CS-GY 6643 I Computer Vision
Spring 2023

Professor Guido Gerig

Project 1.
Remon Roshdy rr3531@nyu.edu

Problem: Implementation of Histogram Equalization

●Include a brief introduction and description of how histogram equalization works.

The histogram of an image is a plot that shows the frequency of occurrence of each pixel intensity in the image, it counts how many pixels each intensity value have.

Histogram equalization is used to enhance the contrast of an image by redistributing the pixel intensities in such a way that the resulting histogram becomes approximately uniform.

We can implement the histogram equalization by implementing the following steps.

1- compute probability distribution function.

The PDF represents the probability of a pixel randomly selected from the original image having a specific intensity value. It tells you how likely it is to encounter a pixel with a particular intensity in the original image. A high PDF value for a specific intensity indicates that many pixels in the image have that intensity.

2- Compute the cumulative distribution function.

The CDF is computed from the histogram. It represents the cumulative probability of pixel intensities. It shows how the cumulative percentage of pixels with a particular intensity value increases from the minimum to the maximum intensity.

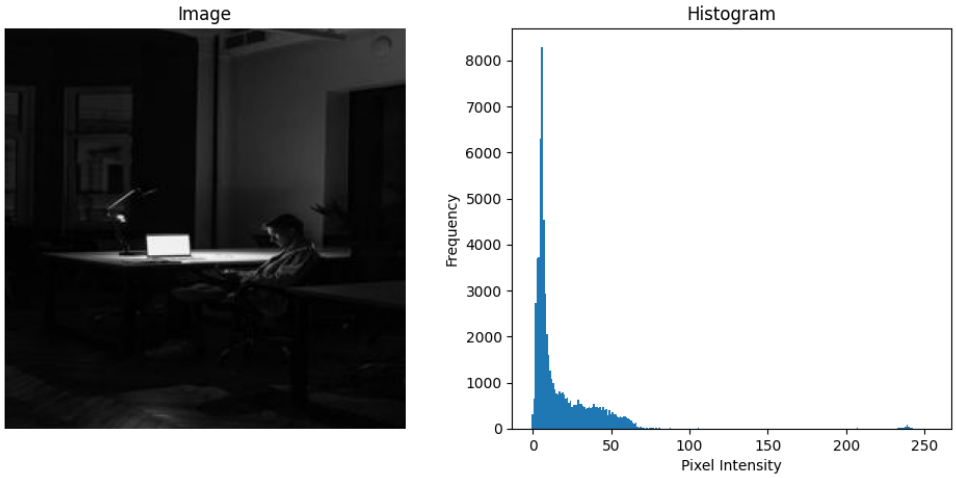
3- Equalize the image.

The main idea of histogram equalization is to map the original pixel intensities to new values such that the CDF of the new values is a straight diagonal line from 0 to 1. This means that pixel values will be more evenly distributed across the entire intensity range, improving contrast of the new equalized image.

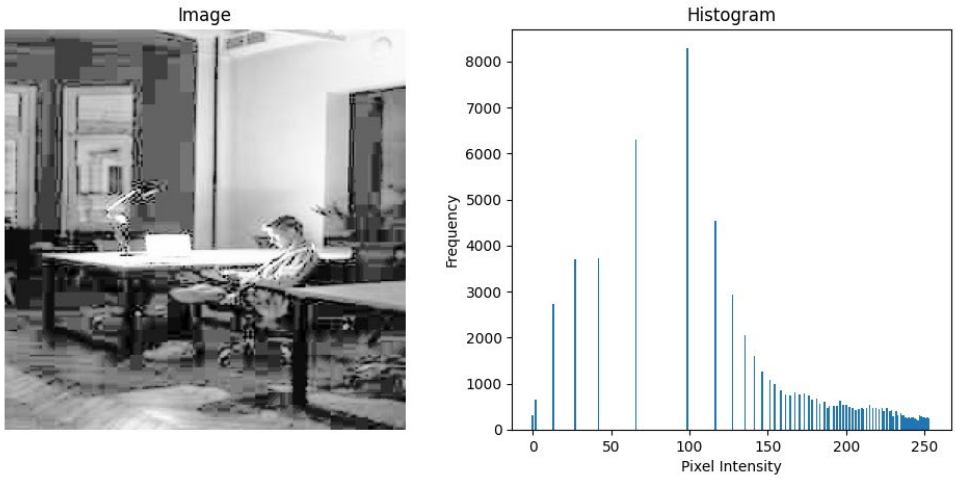
PDF of the Equalized Image: After histogram equalization, the PDF represents the probability of a pixel randomly selected from the equalized image having a specific intensity value. Ideally, in the equalized image, the PDF should be as uniform as possible, meaning that all intensities are equally likely. This uniform distribution enhances the image's contrast.

● Show **indoors.png** before and after histogram equalization, and the corresponding histograms (PDFs).

