CSE 107: Lab 03: Histogram Equalization.

Vedant Sinha
Thursday 10:30 – 1:20
November 10, 2025

Abstract:

The purpose of this lab is to understand how histogram equalization enhances contrast in grayscale images. I analyze images with different lighting and contrast conditions, study their intensity distributions, and derive a cumulative mapping to redistribute pixel values for better visibility. I visualize the histograms and the transformation curve to interpret how the mapping changes contrast and detail. Finally, I compare visual results and basic statistics before and after equalization to evaluate effectiveness and trade-offs.

Results:

Figure 1 contains the dark image and figure 2 contains the histogram of the dark image. Figure 3 contains the transformation used to perform the equalization. Figure 4 contains the equalized dark image and figure 5 contains the histogram of the equalized dark image. The following table contains the mean and standard deviations of the dark image and the equalized version of the dark image.

	Mean pixel value	Standard deviation of the pixel values
Dark image	55.701080	32.155399
Equalized dark image	128.646255	73.568779

Figure 1. The dark image.

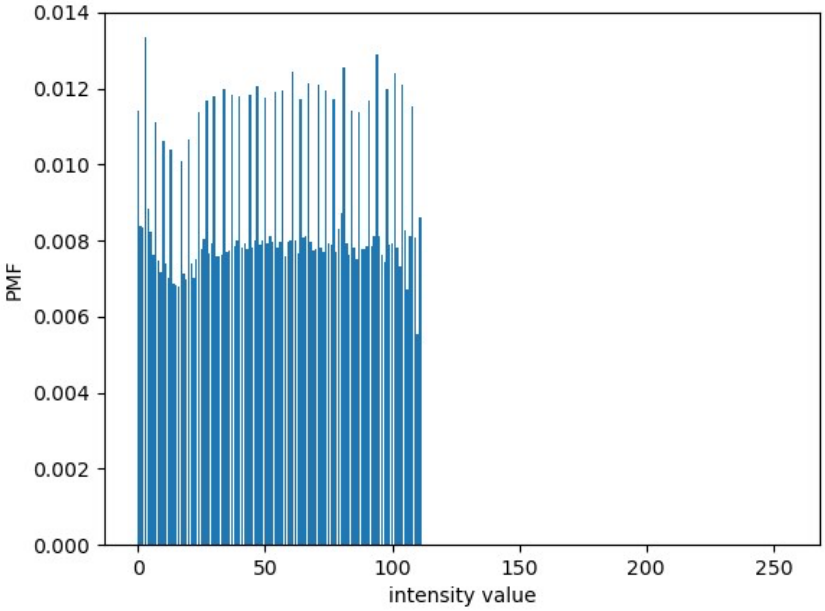


Figure 2. The histogram of the dark image.

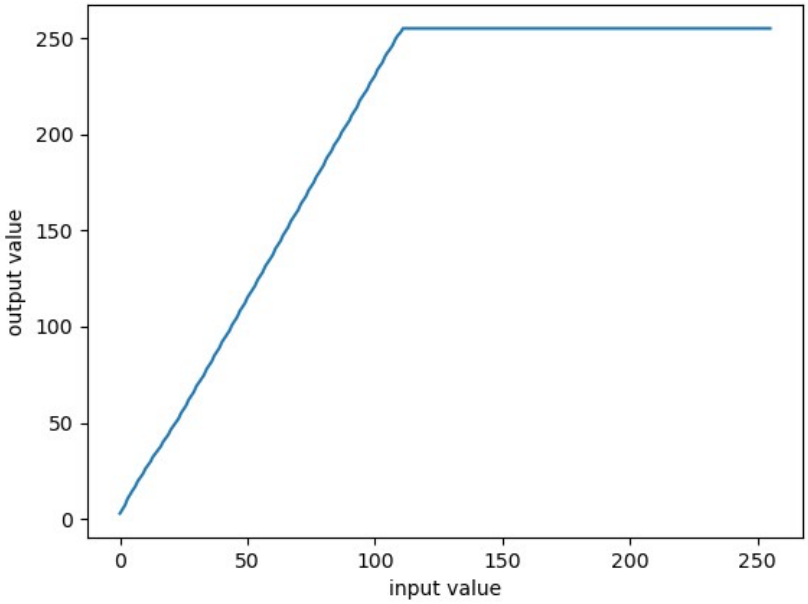


Figure 3. The transform used to perform equalization of the dark image.

Figure 4. The equalized version of the dark image.

