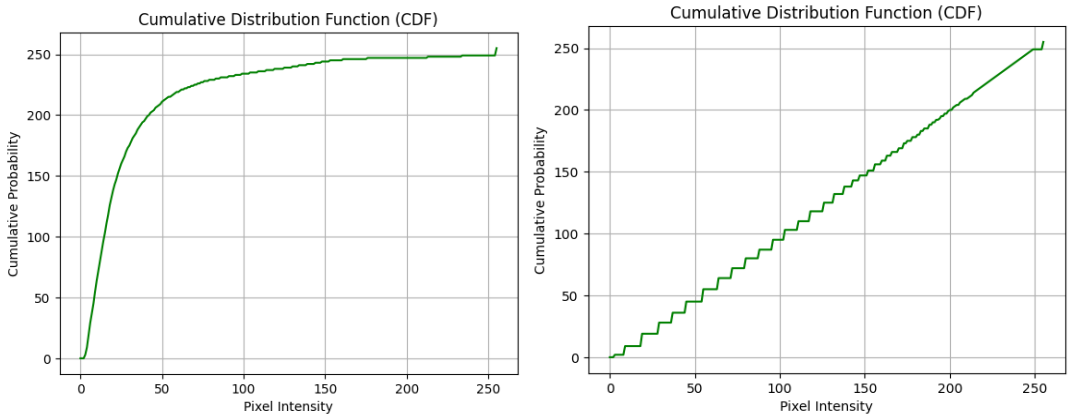


Figure 6: Comparison of CDFs of the original image and the equalized image

It is evident from the above two images that the CDF of the image tends to have a **linear trend** after histogram equalization. When the CDF is a perfect straight line with slope 1 and intercepts 0, it indicates that the image's histogram is perfectly uniform with all intensity values appearing with the same probability. The goal of histogram equalization is to achieve an output image that is as close to perfect uniformity as possible. Thus, it tends to have a **linear trend**.

Original CDF Shape and Interpretation:

The original CDF rises sharply at the start, which reflects the abundance of darker pixels. There are less pixels in the highlight and mid-tone areas, as indicated by the curve's gradual plateauing as it increases in intensity. This shape is typical of a low contrast image, when most of the visual information is concentrated into a small area of dark values. The net effect is an image with poor visibility of details, especially in the darker areas

Equalized CDF Shape and Interpretation:

The CDF of the equalized image approaches a straight line, which suggests a more uniform distribution of pixel intensities. This linearity is a visual representation of the histogram equalization process, which redistributes the pixel intensities in the image.

After equalization, the CDF's more linear trajectory suggests a redistribution of pixel intensities that increases the visibility of details across the spectrum. Mid-tones and highlights that were previously compressed into a narrow band of values are now expanded, enhancing the textural and structural clarity of the image.

A1(e) Applying the algorithm twice

We can expand the application by posing an easy question now that we have applied the histogram equalization technique to an image and obtained an output image with greater contrast than the original. Can we use the histogram equalization procedure to enhance the contrast of the already-corrected image even more? Is it possible to use the algorithm periodically and raise the contrast indefinitely?

Technically, applying histogram equalization to an image that has already been equalized can have different effects, depending on the distribution of the pixel intensities after the first equalization. The answer to the question in this case is that there is no significant change. As we plot the PDF, CDF, and the output image after applying the algorithm again on the corrected image, applying histogram equalization twice does not have any significant impact on the outcome of the image.

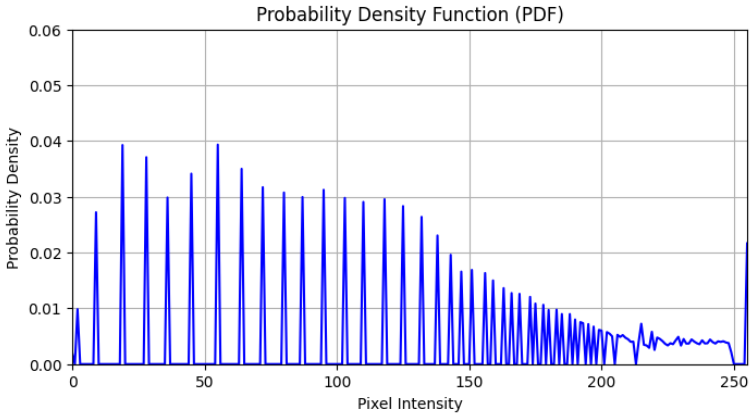


Figure 7: *PDF after applying the algorithm on corrected image*

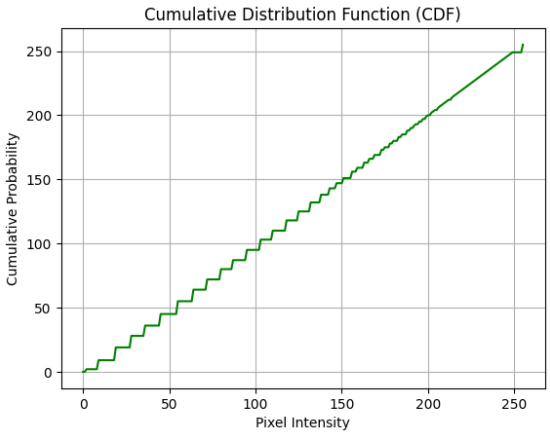


Figure 8: CDF after applying the algorithm on corrected image.

Figure 9: Image after applying histogram equalization twice.

A1(f) Applying Histogram Equalization to another image

Let's try applying histogram equalization to another low contrast image. The image under consideration is this:

Figure 10: Image under consideration

Now we plot the PDF, CDF and the histogram equalized image of the above image.

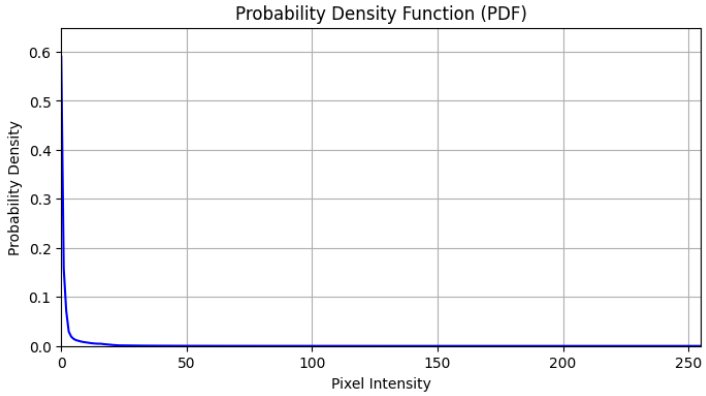


Figure 11: PDF of the image under consideration

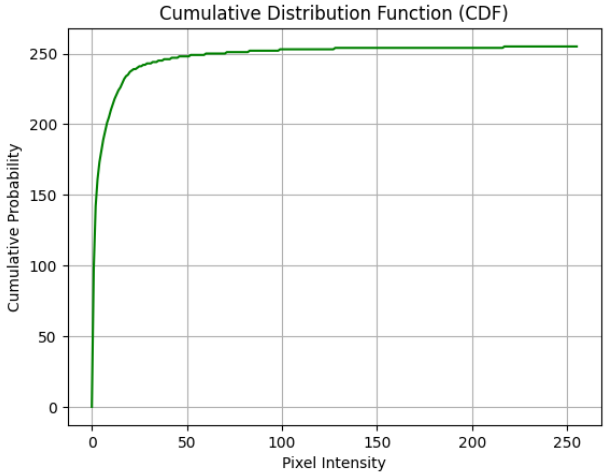


Figure 12: CDF of the image under consideration

Figure 13: Image under consideration after applying histogram equalization

We can see that we have achieved an increase in contrast when comparing the original image and the final image. This is evident from the PDF and CDF of the image after equalization, as shown below.

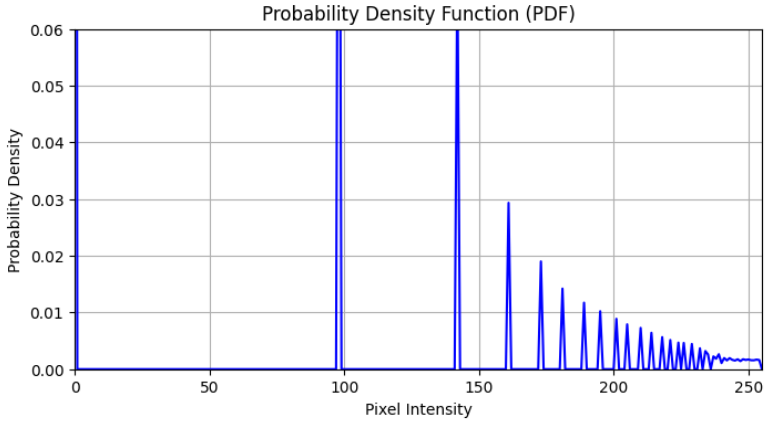


Figure 14: PDF of the image after histogram equalization

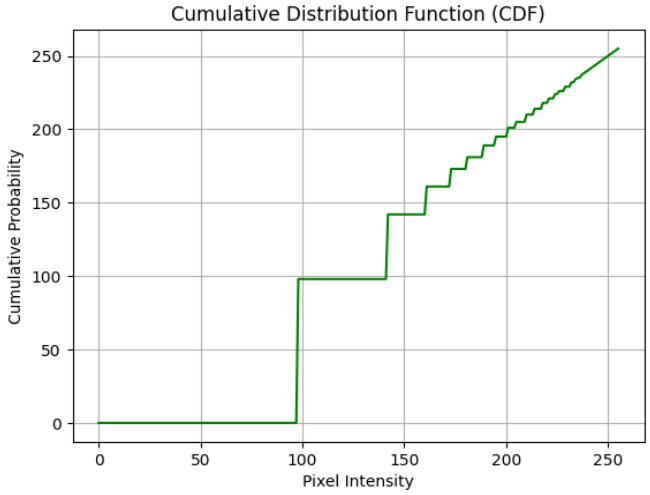


Figure 15: CDF of the image after histogram equalization

Discussion of Results

Original Image and PDF:

The original image appears quite dark, which suggests a lack of contrast, with most of the pixel values concentrated in the lower end of the intensity spectrum. This is supported by the PDF, which has a high peak at the darker end, indicating many pixels with low-intensity values.

The CDF for the original image rises sharply at the beginning and then flattens out. This shape indicates that a significant proportion of the image's pixels are dark, and there is a limited range of mid-tone and bright pixels.

After Histogram Equalization:

The equalized image has a significantly different appearance. It is much brighter, and details that were not visible in the original image are now somewhat visible. This indicates that the range of intensities are expanded therein in improving the contrast.

The PDF of the equalized image is much more spread out across the intensity range, which means that pixels have been remapped to use the full range of possible intensity values. The peaks of the PDF are more evenly distributed, which implies that the pixel intensities are more balanced.

The CDF after equalization has a more linear trend, which is an indicator of a more uniform distribution of pixel intensities. In an ideal scenario, the CDF would be a straight line, meaning that each intensity level is equally probable.

A2) Otsu Image thresholding

Introduction

Image thresholding stands as a fundamental process in the field of Computer Vision, serving as the most rudimentary form of image segmentation. This technique is pivotal for separating the foreground (objects of interest) from the background by converting grayscale images into binary images. The essence of image thresholding lies in its simplicity and effectiveness in object isolation, facilitating tasks such as object detection and image analysis. Among various thresholding methods, Manual Thresholding and Otsu's Thresholding are notably prominent. This report delves into the mechanics, applications, and comparative analysis of these two thresholding techniques.

Delving deeper into Manual Thresholding and Otsu's thresholding in detail would help us better understand the problem at hand.

A2.1) Manual Thresholding

Manual Thresholding is a straightforward approach where a predefined threshold value is employed to segment the image. This technique involves the following steps:

Conversion to Grayscale: Initially, the image is read and converted into a grayscale format to simplify the segmentation process.

Threshold Selection: A threshold value within the range of 0-255 is manually selected to differentiate the foreground from the background.

Pixel-wise Segmentation: The image is iterated pixel by pixel, applying the following criteria:

- Pixels with values greater than the threshold are set to a bright value, typically 255, to represent the foreground.
- Pixels with values less than the threshold are set to a dark value, usually 0, to denote the background.

Binary Image Output: The process yields a binary image distinctly highlighting the foreground against the dark background.

Let's use an example to understand the algorithm. Using three photos, we will apply the manual thresholding algorithm.

Figure 16: First image which is of size 256X256

Consider a 256x256 grayscale image (Figure 16). By selecting a threshold of 128, the image is effectively segmented into two regions: a bright foreground and a dark background, as demonstrated in Figure 17. This segmentation not only simplifies the image but also emphasizes the position and orientation of objects within.

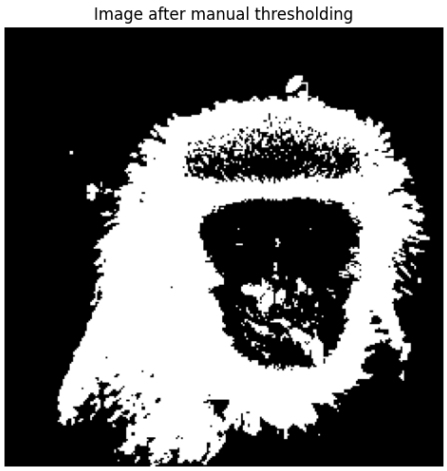


Figure 17: Image after applying manual threshold of 128

Following thresholding, the image is divided into two sections: the white area and the black area. The image displays the object's orientation and position as it appeared in the original picture.

Similar to the first image we can see that we observe the same thing when thresholding is applied to the second and third image, we get images segmented into two parts which is white and black.

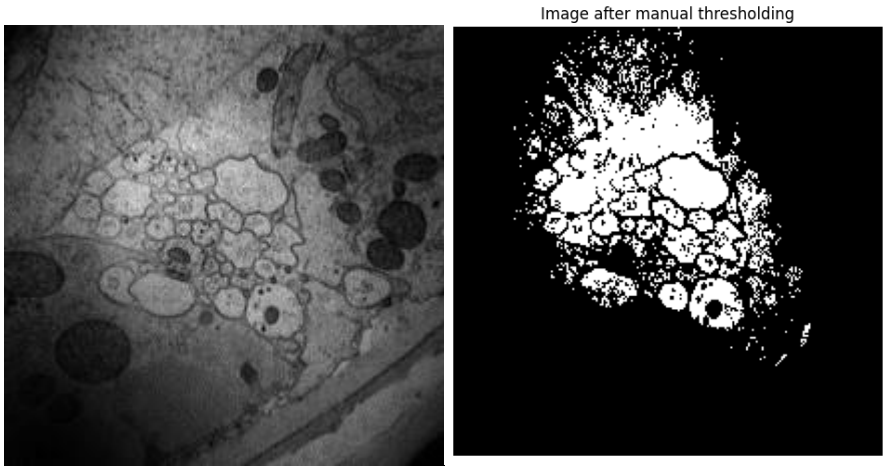


Figure 18: Image thresholding applied on the original microscopy image

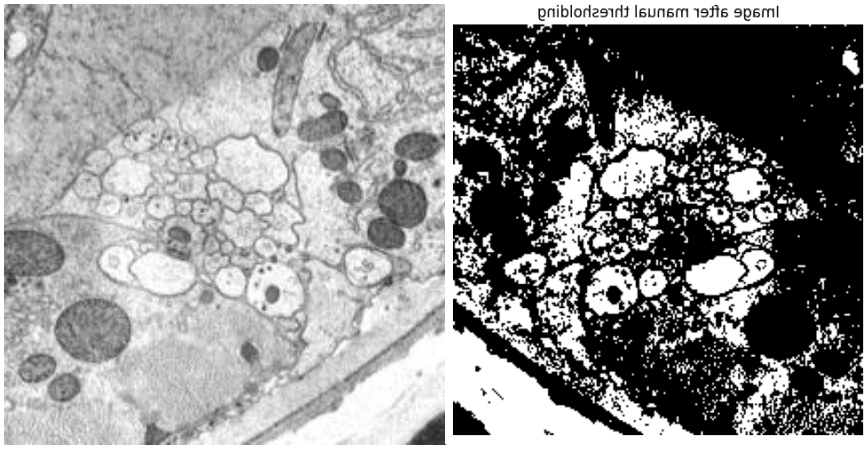


Figure 19: Image thresholding applied on the corrected microscopy image

The results of applying manual thresholding demonstrate its effectiveness in segmenting the foreground from the background, thereby allowing object detection within images. This method's simplicity and straightforward implementation make it accessible for quick segmentation tasks.

However, its reliance on manually selecting threshold values introduces flaws because this can lead to human error and would demand significant domain knowledge to determine the most appropriate threshold for each image. Such a method lacks reproducibility, as a threshold that is optimal for one image may not be suitable for another, underscoring the necessity for a more systematic approach to identify optimal threshold values across diverse set of images. This need has led to the development and adoption of algorithms like Otsu's method, which aim to automate and optimize the thresholding process.

