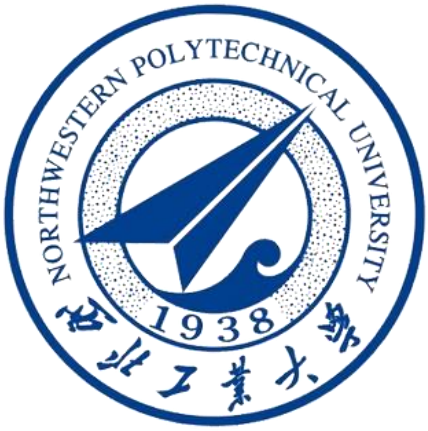
# Experiment Report

# Multimedia Technology




Experiment Information:

| | |
|---|---|
| Number | 1 |
| Name | Histogram equalization |

Student Information:

| | |
|---|---|
| Student ID: | 2021380101 |
| Student Name: | Rheina Trudy Williams |

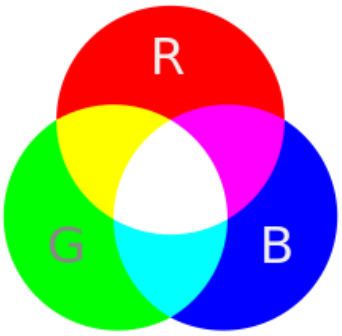
**Northwestern Polytechnical University**

**School of Computer Science**

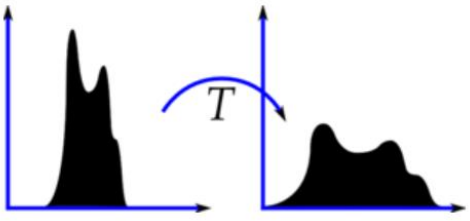
**Autumn 2023**

I. Introduction

Histogram equalization is computer image processing technique to adjust the image's contrast by using histogram. Like a bar chart, a histogram is a graphical depiction of certain data. An image is just a collection of pixels. It can be viewed as a 3D array in RGB format, which consists of three overlapping 2D arrays representing the image's red, green, and blue components. An image with dimensions of axbx3 has a number of rows, b columns, and a total of a*b pixels in a single color array or frame. In this section three represent the colors red, green, and blue within three frames. While an image's histogram will resemble a normal distribution, equalization aims for a uniform distribution. By selecting an appropriate intensity transformation function, the image histogram is consistently distributed throughout the whole intensity axis.




The steps include create the image's histogram, find the image's local minima, divide the histogram based on the local minima, possess the unique gray levels for every histogram split. On every partition, apply the histogram equalization. Because of its great effectiveness and ease of use, histogram equalization is a commonly used contrast-enhancement technique in image processing. It is one of the more advanced techniques for adjusting an image's dynamic range and contrast by transforming it such that its intensity histogram takes on the appropriate form.




Algorithm:
Calculate the input image's pixel value histogram. The values of each pixel $f[x,y]$ in the histogram are assigned to one of L uniformly-spaced bins $h[i]$.

- The image dimension is $M \times N$ and $L$=2^8.

- Determine the cumulative distribution function.
- To create the output image, scale the input image using the cumulative distribution function where CDFmin is the cumulative distribution function's smallest non-zero value.

II. Goals

1. Understand the implementation of histogram equalization.
2. Enhance the contrast of images.
3. Comparing histogram equalization algorithm.
4. Write a program to implement the histogram equalization.