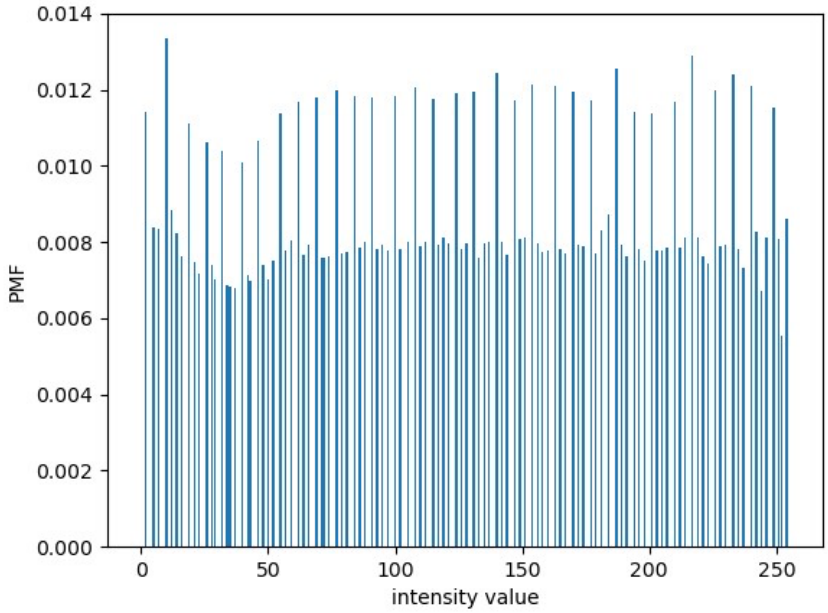
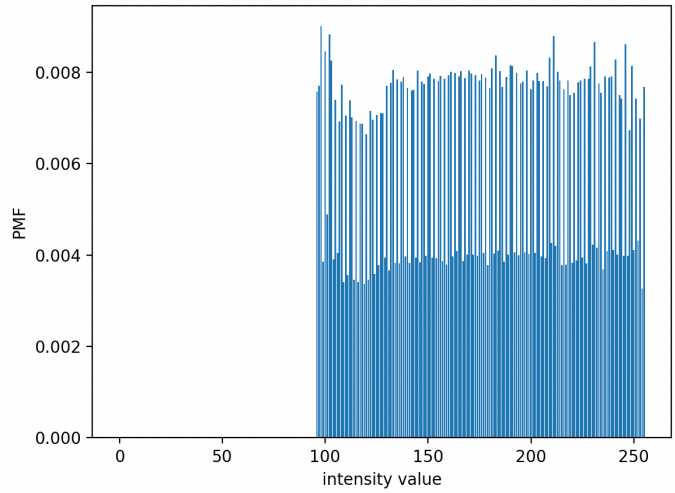
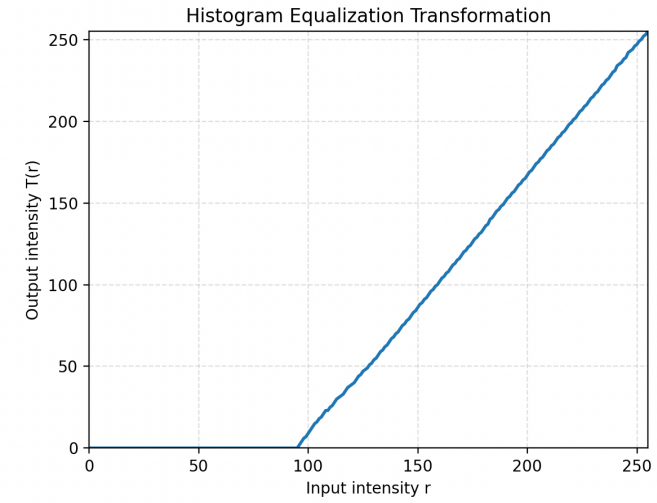
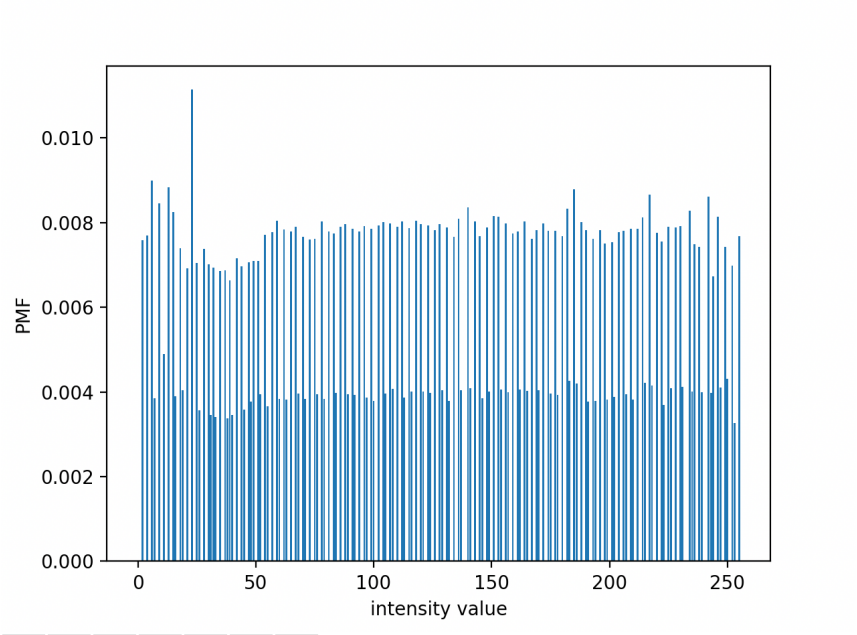