A2.2) Otsu’s thresholding algorithm

Otsu's thresholding algorithm is a method for automatically determining the best threshold value to separate the objects of interest from the background in an image. This algorithm operates under the premise that an image's pixel intensities can be categorized into two groups: foreground and background. By finding a threshold that maximizes the distinction (inter-class variance) between these groups while minimizing the variance within each group (intra-class variance), Otsu's algorithm effectively segments the image.

The algorithm defines the inter-class variance as the weighted sum of the variances of each class (foreground and background). To calculate this, the algorithm first determines the probabilities (weights) and the mean intensity values for each class. These probabilities, w0 and w1, represent the foreground and background respectively, and are calculated by summing up the pixel distribution function (PDF) values up to the threshold t for the foreground, and from t to the maximum intensity level L−1 for the background:

$$w0 = \sum_{i=0}^{t-1} p(i)$$

$$w1 = \sum_{i=t}^{L-1} p(i)$$

where p(i) is the PDF value for intensity level i, and t is the threshold being evaluated.

The means of the pixel values for the foreground and background, denoted as μ0 and μ1 respectively, are computed by weighing the intensity levels with their respective probabilities:

$$\mu 0 = \frac{\sum_{i=0}^{t-1} i * p(i)}{w0}$$

$$\mu 1 = \frac{\sum_{i=t}^{L-1} i * p(i)}{w1}$$

With the class probabilities and mean values determined, the inter-class variance (σ) is calculated using the formula:

$\sigma = w0 * w1 * (\mu 0 - \mu 1)^2$

The algorithm's objective is to identify the threshold t that maximizes this inter-class variance. The selected threshold is then utilized to segment the image into foreground and background, facilitating the isolation of the objects of interest.

The Algorithm

1. Read Image: Load the greyscale image into an array.

2.Compute PDF: Calculate the Probability Density Function (PDF), equivalent to the normalized histogram of the image.

3.Threshold Iteration: For each threshold value in the range 0-255, perform the following steps:

- **Calculate Probabilities**: Determine the probabilities of the foreground (pixels brighter than the threshold) and background (pixels darker than the threshold).
- **Calculate Means:** Compute the means of the foreground and background pixel values.
- **Inter-Class Variance**: Calculate the inter-class variance based on the foreground and background probabilities and means.
- **Maximize Variance:** If the calculated inter-class variance is higher than any previous variance, update the maximum variance and record the corresponding threshold.

4.Apply Threshold: Use the threshold associated with the maximum variance to convert the image into a binary format.

5.Comparison: Compare the optimal threshold value obtained through this method with the threshold value provided by OpenCV's function.

A2(a) Show the histograms for each image

The first task at hand is to display the histograms for the given image so we basically understand the PDF of the images before applying Otsu Thresholding to the given input images below:

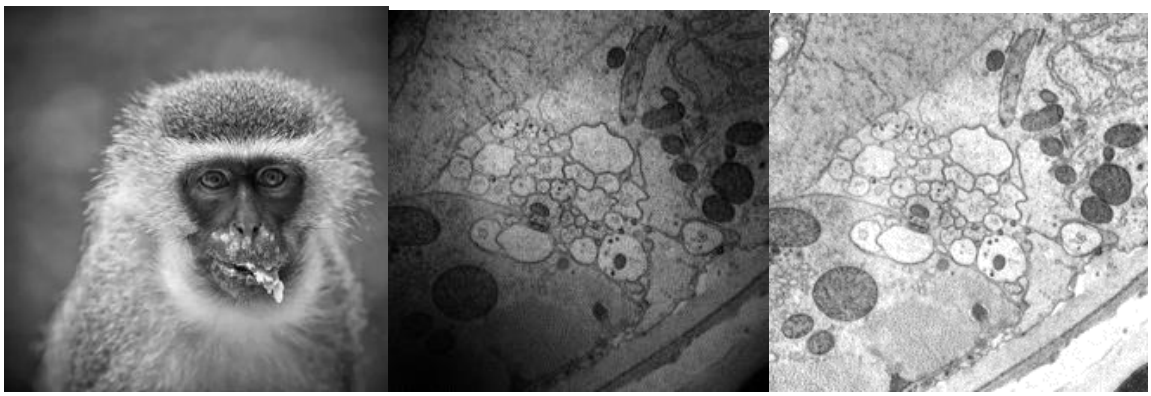


Figure 20: The images under consideration

The first image depicts a grayscale close-up of a monkey, which appears to have a wide range of grey, suggesting good contrast suitable for thresholding. It has significant textural detail, especially in the fur, which may result in a complex histogram and potentially a more challenging thresholding process.

The second and third images are grayscale micrographs with varying degrees of brightness and contrast. They contain a mix of high-density areas (darker regions) and low-density regions (lighter regions), which could result in multi-modal histograms. This characteristic might affect the selection of an optimal threshold value, as there may be more than one peak in the inter-class variance.

These properties of the input images are crucial for the Otsu thresholding process as they influence the histogram's shape and the thresholding outcome.

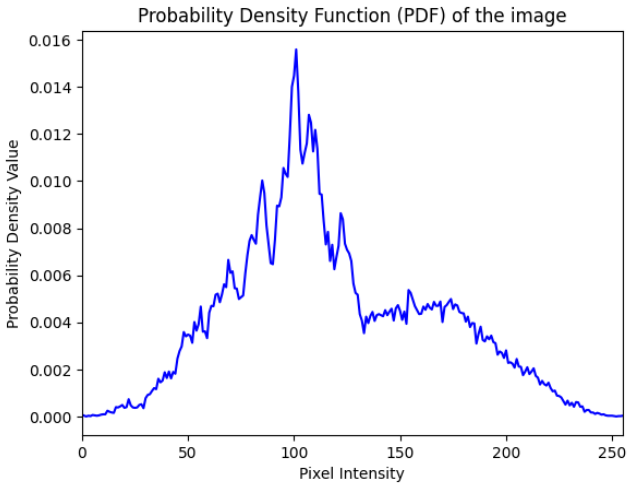


Figure 21: PDF plot of the image1

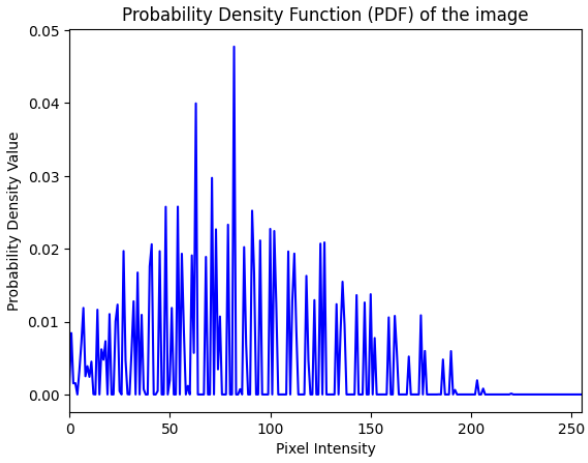


Figure 22: PDF plot of the image2

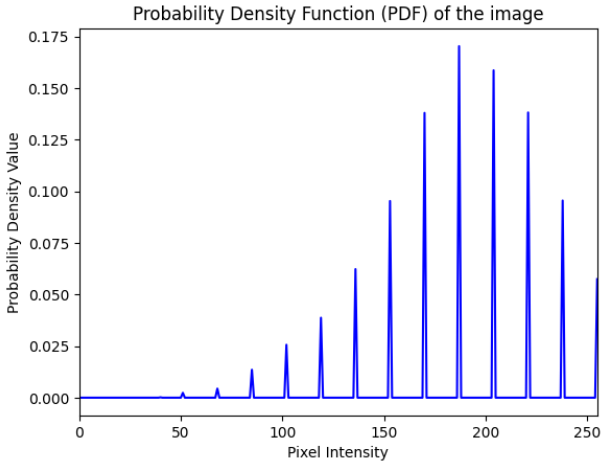


Figure 23: PDF plot of the image3

The Probability Density Function (PDF) plots here depict the normalized frequency of pixel intensities in the images which are given as the input for the process of Otsu thresholding.

1. **Image 1 (PDF):** The bimodal nature suggests that the image probably has two predominant groups of pixel intensities, likely corresponding to the object of interest and the background of the image.

2. **Image 2 (PDF):** This PDF plot is more complex, with multiple peaks at various intensity levels which basically suggests that this image is more textured and has multiple distinct regions.

3. **Image 3 (PDF):** This plot indicates a highly contrasted image with stark differences in pixel intensities, and this is observe because of its sharp peaks. Such an histogram is usually irregular and has multimodal distribution which means that it has varying levels of brightness

A2(b) Show the Inter Class Variance Plot for each image

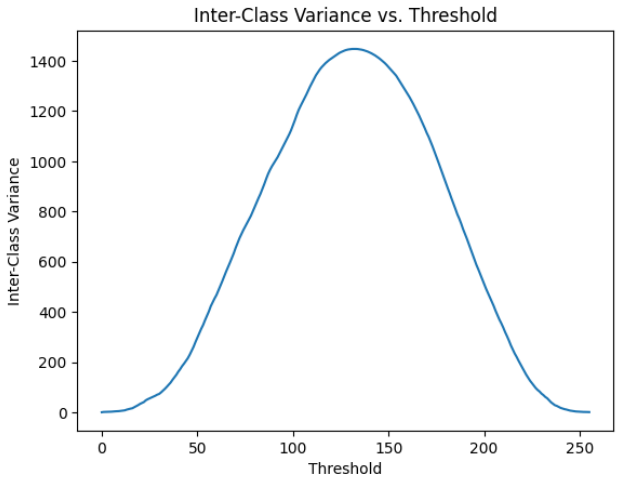


Figure 24: Plot of inter-class variances vs threshold for the image

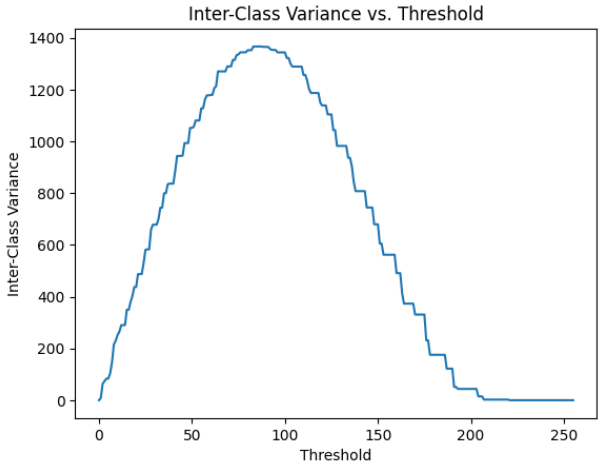


Figure 25: PDF and interclass variance vs threshold plots for the image

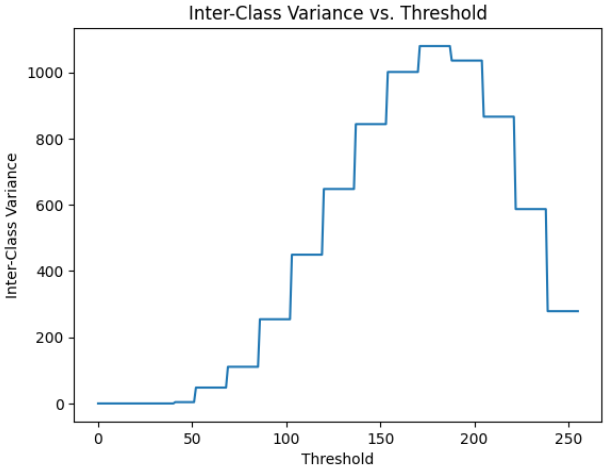


Figure 26: PDF and interclass variance vs threshold plots for the image

The inter-class variance is a key element in Otsu's thresholding algorithm. It represents how the variance between the foreground and background as different threshold values are tested by the algorithm.

1. **Image 1 (Variance Plot):** This inter class variance plot indicates that the threshold value can be observed the peak of the curve where the variance is maximized, which is the optimal point for segmenting the image. In this plot, the peak is quite pronounced and is located around the middle of the scale, suggesting a clear distinction between the two classes in the image at that threshold. This is indicative of an image with well-defined contrast between the object of interest and the background, making it ideal for Otsu's method.

2. Image 2 (Variance Plot): This inter-class variance plot indicates the effectiveness of different thresholds for segmenting an image into foreground and background. The curve's peak suggests the threshold value that maximizes the between-class variance, which is ideal for segmentation. The plot shows a clear peak, which means there's a specific threshold value where the foreground and background are distinctly separated, which should result in a good binary image after applying Otsu's thresholding.

3. Image 3 (Variance Plot): This inter-class variance plot features a plateau rather than a single peak, which indicates multiple threshold levels that could be considered optimal for segmenting the image. This often occurs in images with several regions of interest that have similar levels of variance when separated from the background. Choosing the best threshold in such a scenario may require additional criteria or a multi-threshold approach to effectively distinguish between different objects or features within the image.

A2 (c) State Inter-Class Variance for each image

Inter-class variance of the image upon completion of the algorithm: -

1. Image One: Inter-class variance: 1448.01
2. Image Two: Inter-class variance: 1366.64
3. Image Three: Inter-class variance: 1079.708

A2 (d) Intensity Chosen by the algorithm

Intensity threshold chosen by the algorithm:

1. Image One: Threshold: 132
2. Image Two: Threshold: 86
3. Image Three Threshold: 171

A2 (e) Resulting Binary image produced

Figure 27: Image segmented using the threshold value of 131.

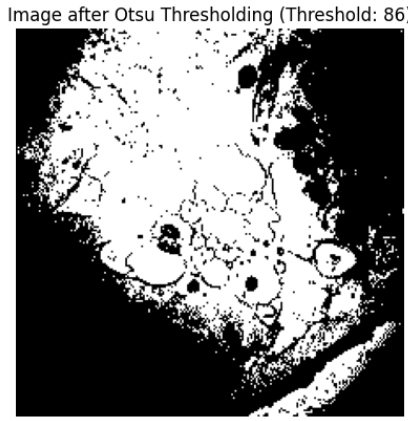


Figure 28: Image segmented by applying Otsu threshold value of 85

Figure 29: Image segmented by applying Otsu threshold value of 88

The results of Otsu's thresholding technique can be discussed in depth by analyzing the binary images in relation to the inter-class variance plots and the chosen threshold values:

1. **Image with Threshold 132**: The inter-class variance plot for this image showed a distinct peak, which indicates a clear threshold for optimal separation. The image has a threshold of 132, the subject appears to be well isolated from the background, suggesting a successful segmentation. This segmentation result shows a strong differentiation between the foreground subject and the background. The threshold of 132 has been determined as the optimal point where the sum of background and foreground pixel intensities has the greatest separation. This has allowed for prominent features of the subject to stand out sharply against the background, which is now mostly black. The stark contrast is indicative of the algorithm effectively distinguishing between the high-intensity values of the subject and the lower intensities of the background.

2. **Image with Threshold 86:** A lower threshold value suggests the peak in the inter-class variance plot was towards the darker intensity values which is true. This has therefore resulted in more detailed extraction of features within the image. However, these features can also be misconstrued based on the inclusion of background noise, indicating that while the algorithm has tried to preserve detail, it may have sacrificed some specificity.

The resulting image captures a greater range of details, as more of the lower-intensity pixels are considered foreground. This can be beneficial for highlighting subtle features but may also incorporate noise — extraneous information that does not represent the subject of interest.

3. **Image with Threshold 171**: A higher threshold suggests the inter-class variance peak was towards the lighter side of the intensity spectrum. This often results in a loss of detail within darker areas but ensures a cleaner separation where the contrast with the background is high.

The segmentation produced by this threshold likely resulted in a binary image where only the brightest regions are classified as the foreground. This can create a clean separation where there is a strong contrast but at the risk of losing important details in less illuminated areas. For some applications, this might be acceptable, but for others, where detail is critical, adjustments to the threshold or additional processing might be needed to retain essential information.