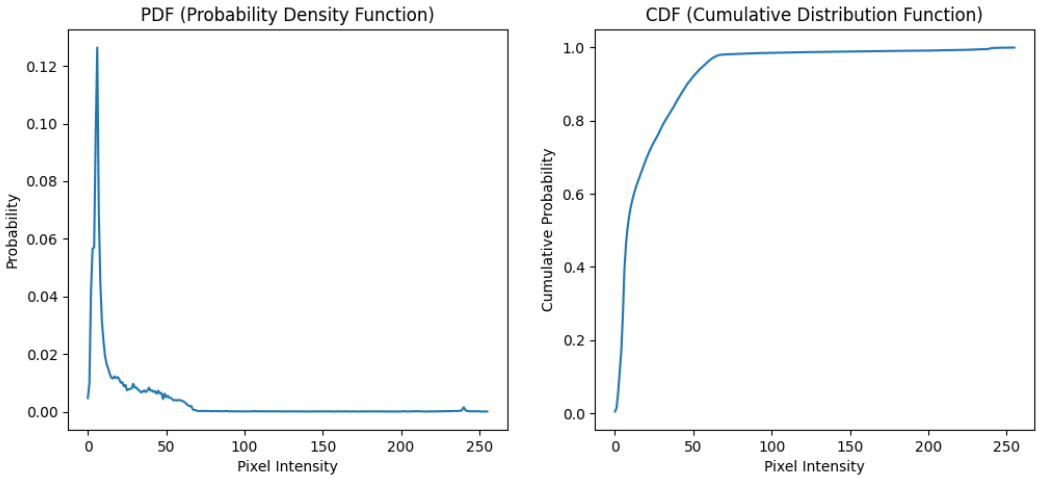
Original image and its Histogram



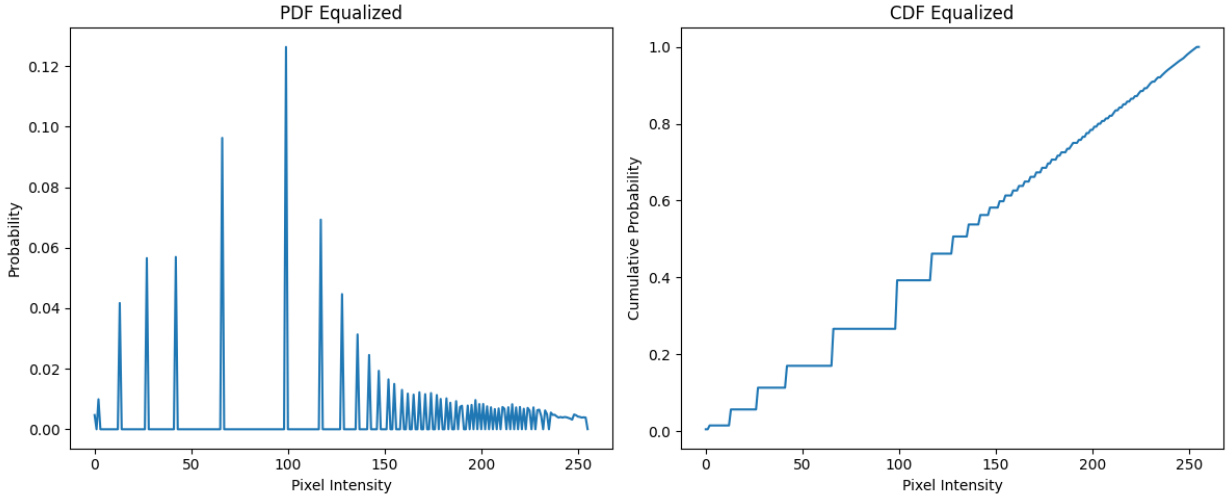
Equalized image & histogram



Original image Pdf & cdf



Equalized image pdf & cdf



● Discuss how the image and histogram have changed and connect it back to your description in 1).

Original image was dark and most of the pixels intensities squeezed toward the left side of the histogram. The pixel intensities value in the range of 0 : 70 which lines up with the original image being dark and we can see that the original image histogram is skewed to the left.

For the equalized image. We can see the image has become brighter and has higher contrast. That’s why the histogram now spread over the entire range of pixel intensities.

The Probability density function tells us how likely it is to encounter a pixel with a particular intensity in the original image. A high PDF value for a specific intensity indicates that many pixels in the image have that intensity which is in line with our original dark image.

the PDF should be as uniform as possible, meaning that all intensities are equally likely. This uniform distribution enhances the image's contrast. But achieving a perfectly uniform PDF in the equalized image is not always possible, especially if the original image has unique characteristics or noise. And as we can see after image equalization the pdf function has so many peaks for all the pixel intensities which indicates that the equalization process did not achieve a uniform distribution of pixel intensities, and it may not be linear as the original image has some regions with specific intensity values that are densely populated, or the original image contains noise or artifacts that the equalization process cannot fully remove.

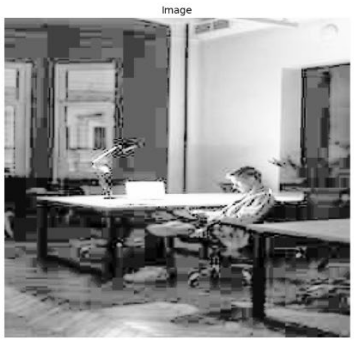
● Show the cumulative distribution function before and after histogram equalization in a side-by-side figure.

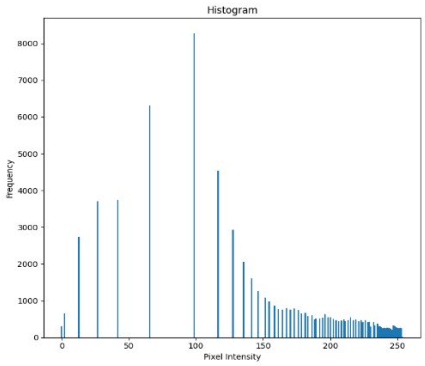
●Describe what you see. Explain the shape of each CDF and relate it back to image contrast and intensity
histogram shape.

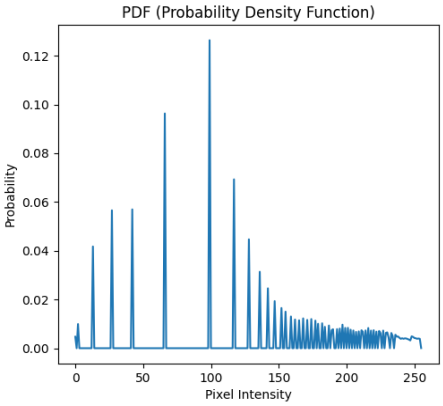
The CDF for the original image has a curve initially increasing slowly and then becoming almost a straight line after a certain point x= 50 indicates that the majority of pixel intensities in the original image are concentrated in the darker range. The nearly straight part of the CDF beyond that point indicates that the brighter intensities are more uniformly distributed.
This kind of CDF is typical for images with predominantly darker regions and fewer brighter regions.

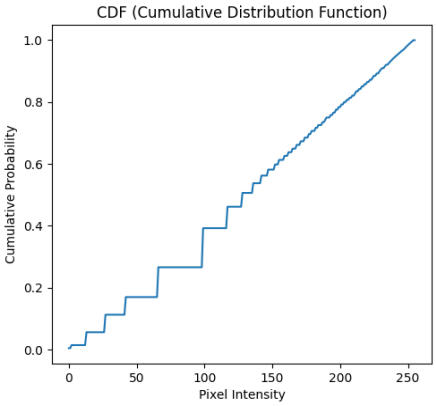
From x = 0 to x = 50, the curve has small steps in the y-direction. This indicates that the CDF is increasing slowly, which means that pixel intensities in the darker range have been spread out more evenly. From x = 50 to x = 150, the curve features bigger steps in the y-direction. This indicates a more rapid increase in cumulative probability which implies that the middle range of pixel intensities has been redistributed and expanded, resulting in better contrast in that range.
For the rest of the intensities range, the curve consists of very tiny steps and eventually becomes a straight line, which indicates that the brighter pixel intensities are now more uniformly distributed across the entire intensity range. The near-straight line indicates that pixel intensities in the brighter range are more balanced. Ultimately this indicates the successful histogram equalization process as the goal is to make the cdf of the equalized image as close to a straight diagonal line from 0 to 1 which significantly enhances the images contrast.