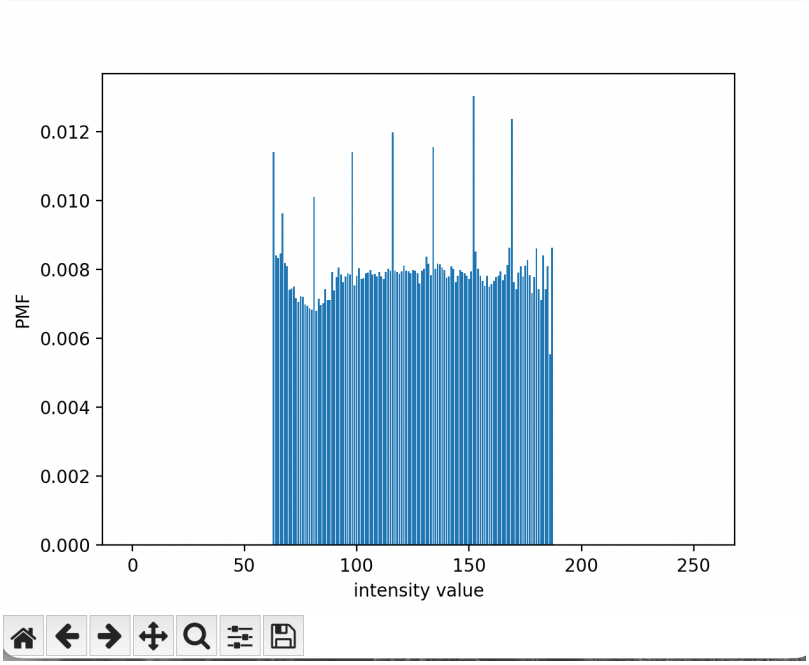
Figure 5. The histogram of the equalized version of the dark image.

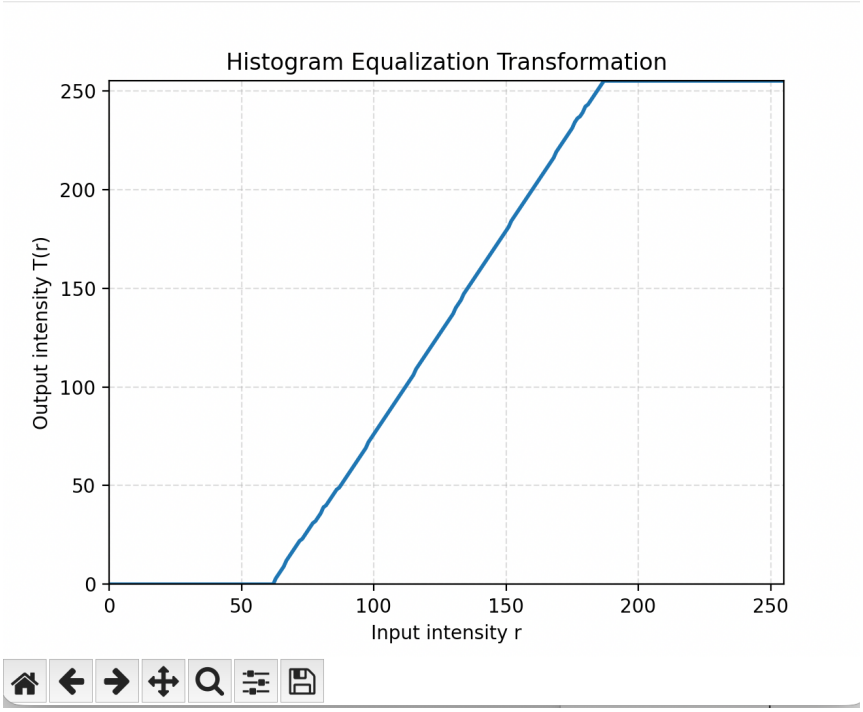
<repeat the above for the light, low contrast, and high contrast images>


PMF
0.000
0.002
0.004
0.006
0.008
0
50
100
150
200
250
intensity value


Histogram Equalization Transformation
Output intensity T(r)
0
50
100
150
200
250
Input intensity r


0.010
0.008
0.006
0.004
0.002
0.000
PMF
0
50
100
150
200
250
intensity value


0.012
0.010
0.008
0.006
0.004
0.002
0.000
PMF
0
50
100
150
200
250
intensity value


Histogram Equalization Transformation
250
200
150
100
50
0
Output intensity T(r)
0
50
100
150
200
250
Input intensity r