A3(f) Discussion of Results

Photo1:

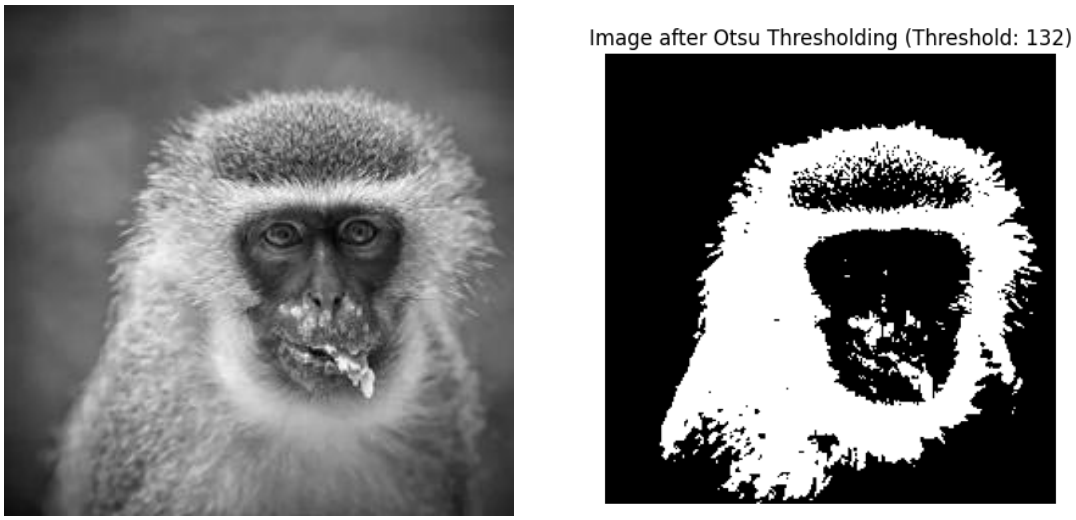


Figure 30: Original Image vs Threshold Image

Photo2:

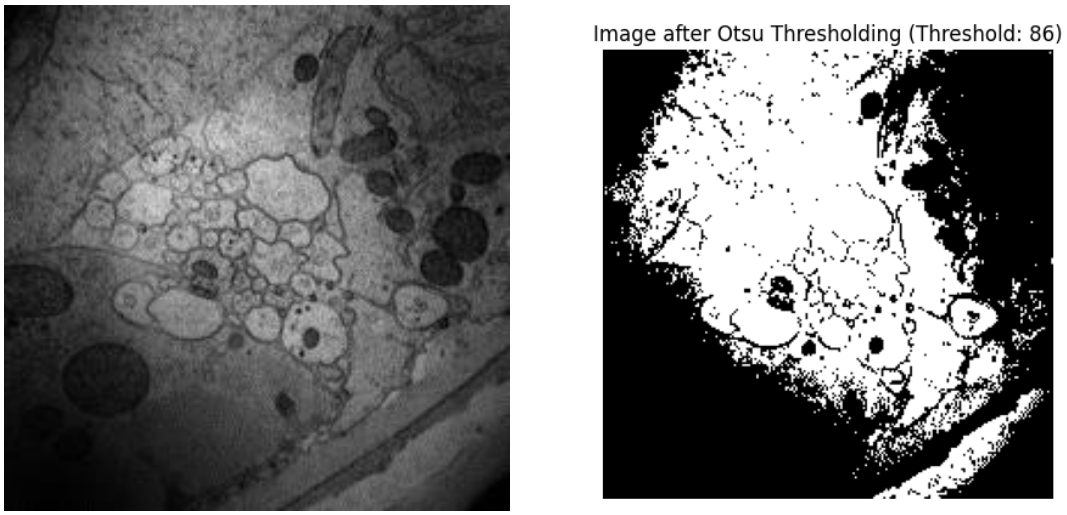


Figure 31: Original Image vs Threshold Image

Photo3:

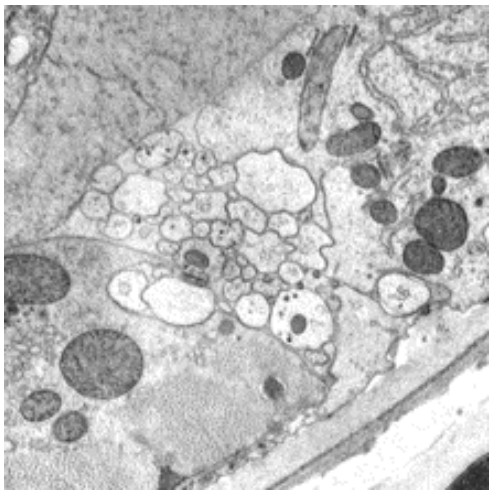


Figure 32: Original Image vs Threshold Image

The automatic threshold results using Otsu's method appear to provide decent segmentation in some cases for the given input images:

1. **For the Image with Threshold 132**: The segmentation seems effective, isolating the subject clearly from the background. The main elements of the image, likely the subject's details, are segmented and suggest that Otsu's algorithm was able to find a threshold that adequately segregates the foreground and background. The binary image shows a strong contrast between the monkey and the background. The main body of the monkey is clearly distinguishable from the background.

Even though the main shape of the monkey is apparent, there could be improvements. The details within the face and the lighter areas of the fur are lost, and considered as a part of the background. Adjusting the thresholding algorithm to preserve more detail, applying smoothing filters or applying different image processing techniques to highlight these areas could improve the result.

2. **For the Image with Threshold 86**: The elements that appear to be separated from each other are likely the cellular or structural elements against the background. It seems that the algorithm was able to distinguish between the denser components (potentially cellular structures) and the lighter background. The lower threshold has highlighted finer details, which is good for capturing more subtle features of the image and this also means that the threshold peaked at a range indicating more dark intensities in original image.

The place where algorithm fails to produce a decent result is where the image has more than two distinct intensities, which the bi-modal assumption of Otsu's method cannot handle effectively. Otsu's method might not be able to find a threshold that accurately separates these objects from the background. Thus, if you notice clearly there are some areas where the image has not successfully differentiated between foreground and background.

Also, if you notice the algorithm included some noise and considered it as objects while looking at the subtle features.

If the noise is misclassifying background pixels as part of the foreground, pre-processing steps such as noise reduction or applying a median filter could improve the result.

3. **For the Image with Threshold 171:** The binary image after applying the Otsu thresholding seems to show a distinction between the darker and lighter areas, which suggests that cellular structures are being separated from the background to a certain extent. The threshold appears to have been successful in identifying the dense cellular regions and the lighter regions. The high threshold has provided a clean separation for high-contrast areas because the input image was basically brightened and this might have led the algorithm to exclude certain details in the darker regions of the image.

The algorithm may have failed partially because it seems to have included a lot of noise (white noise) in the binary image. The threshold value of 171 may not be optimal for this image, as it has resulted in fragmented cellular structures and potentially significant information loss. This occurs due to Otsu’s bimodal representation and the image having multiple grayscale values.

There might be improvements to be made in terms of reducing the noise and fine-tuning the threshold to better capture the edges of the structures. Post-processing techniques could be used to clean up the image and improve the segmentation.

A3): Histogram Matching and Its Application

Histogram matching, also known as histogram specification, is an advanced process which aims to transform the histogram of a source image so that it matches the histogram of a specified target image. Unlike histogram equalization, which seeks to flatten the histogram, thereby enhancing the overall contrast of the image, histogram matching is designed to adjust the pixel distribution of an image to match a particular reference.

The essence of histogram matching lies in its capacity to standardize images based on a desired reference, making it particularly useful in scenarios where consistent image quality and appearance are necessary.

Methodology

To explore the practicality and impact of histogram matching, we implement this technique using multiple python libraries and histogram matching code. The process involved the following steps:

1. Source and Target Image Selection: Here we start by selecting a source image that requires contrast adjustment. Correspondingly, for each source image, a target image with an optimal contrast profile is chosen. The target images are selected based on two criteria:

- Similar scene content: Target images that shared content characteristics with the source images.
- Different scene content: Target images with completely different content to assess the method.

2. Histogram Matching Implementation:

1. **Histogram Computation:** Calculate the histograms for the source image and for the target image by adding the pixel intensities.

2. **Histogram Normalization:** Normalize the histograms to create probability distributions (PDF), which are then used to compute the cumulative distribution functions (CDFs).

3. **CDF Generation:** Generate the CDF for both images by summing the normalized histogram values (PDF). This step essentially maps the intensity levels in the images to a cumulative probability of occurrence.

4. **Mapping Function Construction:** Construct a mapping function by associating each intensity level in the PDF of source image with an intensity level in the PDF of matching image such that the CDF values are as close as possible, minimizing the difference between the two histograms.

5. **Intensity Transformation:** Apply the mapping function to the source image, altering each pixel's intensity to the corresponding intensity in the target image's distribution. The final image is synthesized with the transformed pixel values, resulting in an image whose histogram matches the target histogram.

Results and Discussion

The following figures present a visual comparison of the original and matched images, along with their respective histograms:

(a.)Similar Scene Content

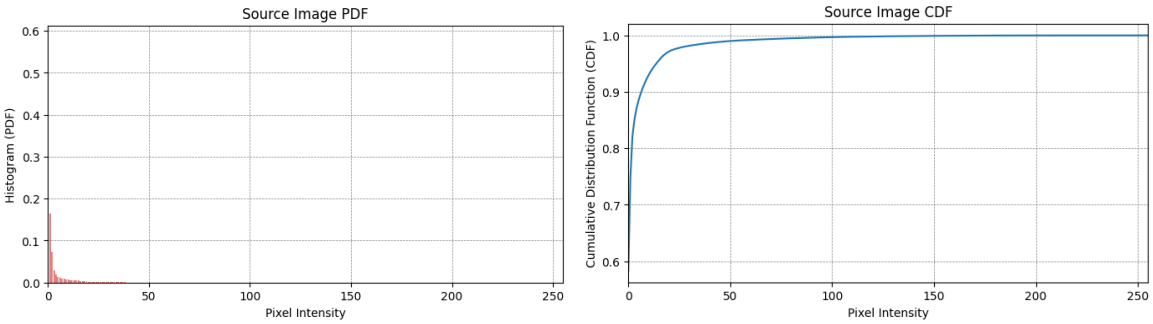


Figure 33: Source Image PDF and CDF(Histogram)

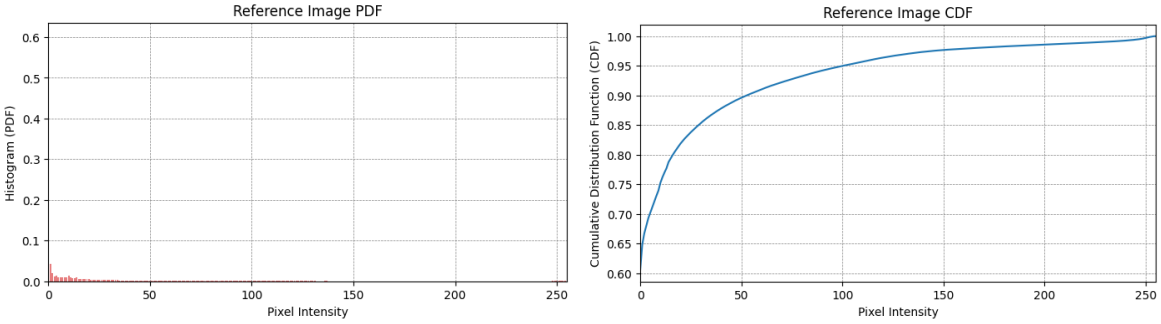


Figure 34: Reference Image PDF and CDF(Histogram)

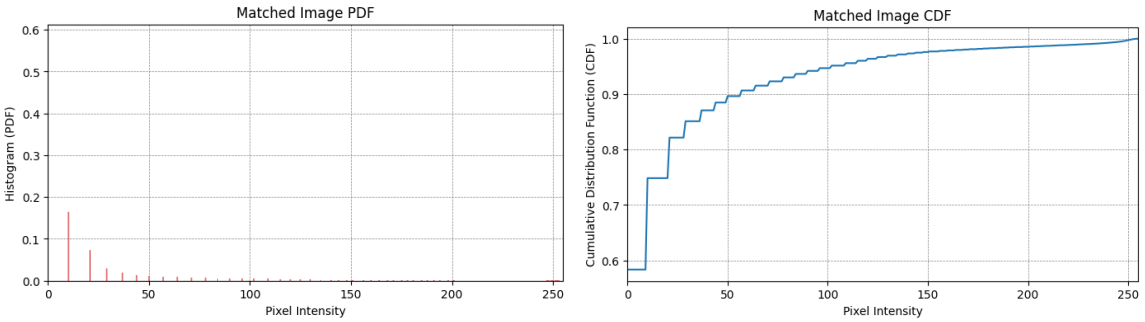


Figure 35: Matched Image PDF and CDF(Histogram)

Figure 36: Representation of Source, Reference and Matched

(b.)Two Different Scenes

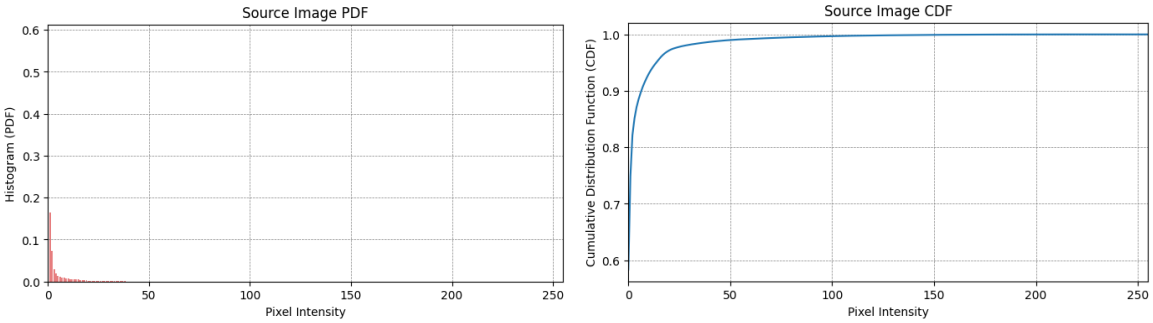


Figure 37: Source Image PDF and CDF(Histogram)

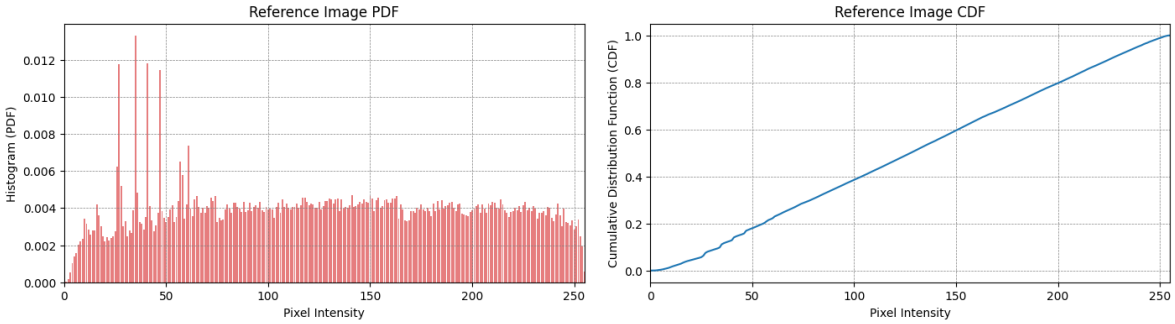


Figure 38: Reference Image PDF and CDF(Histogram)

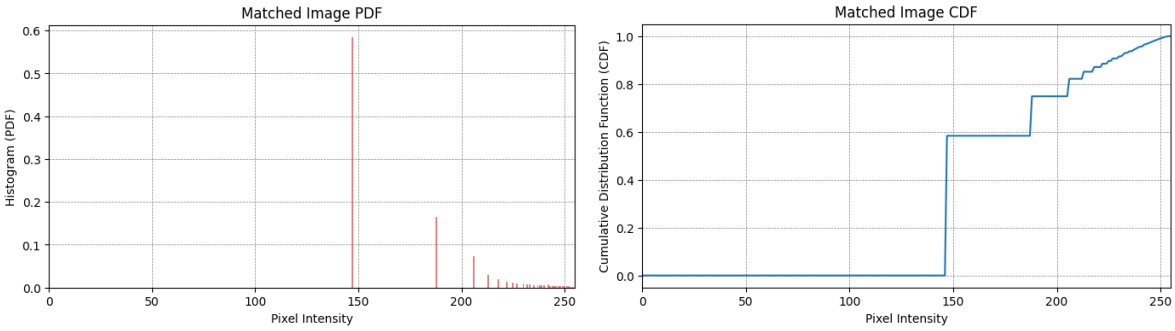


Figure 39: Matched Image PDF and CDF(Histogram)

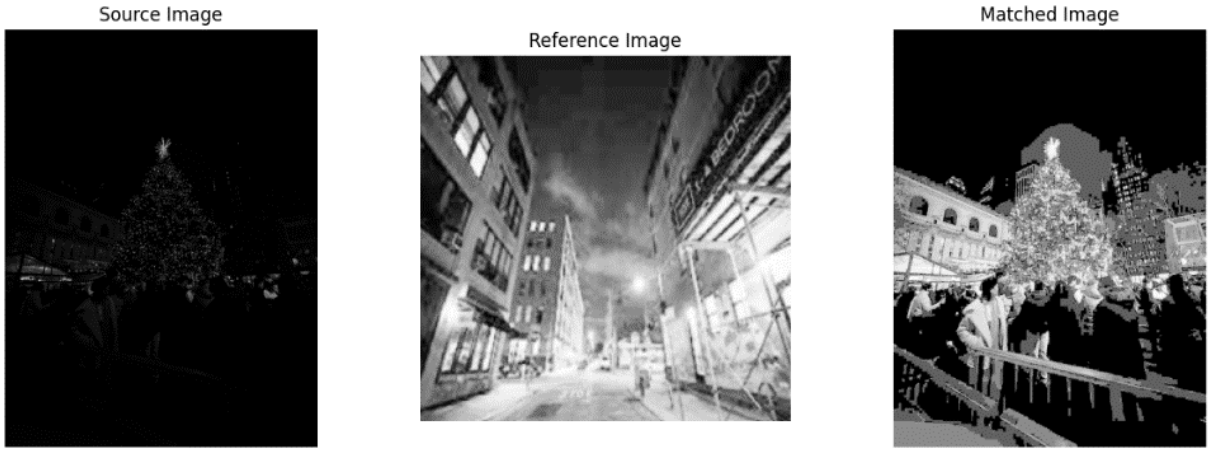


Figure 40: Representation of Source, Reference and Matched

The images showcase the transformation from the source to the matched image, highlighting the adaptation of the histogram. It is observed that when the target image contains similar scene content, the histogram matching yields visually coherent results with a natural appearance. Conversely, when the target image's scene content changes, the matched image retains the structure of original image but displays a high contrast as compared to before.