● Reapply the histogram equalization procedure on the corrected image. Show and discuss the results.

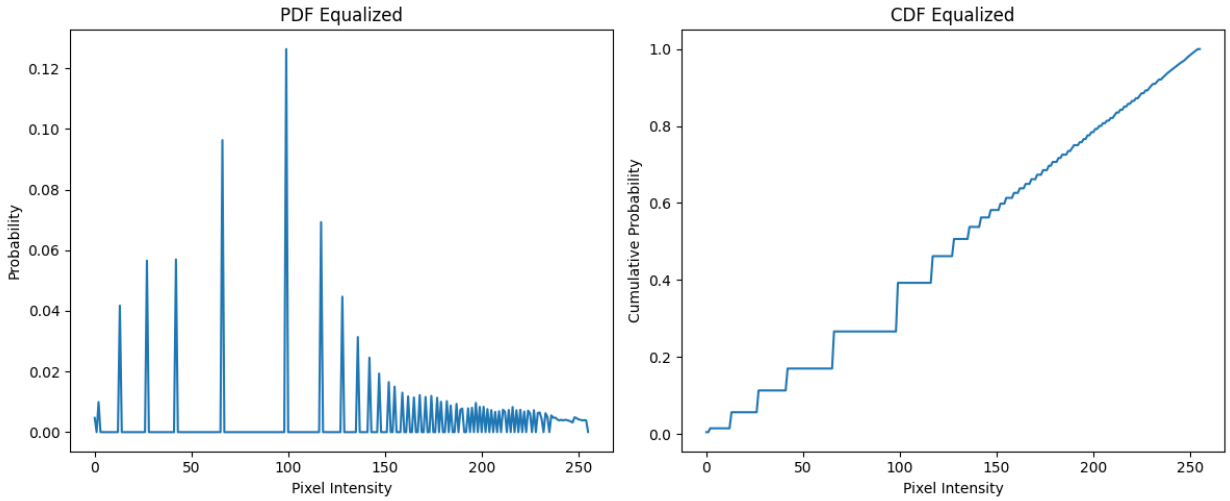


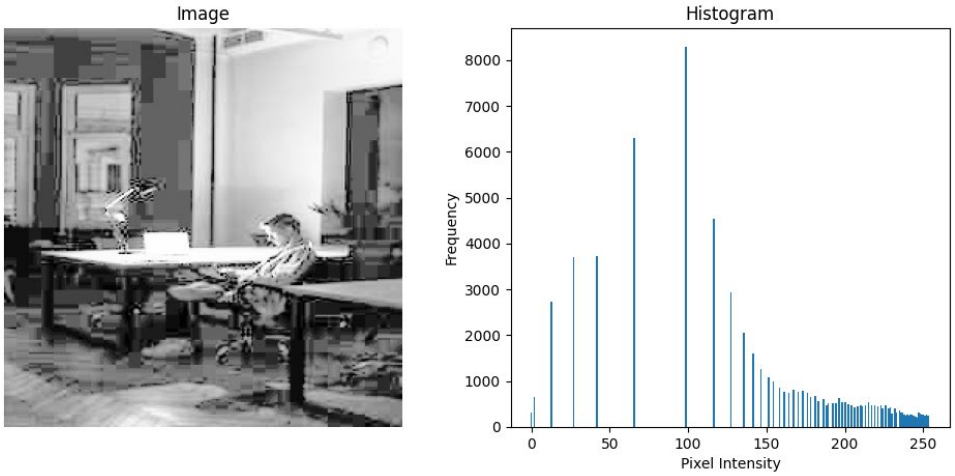






Equalized image 2nd time



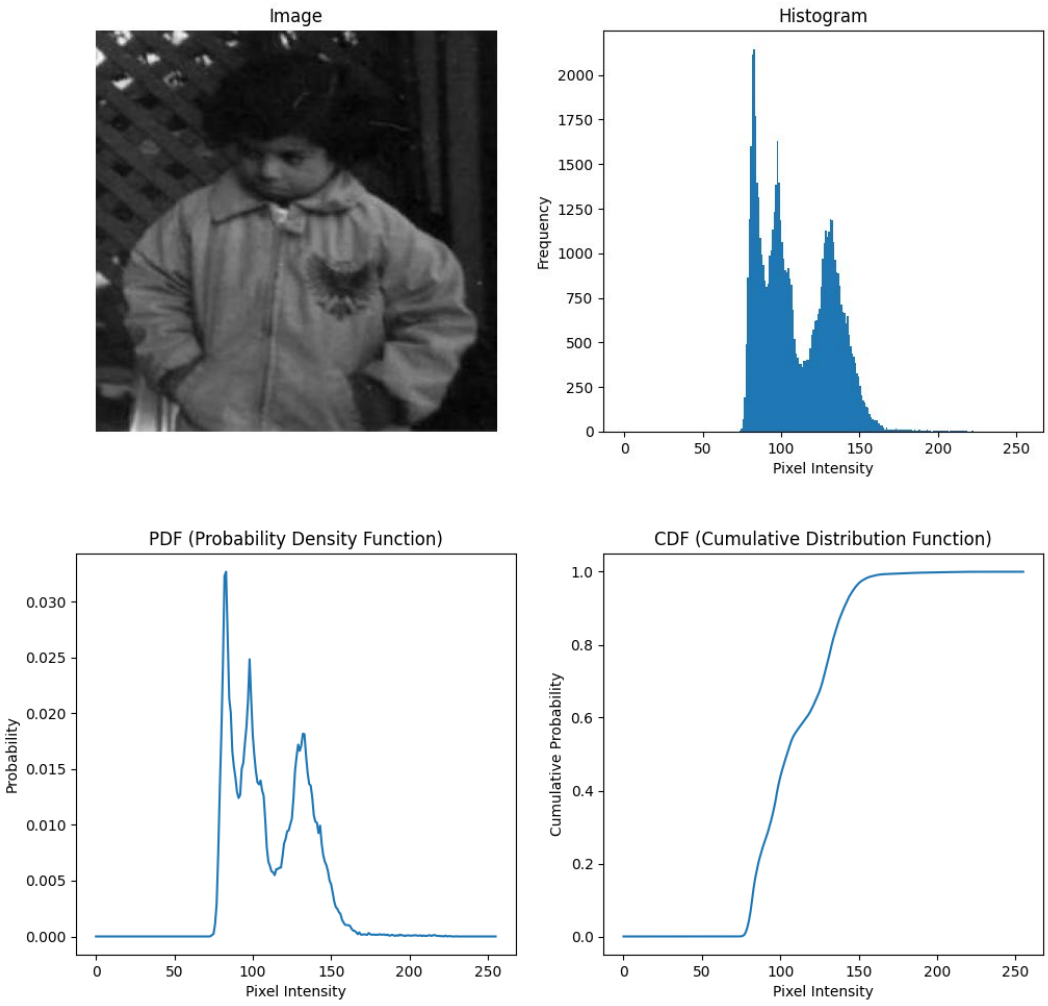


From the above graphs, noted that PDF, CDF and the Histogram of the equalized image did not change after reapplying the Histogram equalization for the second time.

Therefore I can conclude that applying histogram equalization to an image that already undergone the same process did not lead to any extra improvement in the contrast of the image. This is because histogram equalization is designed to enhance contrast by redistributing pixel intensities once, and doing it repeatedly may not yield significant benefits and may actually lead to unwanted results.

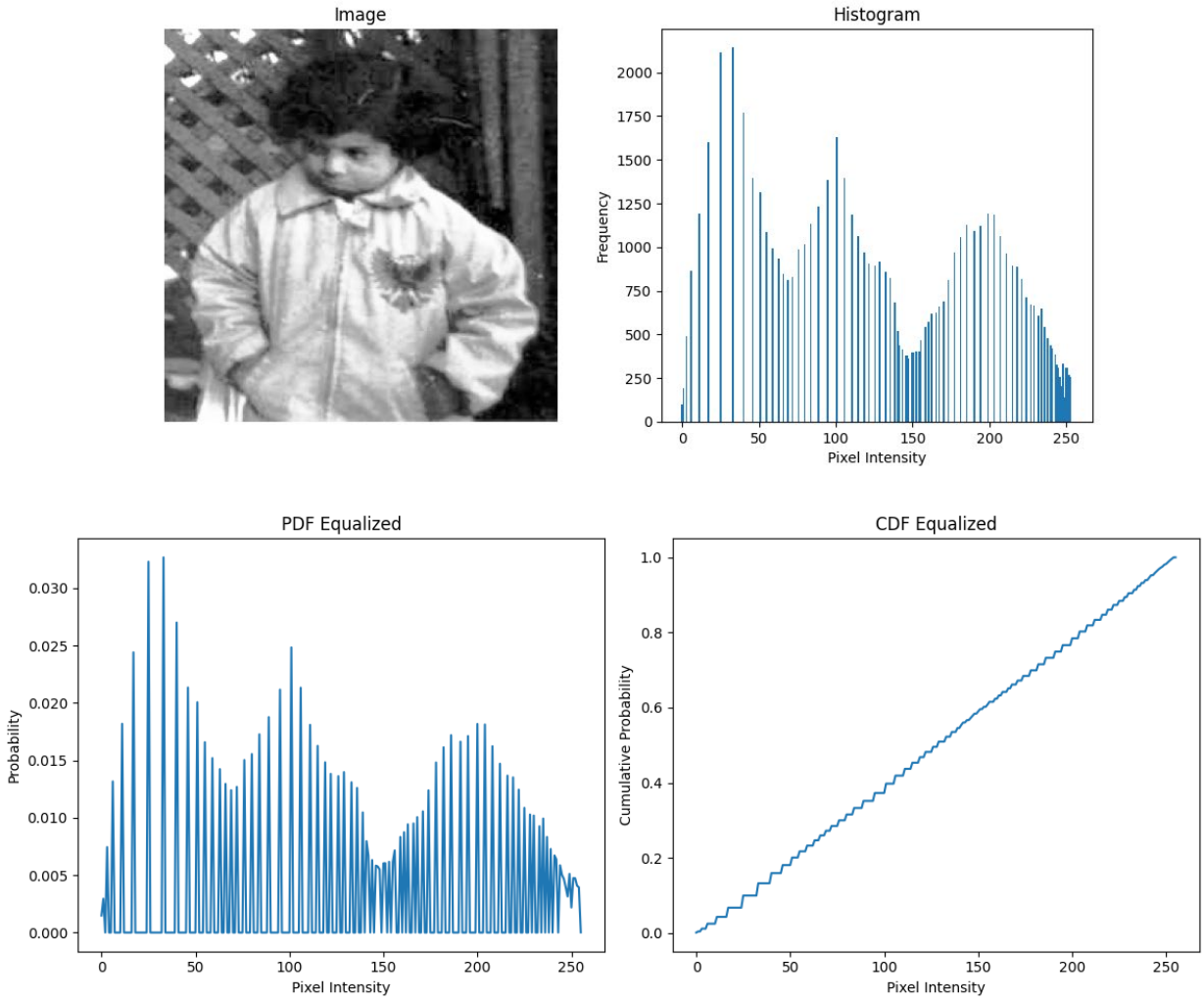
● Apply histogram equalization to another low contrast image (greyscale). Show and discuss the results.

.



As we can see from the above histogram, the original image has low contrast and we can see all pixels intensities are condensed toward the middle of the histogram. Which also appears on the peaks of pixel intensities in the probability density function. The CDF also shows that for pixel intensity from 0 – 80 is steady and straight line as there is not much of pixels in that range and gradually increases to reach the peak within the range 80 – 150 where the majority of pixel intensities in the original image are concentrated in that middle range. The rest of pixel intensities are almost uniformly distributed.

Equalized image.



The equalized image is now more brighter and clear than the original image and its histogram is uniformly distributed and the photo is more appealing than the original low contrast image.

Again achieving a perfectly uniform PDF in the equalized image is not always possible, especially if the original image has unique characteristics or noise. And as we can see after image equalization the pdf function has so many peaks for all the pixel intensities

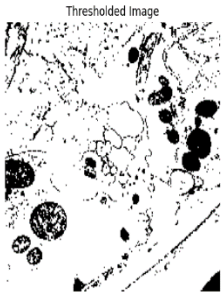
which indicates that the equalization process did not achieve a uniform distribution of pixel intensities.

