COMPUTER VISION
PROJECT- 1

Meet oswal
mo2532@nyu.edu

INDEX

Q1. Histogram Equalization:

a) Introduction to histogram equalization and design-implementation of it.

Introduction to Histogram Equalization:

Histogram equalization is a technique used in image processing to enhance the contrast and improve the visual appearance of images. The primary goal of histogram equalization is to redistribute the intensity values of an image in such a way that the histogram of the output image becomes more uniformly distributed across the available intensity levels.

Here's a brief description of how histogram equalization works:

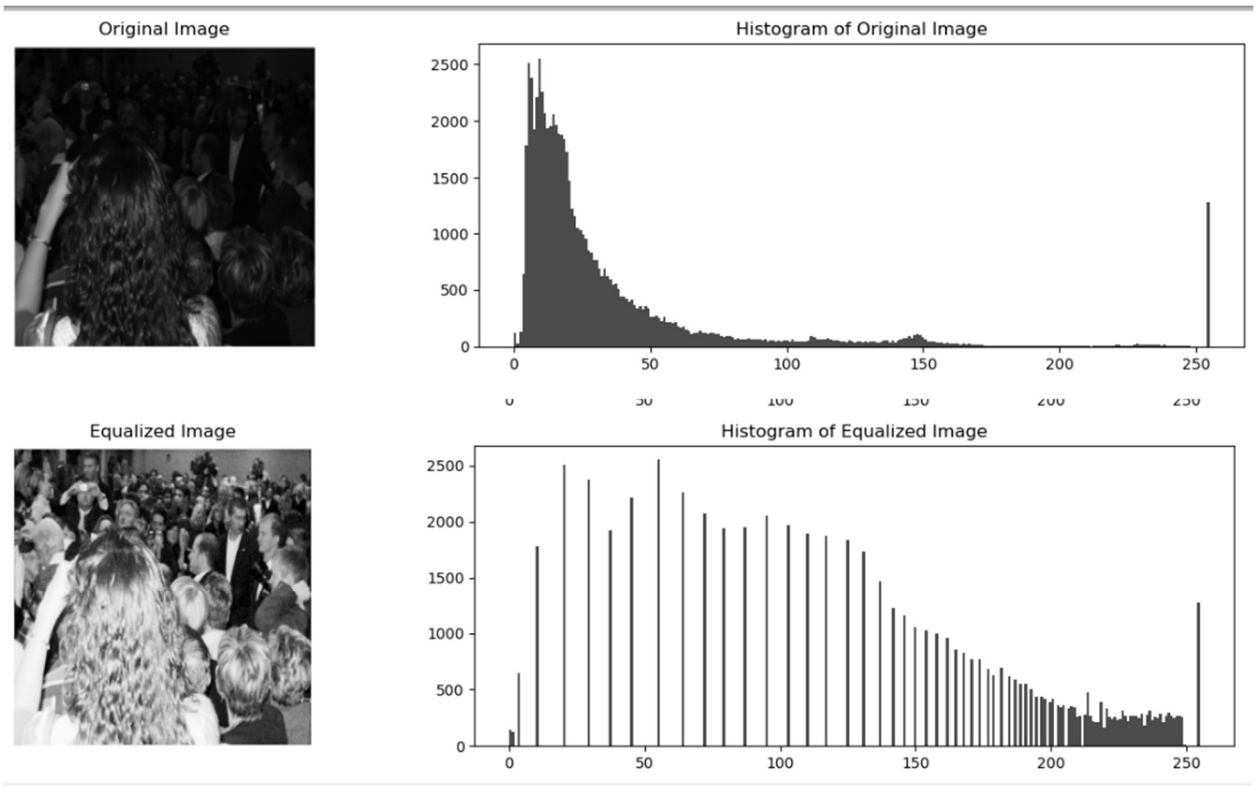
- **Compute Histogram:** The process begins by computing the histogram of the input image. The histogram represents the frequency distribution of pixel intensities in the image. It shows how many pixels in the image have a particular intensity level.
- **Compute Cumulative Distribution Function (CDF):** After computing the histogram, the cumulative distribution function (CDF) is calculated. The CDF represents the cumulative sum of the histogram values. It indicates the cumulative probability of encountering pixel intensities up to a certain level.
- **Equalization Transformation**: Histogram equalization involves applying a transformation function to the pixel intensities of the input image. This transformation aims to spread out the intensity values so that they cover the entire available range more evenly.
- **Mapping Intensities**: The transformation function maps each pixel intensity from its original value in the input image to a new value in the output image. This mapping is determined based on the desired cumulative distribution of pixel intensities.
- **Output Image:** The result of histogram equalization is an output image with improved contrast and enhanced details. Dark regions become darker, bright regions become brighter, and the overall distribution of intensity values becomes more balanced.

Designing and Implementation of code:

- **Image Preprocessing:** The code reads the input image ('people.png') and converts it to grayscale. The image is resized to a 256x256 resolution. The pixel values are normalized to the range [0, 255] using min-max scaling.
- **Create PDF Function:** This function computes the Probability Density Function (PDF) of the input image. It initializes a histogram with 256 bins and calculates the frequency of each pixel intensity. The histogram is then normalized by dividing by the total number of pixels in the image to obtain the PDF.
- **Create CDF Function**: This function calculates the Cumulative Distribution Function (CDF) from the PDF. It initializes an array for the CDF and computes each cumulative sum of the PDF values.
- **Histogram Equalization Function**: This function performs histogram equalization on the input grayscale image. It computes the PDF and CDF of the input image using the previously defined functions. The image is flattened to a 1D array, and each pixel value is mapped to its corresponding value in the CDF, effectively equalizing the histogram. The equalized image is reshaped to its original dimensions and returned as the output.

- **Display Original and Processed Images with their respective histograms**: Matplotlib is used to create a figure with subplots for the original and equalized images, along with their histograms.

b) Image before and after histogram equalization and their respective histograms.



c) Changes before and after the histogram equalization process:

After applying histogram equalization to the original grayscale image, several changes become evident both in the image itself and in its corresponding histogram. Here's how the image and histogram have changed, connected back to the description of equalization:

Image Changes:

- **Enhanced Contrast:** Histogram equalization aims to spread out the intensity values of an image to cover the entire available range more evenly. As a result, areas of the image that were previously low-contrast or washed out may now exhibit improved contrast and detail.
- **Brightening/Darkening:** Histogram equalization redistributes pixel intensities such that dark regions become darker and bright regions become brighter. This adjustment contributes to the overall improvement in image contrast and visual clarity.