III. Tools

Pycharm community 2023.3.1

IV. Code

```
import numpy as np
from matplotlib import pyplot as plt
from PIL import Image
import cv2

def read_image(file_path):
    img = Image.open(r"C:\Users\Rheina
Trudy\Documents\UNI\SEMESTER
5\Multimedia\Experiments\images\1.jpg").convert('L')  # Convert to
grayscale
    return np.array(img)

def calculate_histogram(image):
    histogram = np.zeros(256)
    for pixel_value in range(256):
        mask = (image == pixel_value)
        histogram[pixel_value] = np.sum(mask)
    return histogram

def normalize_histogram(histogram, total_pixels):
    return histogram / total_pixels

def cumulative_distribution_function(normalized_histogram):
    return np.cumsum(normalized_histogram)

def histogram_equalization(image, cdf):
    return np.interp(image, np.arange(256), cdf *
255).astype(np.uint8)

def plot_cumulative_histogram(cdf):
    plt.plot(range(256), cdf, color='g')
    plt.grid(True)
    plt.ylabel('Cumulative Distribution Function (CDF)')
    plt.xlabel('Level of intensity')
    plt.title('Histogram Cumulative')
    plt.show()

def plot_histogram_and_cdf(image, cdf):
    hist, bins = np.histogram(image.flatten(), 256, [0, 256])
```

```
    cdf_normalized = cdf * hist.max() / cdf.max()

    plt.plot(cdf_normalized, color='b')
    plt.hist(image.flatten(), 256, [0, 256], color='r')
    plt.xlim([0, 256])
    plt.legend(('CDF', 'Histogram'), loc='upper left')
    plt.show()

def plot_original_histogram(image):
    plt.figure(figsize=(12, 6))

    # Original Image
    plt.subplot(231)
    plt.imshow(image, cmap='gray')
    plt.title('Original Image')

    # Original Histogram
    plt.subplot(232)
    hist = calculate_histogram(image)
    plt.bar(range(256), hist, color='r')
    plt.grid(True)
    plt.ylabel('Pixels with same intensity')
    plt.xlabel('Level of intensity')
    plt.title('Original Histogram')

    plt.show()

def plot_normalized_histogram(histogram):
    plt.bar(range(256), histogram, color='b')
    plt.grid(True)
    plt.ylabel('Pixels with same intensity (normalized)')
    plt.xlabel('Level of intensity')
    plt.title('Normalized Histogram')
    plt.show()

def show_comparison(original_img, custom_img, built_in_img,
original_hist, custom_hist, built_in_hist):
    plt.figure(figsize=(15, 8))

    # Original Image + Original Histogram
    plt.subplot(331)
    plt.imshow(original_img, cmap='gray')
    plt.title('Original Image')
    plt.subplot(334)
    plt.hist(original_img.flatten(), 256, [0, 256], color='r')
    plt.grid(True)
    plt.ylabel('Pixels with same intensity')
    plt.xlabel('Level of intensity')
    plt.title('Original Histogram')

    # Enhanced Image + Enhanced Histogram (Written by me)
    plt.subplot(332)
    plt.imshow(custom_img, cmap='gray')
    plt.title('Enhanced Image (Written by me)')
    plt.subplot(335)
    plt.hist(custom_img.flatten(), 256, [0, 256], color='r')
    plt.grid(True)
    plt.ylabel('Pixels with same intensity')
    plt.xlabel('Level of intensity')
    plt.title('Equalized Histogram (Written by me)')
```