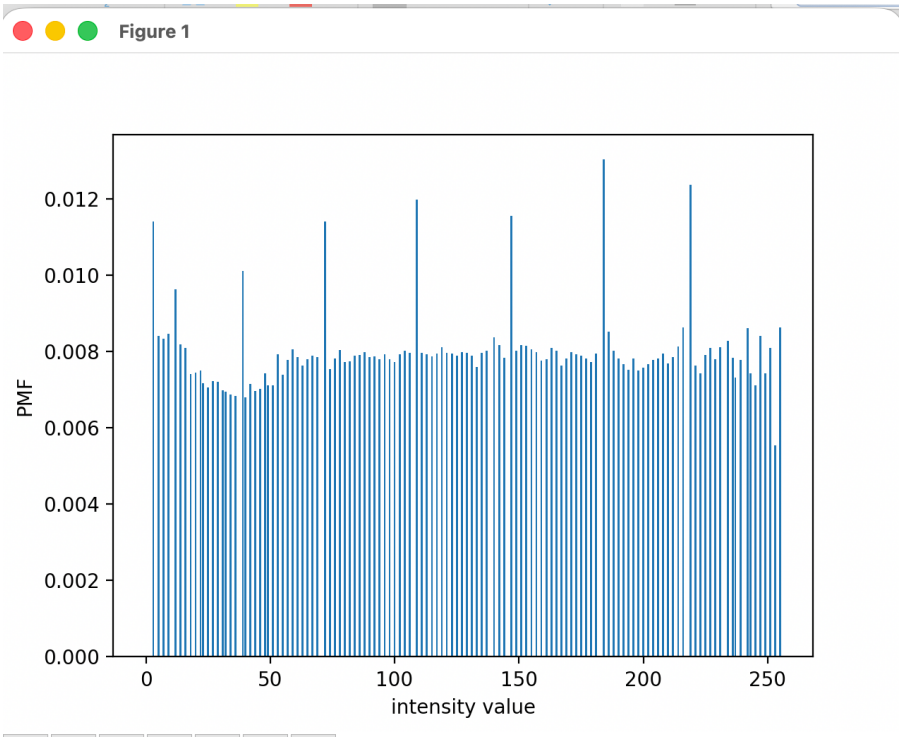
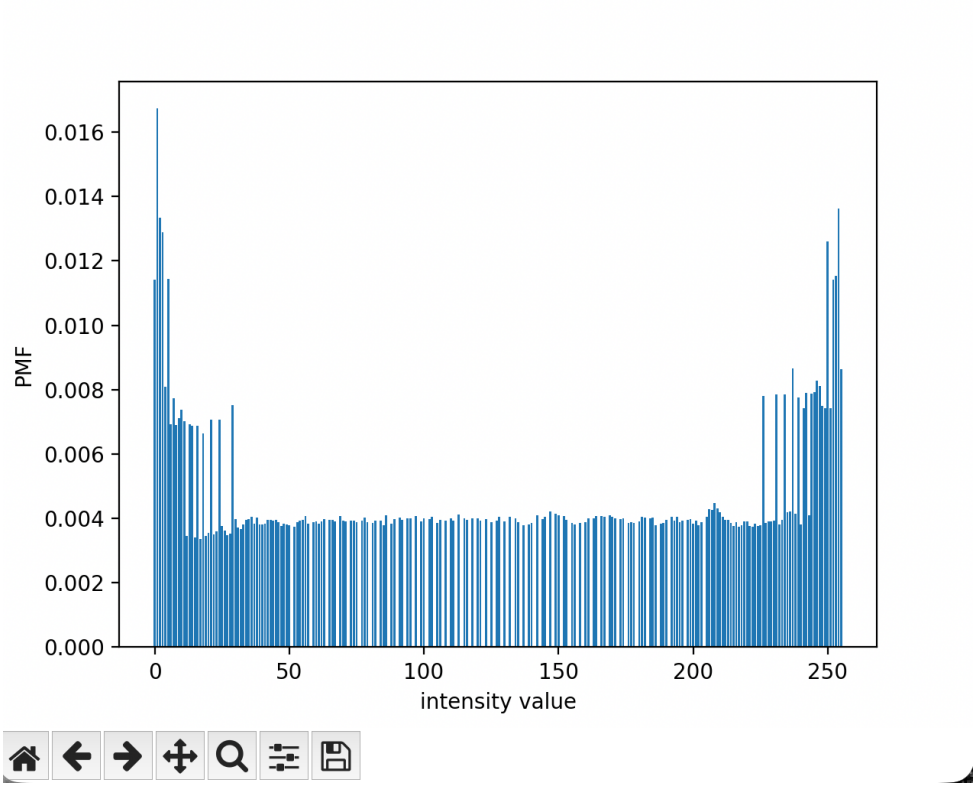


Figure 1
PMF
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
50
100
150
200
250
intensity value


0.016
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
PMF
0
50
100
150
200
250
intensity value