Conclusion

The conducted experiments underscore the ability of histogram matching in achieving desired contrast characteristics in digital images. The method's efficacy is highly dependent on the selection of a suitable target image, which has a major impact on the visual outcome.

Same Images for Histogram Matching:

In the first set, the same image is used as both the source and the reference. The result of histogram matching in such a case might be trivial, as the source and reference histograms are already the same, thus the matched image will be very similar or identical to the source and reference images. The PDFs (Probability Density Functions) and CDFs (Cumulative Distribution Functions) of the source and reference will overlap, and the matched image's histogram will align with them as well.

Different Images for Histogram Matching:

In the second set, different images are used as the source and reference. In this case, the purpose of histogram matching is transforming the pixel values of the source image so that its histogram aligns with that of the reference image. This is done by calculating a mapping function from the source's CDF to the reference's CDF. The matched image will have a similar distribution of intensities as the reference image, which may result in the source image adopting the overall "look" in terms of brightness and contrast of the reference image.

Here we can clearly see that the original image that has been subjected to histogram matching, appears quite dark with most of its details not clearly visible, suggesting a limited dynamic range and a concentration of pixel values in the darker end of the intensity spectrum. Now the reference image used for matching is completely different and has a wider dynamic range as indicated by the brighter appearance and more varied intensity levels.

Applying matching here results in a matched image that displays a significantly increased dynamic range, evident from the brighter and more visual details. The lighting in the scene appears more balanced, and features that were previously hidden in the shadows of the source image are now visible. The transformation has introduced visual elements from the reference image's intensity distribution, such as the overall brightness and contrast levels.

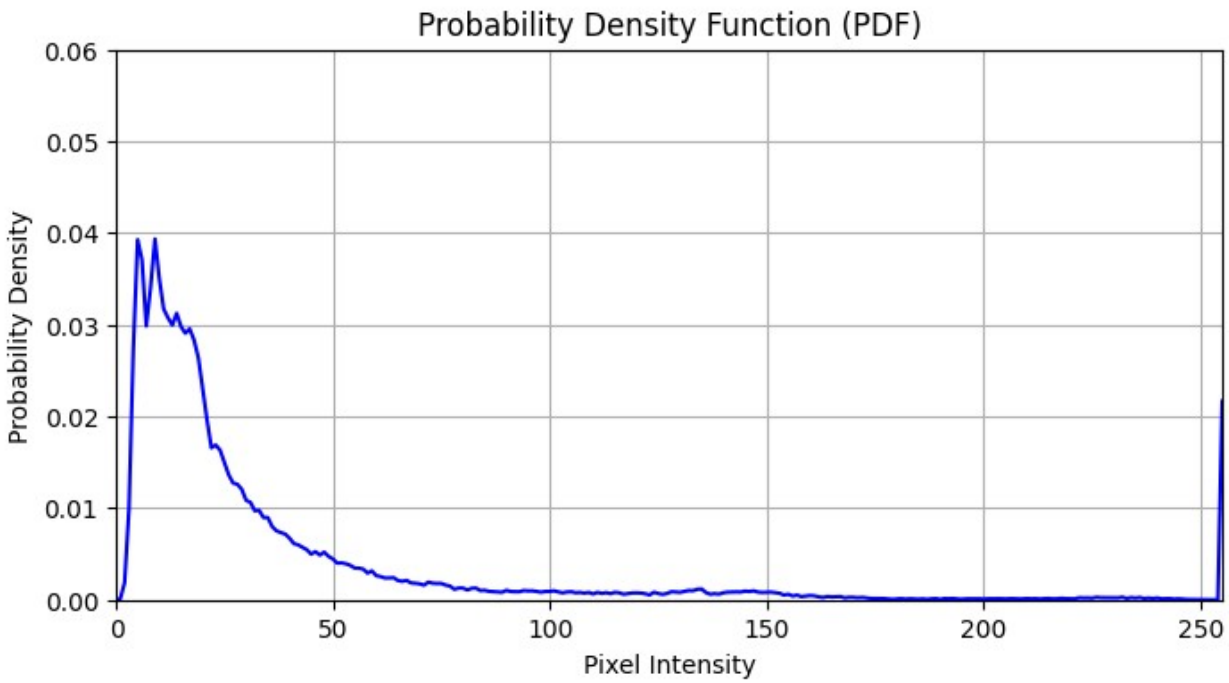
4.) PYTHON Code

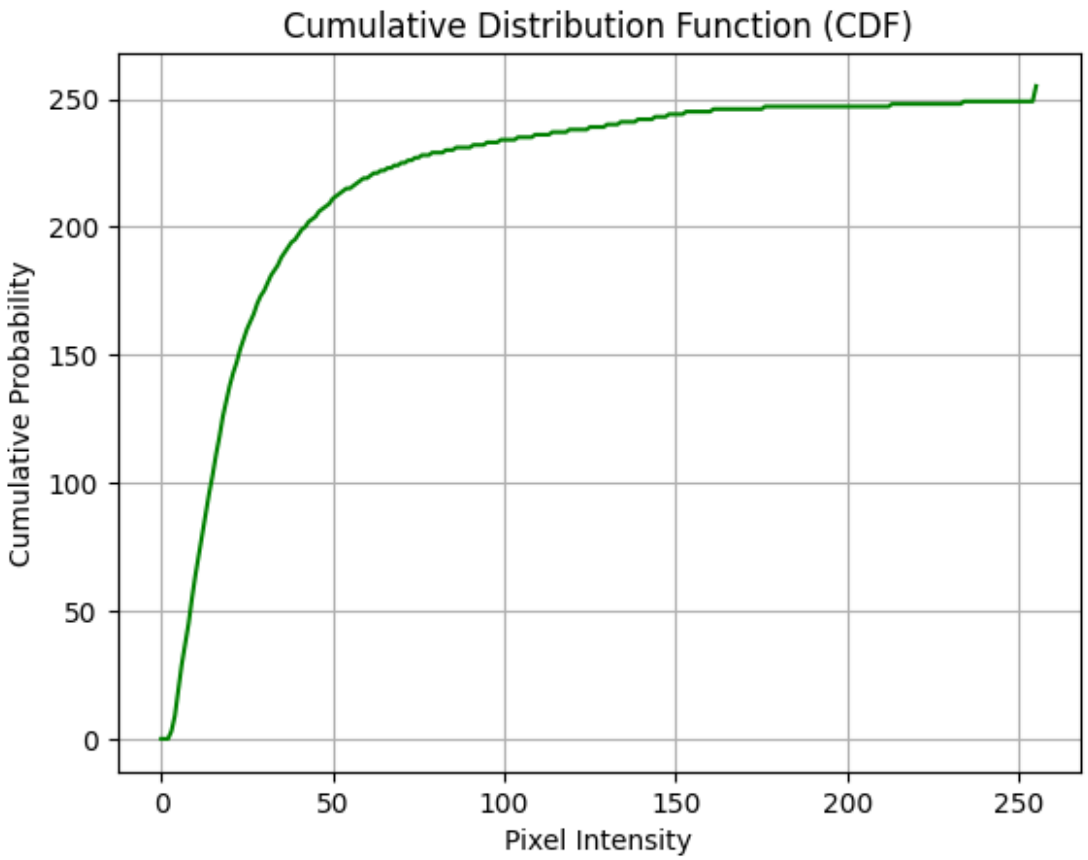
Note:

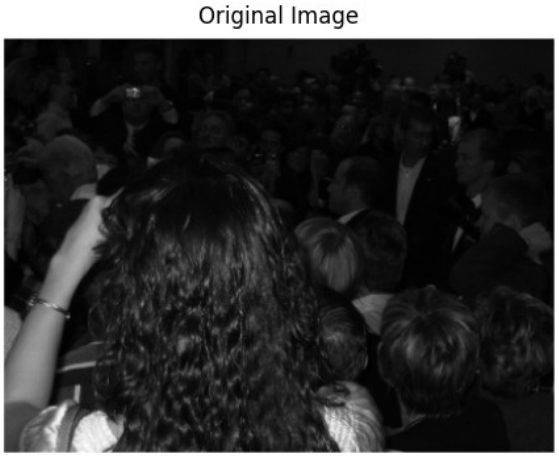
Here is the GitHub Link to all the python code and input images.

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

def create_pdf(im_in):#Function to create the pdf using the image
input using normalized pdf formula
    histogram = np.zeros(256)
    for pixel in im_in:
        histogram[pixel] += 1
    total_pixels = im_in.size
    pdf = histogram / total_pixels
    return pdf

def create_cdf(pdf):#Function to create CDF which basically a
cumulitive sum of pdf values
    cdf = np.zeros_like(pdf)
    cdf[0] = pdf[0]
    for i in range(1, len(pdf)):
      cdf[i] = cdf[i-1] + pdf[i]
    return cdf

def histogram_equalization(im_in):# Histogram equalization function
that equalizes an input image to enhance contrast , returning the
equalized image, its original PDF, and normalized CDF.
    im_array = np.array(im_in)
    flat_im_array = im_array.flatten()
    pdf = create_pdf(flat_im_array)
    cdf = create_cdf(pdf)
    cdf_normalized = (cdf - cdf.min()) * 255 / (cdf.max() - cdf.min())
    cdf_normalized = cdf_normalized.astype('uint8')
    im_out = cdf_normalized[flat_im_array]
    im_out = np.reshape(im_out, im_array.shape)
    return im_out, pdf, cdf_normalized

def plot_pdf(pdf):# Function to plot the pdf
    plt.figure(figsize=(8, 4))
    plt.plot(pdf, color='blue')
    plt.title('Probability Density Function (PDF)')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Probability Density')
    plt.xlim(0, 255)
    plt.ylim(0, 0.06)
    plt.grid(True)
    plt.show()

def plot_cdf(cdf):#function to plot the cdf
    plt.plot(cdf, color='green')
    plt.title('Cumulative Distribution Function (CDF)')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Cumulative Probability')
```

```
    plt.grid(True)
    plt.show()

img_path = '/content/people.png'
img = Image.open(img_path).convert('L')

equalized_img, pdf, cdf = histogram_equalization(img)

plot_pdf(pdf)
plot_cdf(cdf)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(equalized_img, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

plt.show()
```





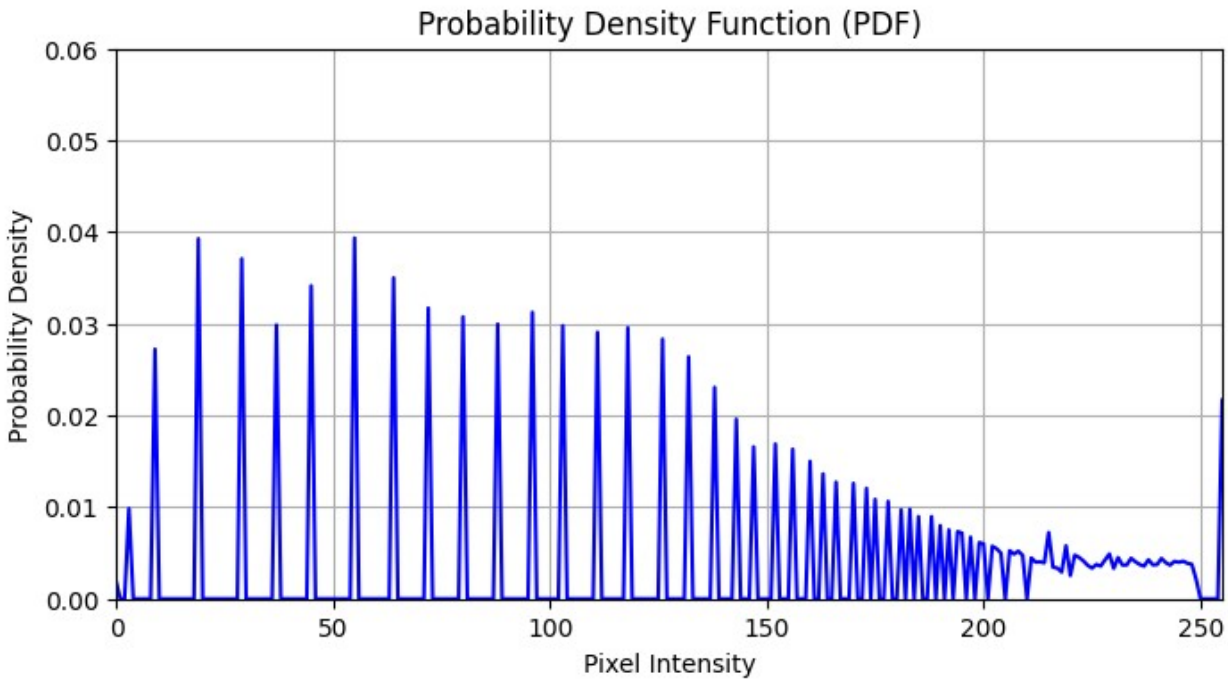


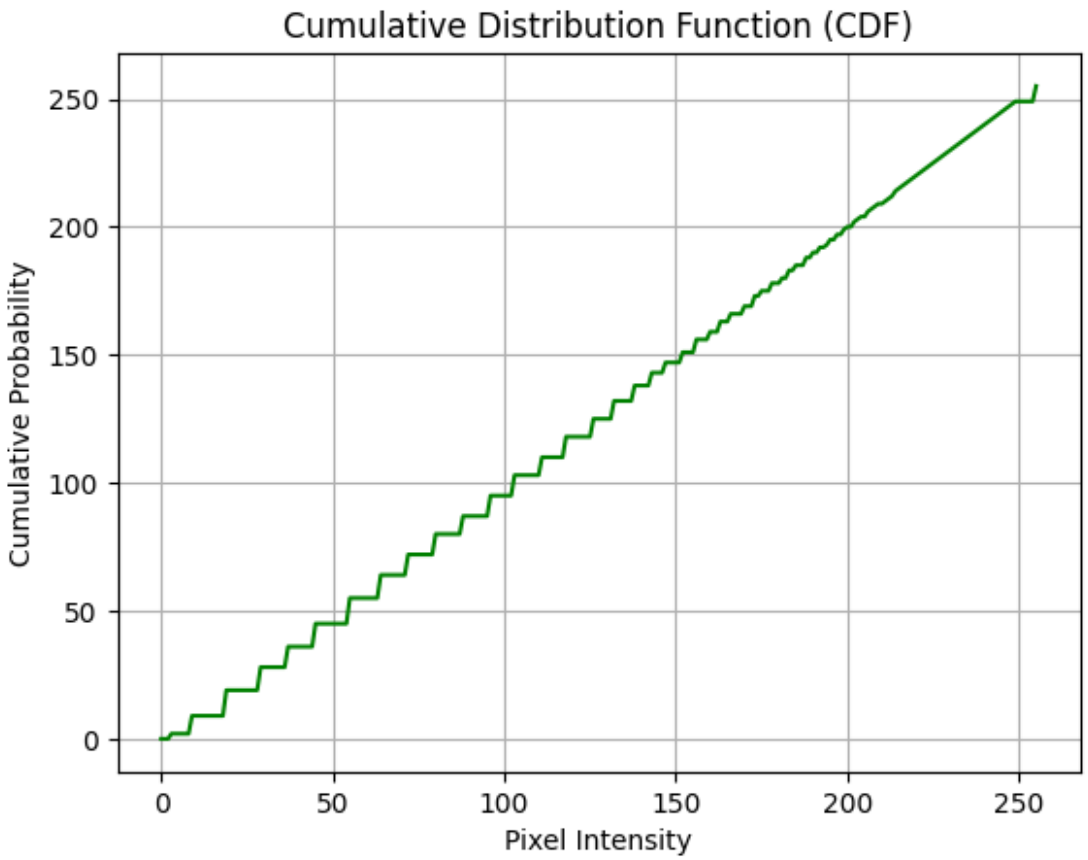

```
from PIL import Image
import numpy as np

# Python code to save the output image to again performm an action on
it
equalized_image = Image.fromarray(equalized_img.astype(np.uint8))
output_path = '/content/equalized_image.png'
```