Histogram Changes:

- **Spread of Pixel Intensities:** Before equalization, the histogram may have exhibited peaks and valleys, indicating uneven distribution of pixel intensities across the available range. After

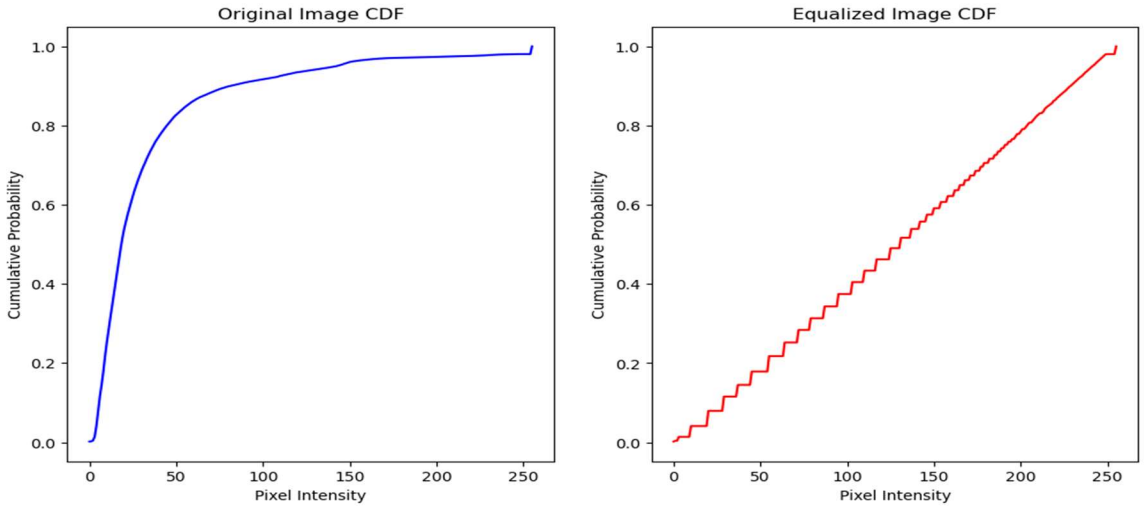
equalization, the histogram becomes more uniformly distributed, with a smoother curve spanning a wider range of intensity levels.

- **Enhanced Dynamic Range:** The histogram's expanded spread of pixel intensities reflects the increased dynamic range achieved through equalization. More pixel intensity values are utilized across the available range, leading to a more comprehensive representation of image features and details.

Connection to Equalization:

- **Histogram equalization redistributes pixel intensities** based on their cumulative distribution function (CDF). The process aims to transform the image's intensity distribution to achieve a **more uniform distribution**, ideally resembling a flat histogram.
- By mapping pixel intensities according to the CDF, **areas of the image with initially limited contrast are expanded**, while areas with high contrast are compressed. This transformation effectively **enhances the image's overall contrast and visibility of features**.

d) Cumulative distributed function before and after histogram equalization:



In the side-by-side figure, the left subplot shows the CDF of the original image, while the right subplot shows the CDF of the equalized image.

Here's what we can observe and interpret:

Original Image CDF:

- The original image CDF might exhibit a non-linear shape, with certain intensity levels dominating the distribution while others are less represented.
- Areas of low contrast in the original image may result in regions where the CDF has relatively shallow slopes.
- Peaks and valleys in the CDF can indicate areas of high and low pixel intensity concentration in the original image, respectively.

Equalized Image CDF:

- After histogram equalization, the CDF tends to be more linear and evenly distributed across the intensity range.
- The slope of the equalized image CDF is typically steeper, indicating a more balanced distribution of pixel intensities.
- The transformation aims to stretch the intensity range to cover the entire spectrum, resulting in enhanced contrast and visibility of details in the equalized image.
- The shape of the CDF before and after histogram equalization reflects the changes in image contrast and the redistribution of pixel intensities achieved through the equalization process.
- A smoother and more linear CDF in the equalized image indicates improved contrast and a wider dynamic range compared to the original image.

e) Reapplying Histogram Equalization on the corrected image:

After conducting further analysis by reapplying histogram equalization to the corrected image, it was observed that there were no discernible or significant changes in the visual characteristics of the image. This observation is supported by several key factors and considerations:

- **Effectiveness of Previous Enhancements:** The effectiveness of the previous enhancements, particularly histogram equalization, may have already maximized the contrast and clarity of the image to a satisfactory level. The initial equalization process likely addressed issues related to uneven illumination and contrast, resulting in a visually improved image.
- **Limited Impact of Reapplication:** Reapplying histogram equalization to an image that has already undergone comprehensive preprocessing may yield marginal or imperceptible changes. As the image's characteristics and features were already optimized during the initial processing, further equalization may not result in noticeable visual improvements.

f) Applying Histogram Equalization on another low contrast image:

Observations:

- **Increased Contrast and Sharpness:** Histogram equalization redistributes pixel intensities across the entire dynamic range, which effectively enhances the contrast between different features in the image. As a result, areas of low contrast become more distinct, leading to an overall increase in image sharpness and clarity.
- **Expansion of Histogram Distribution Horizontally:** Histogram equalization stretches the distribution of pixel intensities horizontally across the entire range of possible values. This expansion ensures that the full range of intensity levels is utilized, allowing for better representation of subtle variations in brightness and contrast within the image.
- Histogram equalization effectively enhances image contrast and sharpness by redistributing pixel intensities across a wider range of values. The expansion of the histogram distribution horizontally allows for better utilization of the available intensity levels.

Q2. Otsu Thresholding:

Introduction to Otsu Thresholding:

Otsu's thresholding, also known as Otsu's method or Otsu's algorithm, is a widely used technique in image processing for automatic image thresholding. The core idea behind Otsu's thresholding is to find the optimal threshold value that maximizes the separation between object and background in an image histogram. It accomplishes this by minimizing the intra-class variance of pixel intensities within each segmented region.

Here's a brief overview of how Otsu's thresholding works:

- **Compute Histogram:** Otsu's method begins by computing the histogram of pixel intensities in the input grayscale image.
- **Calculate Intra-Class Variance:** Otsu's algorithm iteratively evaluates the intra-class variance for all possible threshold values between the minimum and maximum pixel intensities. The intra-class variance measures the spread of pixel intensities within each segmented region.
- **Threshold Selection:** Once the intra-class variance is computed for each potential threshold value, Otsu's method selects the threshold that maximizes the inter-class variance or minimizes the intra-class variance. This threshold value effectively separates the foreground and background regions, optimizing the segmentation process.
- **Threshold Application:** Finally, the selected threshold value is applied to the input grayscale image to produce a binary image where pixels with intensities above the threshold belong to the foreground region, while pixels below the threshold belong to the background.