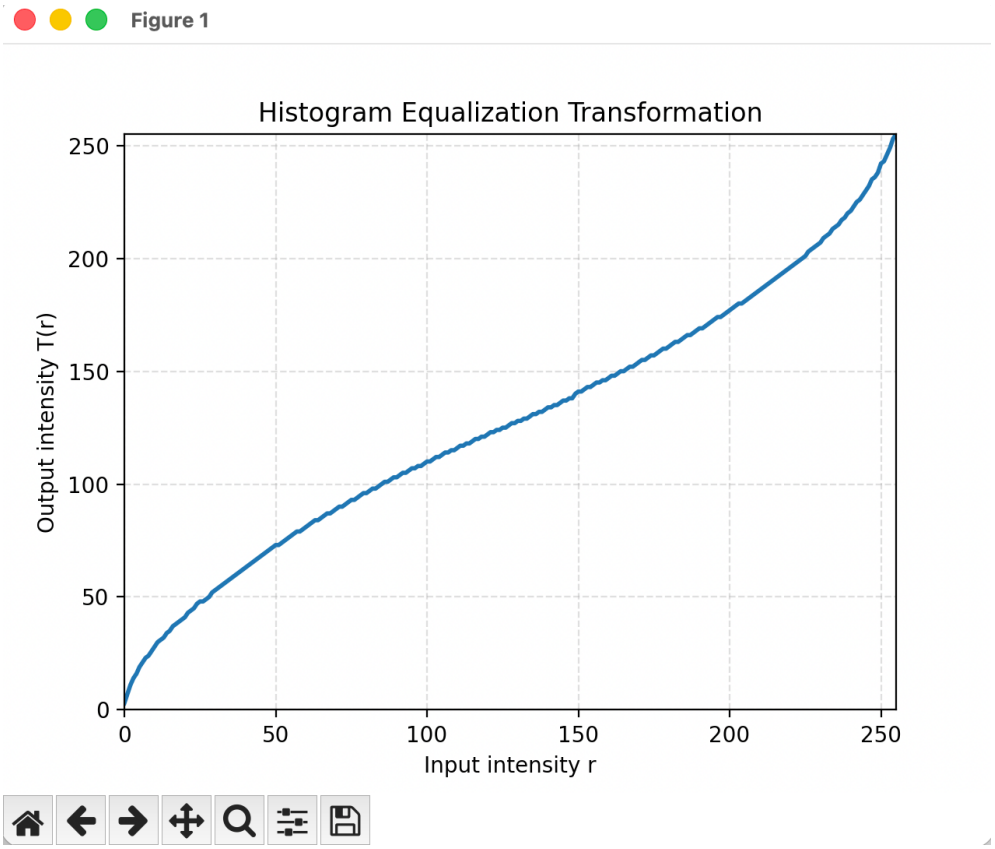
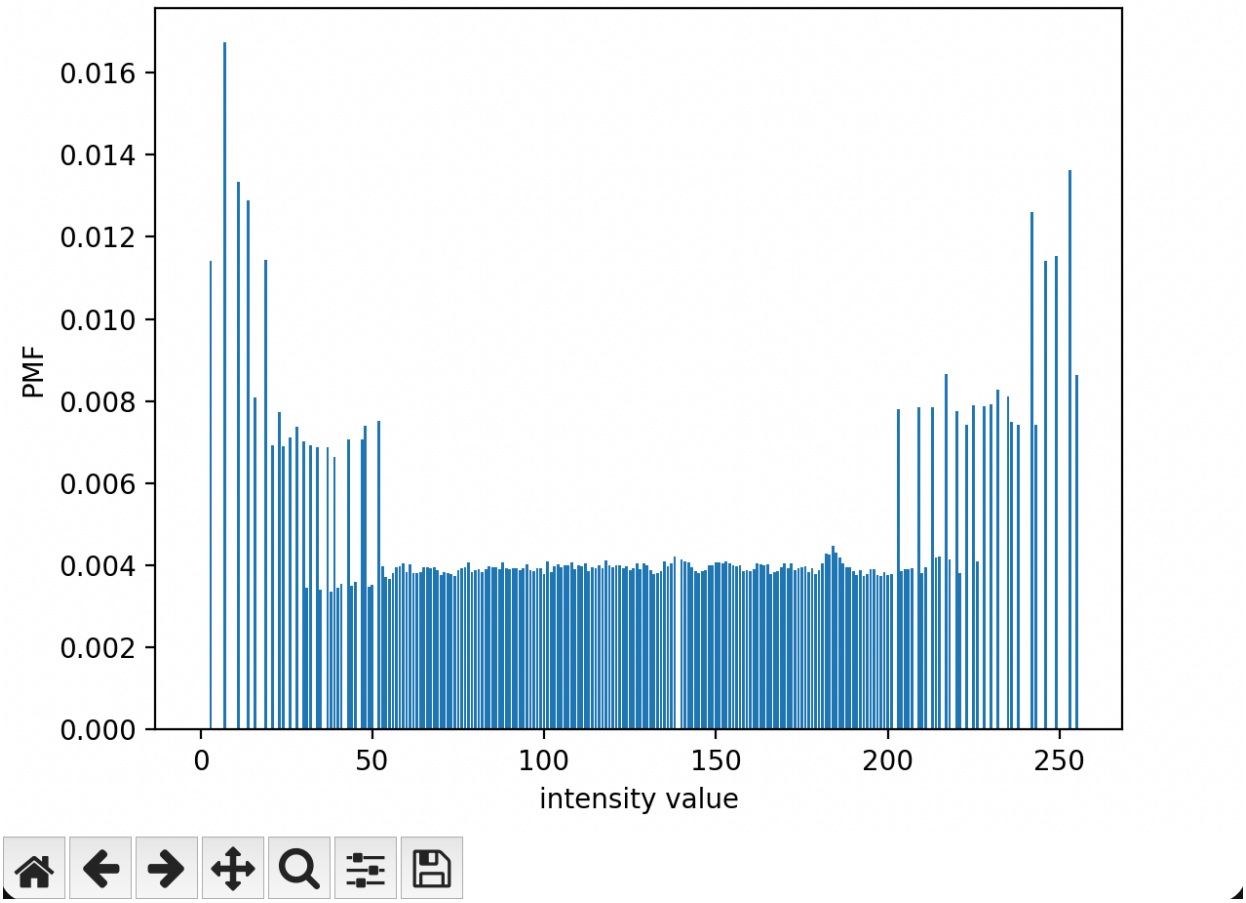