The cdf for the equalized image is nearly straight line which means pixel intensities are more balanced and spread over the entire range. Which means the contrast of the image enhanced and our histogram equalization have successfully enhances the image.

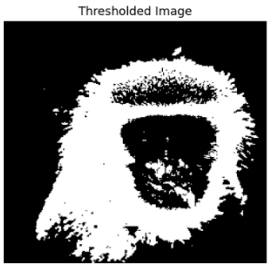
2- Manual Theshold

Write code to generate a **binary image** using a **manually chosen threshold**. Show the resulting binary image given a threshold of your choice, and add the threshold value to the report.

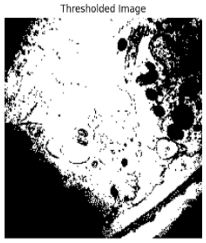
I applied the below threshold to multiple images and the results are below.



Threshold 150



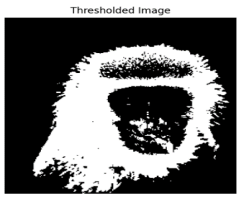
Threshold 130

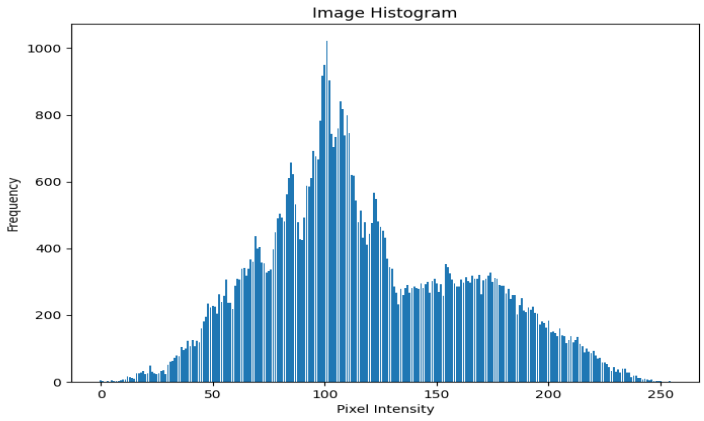


Threshold 70

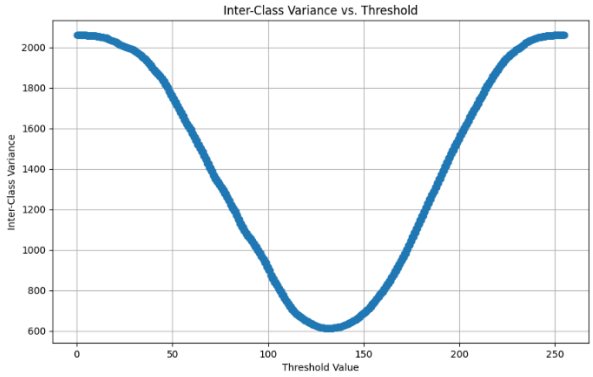
Manual thresholding in binary image techniques involves selecting a threshold value by hand to separate objects from the background in an image. This method can be effective in many cases as it is easy to implement and simple and we got to choose the the threshold, But as we can see from the results above It may not work well for images with complex lighting, variations in contrast, or noisy backgrounds. In such cases, an adaptive thresholding approach may be more effective.

Otsue Method Automatic Threshold



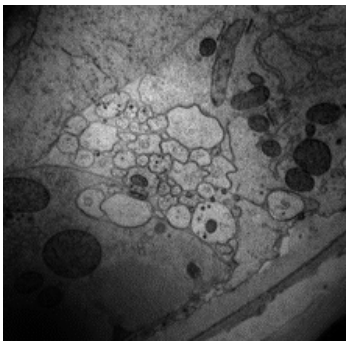


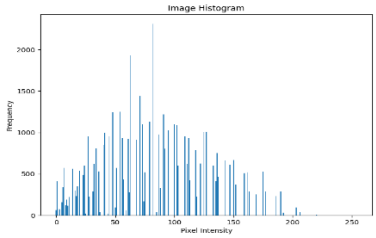
.

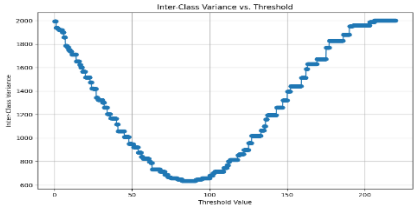


class variance : 613.06 threshold 131.1

b2_b image original



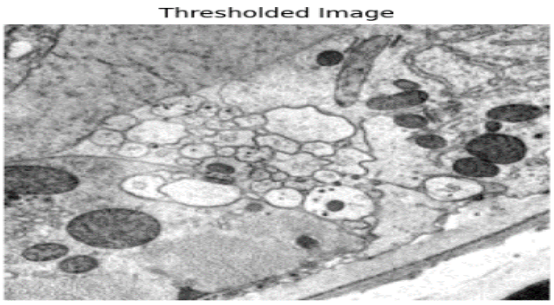


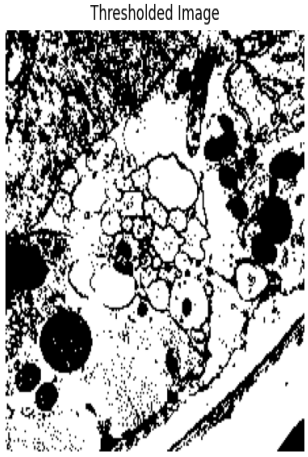


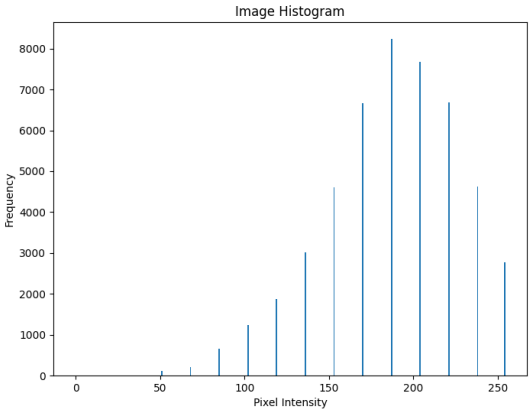
class variance , color threshold

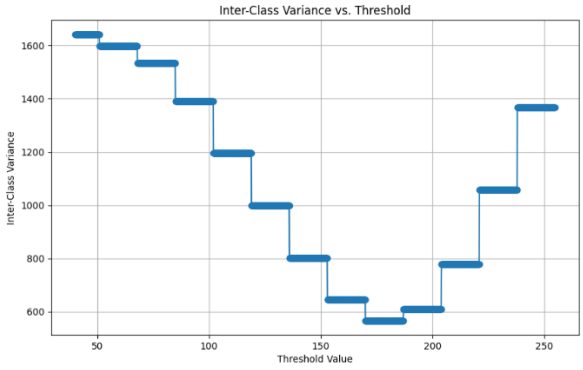
635.96 85.1

B2_c original image









class variance , threshold

trace: 566.166 170.00000000000185

Otsu's algorithm produces a decent result in many cases. However, its effectiveness depends on the characteristics of the image. Otsu's algorithm can produce good results when the image has a clear bimodal histogram, where pixel intensities are divided into two distinct classes like foreground and background or when the image has distinct objects with distinct pixel intensity.