```
equalized_image.save(output_path)

print(f"Image saved to {output_path}")
```

```
Image saved to /content/equalized_image.png
```

```
img_path = '/content/equalized_image.png'
img = Image.open(img_path).convert('L')

equalized_img, pdf, cdf = histogram_equalization(img)

plot_pdf(pdf)
plot_cdf(cdf)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(equalized_img, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

plt.show()
```



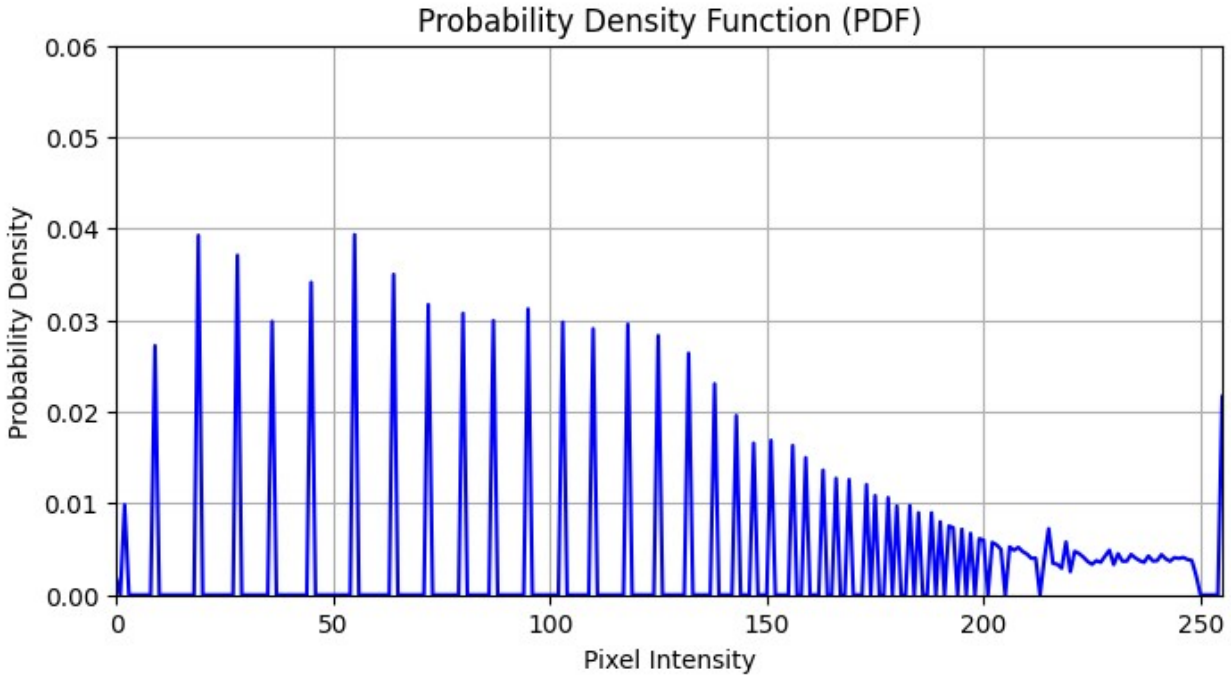


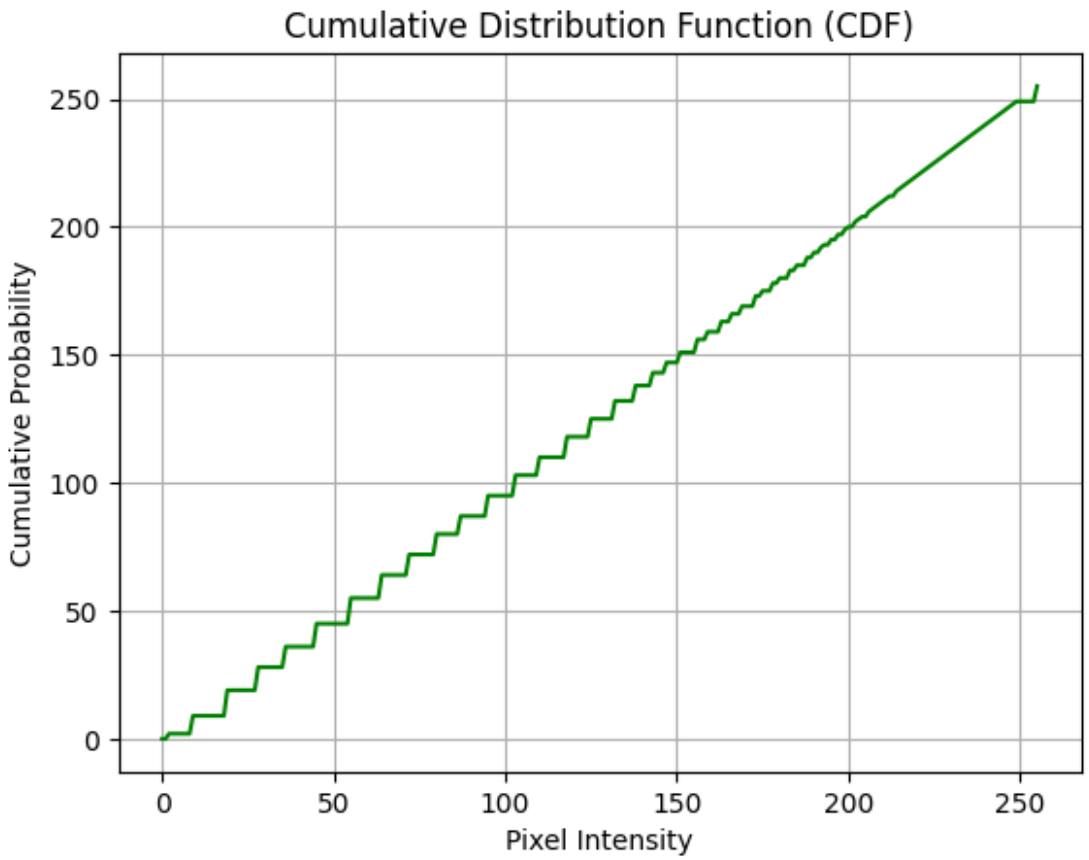

```
from PIL import Image
import numpy as np

# Python code to save the output image to again performm an action on
it
equalized_image_pil = Image.fromarray(equalized_img.astype(np.uint8))

output_path = '/content/equalized_image2.png'
```

```
equalized_image_pil.save(output_path)

print(f"Image saved to {output_path}")
```

```
Image saved to /content/equalized_image2.png
```

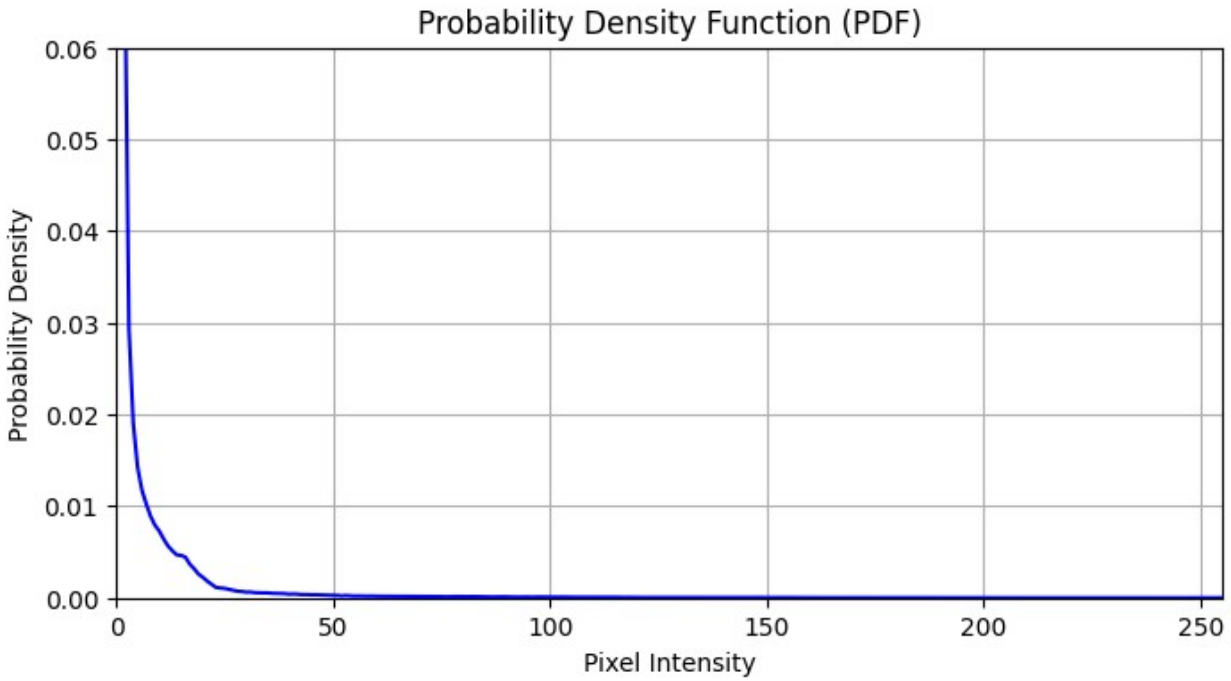
```
img_path = '/content/equalized_image2.png'
img = Image.open(img_path).convert('L')

equalized_img, pdf, cdf = histogram_equalization(img)

plot_pdf(pdf)
plot_cdf(cdf)
```

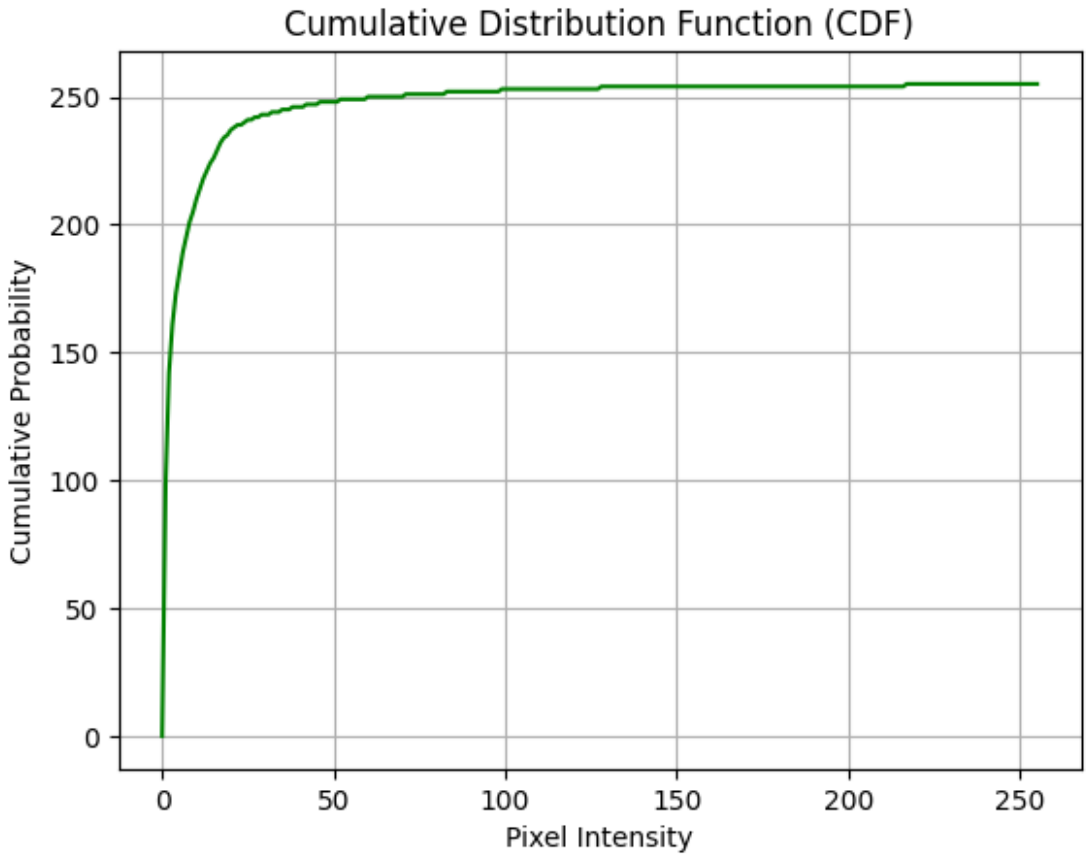





```
uploaded_img_path = '/content/treec.jpg'
img = Image.open(uploaded_img_path)

img_gray = img.convert('L')
img_array = np.array(img_gray)

plt.imshow(img_array, cmap='gray')
plt.title('Grayscale Image')
plt.axis('off')
plt.show()

output_path = '/content/grayscale_image.jpg'
img_gray.save(output_path)
```


```
img_path = '/content/grayscale_image.jpg'
img = Image.open(img_path).convert('L')

equalized_img, pdf, cdf = histogram_equalization(img)

plot_pdf(pdf)
plot_cdf(cdf)
```






```
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
```

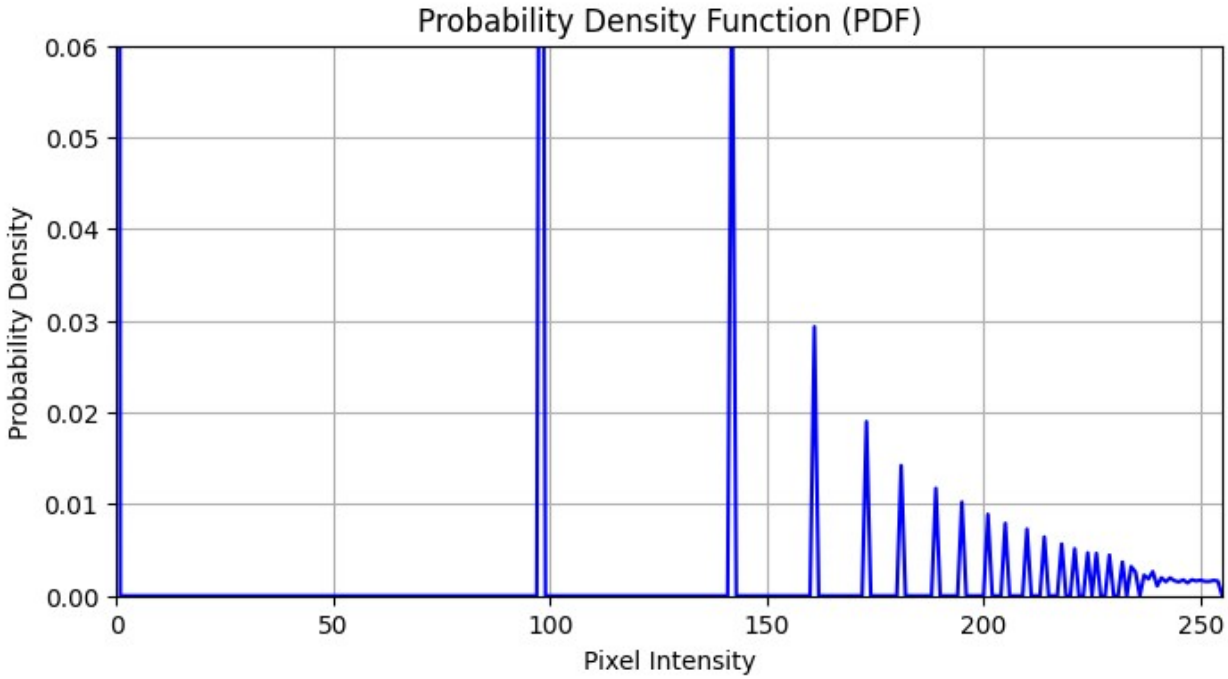
```
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(equalized_img, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

plt.show()
```

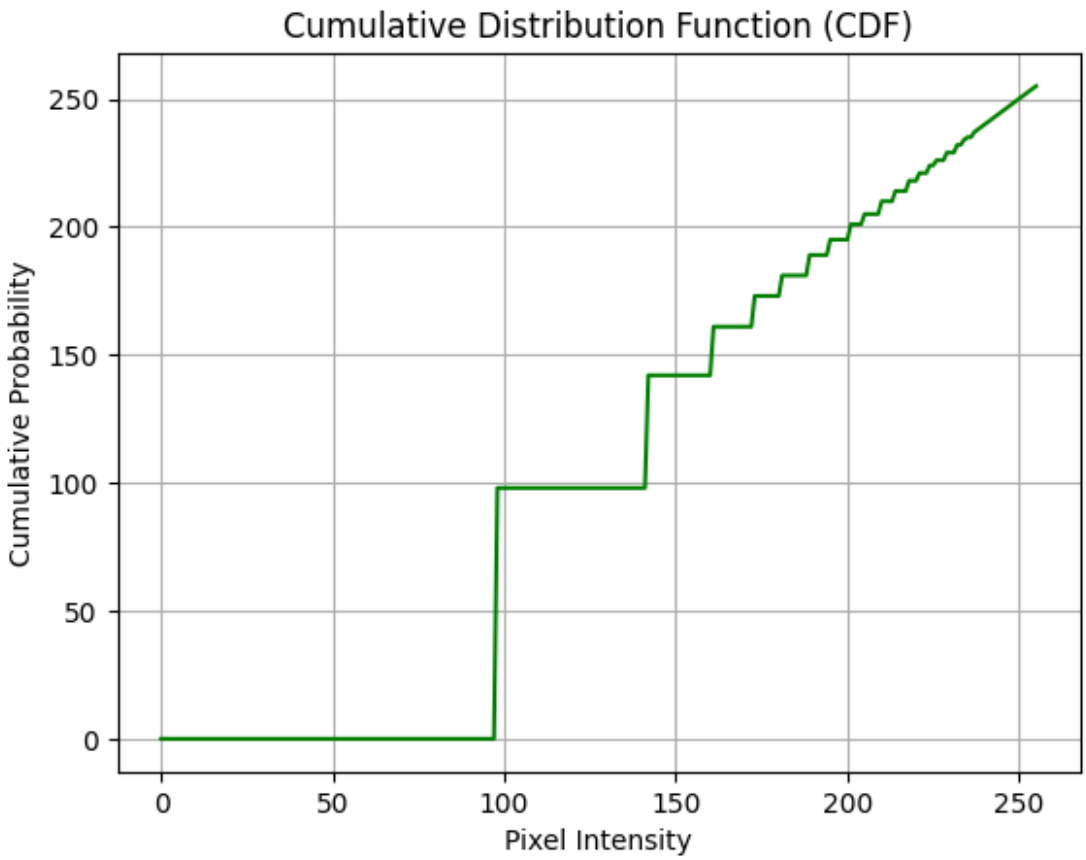

```
from PIL import Image
import numpy as np
# Python code to save the output image to again performm an action on
it
equalized_image_pil = Image.fromarray(equalized_img.astype(np.uint8))
output_path = '/content/equalized_image3.png'