Figure 1
Histogram Equalization Transformation
Output intensity T(r)
Input intensity r
0
50
100
150
200
250



Questions:

1) Visual improvements
Yes, overall the equalized images are visually improved:
Dark image: shadow regions lift and midtones become more visible; details emerge without blowing out highlights.
Light image: bright regions compress and midtones deepen; washed-out areas gain texture.
Low contrast image: the full range spreads out; local details become noticeably clearer.
High contrast image: extremes are tamed and midtones expand; some highlight/shadow detail recovers,
2) Histogram differences
Before: the dark image's histogram is bunched at low intensities, the light image at high intensities, and the low-contrast image tightly concentrated around the center; the high-contrast image often shows heavy mass at both extremes.
After equalization: histograms spread across 0–255 and are closer to uniform. Regions with dense original mass get stretched, while sparse regions remain comparatively less populated.
3) Mean and standard deviation changes

Mean: values move toward mid-gray, especially for dark and light images. This reflects a more balanced use of the dynamic range.
Standard deviation: generally increases for dark, light, and low-contrast images, indicating expanded contrast. For the high-contrast image, the standard deviation typically decreases from a very large value down toward the equalized level, reflecting compression of extreme tails and expansion of midtones.
4) Transformation shape vs. the textbook equation
The transformation is monotonic and matches what I expect from a scaled cumulative distribution function: $T(r) = (L-1)\cdot CDF(r)$.
Where the original histogram is dense, the CDF rises steeply and T(r) has a larger slope (strong local stretching). Where the histogram is sparse, T(r) is flatter (less change). The plotted curve reflects this CDF-driven behavior.
5) Most difficult part
Getting the mapping exactly right, ensuring correct indexing, handling clipping/casting, and keeping types consistent was the trickiest.

```
# Import pillow
from PIL import Image, ImageOps

# Import numpy
import numpy as np
from numpy import asarray

##############################################################################
# Perform histogram equalization on the dark image.
##############################################################################

# Read the dark image from file.
dark_im = Image.open('Lab_03_image1_dark.tif')

# Show the image.
dark_im.show()

# Create numpy matrix to access the pixel values.
# NOTE THAT WE WE ARE CREATING A FLOAT32 ARRAY SINCE WE WILL BE DOING
# FLOATING POINT OPERATIONS IN THIS LAB.
dark_im_pixels = asarray(dark_im, dtype=np.float32)

# Import compute_histogram from My_HE_functions.
from My_HE_functions import compute_histogram

# Compute the histogram of the dark image.
dark_hist = compute_histogram( dark_im_pixels )

# Import plot_histogram from My_HE_functions.
from My_HE_functions import plot_histogram

# Plot the histogram for the dark image.
plot_histogram( dark_hist )

print('Dark image has mean = %f and standard deviation = %f' % \
    (np.mean(dark_im_pixels), np.std(dark_im_pixels)))

# Import equalize from My_HE_functions.
from My_HE_functions import equalize

# Apply histogram equalization to the dark image.
equalized_dark_im_pixels = equalize( dark_im_pixels );

# Create an image from numpy matrix equalized_dark_image_pixels.
equalized_dark_image = Image.fromarray(np.uint8(equalized_dark_im_pixels.round()))

# Show the equalized image.
equalized_dark_image.show()

# Save the equalized image.
equalized_dark_image.save('equalized_dark_image.tif');

# Compute the histogram of the equalized dark image.
equalized_dark_hist = compute_histogram( equalized_dark_im_pixels )
```

```
# Plot the histogram for the equalized dark image.
plot_histogram( equalized_dark_hist )

print('Equalized dark image has mean = %f and standard deviation = %f' % \