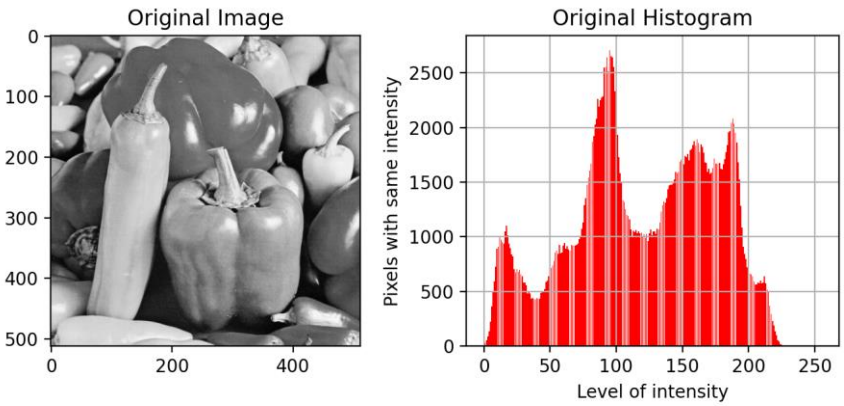
```
    # Enhanced Image + Enhanced Histogram (Built-in using OpenCV)
    plt.subplot(333)
    plt.imshow(built_in_img, cmap='gray')
    plt.title('Enhanced Image (Built-in using OpenCV)')
    plt.subplot(336)
    plt.hist(built_in_img.flatten(), 256, [0, 256], color='r')
    plt.grid(True)
    plt.ylabel('Pixels with same intensity')
    plt.xlabel('Level of intensity')
    plt.title('Equalized Histogram (Built-in using OpenCV)')

    # Comparison: Enhanced Image (Written by me) vs Enhanced Image
(Built-in using OpenCV)
    plt.subplot(337)
    plt.imshow(custom_img, cmap='gray')
    plt.title('Enhanced Image (Written by me)')
    plt.subplot(338)
    plt.imshow(built_in_img, cmap='gray')
    plt.title('Enhanced Image (Built-in using OpenCV)')

    # Histogram: Enhanced Image (Written by me) vs Enhanced Image
(Built-in using OpenCV)
    plt.subplot(339)
    plt.hist(custom_img.flatten(), 256, [0, 256], color='r',
alpha=0.5, label='Written by me')
    plt.hist(built_in_img.flatten(), 256, [0, 256], color='b',
alpha=0.5, label='Built-in OpenCV')
    plt.grid(True)
    plt.ylabel('Pixels with same intensity')
    plt.xlabel('Level of intensity')
    plt.title('Comparison: Equalized Histogram')
    plt.legend()

    plt.tight_layout()
    plt.show()

# Using custom function to read the image
img = read_image("path/to/your/image.jpg")
plot_original_histogram(img)
hist = calculate_histogram(img)

# Normalize
total_pixels = img.shape[0] * img.shape[1]
norm_hist = normalize_histogram(hist, total_pixels)

# Histogram and CDF
plot_histogram_and_cdf(img, norm_hist)

# Normalized histogram
plot_normalized_histogram(norm_hist)

# CDF calculation
cdf = cumulative_distribution_function(norm_hist)

# Cumulative histogram
plot_cumulative_histogram(cdf)

# Equalization
img_equalized = histogram_equalization(img, cdf)

# Enhanced image + histogram (written by me)
```

```
plt.subplot(121)
plt.imshow(img_equalized, cmap='gray')
plt.title('Enhanced Image (Written by me)')
plt.subplot(122)
plt.hist(img_equalized.flatten(), 256, [0, 256], color='r')
plt.grid(True)
plt.ylabel('Pixels with same intensity')
plt.xlabel('Level of intensity')
plt.title('Equalized Histogram (Written by me)')
plt.show()

# Equalization using OpenCV built-in function
img_histeq = cv2.equalizeHist(img)

# Image + histogram (Built-in using OpenCV)
plt.subplot(121)
plt.imshow(img_histeq, cmap='gray')
plt.title('Enhanced Image (Built-in using OpenCV)')
plt.subplot(122)
plt.hist(img_histeq.flatten(), 256, [0, 256], color='r')
plt.grid(True)
plt.ylabel('Pixels with same intensity')
plt.xlabel('Level of Intensity')
plt.title('Equalized Histogram (Built-in using OpenCV)')
plt.show()

# Compare
show_comparison(img, img_equalized, img_histeq, hist, norm_hist,
cv2.calcHist([img_histeq], [0],

None, [256], [0, 256]))
```

V. Result and analysis

1) fig 1.jpg

a. Histogram and the original image




Explanation:

The original image histogram is obtained through the calculate_histogram function and then visualized using the plot_original_histogram function.