equalized_image_pil.save(output_path)

print(f"Image saved to {output_path}")
```

```
Image saved to /content/equalized_image3.png
```

```
img_path = '/content/equalized_image3.png'
img = Image.open(img_path).convert('L')

equalized_img, pdf, cdf = histogram_equalization(img)
```

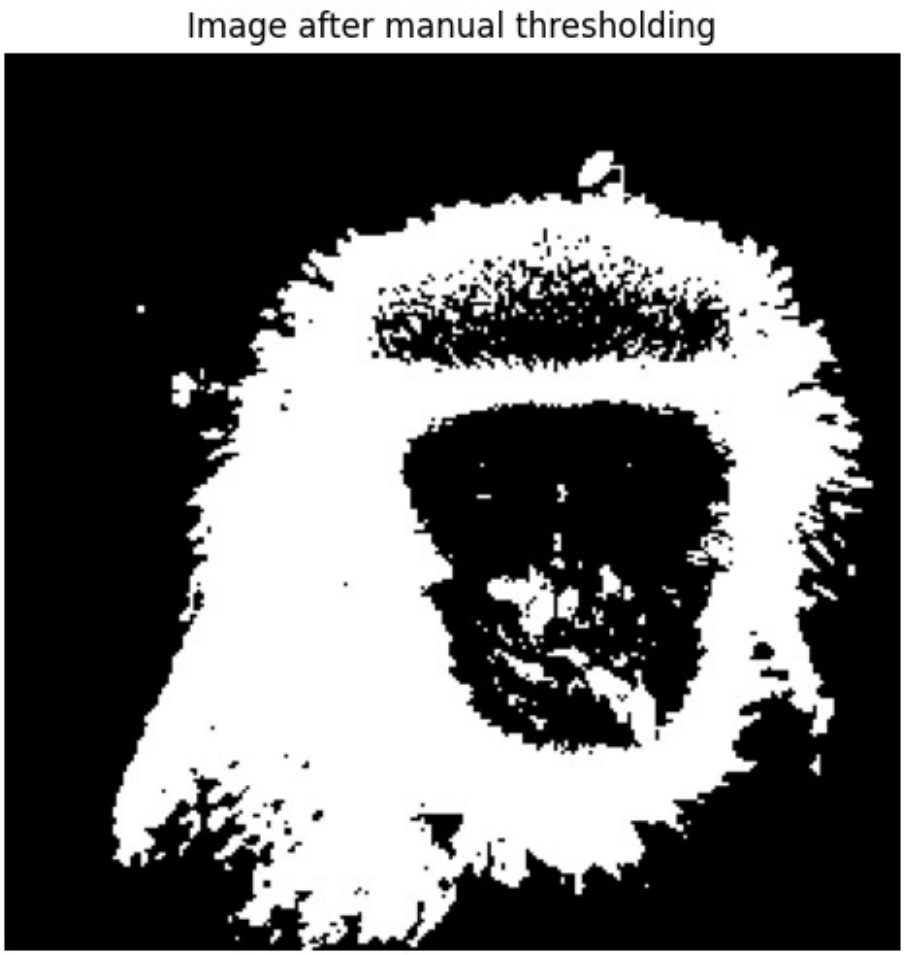
```
plot_pdf(pdf)
plot_cdf(cdf)
```



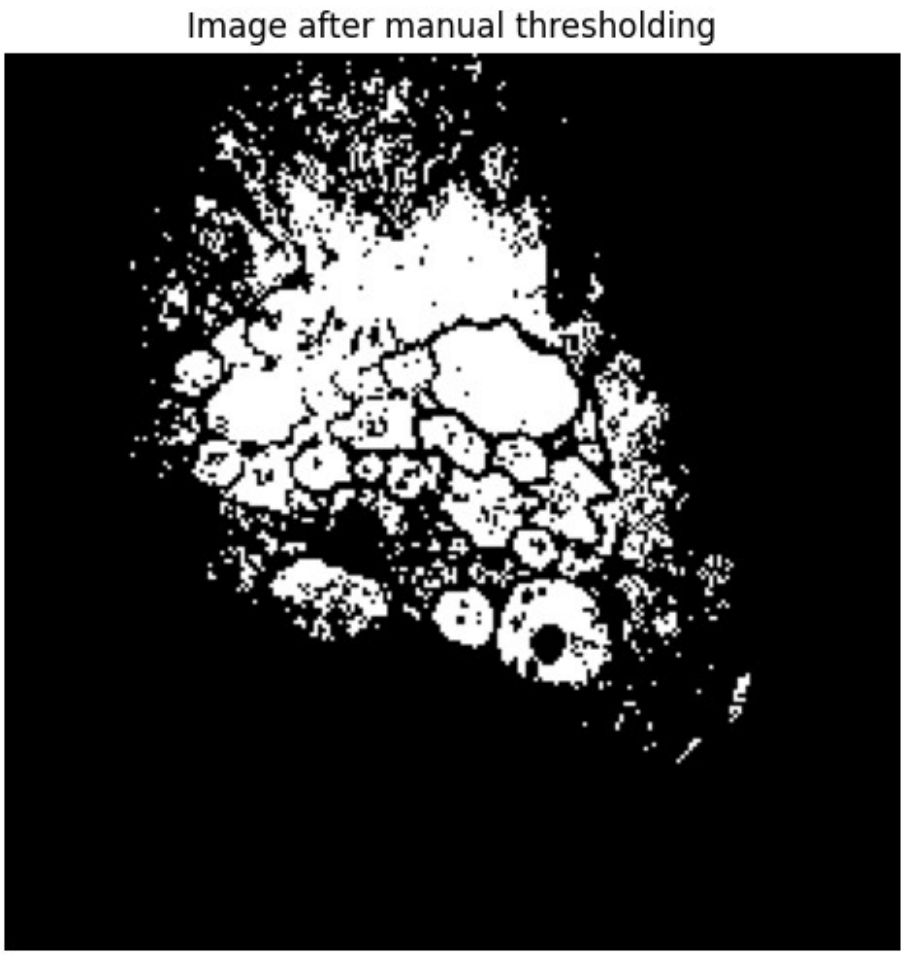

Cumulative Distribution Function (CDF)
Cumulative Probability
Pixel Intensity
0
50
100
150
200
250

Manual Thresholding

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import math

def manual_threshold(im_in, threshold):# Funtion to perfrom manual
threshold on the input image
    manual_thresh_img = np.array(im_in)
    manual_thresh_img[im_in > threshold] = 255
    manual_thresh_img[im_in <= threshold] = 0
    return manual_thresh_img

def plot_image(img_data, im_size):#Function to plot the input image
    equalized_image = img_data.reshape((im_size,
im_size)).astype(np.uint8)
    plt.figure(figsize=(6, 6))
    plt.imshow(equalized_image, cmap='gray', vmin=0, vmax=255)
    plt.title('Image after manual thresholding')
    plt.axis('off')
    plt.show()

img_path = 'b2_a.png' #Code to take an input image and apply functions
to it to perfrom manual thresholding
img = Image.open(img_path).convert('L')

img_data = np.array(img).flatten()
im_size = img.size[0]

threshold_img_manual = manual_threshold(img_data, 128)#Threshold
value(128)

plot_image(threshold_img_manual, im_size)
```




```
img_path = 'b2_b.png'#Code to take an input image and apply functions
to it to perfrom manual thresholding
img = Image.open(img_path).convert('L')

img_data = np.array(img).flatten()
im_size = img.size[0]

threshold_img_manual = manual_threshold(img_data, 128)#Threshold
value(128)

plot_image(threshold_img_manual, im_size)
```



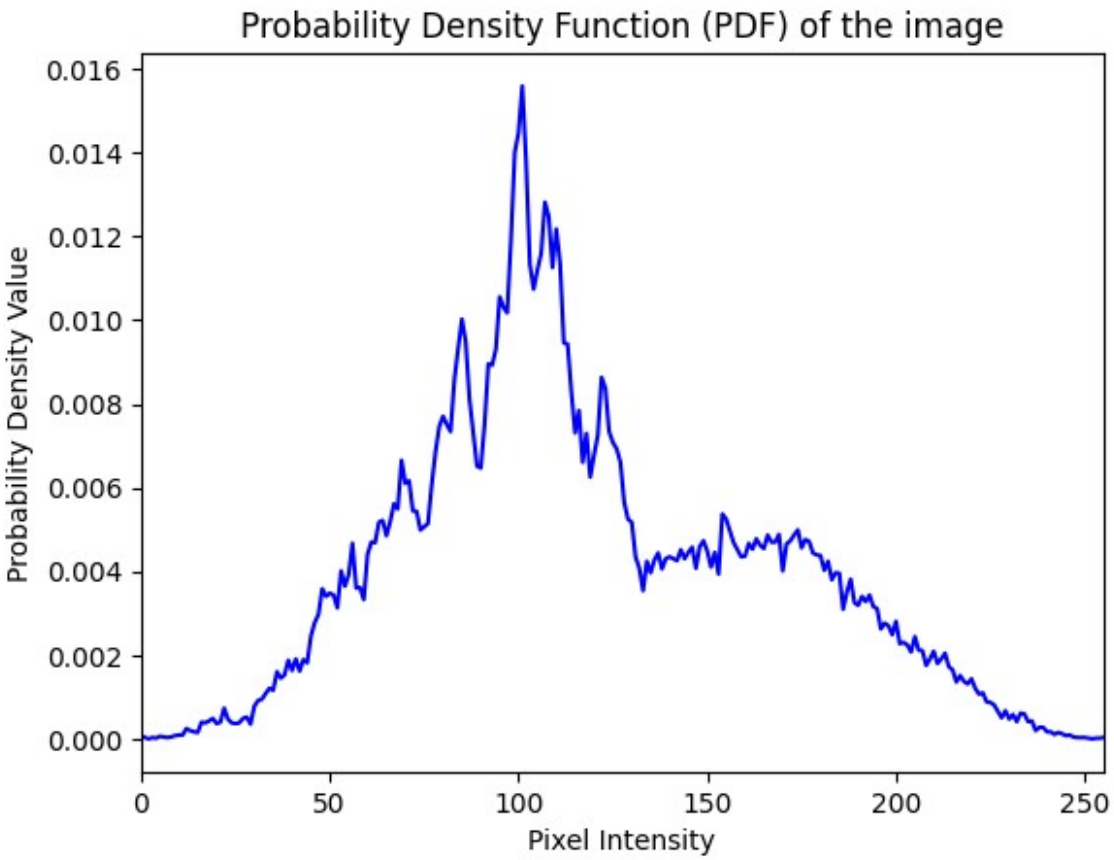

```
img_path = 'b2_c.png'#Code to take an input image and apply functions
to it to perfrom manual thresholding
img = Image.open(img_path).convert('L')

img_data = np.array(img).flatten()
im_size = img.size[0]

threshold_img_manual = manual_threshold(img_data, 220)#Threshold
value(220)

plot_image(threshold_img_manual, im_size)
```

Otsu Thresholding

```
def create_pdf(im_in):#Function to create the pdf using the image
input
    no_of_pixels = [0] * 256
    total_pixels = len(im_in)

    for pixel_value in im_in:
        no_of_pixels[pixel_value] += 1

    pdf = [count / total_pixels for count in no_of_pixels]
    return pdf

def plot_pdf(pdf):#:#Function to plot the pdf for the input image
    plt.plot(range(256), pdf, color='b')
    plt.title('Probability Density Function (PDF) of the image')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Probability Density Value')
    plt.xlim(0, 255)
    plt.show()
```

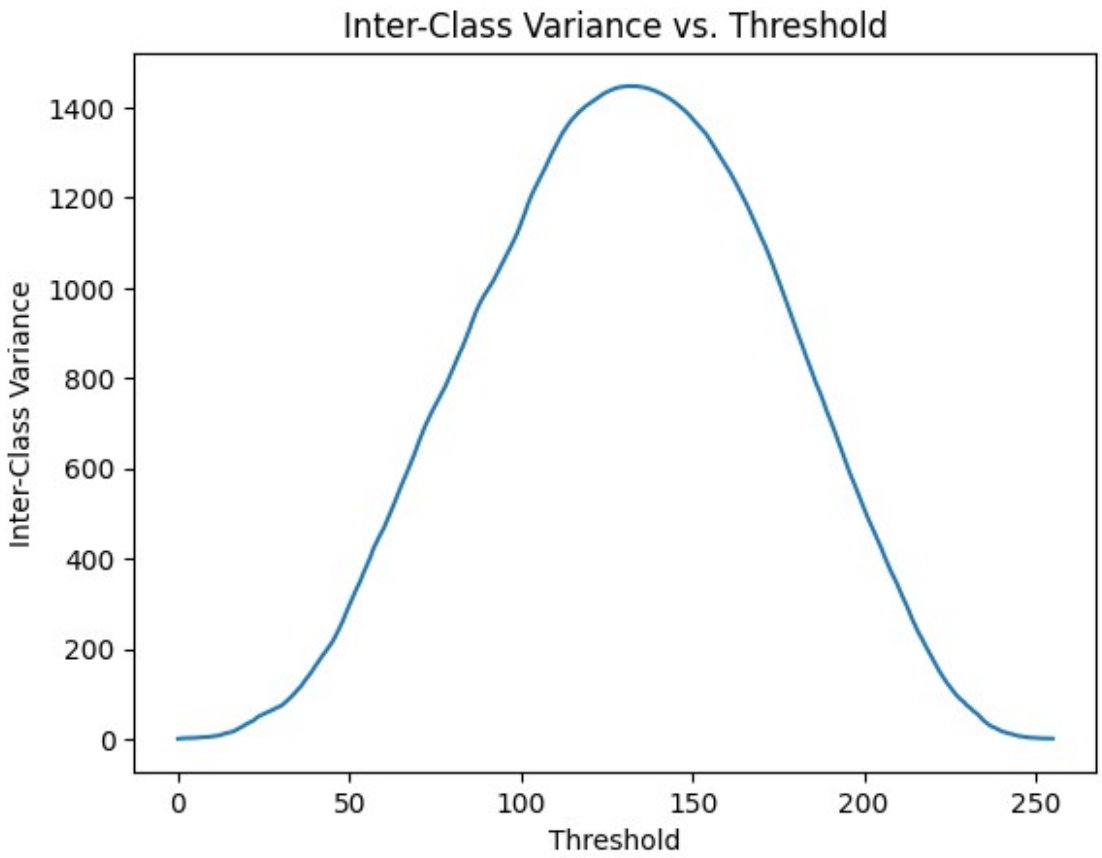
```
def otsu_threshold(im_in):#Function to perfrom Otsu
threshold(automatically finidng optimal threshold for an image)
    pdf = create_pdf(im_in)
    max_variance = float('-inf')
    otsu_threshold_value = 0
    inter_class_variances = []

    for threshold_value in range(256):
        background_probability = sum(pdf[:threshold_value])
        foreground_probability = sum(pdf[threshold_value:])

        if background_probability == 0 or foreground_probability == 0:
            inter_class_variances.append(0)
            continue

        background_sum = sum(i * p for i, p in
enumerate(pdf[:threshold_value]))
        background_mean = background_sum / background_probability
        foreground_mean = (sum(i * p for i, p in enumerate(pdf)) -
background_sum) / foreground_probability

        inter_class_variance = background_probability *
foreground_probability * ((background_mean - foreground_mean) ** 2)
        inter_class_variances.append(inter_class_variance)

        if inter_class_variance > max_variance:
            max_variance = inter_class_variance
            otsu_threshold_value = threshold_value

    otsu_threshold_image = [255 if pixel > otsu_threshold_value else 0
for pixel in im_in]
    return otsu_threshold_image, otsu_threshold_value, max_variance,
inter_class_variances

def plot_image_with_threshold(im_in, threshold_value,
im_size):#Fucntion to plot the thresholded image with the threshold
value
    reshaped_image = np.array(im_in).reshape((im_size,
im_size)).astype(np.uint8)

    plt.imshow(reshaped_image, cmap='gray')
    plt.title(f'Image after Otsu Thresholding (Threshold:
{threshold_value})')
    plt.axis('off')
    plt.show()


def plot_inter_class_variance(inter_class_variances):#Fucntion to plot
the inter classs variance vs threshold plot to identify the peak in
the plot
```

```
    plt.plot(range(256), inter_class_variances)
    plt.title('Inter-Class Variance vs. Threshold')
    plt.xlabel('Threshold')
    plt.ylabel('Inter-Class Variance')
    plt.show()

img_path = '/content/b2_a.png'  #Python Code to take the input image
and apply fucntions such that it results in thresholded image, and
required plots
img = Image.open(img_path).convert('L')

img_data = list(img.getdata())
im_size = int(math.sqrt(len(img_data)))


pdf = create_pdf(img_data)
plot_pdf(pdf)


otsu_threshold_image, otsu_threshold_value,max_variance,
inter_class_variances = otsu_threshold(img_data)
print(f'Otsu Threshold Value: {otsu_threshold_value}')
print(f'Maximum Inter-class Variance: {max_variance}')


plot_image_with_threshold(otsu_threshold_image, otsu_threshold_value,
im_size)
```