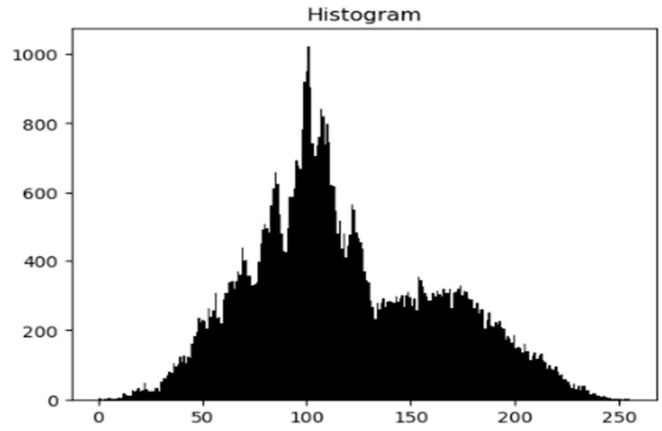
Design and Implementation:

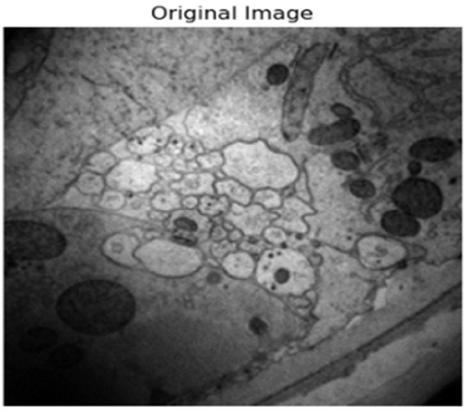
- **Manual Threshold Function:** This function performs manual thresholding on the input image. The threshold set for this function is **120.** Pixels with intensity values greater than or equal to the threshold of 120, are set to 255 (white), and others are set to 0 (black).
- **Otsu Threshold Function:** This function implements Otsu's thresholding algorithm to automatically find the optimal threshold for segmenting the input image. It computes the inter-class variance for all possible threshold values and selects the threshold that maximizes this variance.
- **Reading and Processing Images:** The code reads grayscale images from the specified paths, resizes them to 256x256 pixels, and converts them to unsigned 8-bit integers (uint8). This preprocessing ensures that all images are compatible with the subsequent thresholding operations.
- **Thresholding and Visualization**: For each image, the code computes the PDF (Probability Density Function). Manual thresholding and Otsu's thresholding are applied to each image using the defined functions. The original image, manual thresholding result, and Otsu's thresholding result are displayed alongside their histograms for visual comparison. Additionally, a plot of the inter-class variance as a function of the threshold values is generated to illustrate the optimization process performed by Otsu's algorithm.
- **Inter-Class Variance Calculation:** Within the loop iterating over different threshold values, Otsu's thresholding function computes the inter-class variance for each threshold and stores

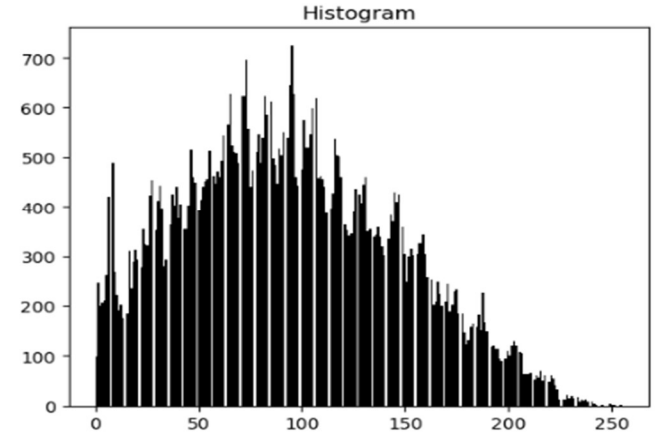
the values in a list. This variance plot helps visualize the threshold selection process and the relationship between threshold values and inter-class variance.

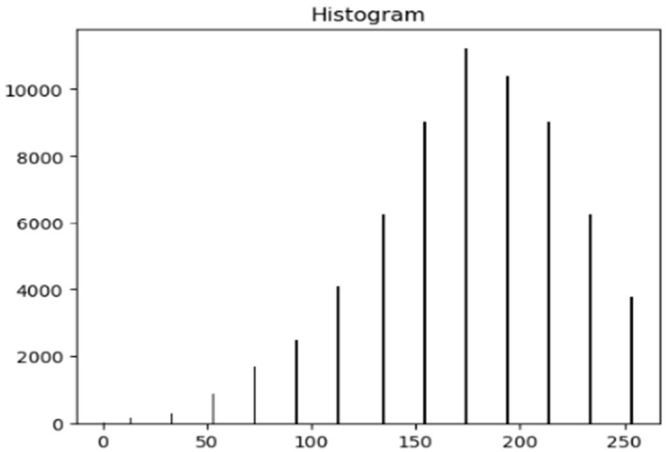
- **Output and Conclusion:** The output includes visual representations of the original image, manual thresholding, and Otsu's thresholding results, along with their respective histograms and inter-class variance plots. The chosen threshold value and inter-class variance are displayed for each image, providing insights into the effectiveness of Otsu's thresholding algorithm.

a) Original Image Histogram: Image 1, 2, 3 from up to down respectively.



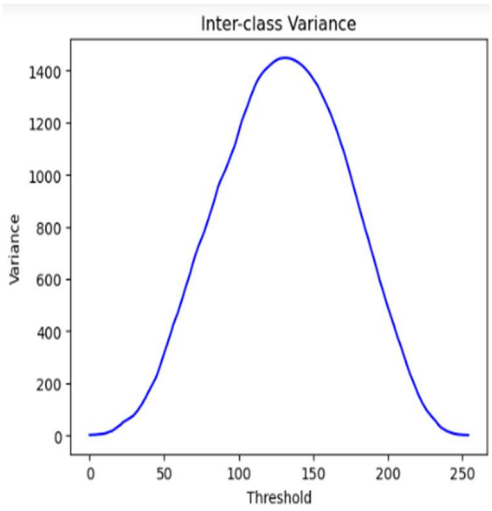


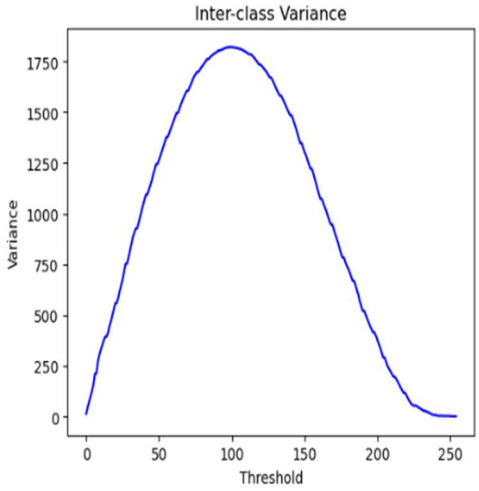


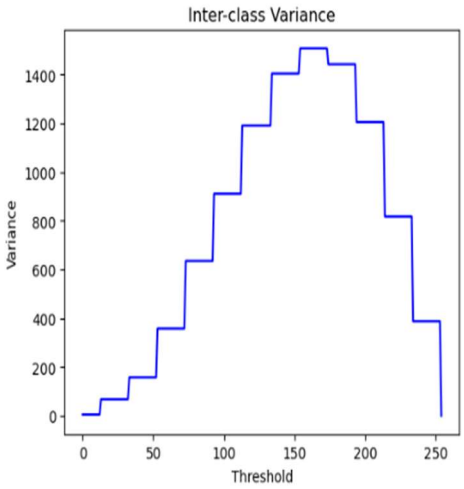


b) Graph Representing the relationship between threshold (0 - 255) and variance at that threshold.