Otsu's algorithm may not produce good results with non-Bimodal Distributions, in cases where the image has a unimodal or nearly unimodal distribution of pixel intensities, the algorithm may not find a threshold that effectively separates the objects from the background. Also, for images with complex scenes, textures, or multiple objects with overlapping intensity distributions, Otsu's algorithm may not be able to find a single threshold that separates all regions of interest. Also, images with noise can affect the histogram and the accuracy of the threshold.

Possible Improvements:

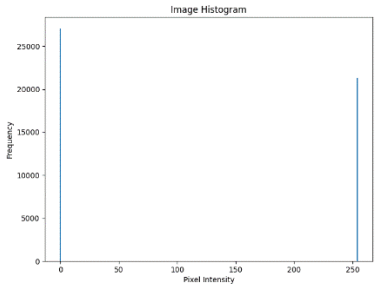
Adaptive Thresholding: In cases where Otsu's algorithm may not perform well, adaptive thresholding methods can be used. These methods calculate a threshold value for each local region of the image, considering local variations in pixel intensities. Also, we can improve the performance of Otsu's algorithm by applying preprocessing techniques such as noise reduction, contrast enhancement, and illumination correction. Also, worth to note that sometimes applying Otsu algo to an image that has already undergone thresholding or processing may or may not produce better results.

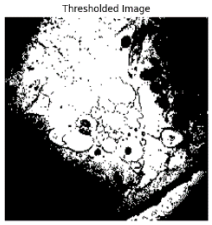
For b2_a the monkey image, Otsu algo provided a decent result but if we notice the face of the monkey is black which matches the background color that made it harder for the algo and we can see that in the image histogram since we do not have 2 very distinct peaks of pixel intensities but Otsu algorithm provided a more decent result than the manual threshold selection.

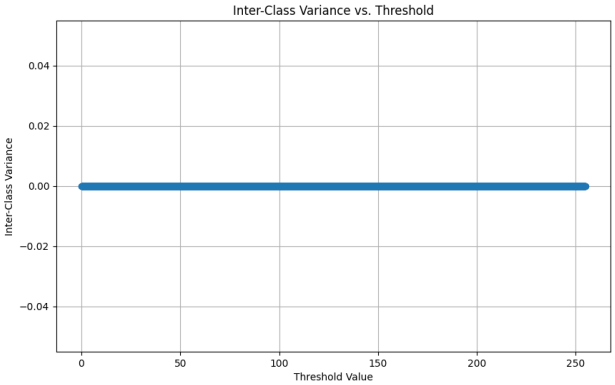
For b2_b the thresholded image provided by otsu algo also has a decent result but again I can see the challenge for the algo to provide a perfect result as from the histogram it is obviously not a biomodal histogram and the background and foreground objects overlap and there are some noise in the photo with dark edges. I tried to run Otsu algorithm on the result of the b2_b image and here is the result

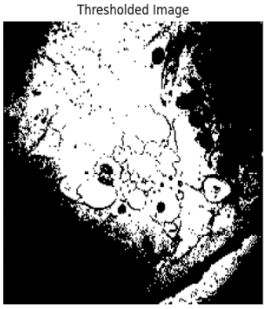
No difference in the resulted image at all as you can see the algorithm in the first run binarized the image completely that’s shown by the histogram and inter class variance.

class variance 0:0 threshold .1









For b2_c image

Again Otsu Algo did a good job but we can see the challenge for the algo as you can see the histogram skewed to the right and there are multiple overlapping colors make it difficult to have background and foreground separation of the pixel.

Histogram Matching.

Figure 1)

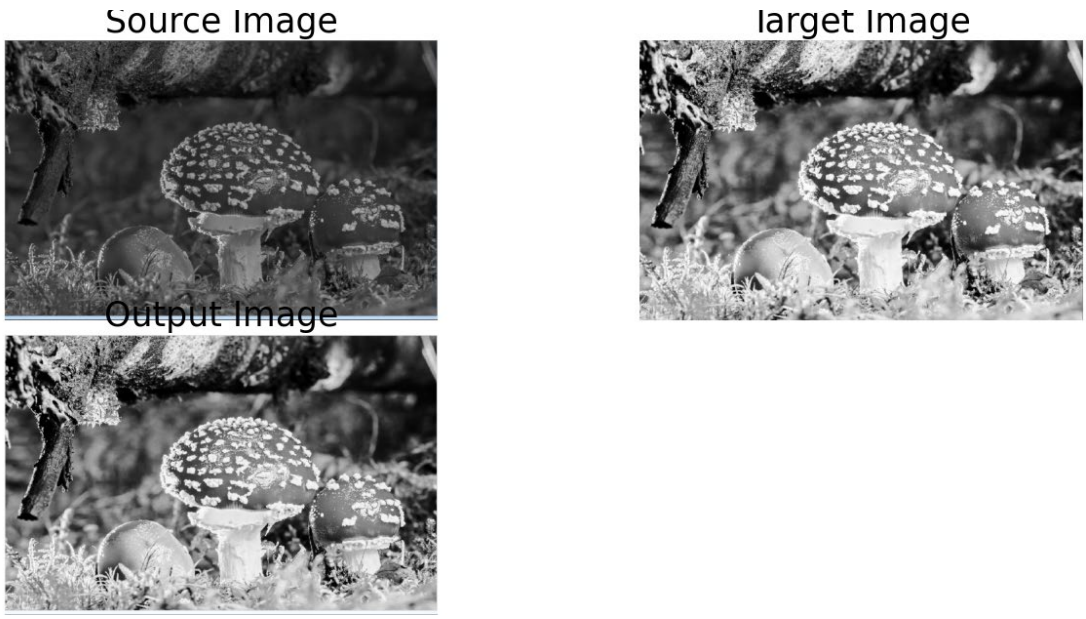


Figure 2)

Histogram matching is a technique used in image processing to improve the contrast and visual quality of an image by adjusting the distribution of pixel intensities in its histogram.