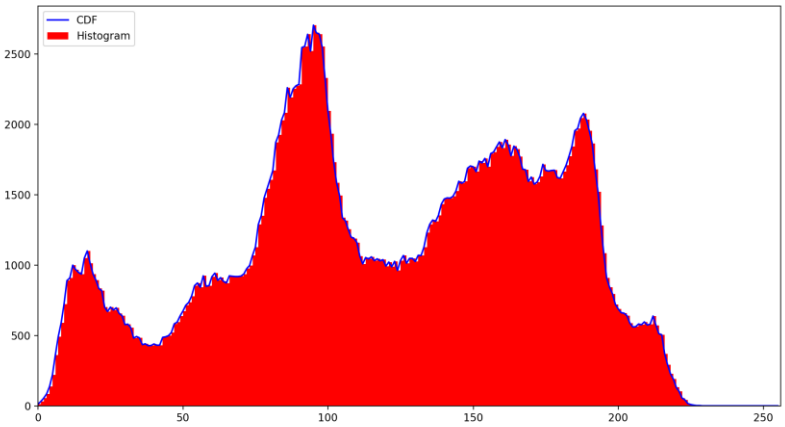
```
def calculate_histogram(image):
    histogram = np.zeros(256)
    for pixel_value in range(256):
```

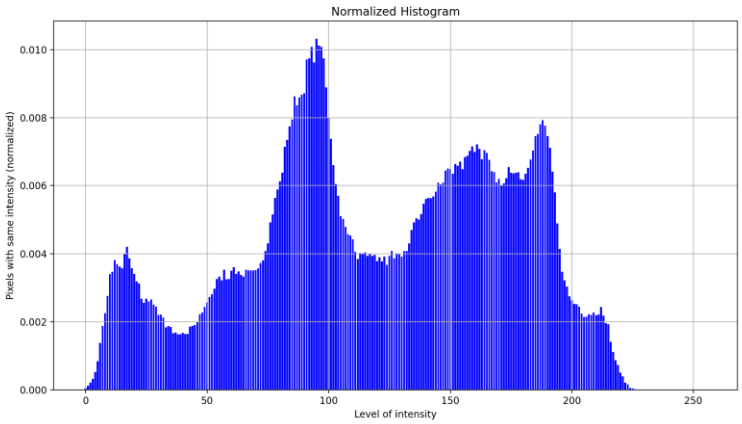
```
        mask = (image == pixel_value)
        histogram[pixel_value] = np.sum(mask)
    return histogram
```

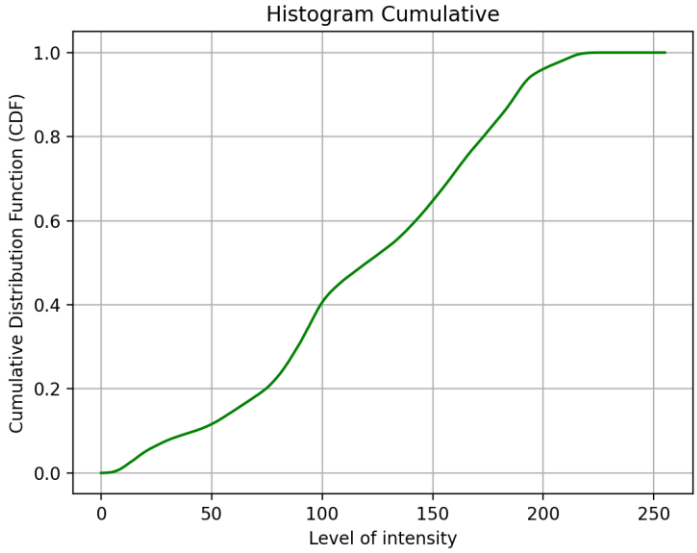
Every potential pixel value in the image which, for an 8-bit image, ranges from 0 to 255 is iterated through by the function. It creates a binary mask for each pixel value using the NumPy equation image == pixel_value, where True denotes pixels with the given intensity. Next, the function counts the number of pixels that have the same intensity using np.sum, and it stores the count in the histogram array.