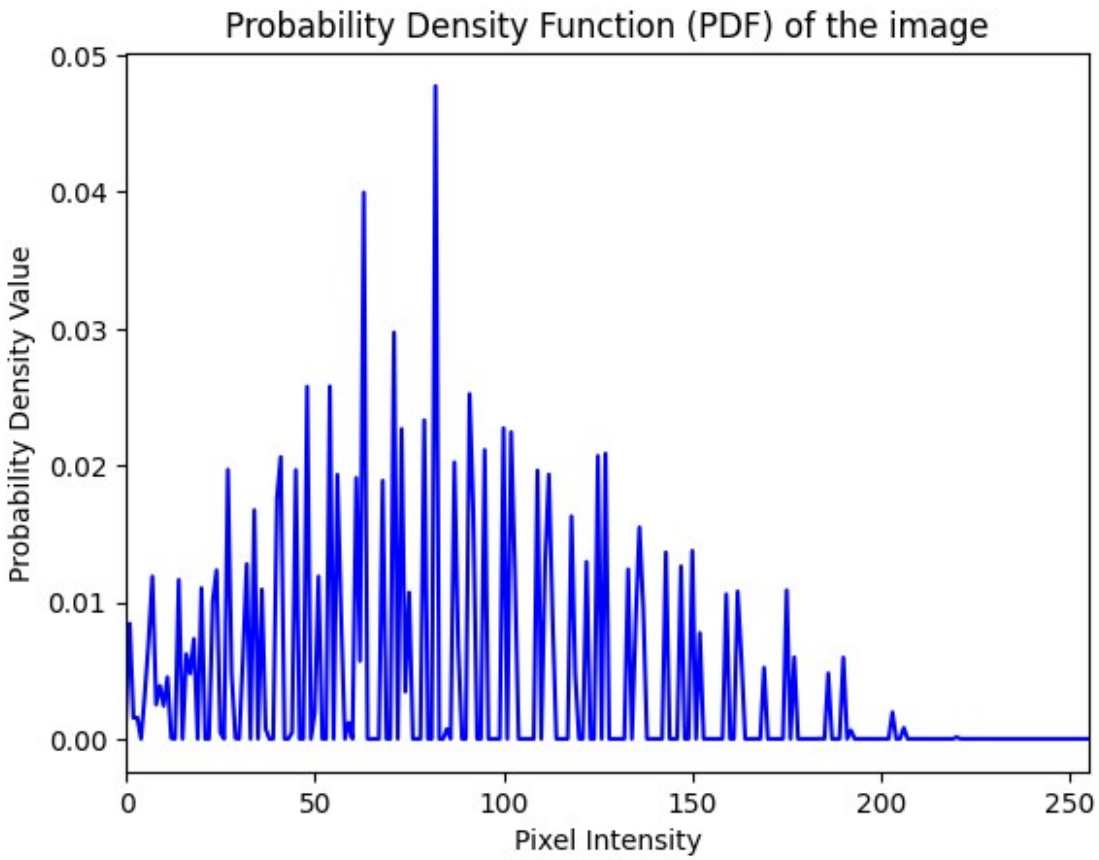



```
Otsu Threshold Value: 132
Maximum Inter-class Variance: 1448.0136082304261
```

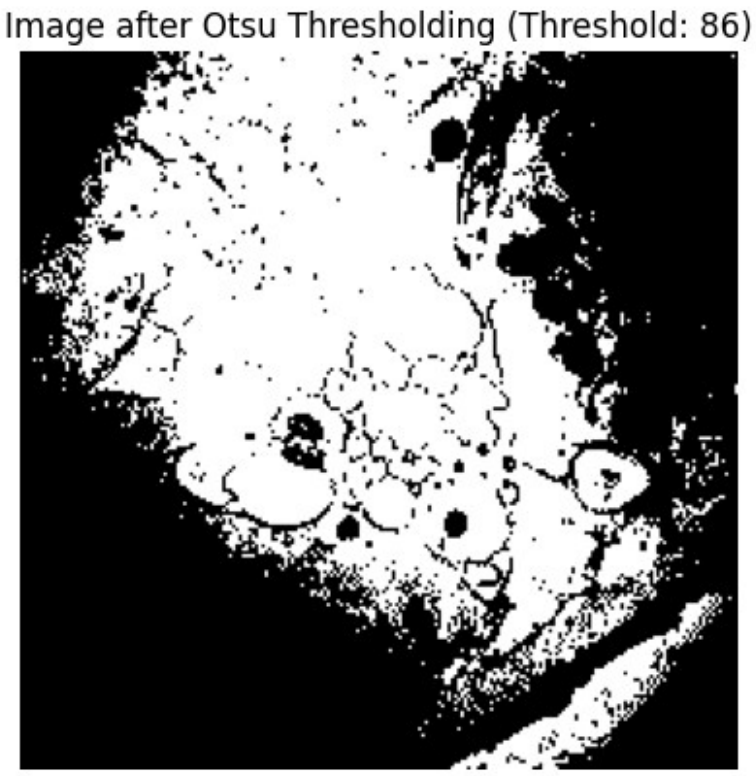

```
# Inter-class variance vs. Threshold
plot_inter_class_variance(inter_class_variances)
```



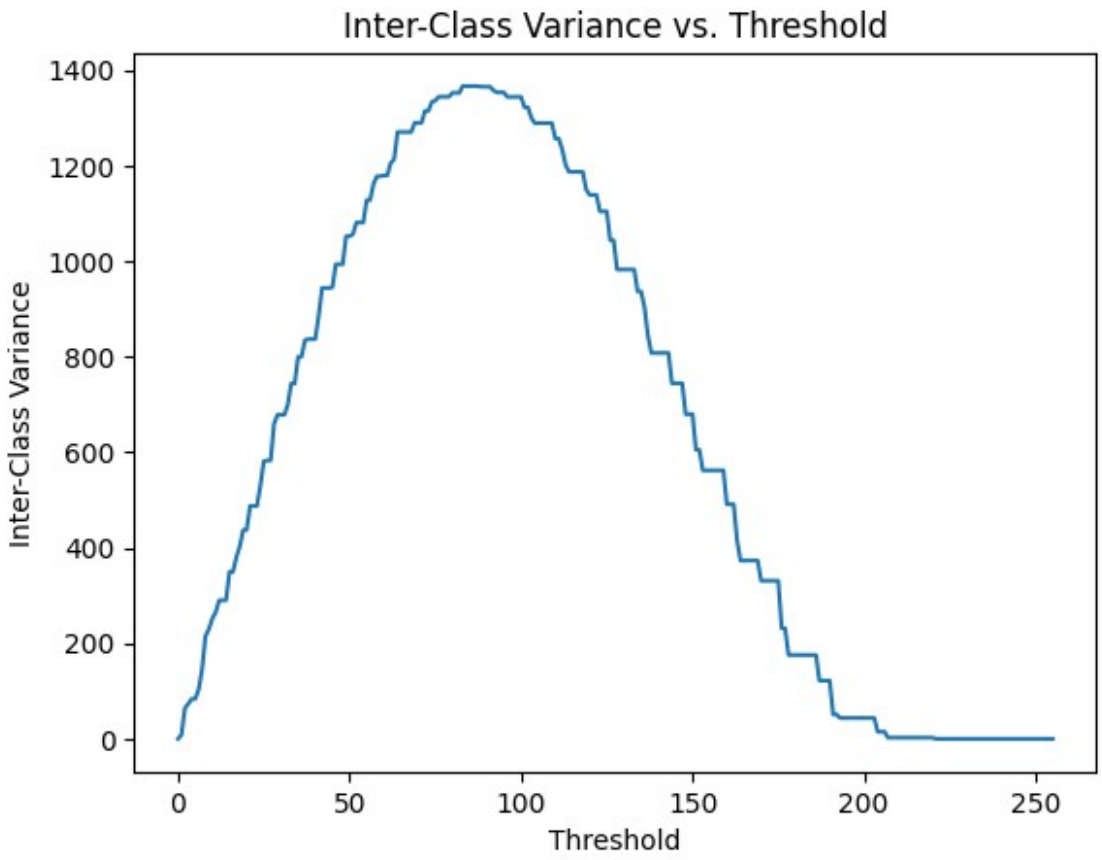

```
img_path = '/content/b2_b.png'  #Python Code to take the input image
and apply fucntions such that it results in thresholded image, and
required plots
img = Image.open(img_path).convert('L')


img_data = list(img.getdata())
im_size = int(math.sqrt(len(img_data)))

pdf = create_pdf(img_data)
plot_pdf(pdf)

otsu_threshold_image, otsu_threshold_value,max_variance,
inter_class_variances = otsu_threshold(img_data)
print(f'Otsu Threshold Value: {otsu_threshold_value}')
print(f'Maximum Inter-class Variance: {max_variance}')

plot_image_with_threshold(otsu_threshold_image, otsu_threshold_value,
im_size)
```



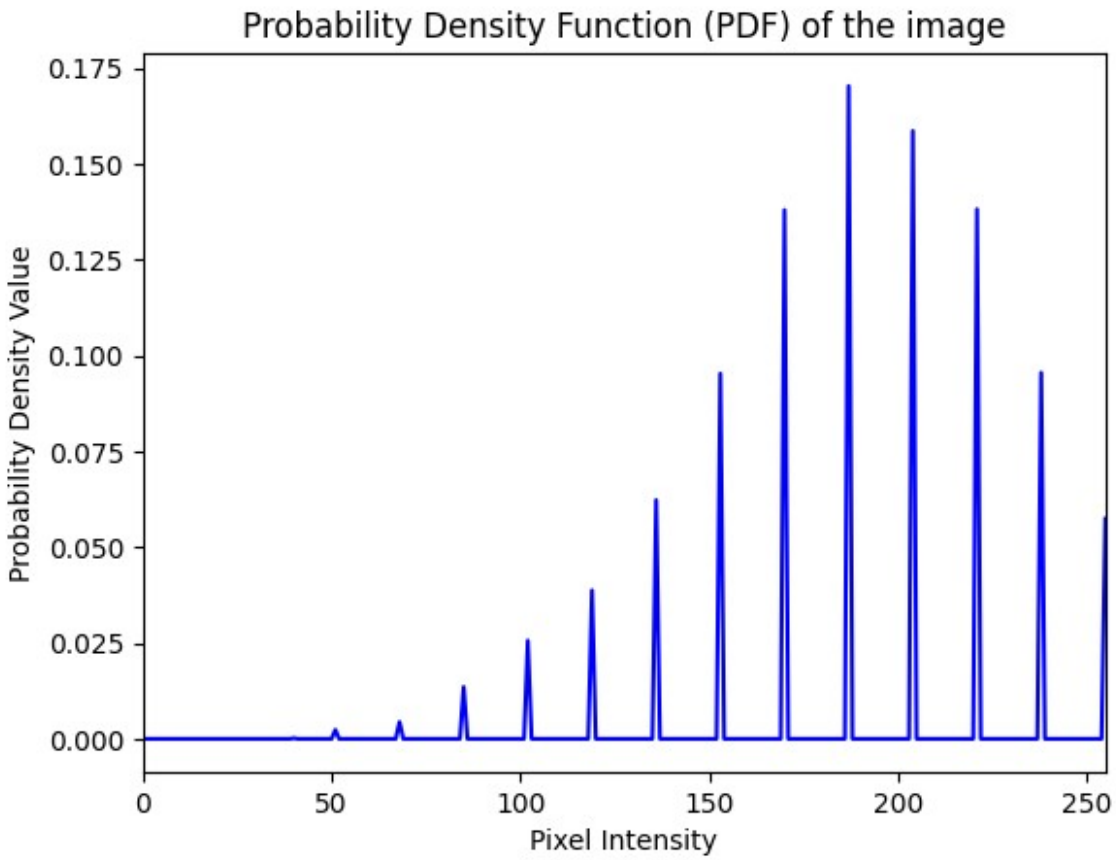

```
Otsu Threshold Value: 86
Maximum Inter-class Variance: 1366.6478182943322
```




```
#  Inter-class variance vs. Threshold Plot
plot_inter_class_variance(inter_class_variances)
```

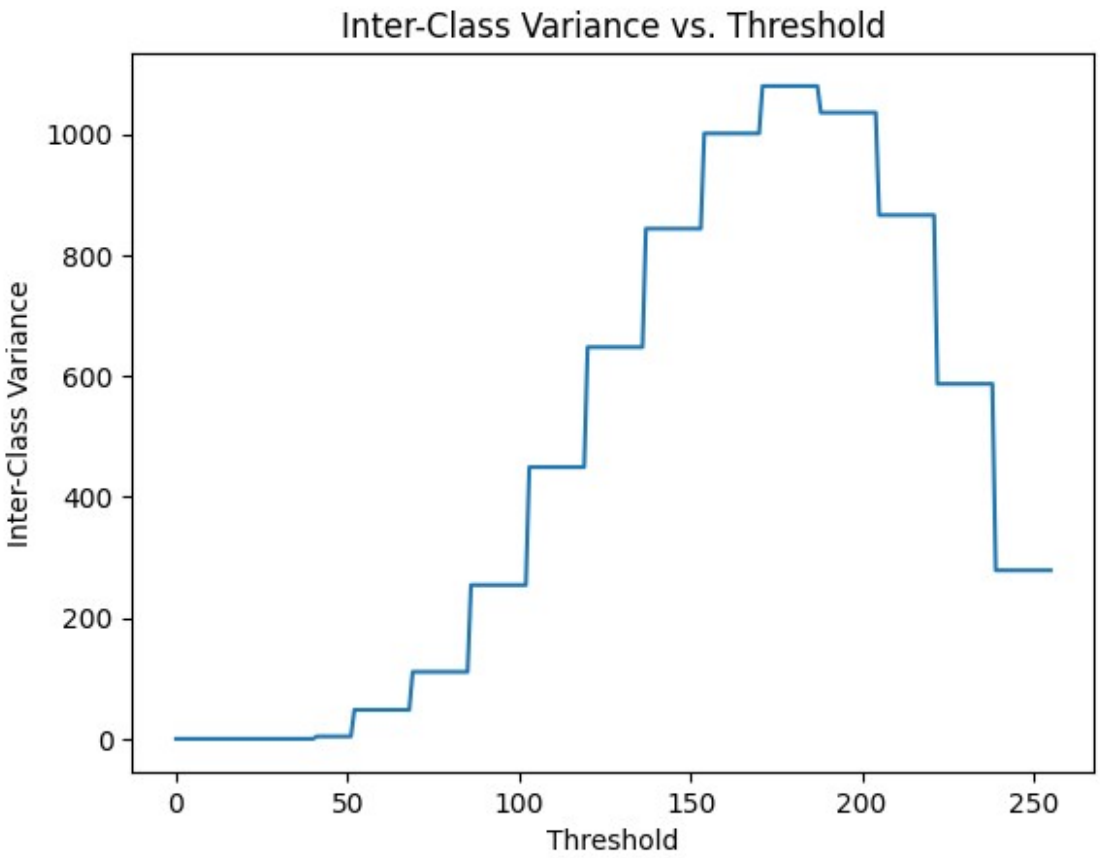



```
img_path = '/content/b2_c.png' #Python Code to take the input image
and apply fucntions such that it results in thresholded image, and
required plots
img = Image.open(img_path).convert('L')

img_data = list(img.getdata())
im_size = int(math.sqrt(len(img_data)))

pdf = create_pdf(img_data)
plot_pdf(pdf)

otsu_threshold_image, otsu_threshold_value,max_variance,
inter_class_variances = otsu_threshold(img_data)
print(f'Otsu Threshold Value: {otsu_threshold_value}')
print(f'Maximum Inter-class Variance: {max_variance}')

plot_image_with_threshold(otsu_threshold_image, otsu_threshold_value,
im_size)
```




```
Otsu Threshold Value: 171
Maximum Inter-class Variance: 1079.7087141560728
```