Process of Histogram Matching:

The process begins by calculating the histogram of the input image. The next step is to compute the cumulative distribution function (CDF) from the histogram. The CDF provides information about the cumulative probability of encountering a pixel with intensity less than or equal to a particular value.

To improve the image, you can specify a target histogram. The target histogram represents the desired distribution of pixel intensities that you want to achieve in the output image. This can be based on another reference image or a desired distribution. Compute the CDF of the target histogram to get the cumulative distribution function of the desired distribution.

Create a mapping function that maps the CDF of the input image to the CDF of the target histogram. This mapping function is used to adjust the pixel values in the input image to match the desired distribution.

Apply the Mapping: Apply the mapping function to each pixel in the input image. This remaps the pixel values according to the desired distribution.

Output Image: The resulting image has improved contrast and a histogram that matches the desired distribution.

I have used the above process and applied to figure 1 where the source image is darker and our target image for the same scene is brighter and look better. My goal is to make the darker image as good as the brighter image. So I applied Histogram matching and mapped the original image pixel intensities to the good image desired distribution. And as we can see in figure 1 the output image improve the original image almost as good as we wanted.

Another application of Histogram matching where I tried to capture the sun theme on a night or evening image to get the sun color effect on the night image.The histogram matching process adjusts the color values in the nighttime image specifically, the color values in the nighttime image are modified to have more warmth, typically shifting them toward reddish and orange tones. This adjustment can make the image appear as though it was captured under sunlight.

Code for Question 1.

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

# Once you close the figure , next figure will show
def create_pdf(im_in):
    # Calculate the histogram of pixel intensities
    histogram, _ = np.histogram(im_in, bins=256, range=(0, 256))

    # Calculate the total number of pixels in the image
    total_pixels = im_in.size

    # Calculate the probability density function (PDF)
    pdf = histogram / total_pixels

    return pdf


def create_cdf(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)  # Compute the PDF of the input image
```

```
    cdf = create_cdf(pdf)  # Compute the CDF from the PDF

    # Normalize the image using the CDF
    equalized_im = cdf[im_in]
    equalized_im = (((equalized_im - np.min(equalized_im)) / (np.max(equalized_im) -
np.min(equalized_im)) * 255)
    .astype( 'uint8'))
    #equalized_im = (equalized_im * 255).astype('uint8')

    return equalized_im
def plot_image_and_histogram(image):
    # Create subplots
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    # Plot the image on the left subplot
    ax1.imshow(image, cmap='gray')
    ax1.set_title('Image')
    ax1.axis('off')

    # Calculate and plot the histogram on the right subplot
    hist, bins = np.histogram(image, bins=256, range=(0, 255))
    ax2.bar(bins[:-1], hist, width=1)
    ax2.set_title('Histogram')
    ax2.set_xlabel('Pixel Intensity')
    ax2.set_ylabel('Frequency')

    # Display the combined subplots
    plt.show()

# Load and convert the image
image_path = 'IMG_5437.JPEG'
image = Image.open(image_path)
if image.mode == 'RGB':
    image = image.convert('L')
# resize image
#image = image.resize((256, 256))
# Convert the image to a NumPy array with the appropriate data type
image_array = np.array(image, dtype=np.uint8)

# Plot original image and its histogram
plot_image_and_histogram(image)

# ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
# Calculate PDF and CDF for the original image
pdf = create_pdf(image_array)
cdf = create_cdf(pdf)
```

```
# Plot the PDF and CDF
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Plot the PDF on the left subplot
ax1.plot(pdf)
ax1.set_title('PDF (Probability Density Function)')
ax1.set_xlabel('Pixel Intensity')
ax1.set_ylabel('Probability')

# Plot the CDF on the right subplot
ax2.plot(cdf)
ax2.set_title('CDF (Cumulative Distribution Function)')
ax2.set_xlabel('Pixel Intensity')
ax2.set_ylabel('Cumulative Probability')

# Display the combined subplots
plt.show()

#++++++++++++++++++++++++++++++++++++++++++++++++


# Perform histogram equalization
equalized_image = histogram_equalization(image_array)

output_image = Image.fromarray(equalized_image)

output_image.save('hala.jpeg')

# Plot the equalized image and its histogram
plot_image_and_histogram(equalized_image)



# Calculate PDF and CDF for the equalized image
pdf_equalized = create_pdf(equalized_image)
cdf_equalized = create_cdf(pdf_equalized)

# Plot the PDF and CDF for the equalized image
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Plot the PDF of the equalized image on the left subplot
ax1.plot(pdf_equalized)
ax1.set_title('PDF Equalized')
ax1.set_xlabel('Pixel Intensity')
ax1.set_ylabel('Probability')
```

```
# Plot the CDF of the equalized image on the right subplot
ax2.plot(cdf_equalized)
ax2.set_title('CDF Equalized')
ax2.set_xlabel('Pixel Intensity')
ax2.set_ylabel('Cumulative Probability')

# Display the combined subplots
plt.tight_layout()
plt.show()
```

Code for question 2. Manual Threshold

```
import numpy as np
import imageio
import matplotlib.pyplot as plt
from PIL import Image
import time
def manual_threshold(img, l):
    # create new image with 1 that has same dimension as the tested image to reserve
space
    new_img = np.ones(img.shape).astype(np.uint8)
    # new_img has all pixels of 1s
    # setting 0 to the pixel above threshold in org image
    new_img[np.where(img < l)] = 0
    manual_thresh_img = Image.fromarray(new_img * 255)
    manual_thresh_img.save('after.png')
    return manual_thresh_img


def plot_Histogram(figure):
    plt.figure(figsize=(8, 6))
    plt.bar(bins[:-1], figure)
    plt.title('Image Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.show()

def plot_image(image):
    plt.imshow(manual_thresh_img, cmap='gray')
    plt.title('Thresholded Image')
    plt.axis('off')
    plt.show()

#  import the image and save it as numpy array for processing
#image_path = 'b2_a.png'
image_path = 'b2_b.png'
#image_path = 'b2_c.png'

#image = Image.open(image_path).convert('L')

image_array = np.array(Image.open(image_path).convert('L'), dtype=np.uint8)