In the original histogram, we can see the spread of intensity level with small difference of levels with the most pixels concentrated around 100.

b. Plot of transformation function






Explanation:
The normalize_histogram function divides each count by the total number of pixels in the picture to scale the histogram values.

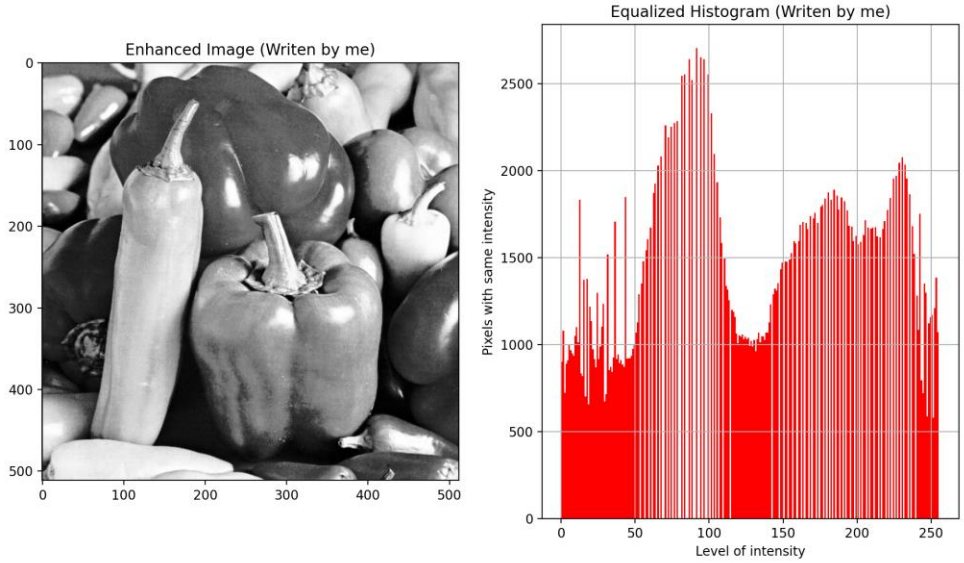
The cumulative probability distribution of a random variable is represented statistically by the cumulative distribution function, or CDF. The cumulative probability of pixel intensities in an image is described by the CDF in the context of image processing and histogram equalization. The CDF at a particular level of pixel intensity for a grayscale image. The normalized histogram values up to that intensity are added together to form k. Mathematically, it is defined as:

$$CDF(k) = \sum_{i=0}^{k} P(i)$$

P(i) = normalized histogram value
i = intensity level
k = range of the image

Using the cumulative_distribution_function function, the CDF is obtained by taking the cumulative sum of the normalized histogram. The likelihood that a pixel in the image has an intensity less than or equal to a given level is represented by this cumulative distribution function. Plotting the CDF versus the intensity levels allows for the visualization of the cumulative histogram. The accumulation of pixel intensities over the whole range is shown by this cumulative histogram.

c. Enhanced image and its histogram

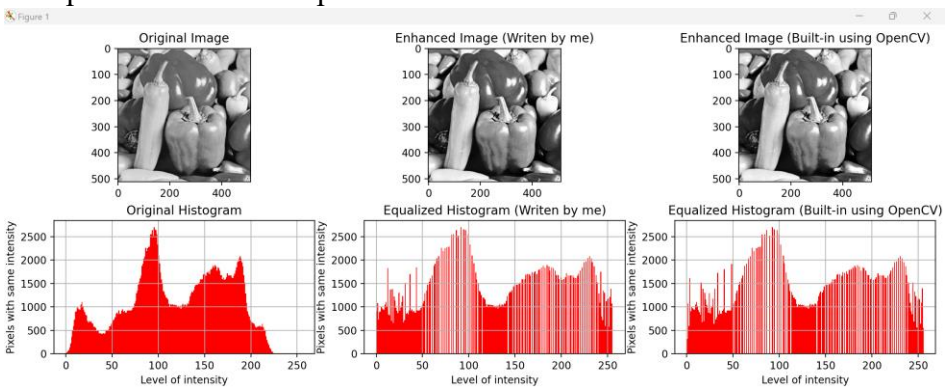



Explanation:

This process results in an equalized image where the distribution of intensities is more uniform, emphasizing details and improving visibility of features that might have been obscured by an initially skewed intensity distribution.