- Image 1, 2, 3 from left to right respectively.







Inter-class Variance: 1448.01
Threshold: 132

Inter-class Variance: 1820.03
Threshold: 99

Inter-class Variance: 1506.47
Threshold: 155

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

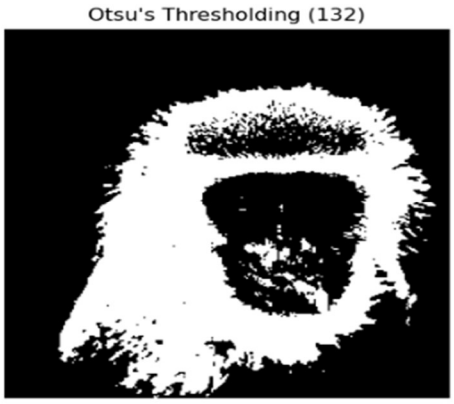
- Image One: Inter-class variance: 1448.01
- Image Two: Inter-class variance: 1820.003
- Image Three: Inter-class variance: 1506.47

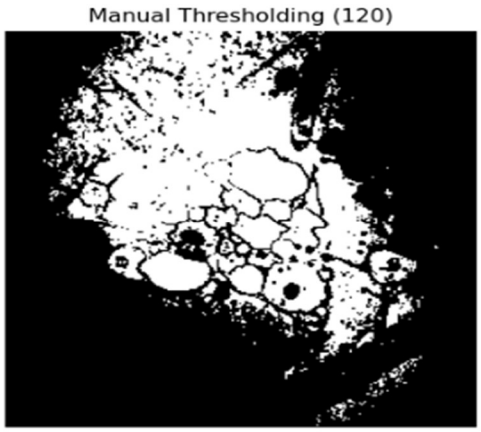
d) Intensity threshold chosen by the algorithm:

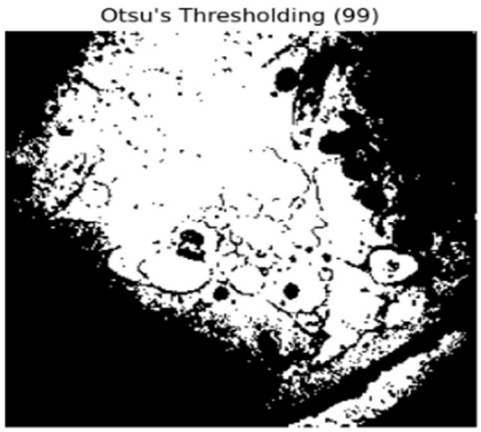
- Image One: Threshold: 132
- Image Two: Threshold: 99
- Image Three: Threshold: 155

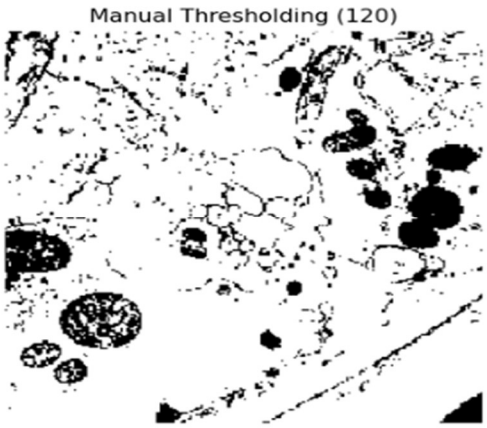
e) Resulting binary image produced by Manual and Otsu Algorithm:

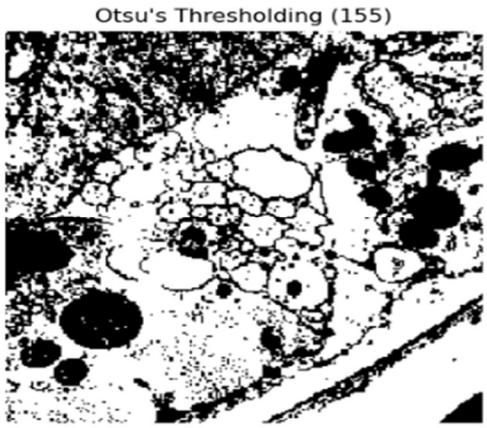
- Image 1, 2, 3 from up to down respectively











f) Discussion on Results:

- **Descent Results?** Yes, The automatic thresholding process separates most of the elements in the image that have distinct intensity characteristics. This separation is evident by the almost clear difference between foreground and background regions or the distinct object segmentation.
- **Elements Separated:** If the automatic thresholding produces a decent result, elements in the image that have different pixel intensities are effectively separated from each other. For

example, in a grayscale image, foreground objects or regions with higher pixel intensities may be separated from the background or regions with lower pixel intensities.

- **Improvements:** the automatic thresholding produces a decent result, but there are still some areas for improvement:
  a. **Fine-tuning the threshold value:** Adjusting the threshold value may lead to better separation, especially if there are subtle variations in pixel intensities within the image.
  b. **Preprocessing:** Applying preprocessing techniques such as noise reduction, contrast enhancement, or morphological operations may improve the quality of the image before thresholding, leading to better segmentation results.
  c. **Adaptive Thresholding:** Implementing adaptive thresholding techniques that dynamically adjust the threshold based on local image characteristics may lead to more robust segmentation results, especially in images with varying illumination or contrast.

Q3. Histogram Mapping:

Introduction to Histogram Mapping:

Histogram mapping, also known as histogram matching, is a technique used in image processing to adjust the distribution of pixel intensities in an image with respect to other (good)image or target distribution. The goal of histogram mapping is to modify the intensity distribution of an image to match a specified target distribution or to achieve a desired visual appearance. This technique is particularly useful for adjusting the visual appearance of an image to match a desired standard or to ensure consistency across multiple images.

The histogram mapping algorithm involves the following steps:

- Compute the histograms of the input image and the reference/target image.
- Calculate cumulative distribution functions (CDFs) for both histograms.
- Map the intensity values of the input image to the corresponding intensity values in the reference/target image based on the CDFs.
- Apply the mapped intensity values to the input image, effectively adjusting its histogram to match the target distribution.

Desing and Implementation:

- **Create PDF**: This function computes the probability density function (PDF) of an input image. It initializes a histogram with 256 bins (one for each possible pixel intensity). It iterates through each pixel in the input image, calculates the pixel intensity, and updates the corresponding histogram bin. Finally, it divides the histogram by the total number of pixels in the image to compute the PDF.
- **Create CDF**: This function computes the cumulative distribution function (CDF) from the input PDF. It initializes an array for the CDF. It iterates through the PDF values and computes the cumulative sum to obtain the CDF.