    (np.mean(equalized_dark_im_pixels), np.std(equalized_dark_im_pixels)))

#############################################################################
# Perform histogram equalization on the light image.
#############################################################################

# Read the light image from file.
light_im = Image.open('Lab_03_image2_light.tif')

# Show the image.
light_im.show()

# Create numpy matrix to access the pixel values.
# NOTE THAT WE WE ARE CREATING A FLOAT32 ARRAY SINCE WE WILL BE DOING
# FLOATING POINT OPERATIONS IN THIS LAB.
light_im_pixels = asarray(light_im, dtype=np.float32)

# Compute the histogram of the light image.
light_hist = compute_histogram( light_im_pixels )

# Plot the histogram for the light image.
plot_histogram( light_hist )

print('\nLight image has mean = %f and standard deviation = %f' % \
    (np.mean(light_im_pixels), np.std(light_im_pixels)))

# Apply histogram equalization to the light image.
equalized_light_im_pixels = equalize( light_im_pixels );

# Create an image from numpy matrix equalized_light_image_pixels.
equalized_light_image = Image.fromarray(np.uint8(equalized_light_im_pixels.round()))

# Show the equalized image.
equalized_light_image.show()

# Save the equalized image.
equalized_light_image.save('equalized_light_image.tif');

# Compute the histogram of the equalized light image.
equalized_light_hist = compute_histogram( equalized_light_im_pixels )

# Plot the histogram for the equalized light image.
plot_histogram( equalized_light_hist )

print('Equalized light image has mean = %f and standard deviation = %f' % \
    (np.mean(equalized_light_im_pixels), np.std(equalized_light_im_pixels)))

#############################################################################
# Perform histogram equalization on the low contrast image.
```

```
##############################################################################

# Read the low contrast image from file.
low_contrast_im = Image.open('Lab_03_image3_low_contrast.tif')

# Show the image.
low_contrast_im.show()

# Create numpy matrix to access the pixel values.
# NOTE THAT WE WE ARE CREATING A FLOAT32 ARRAY SINCE WE WILL BE DOING
# FLOATING POINT OPERATIONS IN THIS LAB.
low_contrast_im_pixels = asarray(low_contrast_im, dtype=np.float32)

# Compute the histogram of the low contrast image.
low_contrast_hist = compute_histogram( low_contrast_im_pixels )

# Plot the histogram for the low contrast image.
plot_histogram( low_contrast_hist )

print('Low contrast image has mean = %f and standard deviation = %f' % \
    (np.mean(low_contrast_im_pixels), np.std(low_contrast_im_pixels)))

# Apply histogram equalization to the low contrast image.
equalized_low_contrast_im_pixels = equalize( low_contrast_im_pixels );

# Create an image from numpy matrix equalized_low_contrast_image_pixels.
equalized_low_contrast_image = Image.fromarray(np.uint8(equalized_low_contrast_im_pixels.round()))

# Show the equalized image.
equalized_low_contrast_image.show()

# Save the equalized image.
equalized_low_contrast_image.save('equalized_low_contrast_image.tif');

# Compute the histogram of the equalized low contrast image.
equalized_low_contrast_hist = compute_histogram( equalized_low_contrast_im_pixels )

# Plot the histogram for the equalized low contrast image.
plot_histogram( equalized_low_contrast_hist )

print('Equalized low contrast image has mean = %f and standard deviation = %f' % \
    (np.mean(equalized_low_contrast_im_pixels), np.std(equalized_low_contrast_im_pixels)))

##############################################################################
# Perform histogram equalization on the high contrast image.
##############################################################################

# Read the high contrast image from file.
high_contrast_im = Image.open('Lab_03_image4_high_contrast.tif')

# Show the image.
high_contrast_im.show()

# Create numpy matrix to access the pixel values.
```