# create the histogram of the image.
hist, bins = np.histogram(image_array, bins=255, range=(0, 255))
```

```
# Plot the histogram
plot_Histogram(hist)

# Threshold the image and save it as binary (0 and 255)
manual_thresh_img = manual_threshold(image_array, 70)  # Apply the threshold

# Display the thresholded image
plot_image(manual_thresh_img)
```

Otsu Method Code

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

from skimage.io import imread
from skimage.color import rgb2gray
from skimage.filters import threshold_otsu

def threshold_otsu_impl(img, nbins=0.1):
    # validate grayscale
    if len(img.shape) == 1 or len(img.shape) > 2:
        print("must be a grayscale image.")
        return

    # validate multicolored
    if np.min(img) == np.max(img):
        print("the image must have multiple colors")
        return

    all_colors = img.flatten()
    total_weight = len(all_colors)
    least_variance = -1
    least_variance_threshold = -1

    # create an array of all possible threshold values which we want to loop through
    color_thresholds = np.arange(np.min(img) + nbins, np.max(img) - nbins, nbins)
    # dictionary to store the threshold and within class variance
    mydic = {}
    # loop through the thresholds to find the one with the least within class variance
    for color_threshold in color_thresholds:
        bg_pixels = all_colors[all_colors < color_threshold]
        weight_bg = len(bg_pixels) / total_weight
        variance_bg = np.var(bg_pixels)
```

```
        fg_pixels = all_colors[all_colors >= color_threshold]
        weight_fg = len(fg_pixels) / total_weight
        variance_fg = np.var(fg_pixels)

        within_class_variance = weight_fg * variance_fg + weight_bg * variance_bg

        mydic[color_threshold] = within_class_variance

        if least_variance == -1 or least_variance > within_class_variance:
            least_variance = within_class_variance
            least_variance_threshold = color_threshold
            print("class variance ,      color threshold ")
            print("trace:", within_class_variance, color_threshold)

#   plot the dictionary
    # Threshold values and within-class variance from the dictionary
    thresholds = list(mydic.keys())
    variances = list(mydic.values())

    # Create a plot of threshold vs. variance
    plt.figure(figsize=(10, 6))
    plt.plot(thresholds, variances, marker='o', linestyle='-')
    plt.title('Inter-Class Variance vs. Threshold')
    plt.xlabel('Threshold Value')
    plt.ylabel('Inter-Class Variance')
    plt.grid(True)
    plt.show()


    Otsu_thresh_img = np.ones(img.shape).astype(np.uint8)
    # new_img has all pixels of 1s
    # setting 0 to the pixel above threshold in org image
    Otsu_thresh_img[np.where(img < least_variance_threshold)] = 0
    Otsu_thresh_img = Image.fromarray(Otsu_thresh_img * 255)
    Otsu_thresh_img.save('after.png')
    return Otsu_thresh_img

    #return least_variance_threshold
def plot_Histogram(figure,bins):
    plt.figure(figsize=(8, 6))
    plt.bar(bins[:-1], figure)
    plt.title('Image Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.show()
```

```
def plot_image(img):
    plt.imshow(img, cmap='gray')
    plt.title('Thresholded Image')
    plt.axis('off')
    plt.show()
# choose the image
#image_path = 'after.png'
#image_path = 'b2_a.png'
#image_path = 'b2_b.png'
image_path = 'b2_c.png'
img = Image.open(image_path).convert('L')

plot_image(img)

image_array = np.array(img, dtype=np.uint8)

# create the histogram of the image.
hist, bins = np.histogram(image_array, bins=255, range=(0, 255))

# Plot the histogram
plot_Histogram(hist,bins)

otsu_thresholding = threshold_otsu_impl(image_array, nbins=0.1)


plot_image(otsu_thresholding)
```

Histogram Matching Code.

```
import cv2
from cv2 import imread
from skimage.exposure import cumulative_distribution
from skimage.color import rgb2gray
import matplotlib.pylab as plt
import numpy as np

def histogram_matching(cdf_image, cdf_target, image):
    # Calculate the mapping to match the input CDF to the target CDF
    mapping = np.interp(cdf_image, cdf_target, np.arange(256))

    # Create a dictionary to replace pixel values
    pixel_replacement = {i: mapping[i] for i in range(256)}

    # Map pixel values using the dictionary
    mapped_pixels = np.arange(0, 256)
    for (key, value) in pixel_replacement.items():
        mapped_pixels[key] = value

    # Apply the mapping to the input image to get the matched image
    shape = image.shape
    matched_image = np.reshape(mapped_pixels[image.ravel()], shape)
    return matched_image

def calculate_cdf(image):
    cdf, bins = cumulative_distribution(image)
    for i in range(bins[0]):
        cdf = np.insert(cdf, 0, 0)
    for i in range(bins[-1] + 1, 256):
        cdf = np.append(cdf, 1)
    return cdf

# Load the source and target images

##  for color image to add a theme color from one image to another
#source_image = imread('sourceimage.png').astype(np.uint8)

#target_image = imread('Templatsunset.jpg').astype(np.uint8)
```

```
# here grey images to improve a dark image  based on good image

source_image = imread('sourceimg555.PNG').astype(np.uint8)
target_image = imread('referenceimg5555.PNG').astype(np.uint8)

source_image = cv2.cvtColor(source_image, cv2.COLOR_BGR2RGB)
target_image = cv2.cvtColor(target_image, cv2.COLOR_BGR2RGB)

# Create a new image for the matched result
matched_image = np.zeros(source_image.shape).astype(np.uint8)

# Match the histograms for each color channel separately
for channel in range(3):
    cdf_source = calculate_cdf(source_image[..., channel])
    cdf_target = calculate_cdf(target_image[..., channel])
    matched_image[..., channel] = histogram_matching(cdf_source, cdf_target,
source_image[..., channel])

# Display the input, target, and matched images
plt.figure(figsize=(20, 17))
plt.subplots_adjust(left=0, top=0.95, right=1, bottom=0, wspace=0.05, hspace=0.05)
plt.subplot(221), plt.imshow(source_image), plt.axis('off'), plt.title('Source Image',
size=25)
plt.subplot(222), plt.imshow(target_image), plt.axis('off'), plt.title('Target Image', size=25)
plt.subplot(223), plt.imshow(matched_image[..., :3]), plt.axis('off'), plt.title('Output
Image', size=25)
plt.show()
```