After enhancement, there is more contrast between intensity level, with most intensity focused at level 100.

d. Comparison between implementation and intrinsic function



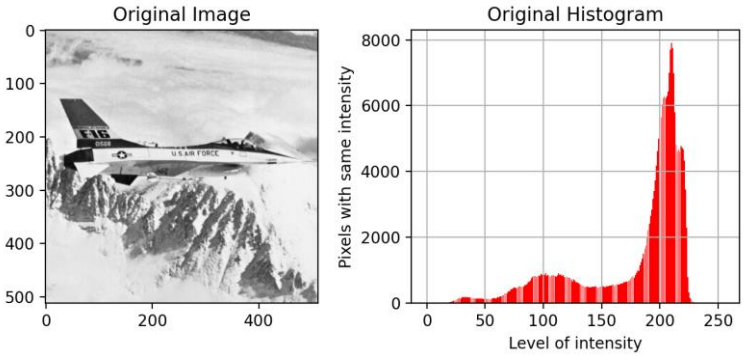

Explanation:

When compared to the original image, the enhanced photographs in both situations show better contrast and detail visibility. In this experiment, I output the result of the enhanced image equalized using my own code and the equalized image from the built-in function of OpenCV. Both approaches seek to accomplish histogram equalization, the improved photos generated by the OpenCV function and the custom implementation written by me seem identical in the overall graph. However, there are small variances in the pixel values as a result of different normalization or interpolation techniques. This can be seen that some level of intensity has different pixels with the same intensity.

2) fig 2.jpg

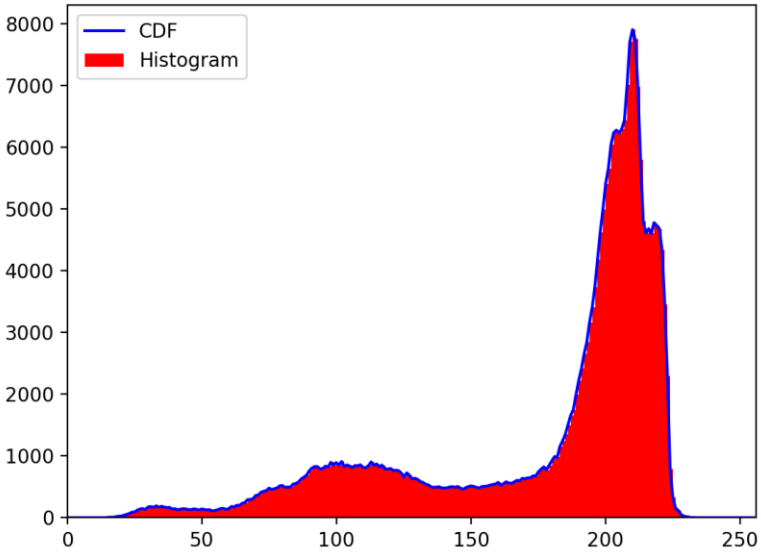
The process done in fig2.jpg is the same as the process done in fig1.jpg. The difference is that fig2.jpg is a colorized image. Therefore, to simplify the process, the image is first converted into grayscale image.