test_HistogramEqualization.py

```
# NOTE THAT WE WE ARE CREATING A FLOAT32 ARRAY SINCE WE WILL BE DOING
# FLOATING POINT OPERATIONS IN THIS LAB.
high_contrast_im_pixels = asarray(high_contrast_im, dtype=np.float32)

# Compute the histogram of the high contrast image.
high_contrast_hist = compute_histogram( high_contrast_im_pixels )

# Plot the histogram for the high contrast image.
plot_histogram( high_contrast_hist )

print('High contrast image has mean = %f and standard deviation = %f' % \
    (np.mean(high_contrast_im_pixels), np.std(high_contrast_im_pixels)))

# Apply histogram equalization to the high contrast image.
equalized_high_contrast_im_pixels = equalize( high_contrast_im_pixels );

# Create an image from numpy matrix equalized_high_contrast_image_pixels.
equalized_high_contrast_image = Image.fromarray(np.uint8(equalized_high_contrast_im_pixels.round()))

# Show the equalized image.
equalized_high_contrast_image.show()

# Save the equalized image.
equalized_high_contrast_image.save('equalized_high_contrast_image.tif');

# Compute the histogram of the equalized high contrast image.
equalized_high_contrast_hist = compute_histogram( equalized_high_contrast_im_pixels )

# Plot the histogram for the equalized high contrast image.
plot_histogram( equalized_high_contrast_hist )

print('Equalized high contrast image has mean = %f and standard deviation = %f' % \
    (np.mean(equalized_high_contrast_im_pixels), np.std(equalized_high_contrast_im_pixels)))
```

My_HE_functions.py

```python
# MyHEFunctions.py

# Import numpy
import numpy as np

def compute_histogram( image_pixels ):

    # compute_histogram  Computes the normalized histogram (PMF) of a grayscale image.
    #
    # Syntax:
    #   hist = compute_histogram( image_pixels )
    #
    # Input:
    #   image_pixels = A 2D numpy array (float or int) with values in [0, 255].
    #
    # Output:
    #   hist = A length-256 numpy vector where hist[i] is the probability of intensity i.
    #
    # History:
    #   Implemented for Lab 03

    if image_pixels is None:
        raise ValueError("image_pixels must not be None")

    if not isinstance(image_pixels, np.ndarray):
        raise TypeError("image_pixels must be a numpy ndarray")

    if image_pixels.ndim != 2:
        raise ValueError("image_pixels must be a 2D grayscale image")

    # Ensure values are within [0, 255] and convert to integer indices
    clipped_pixels = np.clip(image_pixels, 0, 255)
```

```
    intensity_indices = clipped_pixels.astype(np.int32).ravel()

    # Count occurrences for each intensity using bincount for efficiency
    counts = np.bincount(intensity_indices, minlength=256).astype(np.float64)

    num_pixels = image_pixels.size
    if num_pixels == 0:
        # Avoid division by zero; return uniform zeros histogram
        return np.zeros(shape=(256,), dtype=np.float64)

    hist = counts / float(num_pixels)
    return hist

def equalize( in_image_pixels ):

    # equalize  Applies histogram equalization to a grayscale image and plots the
    #           intensity transformation function.
    #
    # Syntax:
    #   out_image_pixels = equalize( in_image_pixels )
    #
    # Input:
    #   in_image_pixels = A 2D numpy array (float or int) with values in [0, 255].
    #
    # Output:
    #   out_image_pixels = A 2D numpy array (float32) of the equalized image
    #                      with values in [0, 255].
    #
    # History:
    #   Implemented for Lab 03

    if in_image_pixels is None:
        raise ValueError("in_image_pixels must not be None")
```

```python
    if not isinstance(in_image_pixels, np.ndarray):
        raise TypeError("in_image_pixels must be a numpy ndarray")

    if in_image_pixels.ndim != 2:
        raise ValueError("in_image_pixels must be a 2D grayscale image")

    # Compute the normalized histogram and its cumulative distribution function (CDF)
    hist = compute_histogram(in_image_pixels)
    cdf = np.cumsum(hist)

    # Create the intensity mapping: T(r) = round(255 * CDF(r))
    mapping = np.round(255.0 * cdf).astype(np.uint8)

    # Apply mapping to the input image
    clipped_pixels = np.clip(in_image_pixels, 0, 255).astype(np.int32)
    equalized_pixels_uint8 = mapping[clipped_pixels]

    # Convert to float32 to match lab expectations of float operations
    out_image_pixels = equalized_pixels_uint8.astype(np.float32)

    # Plot the transformation function T(r) vs r
    import matplotlib.pyplot as plt
    r_values = np.arange(256)
    plt.plot(r_values, mapping, linewidth=2)
    plt.title('Histogram Equalization Transformation')
    plt.xlabel('Input intensity r')
    plt.ylabel('Output intensity T(r)')
    plt.xlim([0, 255])
    plt.ylim([0, 255])
    plt.grid(True, linestyle='--', alpha=0.4)
    plt.show()

    return out_image_pixels
```

```
def plot_histogram( hist ):
    # plot_histgram  Plots the length 256 numpy vector representing the normalized
    # histogram of a grayscale image.
    #
    # Syntax:
    #   plot_histogram( hist )
    #
    # Input:
    #   hist = The length 256 histogram vector..
    #
    # Output:
    #   none
    #
    # History:
    #   S. Newsam     10/25/2025   created

    # Import plotting functions from matplotlib.
    import matplotlib.pyplot as plt

    plt.bar( range(256), hist )

    plt.xlabel('intensity value');

    plt.ylabel('PMF');

    plt.show()
```