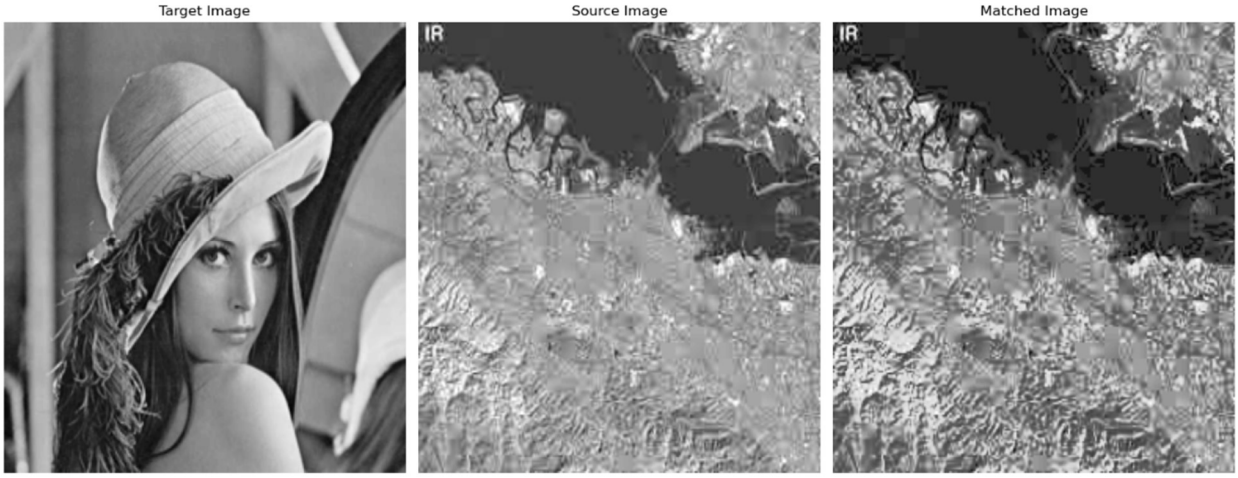
- **Histogram Matching**: This function performs histogram matching between the target and source images. It computes the PDF and CDF for both the target and source images. It calculates a mapping function to map the pixel intensities of the source image to those of the target image. This mapping function ensures that the CDFs of the source and target images are aligned. It applies the mapping function to the source image to generate the matched image.
- **Image Processing**: The code reads the target and source images from the specified file paths. It resizes both images to 256x256 pixels and converts them to grayscale. The pixel intensity values of both images are normalized to the range [0, 255] using min-max scaling. Histogram matching is performed between the target and source images. The matched image, along with the target and source images, is displayed using Matplotlib.
- **Visualization**: Matplotlib is used to visualize the target image, source image, and the matched image side by side in a single figure. Each image is displayed in a subplot, with titles indicating their respective roles.

Result of Histogram Matching:

- Histogram Matching with similar scene content:

- Histogram Matching with different scene content:

- Histogram Matching with totally different scene content:



Result Discussion:

1. Histogram Matching with Totally Different Scene Content:
- **Observation**: Histogram matching produced satisfactory results even though the scenes in the images were entirely different.
- **Explanation**: Despite the disparity in scene content, the histogram matching algorithm successfully adjusted the pixel intensity distribution of the source image to match that of the target image.
- **Conclusion**: Histogram matching is effective even when dealing with images with unrelated scene content, making it a versatile technique for various applications.

2. Histogram Matching with Different Scene Content:
- **Observation**: Histogram matching yielded good results for images with different scenes but some common elements.
- **Explanation**: Although the scenes differed, the presence of common elements facilitated the adjustment of the pixel intensity distribution, leading to visually appealing matched images.
- **Conclusion**: Histogram matching can handle images with diverse scene content, provided there are common visual elements that aid in the matching process.

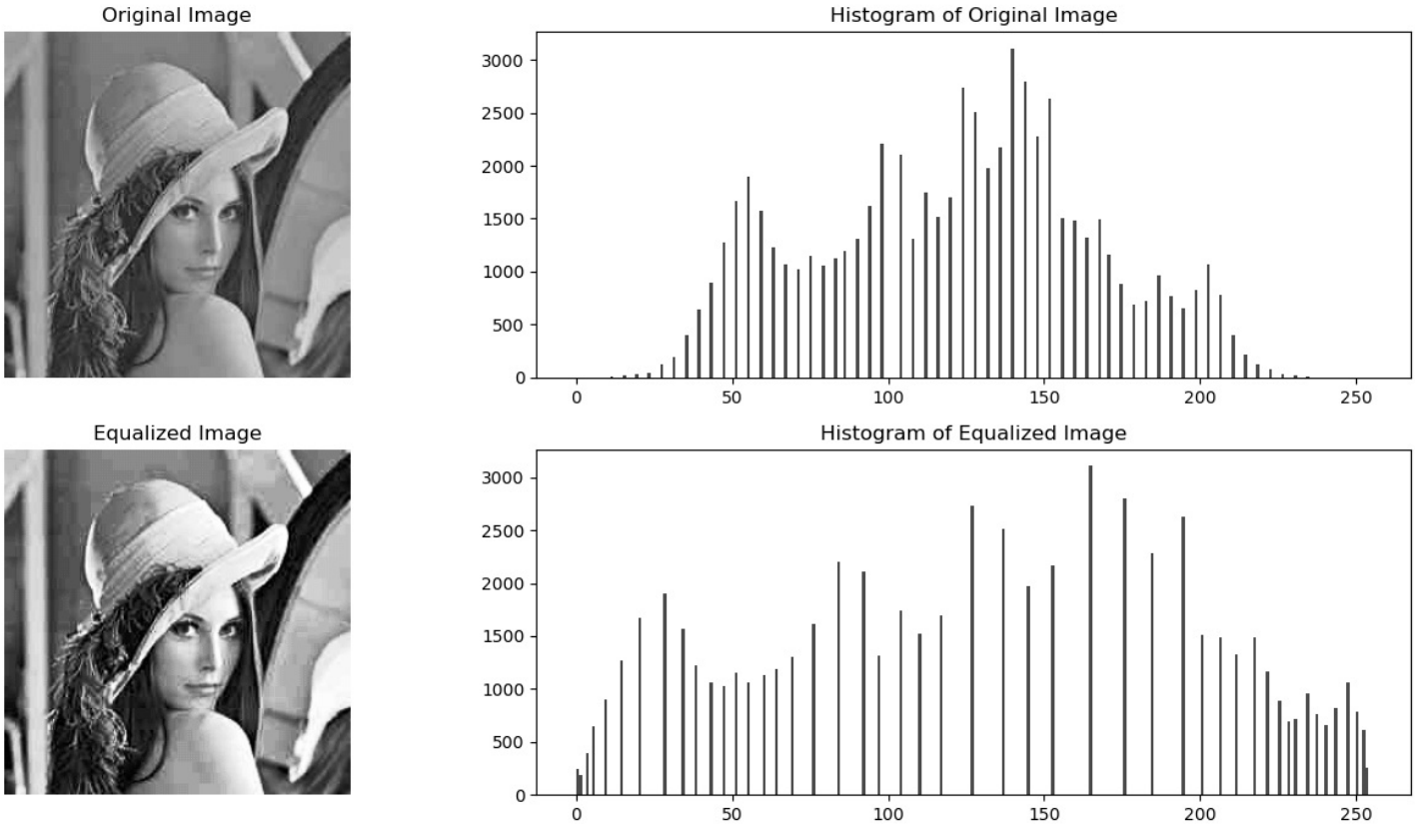
3. Histogram Matching with Same Scene Content:
- **Observation**: Histogram matching produced satisfactory results for images with many common elements in the scenes.
- **Explanation**: The histogram matching algorithm successfully aligned the pixel intensity distributions of the source and target images, resulting in matched images with improved visual consistency.
- **Conclusion**: Histogram matching can handle images with same scene contents, with similar common visual elements that aid in the matching process.