a. Histogram and the original image

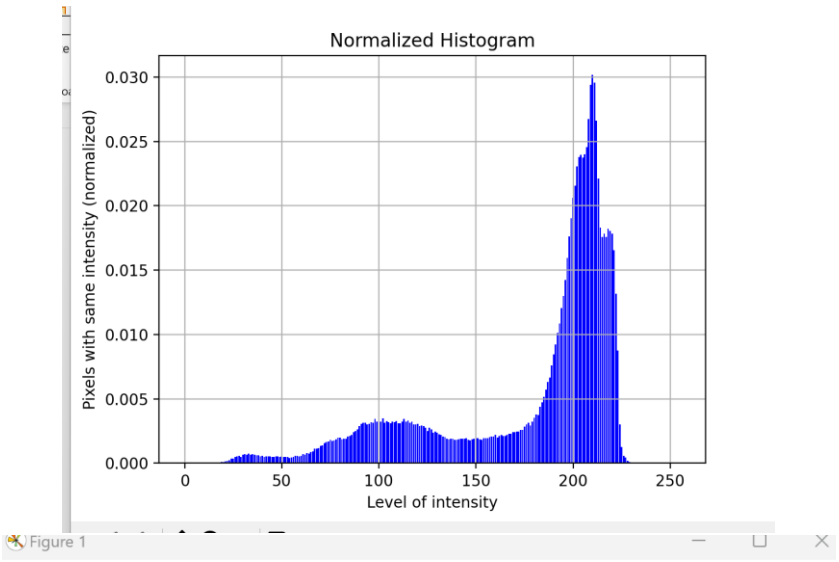


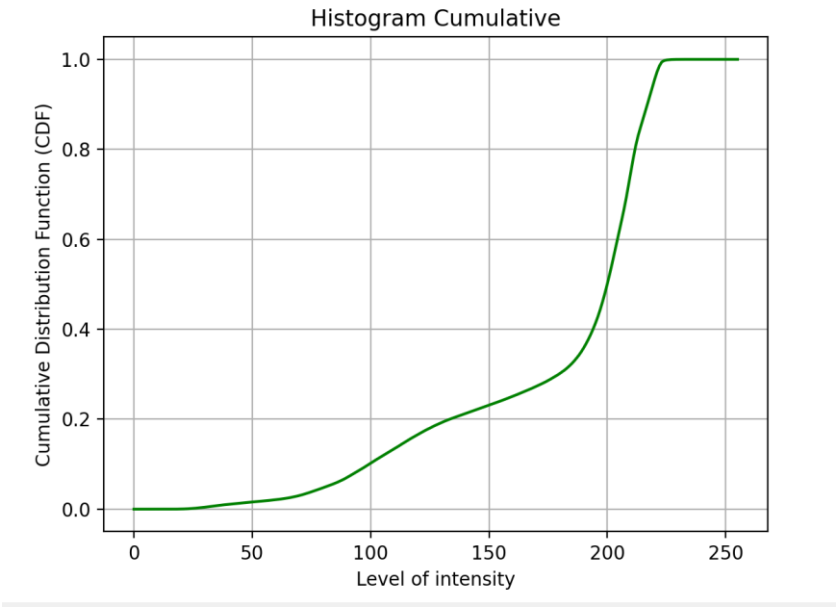

In the original image, the intensity level is spread out with small differences in intensity levels.