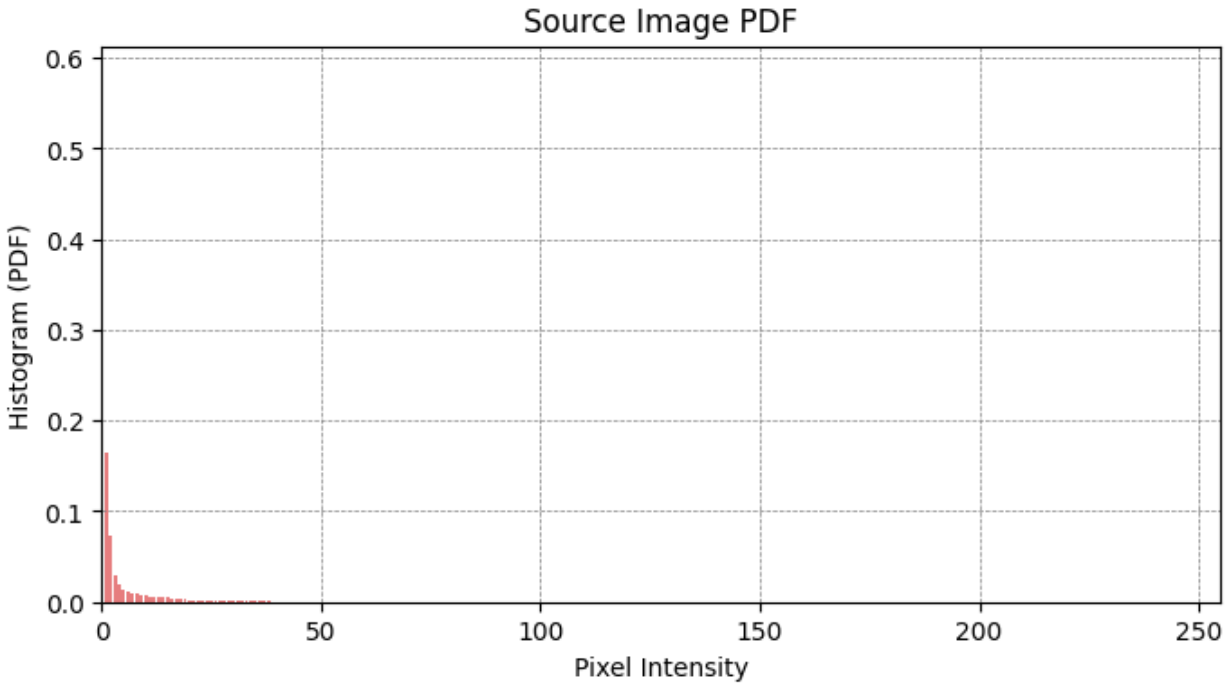

```
# Inter-class variance vs. Threshold Plot
plot_inter_class_variance(inter_class_variances)
```

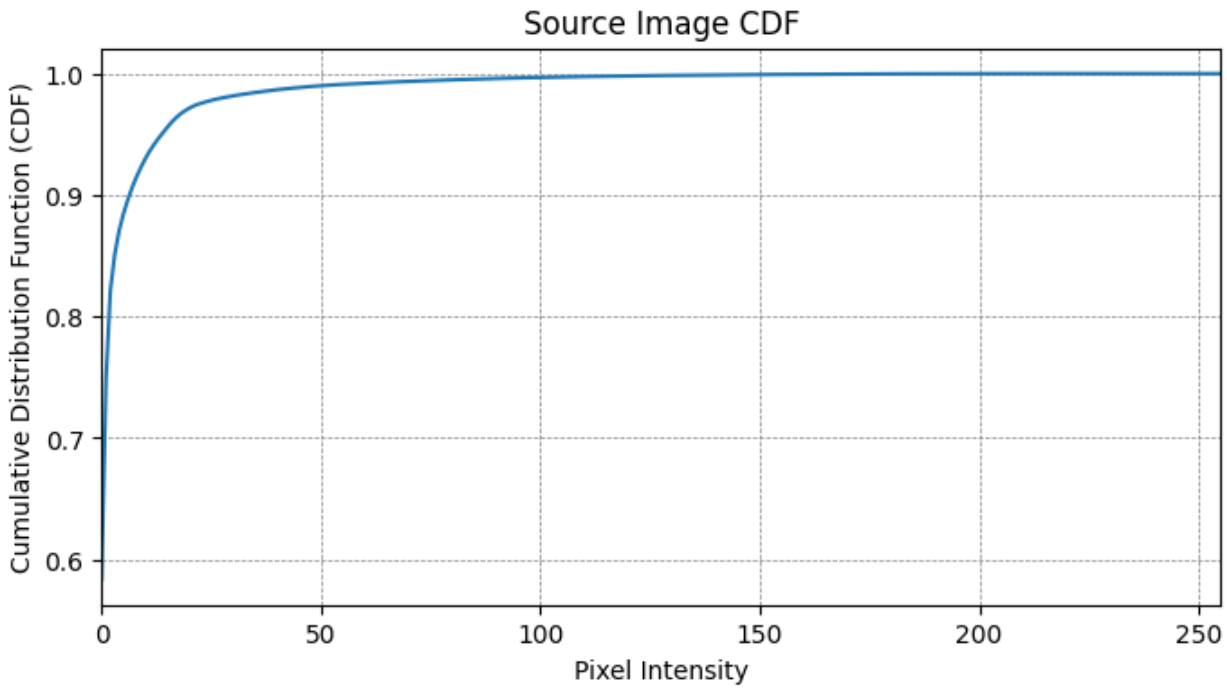

Inter-Class Variance vs. Threshold
Inter-Class Variance
Threshold
0
200
400
600
800
1000
0
50
100
150
200
250

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import math

def create_pdf(im_in):#Function to create PDF
    histogram = np.zeros(256)
    for pixel in im_in:
        histogram[pixel] += 1
    total_pixels = im_in.size
    pdf = histogram / total_pixels
    return pdf

def create_cdf(pdf):#Function to create CDF
    cdf = np.zeros_like(pdf)
    cdf[0] = pdf[0]
    for i in range(1, len(pdf)):
      cdf[i] = cdf[i-1] + pdf[i]
    return cdf

def plot_pdf(image, title):#Function to plot the PDF of an image

    pdf = create_pdf(image)
    plt.figure(figsize=(8, 4))
    plt.bar(np.arange(256), pdf, color='tab:red', alpha=0.6)
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Histogram (PDF)')
    plt.title(title + ' PDF')
    plt.xlim(0, 255)
    plt.grid(True, which='major', linestyle='--', linewidth='0.5',
color='grey')
    plt.show()

def plot_cdf(image, title):#Function to plot the CDF of an image
    """Plot the CDF of an image."""
    pdf = create_pdf(image)
    cdf = create_cdf(pdf)
    plt.figure(figsize=(8, 4))
    plt.plot(np.arange(256), cdf, color='tab:blue')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Cumulative Distribution Function (CDF)')
    plt.title(title + ' CDF')
    plt.xlim(0, 255)
    plt.grid(True, which='major', linestyle='--', linewidth='0.5',
color='grey')
    plt.show()

def histogram_matching(source, reference):#Function to perfrom
Histogram Matching such that it returns matched image after taking
input source image and reference image,adjusting its contrast.
```

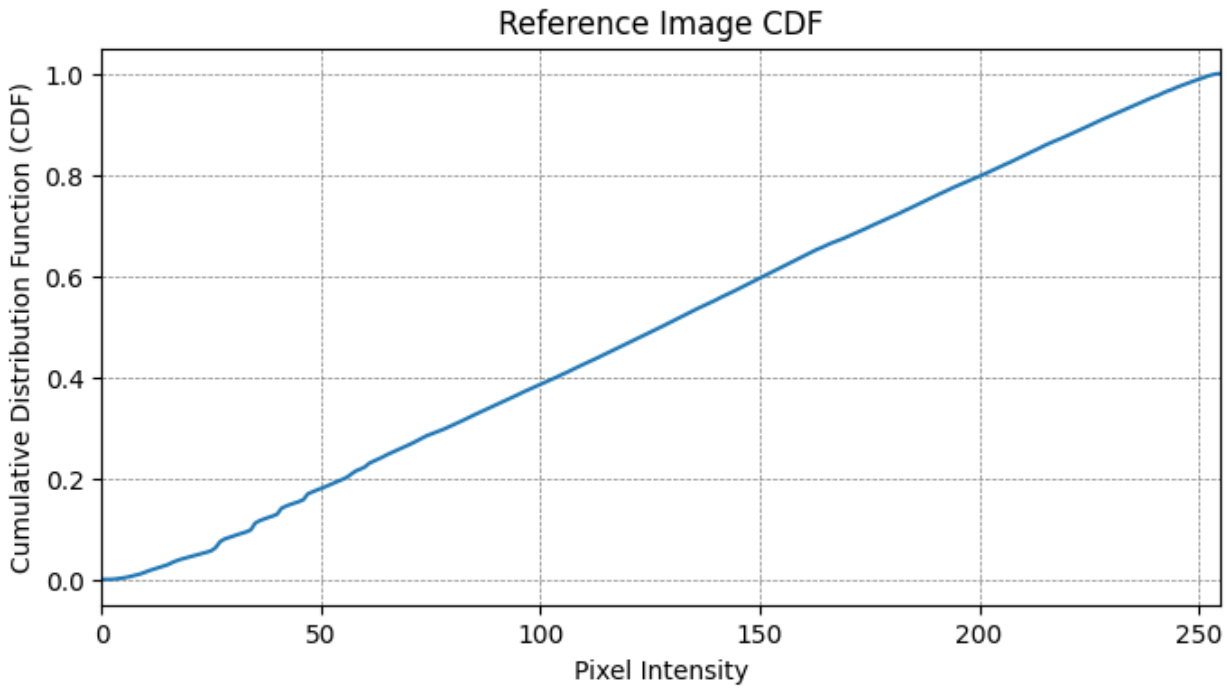
```
    # Compute PDFs and CDFs
    source_pdf = create_pdf(source.flatten())
    source_cdf = create_cdf(source_pdf)
    reference_pdf = create_pdf(reference.flatten())
    reference_cdf = create_cdf(reference_pdf)

    mapping_function = np.zeros(256, dtype=np.uint8)
    for pixel_value in range(256):
        closest_idx = np.argmin(np.abs(reference_cdf -
source_cdf[pixel_value]))
        mapping_function[pixel_value] = closest_idx

    matched_image =
mapping_function[source.flatten()].reshape(source.shape)

    return source, reference, matched_image, source_pdf, source_cdf,
reference_pdf, reference_cdf

def display_images(source, reference, matched):
    fig, axes = plt.subplots(1, 3, figsize=(18, 6))
    axes[0].imshow(source, cmap='gray')
    axes[0].set_title('Source Image')
    axes[1].imshow(reference, cmap='gray')
    axes[1].set_title('Reference Image')
    axes[2].imshow(matched, cmap='gray')
    axes[2].set_title('Matched Image')
    for ax in axes:
        ax.axis('off')
    plt.show()

def plot_source(source):
    source_pdf = create_pdf(source.flatten())
    source_cdf = create_cdf(source_pdf)

    plot_pdf(source.flatten(), 'Source Image')
    plot_cdf(source.flatten(), 'Source Image')

def plot_reference(reference):
    reference_pdf = create_pdf(reference.flatten())
    reference_cdf = create_cdf(reference_pdf)

    plot_pdf(reference.flatten(), 'Reference Image')
    plot_cdf(reference.flatten(), 'Reference Image')

def plot_matched(matched):
    matched_pdf = create_pdf(matched.flatten())
    matched_cdf = create_cdf(matched_pdf)

    plot_pdf(matched.flatten(), 'Matched Image')
    plot_cdf(matched.flatten(), 'Matched Image')
```

```
source_img_path = '/content/treec.jpg'
reference_img_path = '/content/imgcv2.png'
source_img = Image.open(source_img_path).convert('L')
reference_img = Image.open(reference_img_path).convert('L')
source_array = np.array(source_img)
reference_array = np.array(reference_img)

source, reference, matched, source_pdf, source_cdf, reference_pdf,
reference_cdf = histogram_matching(source_array, reference_array)

plot_source(source)
```

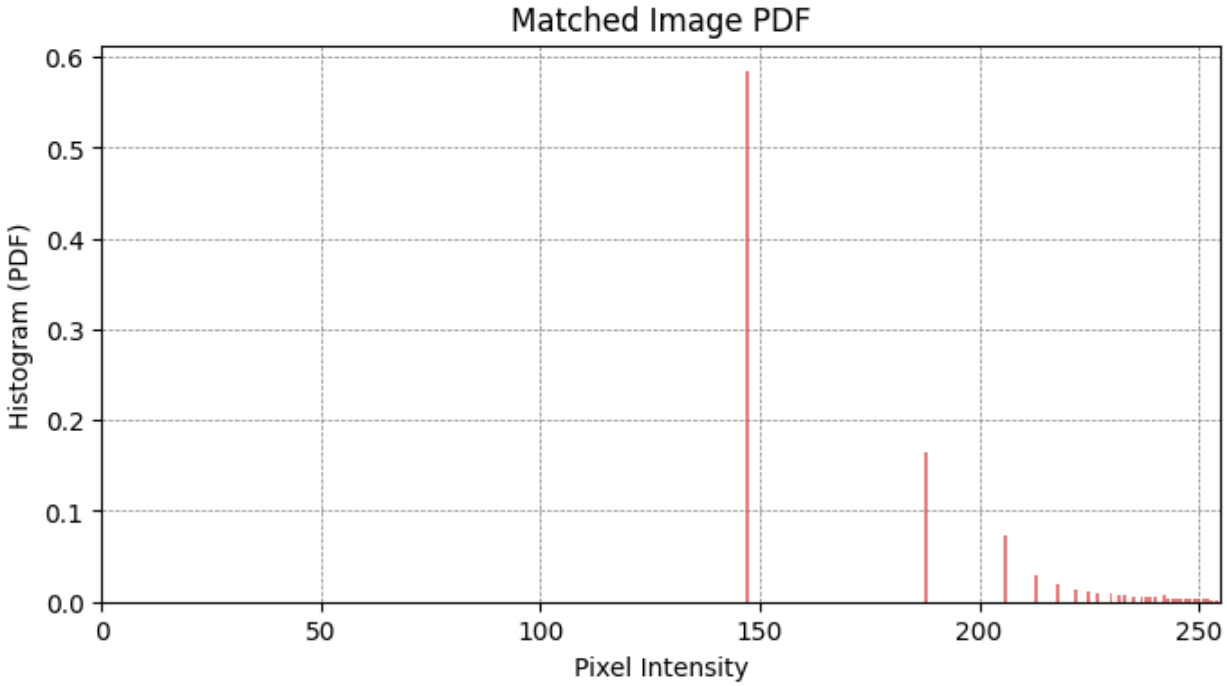


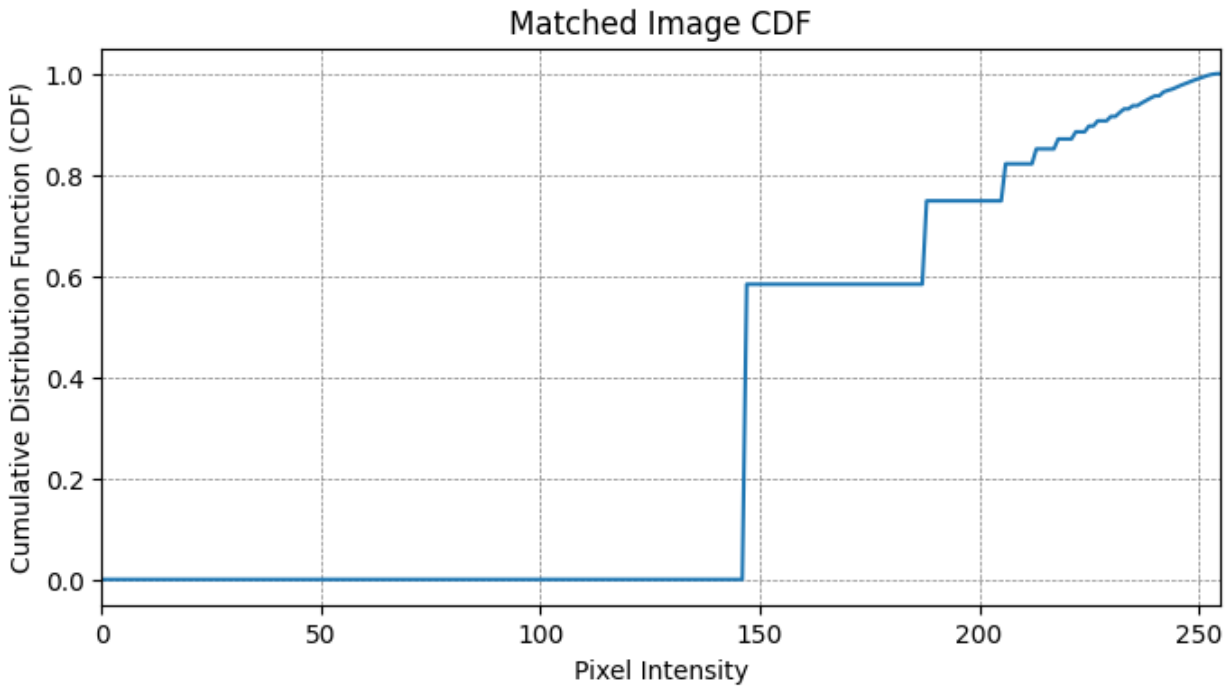



```
plot_reference(reference)
```




```
plot_matched(matched)
```






```
# Display images
display_images(source, reference, matched)
```