Q1. Implement histogram equalization and apply to image people.png

In [126]:

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

def create_pdf(im_in):
    # Initialize histogram
    histogram = np.zeros(256, dtype=np.float32)
    # Calculate histogram
    height, width = im_in.shape
    for y in range(height):
        for x in range(width):
            pixel_value = im_in[y, x]
            histogram[pixel_value] += 1

    # Calculate probability density function (pdf)
    pdf = histogram / (height * width)

    return pdf

def create_cdf(pdf):
    # Calculate cumulative distribution function (cdf)
#     cdf = np.cumsum(pdf)
    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):
    pdf = create_pdf(im_in)
    cdf = create_cdf(pdf)

    flattened_image = im_in.flatten()

    interpolated_values = np.zeros(flattened_image.shape)
    cdf *= 255
    for i in range(len(flattened_image)):
        interpolated_values[i] =  cdf[flattened_image[i]]

    equalized_im = interpolated_values.reshape(im_in.shape).astype(np.uint8)
    return equalized_im
#*****************************************************************************
# Read the image
image_path = 'people.png'
original_image = Image.open(image_path)
resized_image = original_image.resize((256, 256))
im = np.array(resized_image.convert('L'))
gray_image = ((im - np.min(im)) * (1 / (np.max(im) - np.min(im)) * 255)).astype('uint8')
# Perform histogram equalization
equalized_im = histogram_equalization(gray_image)

# Display original and equalized images
plt.figure(figsize=(14, 7))

# Plot the original image and its histogram
plt.subplot(2, 2, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 2, 2)
plt.hist(gray_image.ravel(), bins=256, color='black', alpha=0.7)
```

```
plt.title('Histogram of Original Image')

# Plot the equalized image and its histogram
plt.subplot(2, 2, 3)
plt.imshow(equalized_im, cmap='gray')
plt.title('Equalized Image')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.hist(equalized_im.ravel(), bins=256, color='black', alpha=0.7)
plt.title('Histogram of Equalized Image')

plt.tight_layout()
plt.show()


# pdf_original = create_pdf(gray_image)
# cdf_original = create_cdf(pdf_original)

# pdf_equalized = create_pdf(equalized_im)
# cdf_equalized = create_cdf(pdf_equalized)

# # Plot CDFs side by side
# plt.figure(figsize=(12, 6))

# # Original Image CDF
# plt.subplot(1, 2, 1)
# plt.plot(cdf_original, color='blue')
# plt.title('Original Image CDF')
# plt.xlabel('Pixel Intensity')
# plt.ylabel('Cumulative Probability')