b. Plot of transformation function

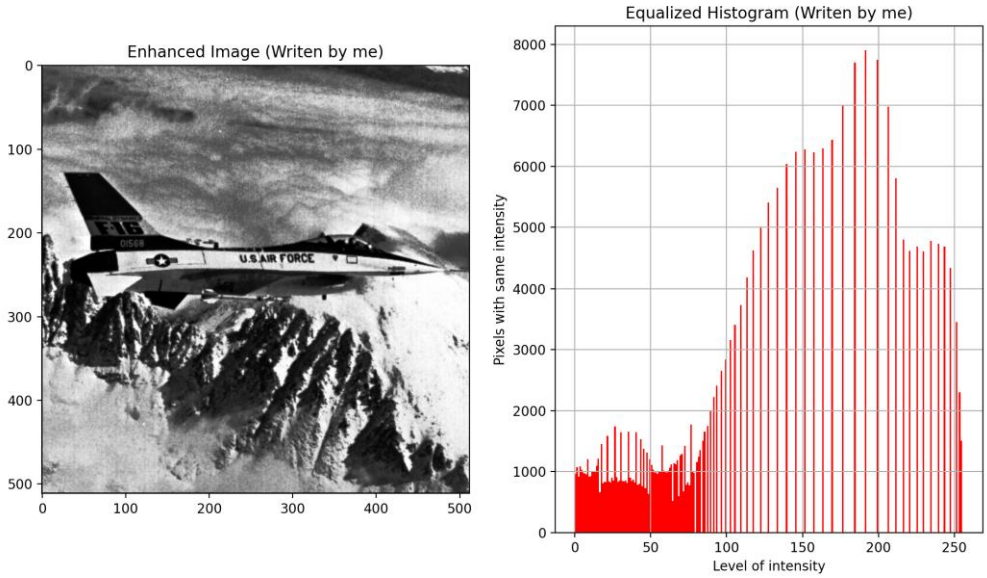



The function calculates the CDF graph.



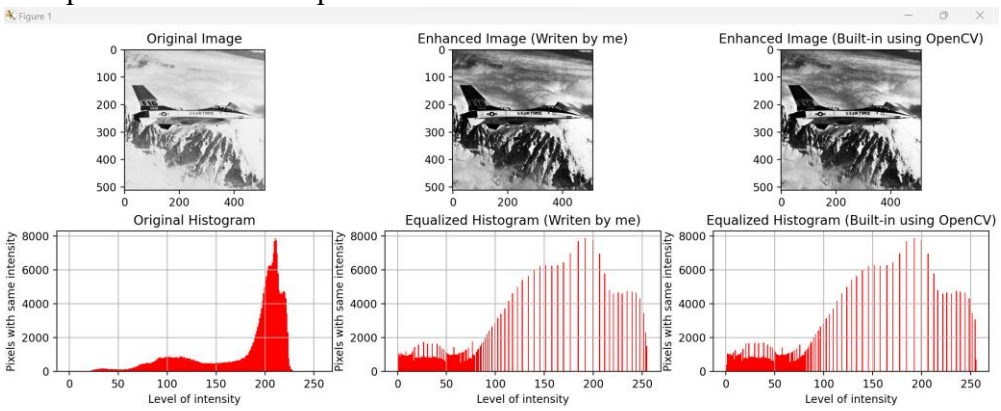




c. Enhanced image and its histogram




In this enhanced histogram, we can see that there are more high level intensity pixel than low level intensity, especially starting from 100 and reaching its peak around the intensity level of 200. Low intensity level is around 0-50 pixels. Additionally, the level intensity is more spread out and diverse.

d. Comparison between implementation and intrinsic function




Explanation:
After equalization, the picture looks clearer. Both techniques successfully reshape the pixel intensities, the overall enhanced histogram looks similar. However, only some intensity levels have small differences in the number of pixels with the same intensity because of the variations in the underlying algorithms and interpolation techniques.

## VI. Conclusion

This image processing experiment on histogram equalization gives me understanding on contrast enhancement methods and how they are applied. Histogram equalization is a technique to improve the contrast of images using Cumulative Distribution Function (CDF), representing the cumulative probability

distribution of pixel intensities in a grayscale image. Understanding the fundamentals of histogram equalization, improving image contrast, and contrasting the outcomes of manual implementation with an integrated OpenCV function were the main objectives. To produce a more uniform distribution of pixel intensities, the experiment involved computing image histograms, normalizing them, figuring out cumulative distribution functions, and using histogram equalization. The contrast enhancement produced by the manual implementation and the built-in OpenCV function was successful in bringing out more details and enhancing the features' overall visibility in the photos.

At the end, I compared the result of my implemented function and the built-in function by OpenCV. The results in overall are similar with the almost the same shapes, the enhanced image brings more disparity in the intensity level to bring better contrast. Only some intensity levels have different pixel contents. This is because both functions have different calculation and interpolation technique. However, both techniques are successful in enhancing the images.

The experiment gives me understanding in many areas of computer graphics. During the experiment, I had some problems programming the CDF function, but after analysing and reading the guidance, the experiment is finished successfully.