# # Equalized Image CDF
# plt.subplot(1, 2, 2)
# plt.plot(cdf_equalized, color='red')
# plt.title('Equalized Image CDF')
# plt.xlabel('Pixel Intensity')
# plt.ylabel('Cumulative Probability')
```

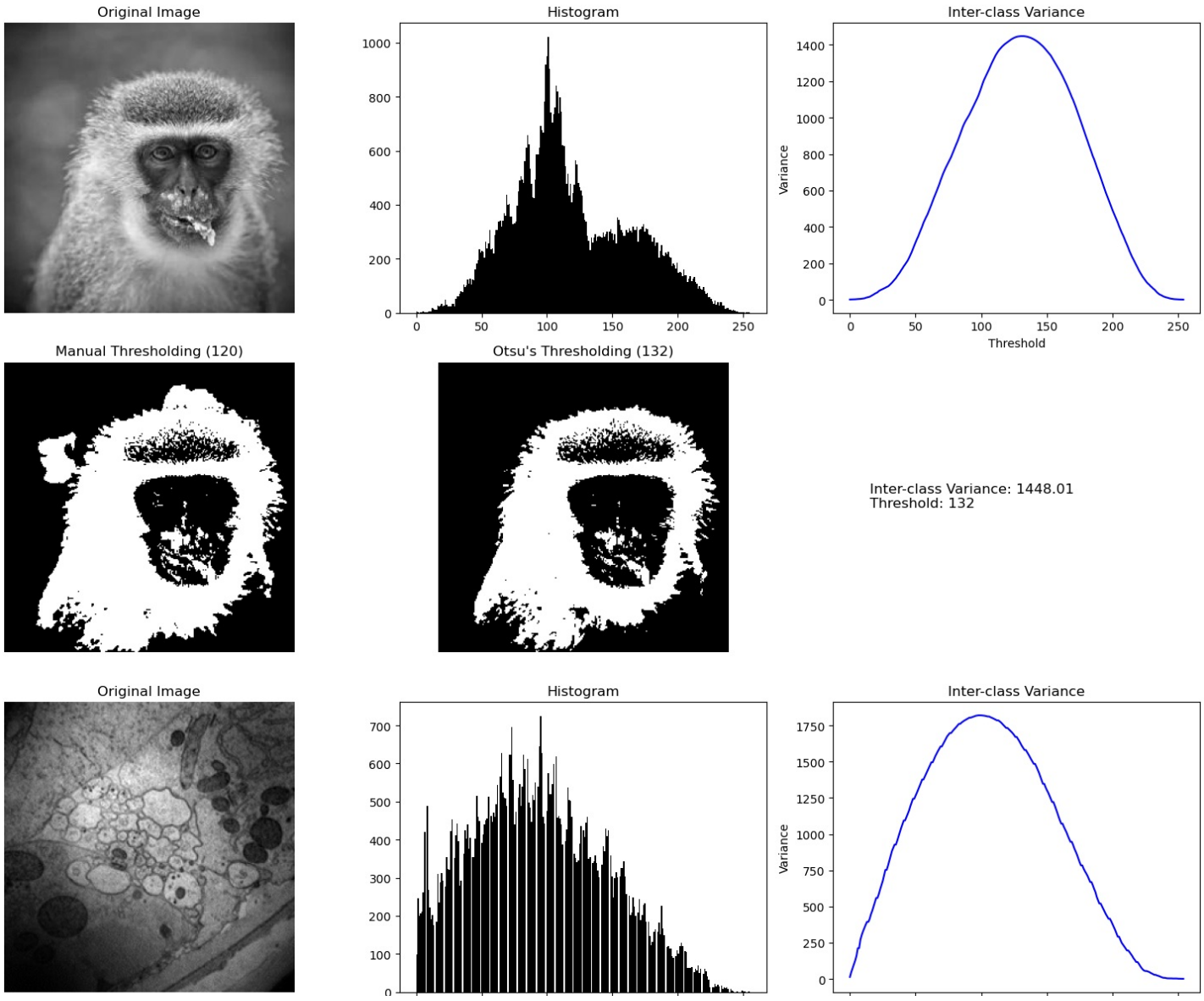


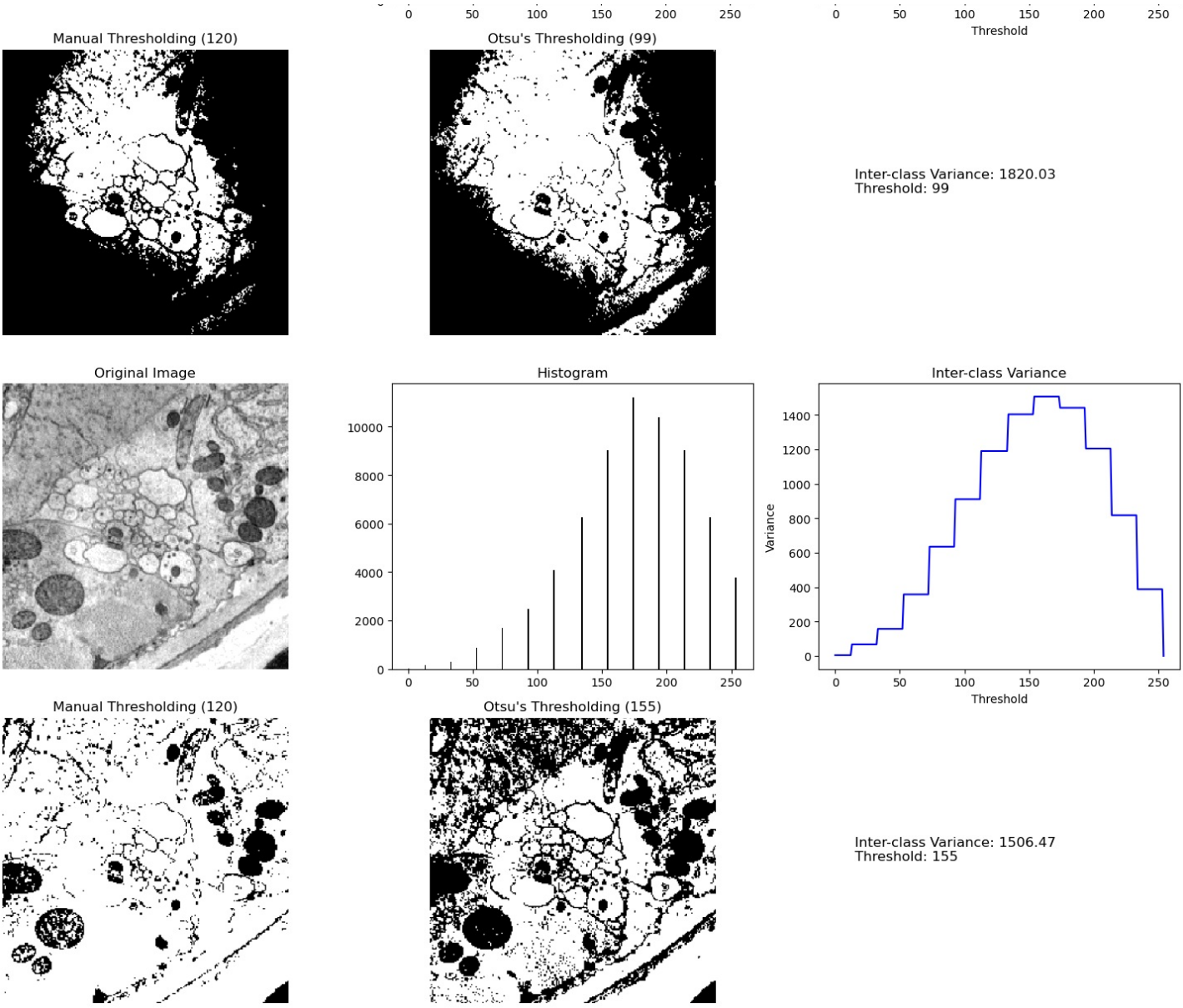
Q2.Implementing Otsu thresholding algorithm to generate binarized images

In [137]:

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

def manual_threshold(im_in, threshold):
    manual_thresh_img = np.where(im_in >= threshold, 255, 0).astype(np.uint8)
    return manual_thresh_img
#*****************************************************************************
def otsu_threshold(im_in, pdf):


    # Initialize variables
    max_var = 0
    threshold = 0
    inter_class_varaiances = []
    # Iterate through all possible thresholds
    for t in range(1, 256):
        # Compute class probabilities and means
        w0 = np.sum(pdf[:t])
        w1 = np.sum(pdf[t:])
        epsilon = 1e-10
        w0 = max(w0, epsilon)
        w1 = max(w1, epsilon)
        mu0 = np.sum(np.arange(t) * pdf[:t]) / w0
        mu1 = np.sum(np.arange(t, 256) * pdf[t:]) / w1

        # Compute inter-class variance
        var = w0 * w1 * (mu0 - mu1) ** 2
        inter_class_varaiances.append(var)

        # Update threshold if variance is greater
        if var > max_var:
            max_var = var
            threshold = t

    # Threshold image
    otsu_thresh_img = np.where(im_in >= threshold, 255, 0).astype(np.uint8)

    return otsu_thresh_img, threshold, max_var, inter_class_varaiances

#*****************************************************************************
# Read images
image_paths = ['b2_a.png', 'b2_b.png', 'b2_c.png']
images = []

for path in image_paths:
    original_image = Image.open(path)
    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
')
    images.append(im_uint8)

# Process each image
for i, image in enumerate(images):
    # Compute PDF
    pdf = create_pdf(image)

    # Manual thresholding
    manual_thresh_img = manual_threshold(image, 120)

    # Otsu's thresholding
    otsu_thresh_img, threshold, inter_class_variance, inter_class_varaiances = otsu_thre
shold(image, pdf)

    # Display results
    fig, axes = plt.subplots(2, 3, figsize=(15, 8))

    # Original Image and Histogram
```

```
    axes[0, 0].imshow(image, cmap='gray')
    axes[0, 0].set_title('Original Image')
    axes[0, 0].axis('off')
    axes[0, 1].hist(image.ravel(), bins=256, color='black')
    axes[0, 1].set_title('Histogram')

    # Manual Thresholding
    axes[1, 0].imshow(manual_thresh_img, cmap='gray')
    axes[1, 0].set_title('Manual Thresholding (120)')
    axes[1, 0].axis('off')

    # Otsu's Thresholding
    axes[1, 1].imshow(otsu_thresh_img, cmap='gray')
    axes[1, 1].set_title(f"Otsu's Thresholding ({threshold})")
    axes[1, 1].axis('off')

    # Inter-class Variance Plot
    thresholds = np.arange(255)
    variances = []
    for t in thresholds:
        _, _,_, variances = otsu_threshold(image, pdf)
    axes[0, 2].plot(thresholds, variances, color='blue')
    axes[0, 2].set_title('Inter-class Variance')
    axes[0, 2].set_xlabel('Threshold')
    axes[0, 2].set_ylabel('Variance')

    # Print inter-class variance and chosen threshold
    axes[1, 2].text(0.1, 0.5, f'Inter-class Variance: {inter_class_variance:.2f}\nThresh
old: {threshold}', fontsize=12)
    axes[1, 2].axis('off')

    plt.tight_layout()
    plt.show()
```





Q3.Histogram Matching

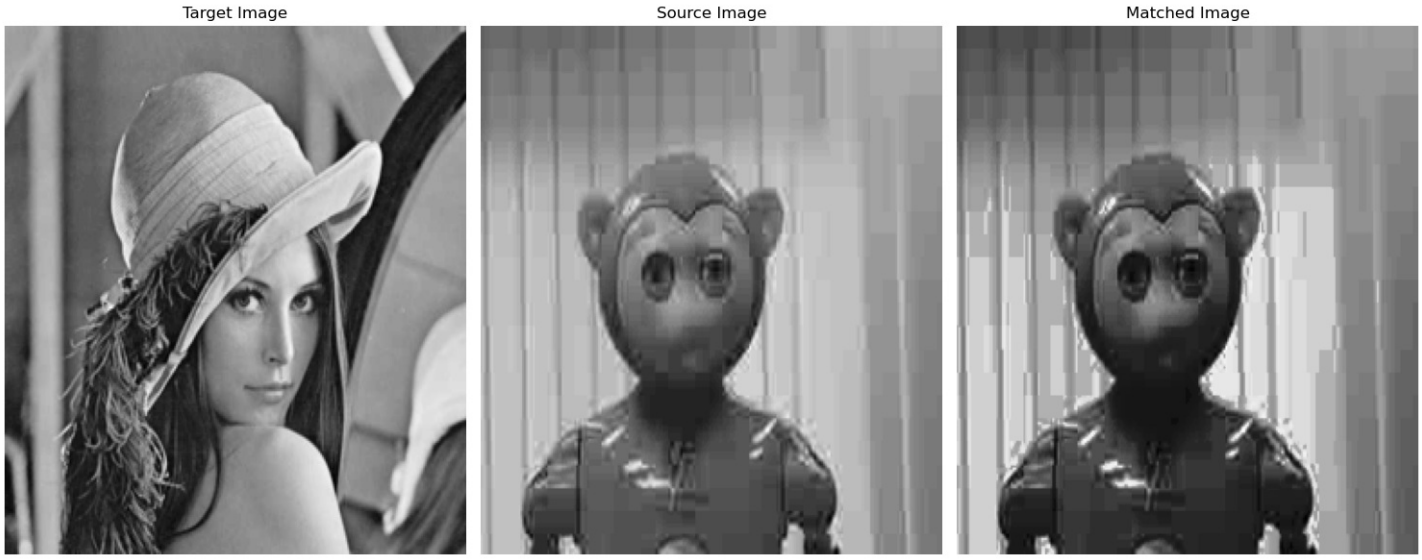
In [140]:

```
def create_pdf(im_in):
    # Initialize histogram
    histogram = np.zeros(256, dtype=np.float32)
    # Calculate histogram
    height, width = im_in.shape
    for y in range(height):
        for x in range(width):
            pixel_value = im_in[y, x]
            histogram[pixel_value] += 1

    # Calculate probability density function (pdf)
    pdf = histogram / (height * width)

    return pdf

def create_cdf(pdf):
    # Calculate cumulative distribution function (cdf)
#     cdf = np.cumsum(pdf)
    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_matching(target,source):
    target_pdf = create_pdf(target)
```

```
    target_cdf = create_cdf(target_pdf)

    source_pdf = create_pdf(source)
    source_cdf = create_cdf(source_pdf)

    flattened_source = target.flatten()
    flattened_test = source.flatten()
    print("Got Test and Source CDF")
    interpolated_values = np.zeros(flattened_source.shape)

    mapping_function = np.zeros(256, dtype=np.uint8)

    # Linear interpolation to compute mapping function
    for i in range(256):
        j = 255
        while j >= 0 and target_cdf[j] > source_cdf[i]:
            j -= 1
        mapping_function[i] = j

    # Apply mapping function to source image
    matched_image = mapping_function[source]

#     mapping_function = np.interp(source_cdf, target_cdf, np.arange(256))

#     # Apply mapping function to source image
#     matched_image = mapping_function[source]

    return matched_image.astype(np.uint8)

#*****************************************************************************
image_path = 'target.png'
original_image = Image.open(image_path)
resized_image = original_image.resize((256, 256))
im = np.array(resized_image.convert('L'))
target = ((im - np.min(im)) * (1 / (np.max(im) - np.min(im)) * 255)).astype('uint8')

image_path = 'source3.png' # 'source2.png' 'source.png'
original_image = Image.open(image_path)
resized_image = original_image.resize((256, 256))
im = np.array(resized_image.convert('L'))
source = ((im - np.min(im)) * (1 / (np.max(im) - np.min(im)) * 255)).astype('uint8')

matched_image = histogram_matching(target,source)

plt.figure(figsize=(15, 6))

# Display Target Image
plt.subplot(1, 3, 1)
plt.imshow(target, cmap='gray', aspect='auto')  # Specify aspect ratio
plt.title('Target Image')
plt.axis('off')

# Display Source Image
plt.subplot(1, 3, 2)
plt.imshow(source, cmap='gray', aspect='auto')  # Specify aspect ratio
plt.title('Source Image')
plt.axis('off')

# Display Matched Image
plt.subplot(1, 3, 3)
plt.imshow(matched_image, cmap='gray', aspect='auto')  # Specify aspect ratio
plt.title('Matched Image')
plt.axis('off')

plt.tight_layout()  # Adjust layout for better visualization
plt.show()
```

```
Got Test and Source CDF
```

Got Test and Source CDF



Got Test and Source CDF

In [ ]: