VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
Faculty of Computer Science and Engineering

DIGITAL IMAGE PROCESSING AND COMPUTER VISION (CO3057)

Assignment

A DEEP INVESTIGATION RELATED TO HISTOGRAM EQUALIZATION IN COMPUTER VISION

Lecturer: Dr. Nguyen Duc Dung
Student: Doan Viet Tu - 1952521

HO CHI MINH CITY, SEMESTER 2022 - 2023

Contents

1 INTRODUCTION

In the field of Computer Vision, the ability to enhance and manipulate images plays a crucial role in various applications, ranging from image processing to object detection and recognition. One fundamental technique used for image enhancement is histogram equalization. Histogram equalization is a method that aims to improve the visual quality and enhance the details of an image by redistributing the pixel intensities.

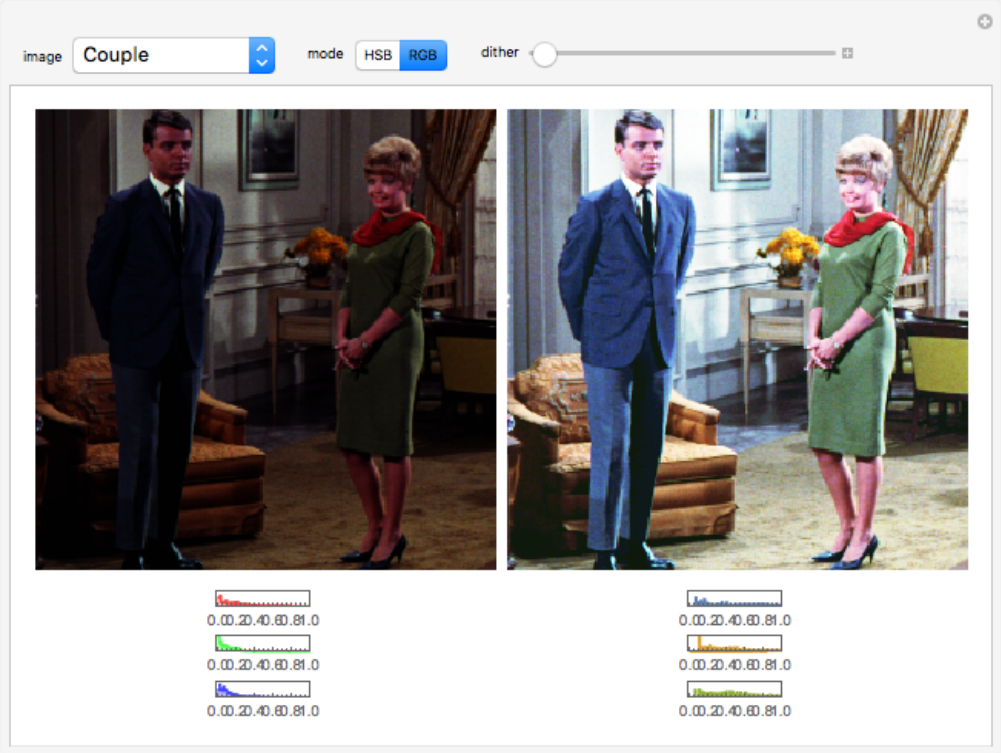

Figure 1: An illustration of RGB-based histogram equalization. (source: wolfram.com)

The concept behind histogram equalization is rooted in the understanding of the histogram of an image. The histogram represents the frequency distribution of pixel intensities in an image, essentially providing a visual summary of the image's tonal distribution. By analyzing and modifying the histogram, I can alter the contrast and enhance the details in the image.

The concept behind histogram equalization is rooted in the understanding of the histogram of an image. The histogram represents the frequency distribution of pixel intensities in an image, essentially providing a visual summary of the image's tonal distribution. By analyzing and modifying the histogram, I can alter the contrast and enhance the details in the image.

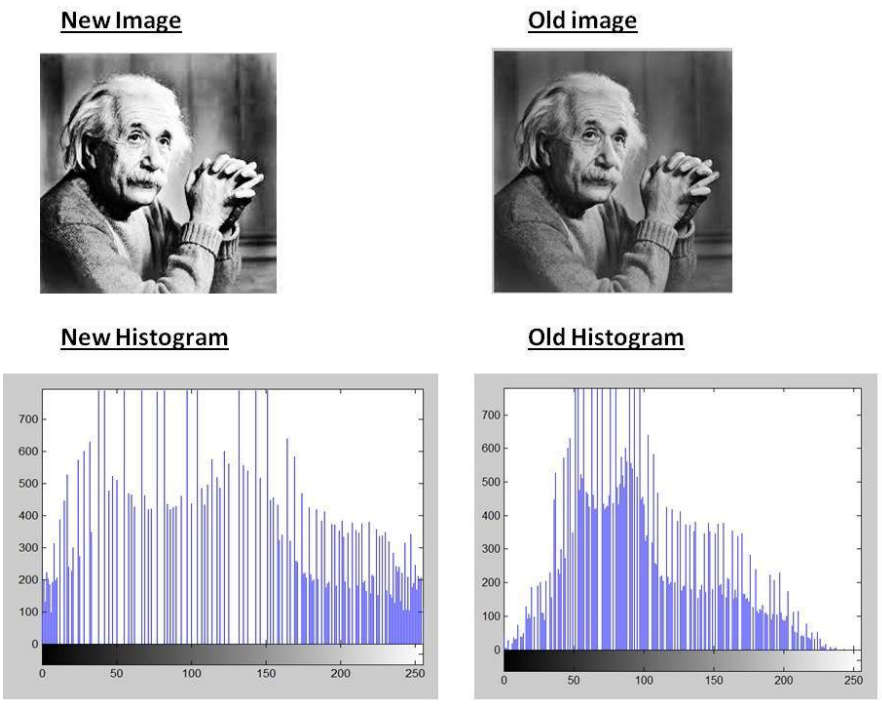

Figure 2: It is useful for images that are almost filled with bright or dark color. Histogram equalization in this situation can spread out the distribution of pixels to balance the contrast. (source: tutorialspoint.com)

Histogram equalization works by stretching the range of intensities in an image, so that the entire available intensity range is better utilized. This technique is particularly effective in scenarios where the image exhibits poor contrast or has a narrow intensity range, causing details to be obscured or washed out. By redistributing the intensities, histogram equalization can improve the overall appearance of the image and reveal details that Ire previously hidden.

In this essay, I will make a deep investigation into the concepts and algorithms of histogram equalization. The study will also be related to the process of histogram equalization, discuss its advantages and limitations, and examine real-world examples where it has been successfully applied in Computer Vision tasks. Additionally, I will make some experiments using C++ language to develop a small program that uses Cumulative Distribution Function (CDF) to test the efficiency of processing the image with histogram equalization.

2 LITERATURE SURVEY

2.1 Concepts and Algorithms

2.1.1 Definition

In histogram equalization, there are various kinds of concepts as well as techniques. They provide different approaches to enhance image contrast, improve visual quality, and reveal details in images. Depending on the specific requirements and constraints of a given application, different variants and adaptations of histogram equalization can be utilized.

Below are some concepts and algorithms of histogram equalization that are famous and commonly used:

- **Cumulative Distribution Function (CDF):** The CDF represents the accumulated probability distribution of pixel intensities in an image. It is computed from the histogram and is a fundamental component in histogram equalization algorithms.
- **Global Histogram Equalization:** This is the basic form of histogram equalization, where the entire image's histogram is considered for equalization. It provides a global enhancement to the image by adjusting the intensity levels.
- **Local Histogram Equalization:** In local histogram equalization, the image is divided into smaller regions or blocks, and histogram equalization is applied independently to each block. This approach helps in preserving local details and reducing artifacts caused by global equalization.
- **Adaptive Histogram Equalization (AHE):** AHE is an extension of histogram equalization where the equalization process is adaptive and varies locally based on the image content. It aims to address the limitations of global histogram equalization by adjusting the equalization based on the local statistics of the image.
- **Contrast-Limited Adaptive Histogram Equalization (CLAHE):** CLAHE is a variant of AHE that overcomes the problem of amplifying noise in areas with low contrast. It limits the contrast enhancement by applying a clipping or thresholding operation on the histogram.
- **Multi-scale Retinex with Color Restoration (MSRCR):** MSRCR is an advanced algorithm that combines histogram equalization with multiscale image decomposition. It aims to enhance the image while preserving its natural appearance and color.
- **Variants for Color Images:** Histogram equalization can be extended to color images by considering each color channel independently or by transforming the image to a different color space before applying equalization.

2.1.2 Applications

Some common examples or real-life applications of the above concepts are:

- **Cumulative Distribution Function (CDF):** The CDF is used in applications like image segmentation. By analyzing the CDF, segmentation algorithms can separate foreground and background regions effectively.

Figure 3: CDF helps identify distinct regions or objects based on the distribution of pixel intensities. (source: superannotate.com)

- **Global Histogram Equalization:** Global histogram equalization finds applications in various fields, including satellite imagery analysis. It can be used to enhance satellite images, making them more visually appealing and improving the visibility of land features, vegetation, and other important details.

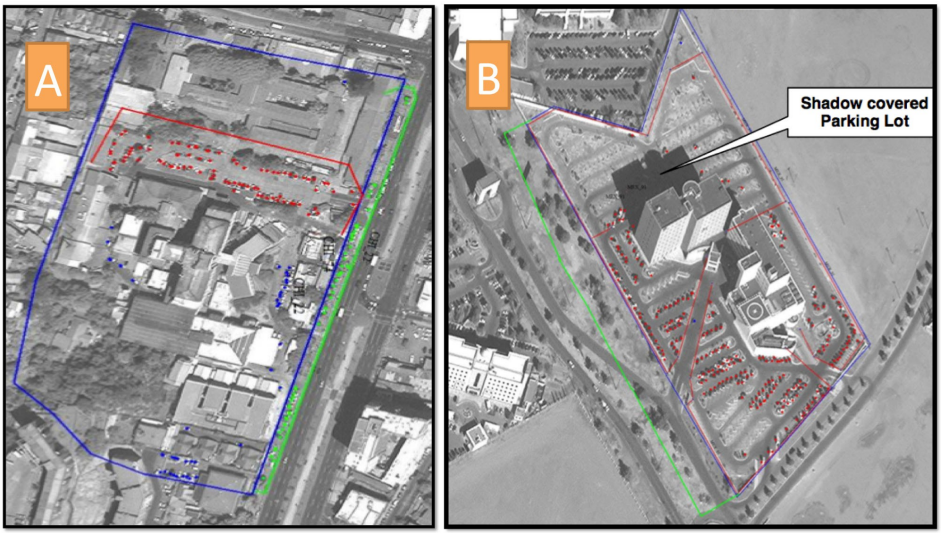

Figure 4: Global histogram equalization can make satellite imagery analysis more affordable. (source: nature.com)

- **Local Histogram Equalization:** Local histogram equalization is widely used in computer vision applications like face recognition. By applying local equalization techniques, the contrast of facial features can be improved, helping in accurate face detection and recognition.

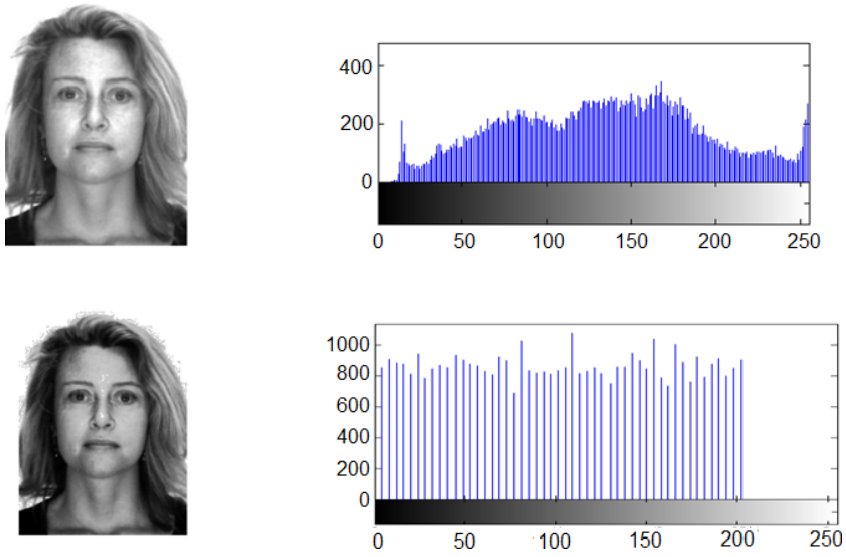

Figure 5: Ramírez-Gutiérrez, Kelsey Aleiandra et al. "A Face Recognition Algorithm using Eigenphases and Histogram Equalization.". [3]

- **Adaptive Histogram Equalization (AHE):** AHE is commonly used in real-time video processing applications such as surveillance systems. It helps enhance visibility in low-light or dynamically changing lighting conditions, making it easier to detect and track objects or activities in the video stream.

Figure 6: AHE can be applied to help the real-life scenario of security surveillance in parking lots. (source: news4sanantonio.com)

- **Contrast-Limited Adaptive Histogram Equalization (CLAHE):** Since CLAHE is a variant of AHE, it has some similar applications with CLAHE. Moreover, CLAHE can also be applied in aerial imaging. It can be used to enhance the contrast in aerial photographs, improving the visibility of ground features and aiding in tasks like land cover classification and target identification.

- **Multi-scale Retinex with Color Restoration (MSRCR):** For the example of applying MSRCR in image restoration, in the below MSRCR output image, the color contrast has been improved compared to Multi-scale Retinex. Still, the color contrast is not as close to the original image.

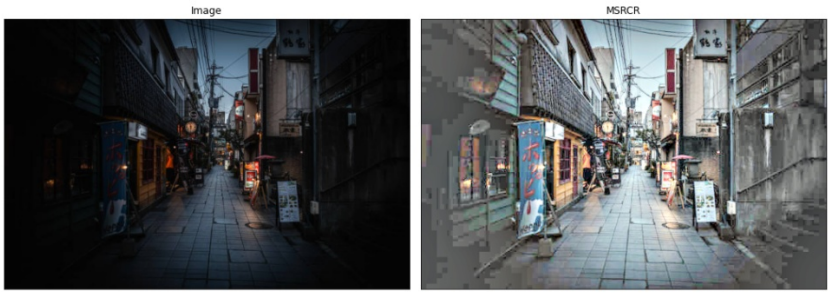

Figure 7: The MSRCR is not a general working method but it can provide enough benefits in some situations. (source: Santha Lakshmi Narayana's Blog)

- **Variants for Color Images:** Color image histogram equalization techniques, such as RGB-based equalization, are used in applications like color image enhancement and display. They help improve the contrast and color balance of images, making them more vibrant and visually appealing.

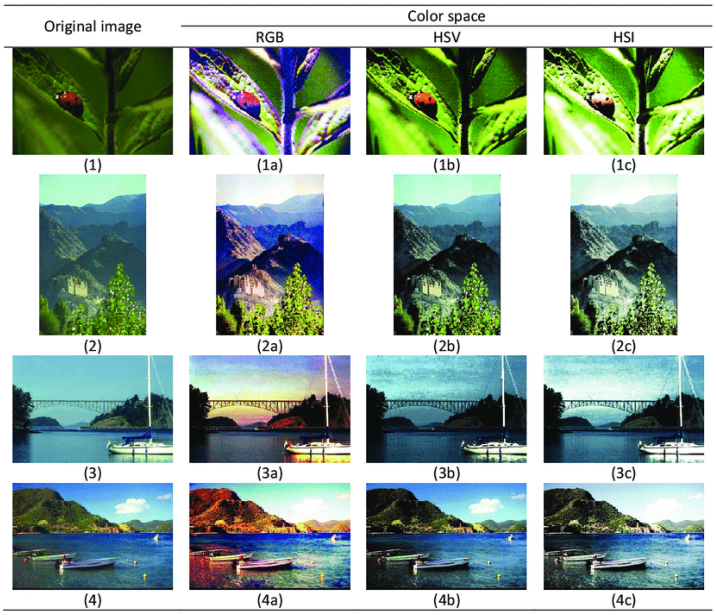

Figure 8: Histogram equalization using different color spaces. (source: researchgate.net)

2.1.3 Summary

In summary, the concepts and algorithms associated with histogram equalization provide a range of approaches to address different application requirements and challenges. Histogram equalization, whether applied globally or locally, offers benefits in various domains. In medical

imaging, it aids in interpreting X-rays and medical scans, while in satellite imagery analysis, it enhances the visibility of land features. Face recognition systems leverage local equalization techniques to improve facial feature contrast, while surveillance systems utilize adaptive equalization to handle changing lighting conditions in real-time video processing. Furthermore, algorithms like CLAHE address the limitations of global equalization by limiting contrast enhancement, making them suitable for applications such as aerial imaging. MSRCR techniques excel in restoring color and details in underwater photography, while variants for color images enhance the contrast and color balance in various visual displays.

The wide range of real-life applications demonstrates the significance and versatility of histogram equalization concepts in Computer Vision. By leveraging these techniques, practitioners can achieve improved image quality, enhanced details, and better interpretability, thereby facilitating tasks such as image analysis, object detection, and image understanding across diverse domains.

2.2 Histogram Equalization Process

Histogram equalization can be made with a fairly straightforward process, using the calculation of Cumulative Distributive Function (CDF), let's take the figure 2 at section 1 as a sample image.

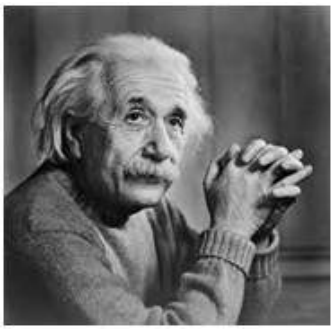

Figure 9: The sample image. (source: tutorialspoint.com)

The histogram of this image has been shown below.

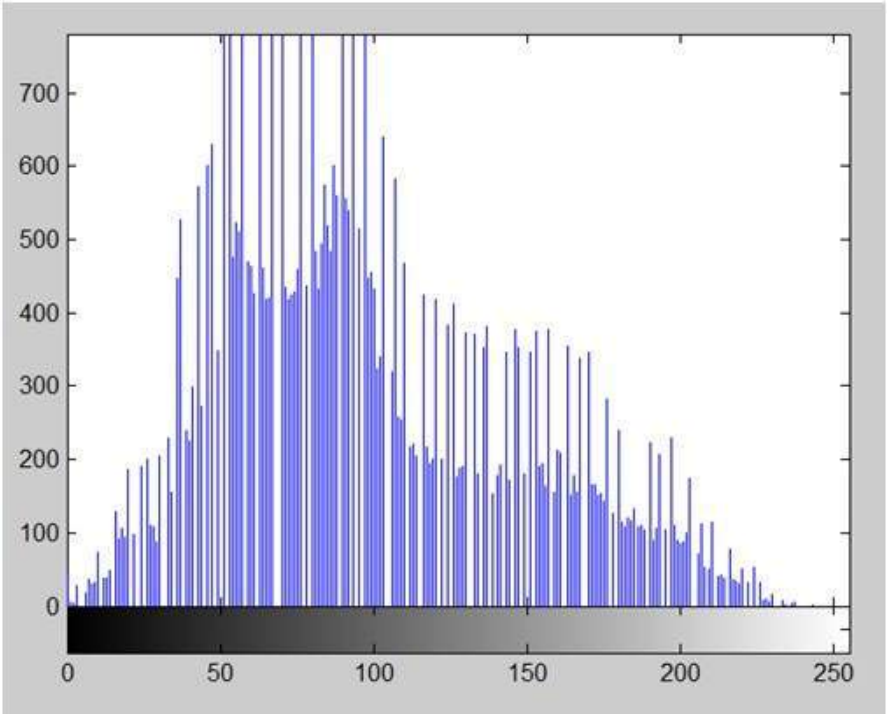

Figure 10: The histogram. (source: tutorialspoint.com)

Now we will perform histogram equalization to it. First we have to calculate the Probability Mass Function (PMF) of all the pixels in this image. PMF is the probability distribution of a discrete random variable, and provides the possible values and their associated probabilities. It is the function $p : R \to [0, 1]$ defined by

$$p_X(x) = P(X = x)$$

for $-\infty < X < \infty$, where P is a probability measure.

Then, we will move next to the next step which involves calculation of Cumulative Distributive Function (CDF) according the gray levels. The CDF of a real-valued random variable X is the function given by

$$F_X(x) = P(X \leq x)$$

where the right-hand side represents the probability that the random variable X takes on a value less than or equal to x.

Gray Level Value	CDF	CDF * (Levels - 1)
0	0.11	0
1	0.22	1
2	0.55	3
3	0.66	4
4	0.77	5
5	0.88	6
6	0.99	6
7	1	7

Let's assume that the CDF calculated in the second step looks like above, then we can multiply the CDF value with (Gray levels (minus) 1). Considering we have an 3 bits per pixel (bpp) image. Then number of levels we have are 8. And 1 subtracts 8 is 7. So we multiply CDF by 7. The table above is the result of this step.

Now what we have is the last step, in which we have to map the new gray level values into number of pixels. Let's assume our old gray levels values has these number of pixels: 2, 4, 6, 8, 10, 12, 14 and 16. Now if we map our new values to, then the table of result below is what we got.

Gray Level Value	New Gray Level Value	Frequency
0	0	2
1	1	4
2	3	6
3	4	8
4	5	10
5	6	12
6	6	14
7	7	16

When applying this technique to our original image, we can get the following image and its following histogram.

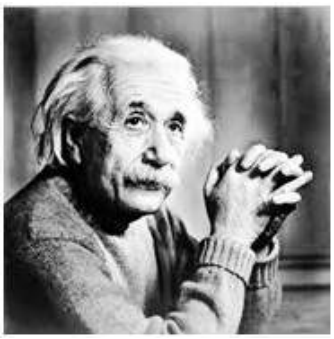

Figure 11: The resulted image. (source: tutorialspoint.com)

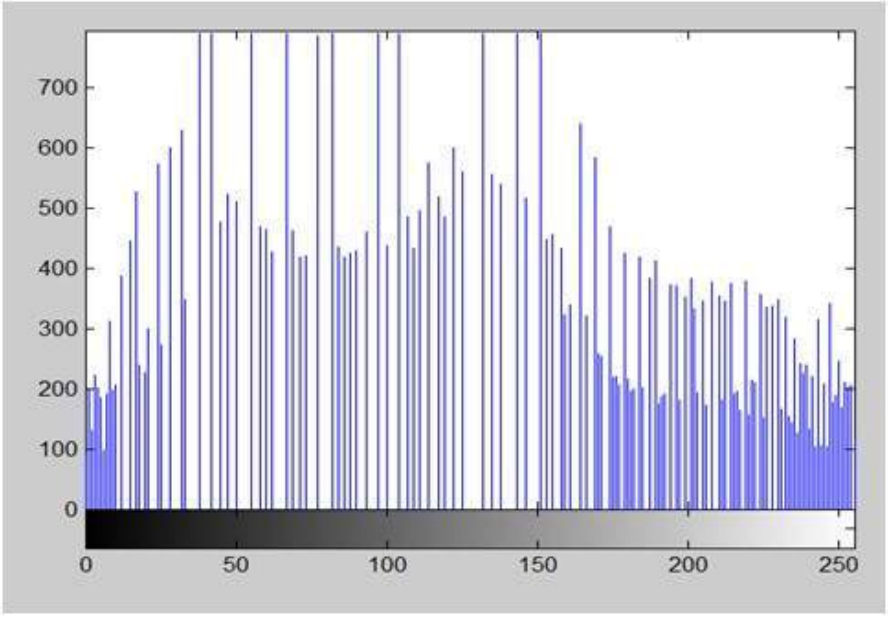

Figure 12: The new histogram. (source: tutorialspoint.com)

From the result, it can be clearly seen that the new image contrast has been enhanced and its histogram has also been equalized. There is also one important thing to be note here that during histogram equalization the overall shape of the histogram changes, where as in histogram stretching the overall shape of histogram remains same.

2.3 Advantages and Limitations

It is important to consider these advantages and limitations when applying histogram equalization in Computer Vision tasks. Depending on the specific requirements and characteristics of the images involved, alternative techniques or adaptations of histogram equalization may be more suitable to address the limitations and achieve desired results.

2.3.1 Advantages

- *Enhances image contrast*: Histogram equalization effectively stretches the intensity range of an image, leading to improved contrast. This makes it easier to distinguish between different objects or features in the image.
- *Enhances visual appearance*: By redistributing pixel intensities, histogram equalization can enhance the overall visual quality of an image. It can make images look more vibrant, with better-defined details and clearer boundaries.
- *Simple and computationally efficient*: Histogram equalization is a relatively straightforward technique to implement, requiring only the computation of a histogram and a cumulative distribution function. It can be efficiently applied to images in real-time or in batch processing scenarios.

- *Applicable to a wide range of images*: Histogram equalization is not limited to specific image types or domains. It can be applied to grayscale images as well as color images, making it a versatile technique in various Computer Vision applications.

2.3.2 Limitations

- *May result in over-enhancement or loss of information*: In certain cases, histogram equalization can lead to over-enhancement, where image features become overly exaggerated. This can result in unnatural-looking images or the loss of important details.
- *Ignores spatial information*: Traditional histogram equalization methods, including global and local approaches, do not consider the spatial relationships between pixels. This means that local variations or structures in the image may not be preserved during the equalization process.
- *Sensitive to noise*: Histogram equalization can amplify noise present in the image, particularly in low-contrast regions or areas with high-frequency noise. This can lead to the introduction of artifacts or unwanted visual distortions in the enhanced image.
- *Lack of adaptivity to specific image content*: Global histogram equalization treats the entire image as a whole, without considering the specific characteristics or content of different regions. This may result in undesirable effects in images with varying lighting conditions or regions with distinct contrast requirements.
- *Impact on color images*: Applying histogram equalization directly to color images can cause color shifts or distortions. Color preservation and maintaining natural color balance can be challenging with traditional histogram equalization techniques.

2.4 Related work

In recent years, histogram equalization has gained significant attention in the field of Computer Vision, with numerous scientific papers exploring its application and effectiveness. These papers aim to address various challenges related to image contrast enhancement and visual quality improvement. This section provides a comprehensive analysis of a few related works recently, summarizing their goals, methodologies, and key findings.

In paper [2], this study investigates the generalizability of published models using publicly available COVID-19 Computed Tomography data and assesses their predictive ability for COVID-19 severity. They employ *histogram equalization*, contrast limited adaptive histogram equalization (*CLAHE*), and a learning Gabor filter for inter-dataset analysis. Their findings demonstrate varying model generalization due to diverse image provenances and acquisition processes. Their best model exhibits high predictive accuracy for lung involvement, especially with expertly labeled stratification data. An ensemble model using a min-max function improves lung involvement prediction, with 75% accuracy for zero lung involvement and 96% accuracy for 75-100% lung involvement, showing an almost linear relationship between these stratifications.

Next, in paper [4], the authors claim that existing image enhancement methods have limitations in simultaneously improving global and local image contrast. To address this challenge, they propose a data-dependent *histogram equalization-based* method that enhances brightness while preserving global contrast and enhancing visibility of details. The approach incorporates spatial information from image context in density estimation for discriminative *histogram equalization*.

To mitigate the impact of non-uniform illumination, their work define spatial information based on edge-preserving smoothing for estimating image reflectance. For the result, this method effectively adjusts background brightness adaptively and reveals hidden image details in dark regions.

One of the latest paper related to this area is [5], which is released in 2023. This study concerns about "Adversarial Examples Detection with Enhanced Image Difference Features based on Local *Histogram Equalization*". They state that, Deep Neural Networks (DNNs) have made significant progress but are vulnerable to adversarial examples. Previous defense methods, such as feature compression and gradient masking, have been proposed. Nevertheless, these methods often provide limited defense against specific attacks, rendering them ineffective against unknown attack techniques. To address this, the authors propose an adversarial example detection framework based on a high-frequency information enhancement strategy. Their framework amplifies feature differences between adversarial and normal examples, improving detection performance without modifying existing models.

The discussed papers collectively contribute to the understanding and utilization of histogram equalization techniques in Computer Vision. They demonstrate the efficacy of histogram equalization in addressing specific challenges and achieving notable results in various domains. These works reinforce the potential of histogram equalization in enhancing image contrast, improving visual quality, and facilitating image analysis tasks. The collective findings inspire further research and advancements in the field of histogram equalization in Computer Vision.

3 DESIRED APPROACH

As mentioned above in the 1. Introduction, in this essay I will try to do some basic experiments from scratch by implementing a small program to apply CDF for histogram equalization. Therefore, let's begin with some theoretical knowledge.

3.1 Image and Pixels

Firstly, it is critical to understand **pixels** and **colour models**. The pixel is the smallest portion of an image or display that a computer is capable of printing or displaying. For gray-scale photographs, an 8-bit data value (with a range of 0 to 255) or a 16-bit data value (with a range of 0 to 65535) is frequently used. While for color images, there are 8-bit, 16-bit, 24-bit, and 30-bit colors available. The 24-bit colors having three 8-bit pixels, one for each of the red, green, and blue intensities, are considered as real colors. For example, a pixel with a value of 255, 0, and 0 is a red pixel.

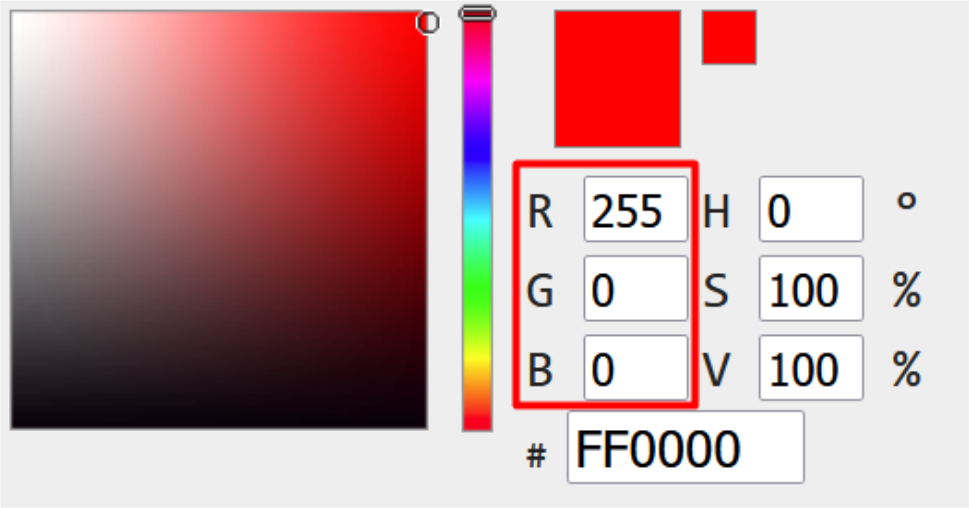

Figure 13: RGB value for a red pixel.

As mentioned above, the 24-bit colours are considered as real colours. Each 8-bit colour contains 8 digits (0 or 1) for each. The smallest number would be 00000000 in 8 digits representation, which is 0 and the largest number would be 11111111 in 8 digits representation, which is 255. Any pixels in 8-bit case could accommodate anything between 0 to 255 as a value for each of the colors.

3.1.1 Gray-scale image

A gray scale image is a digital image in which each pixel only contains one scalar value which is its intensity. The number of possible levels (intensity values) depends on the numerical type encoding the image.

For example, an image encoded with $n = 8$ bits will only have $L = 2^8 = 256$ possible intensity values going from 0 representing black to $L - 1 = 255$ representing white.

3.1.2 Color image

A color image is a digital array of pixel containing a color information. Each image can be decomposed into three different layers according to the three color channels encoded: Red, Green and Blue.

For instance, an 8 eight bits color images encode the Red and Green channel with three bits and the blue with two. Which could encode 256 different colors.

3.2 Cumulative Distribution Function (CDF)

Histogram equalization: is a method which increases the dynamic range of the gray-level in a low-contrast image to cover full range of gray-levels.

Histogram equalization is achieved by having a transformation function $T(r)$, which can be defined to be the **Cumulative Distribution Function (CDF)** of a given **Probability Density Function (PDF)** of a gray-levels in a given image (the histogram of an image can be considered as the **approximation** of the PDF of that image).

The cumulative distribution function (CDF) of random variable X is defined as

$$F_X(x) = P(X \leq x), \text{ for all } x \in R.$$

Both PDFs and CDFs provide likelihoods for random variables. However, PDFs calculate probability densities for $X = x$, while CDFs give the chances for $X \leq x$.

Consider the diagram shown below. The diagram shows the PDF $f(x)$, which gives us a rectangle between the points (a, b) when plotted. $f(x)$ has a value of $1/(b-a)$.

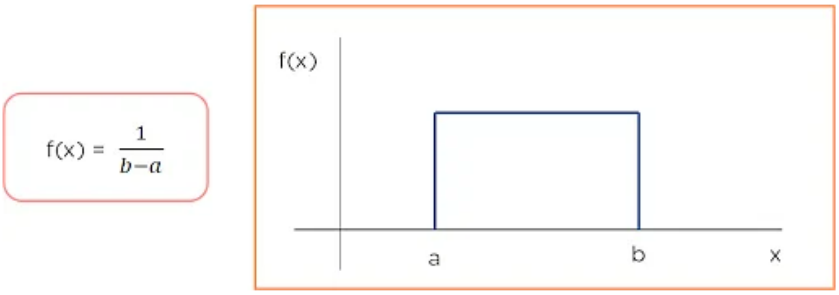

Figure 14: Probability Density Function. (source: simplilearn.com)

Now consider the point c on the $x - axis$. This is the point we need to find the CDF at. According to the definition, we should find the total PDF up to point c. This means that we have to find the area of the rectangle between points a and c.

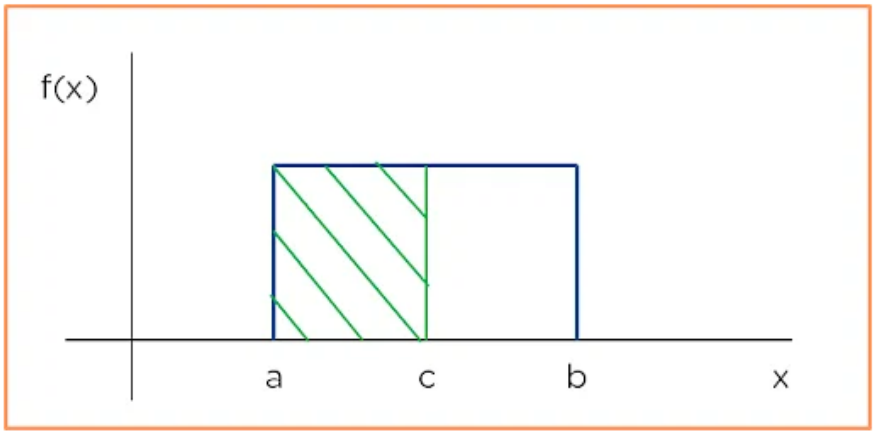

Figure 15: Calculating the CDF. (source: simplilearn.com)

In the end, we can get the CDF as

$$P(X \leq c) = (c - a)f(x)$$
$$(c - a)f(x) = (c - a)/(b - a)$$

As mentioned in section 2.2, which I already clarified the steps of calculation and transformation used in this experiment, I will deploy CDF in C++ for more understandings of this method and also to make some experiments and evaluate my results.

4 EXPERIMENTS

4.1 Description

In this experiment, I'll implement a small CDF-based program in C++ and compare the efficiency with the OpenCV, which is a library of programming functions mainly for real-time computer vision. This implementation can take the advantage of the convenience of Python for reading the image and the effectiveness of C++ for processing the image with histogram equalization.

4.1.1 Overall architecture

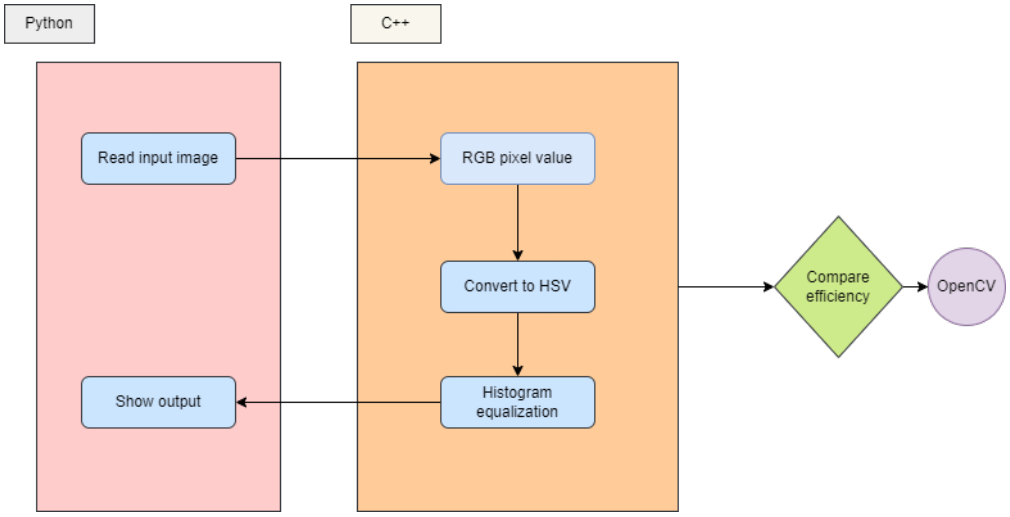

Figure 16: Overall architecture of the experiment.

4.2 How the tool works

4.2.1 Installation

For this tool to work, the installation of **OpenCV** library is required.

- Linux: Install on Linux
- Windows: Install on Windows

For other libraries such as *pybind11*, *pytest*, *matplotlib*, ... please refer to the online documents for more accurate details since it depends on each system and may have compatibility issues.

4.2.2 Demo

Let's see some output images after running the program, they all should be reach the balanced intensity of pixels value.

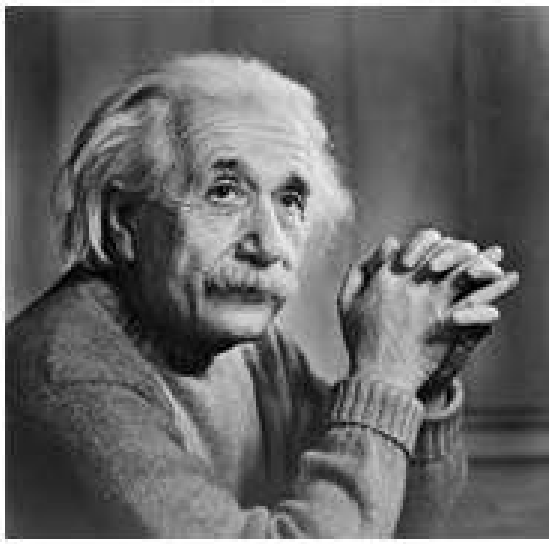

Figure 17: The output sample 1 picturing Albert Einstein.

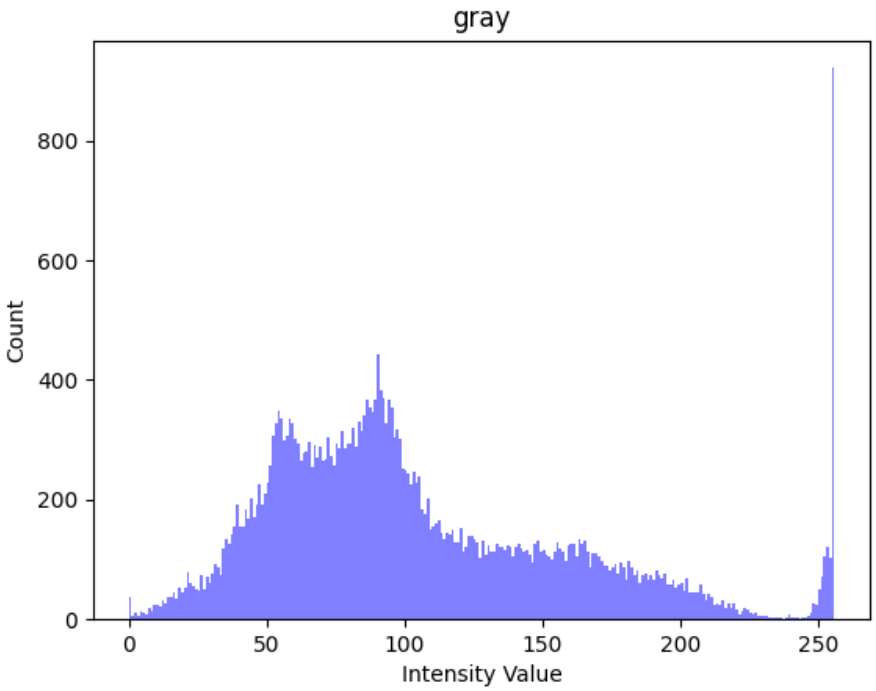

Figure 18: The histogram result of output sample 1.

Figure 19: The output sample 2 capturing a lake.

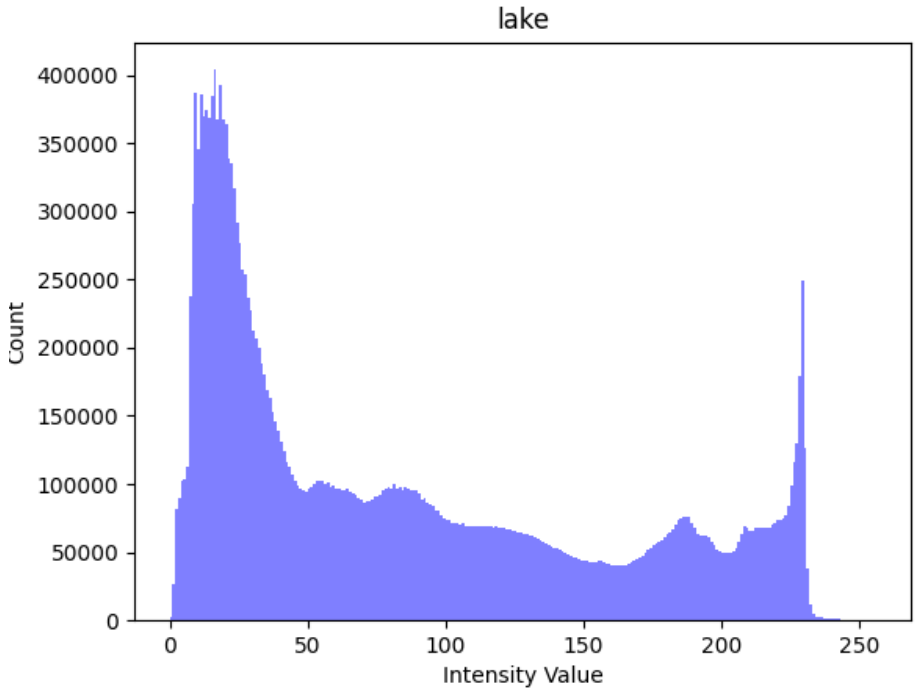

Figure 20: The histogram result of output sample 2.

After running the histogram equalization for some sample inputs, below are the results of the serial and thread histogram execution time, compared to OpenCV.

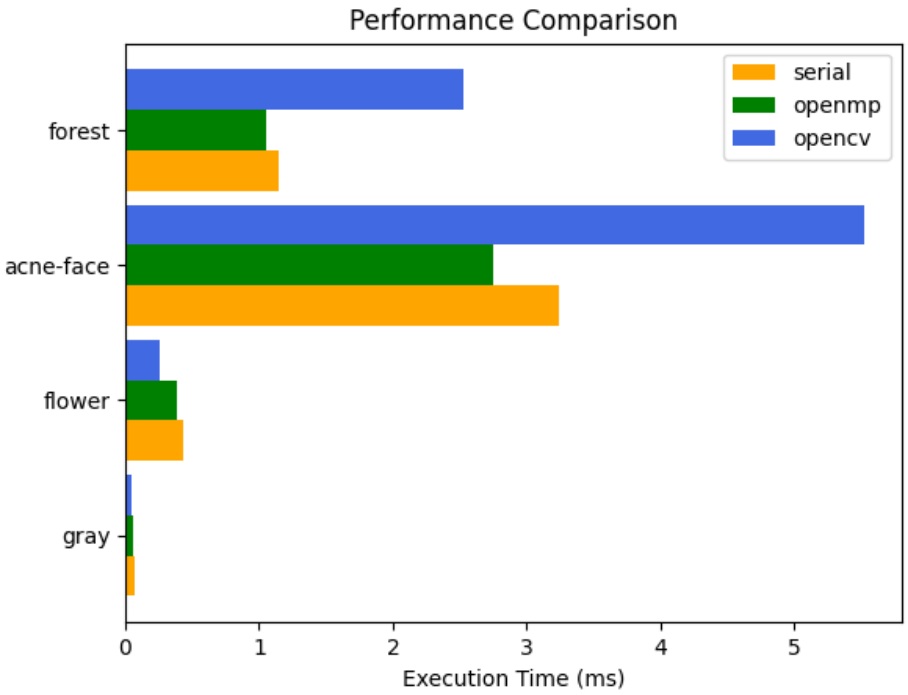

Figure 21: The performance comparison.

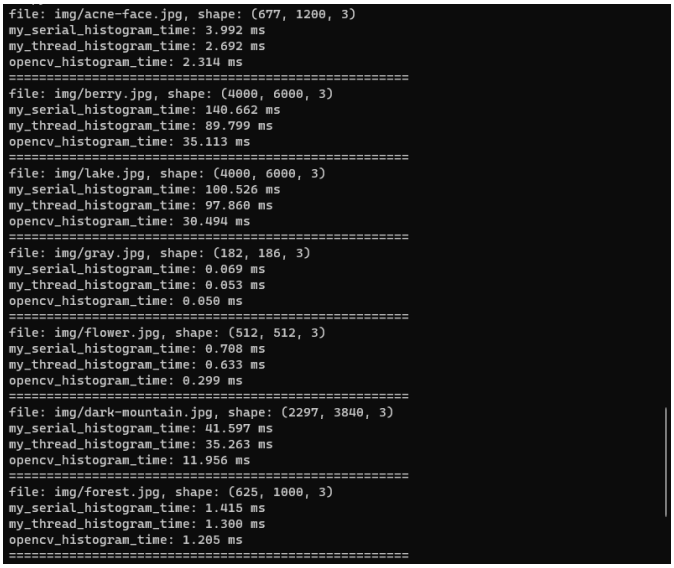

Figure 22: The result in terminal.

4.2.3 Source code

```cpp
#include <iostream>
#include "src/histEqSerial.h"
#define WIDTH 100
#define HEIGHT 100

using namespace std;

int main()
{
    int input[WIDTH * HEIGHT];
    for (int i = 0; i < WIDTH * HEIGHT; i++)
    {
        input[i] = i % 256;
    }

    int output[WIDTH * HEIGHT];

    int histogram[256] = {0};
    for (int y = 0; y < HEIGHT; y++)
    {
        for (int x = 0; x < WIDTH; x++)
        {
            histogram[input[y * WIDTH + x]]++;
        }
    }

    int cdf[256] = {0};
    cdf[0] = histogram[0];
    for (int i = 1; i < 256; i++)
    {
        cdf[i] = cdf[i - 1] + histogram[i];
    }

    float cdf_min = *std::min_element(cdf, cdf + 256);
    for (int i = 0; i < 256; i++)
    {
        cdf[i] = std::round((cdf[i] - cdf_min) * 255.0f / (WIDTH * HEIGHT - cdf_min));
    }

    for (int y = 0; y < HEIGHT; y++)
    {
        for (int x = 0; x < WIDTH; x++)
        {
            output[y * WIDTH + x] = cdf[input[y * WIDTH + x]];
        }
    }

    return 0;
}
```

Figure 23: main.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <cmath>

void _histEqSerial(int *output, const int *input, int width, int height)
{
    int histogram[256] = {0};
    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            histogram[input[y * width + x]]++;
        }
    }

    // Compute the CDF
    int cdf[256] = {0};
    cdf[0] = histogram[0];
    for (int i = 1; i < 256; i++)
    {
        cdf[i] = cdf[i - 1] + histogram[i];
    }

    // Normalize the CDF
    float cdf_min = *std::min_element(cdf, cdf + 256);
    for (int i = 0; i < 256; i++)
    {
        cdf[i] = std::round((cdf[i] - cdf_min) * 255.0f / (width * height - cdf_min));
    }

    // Map the intensity values of the input with CDF
    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            output[y * width + x] = cdf[input[y * width + x]];
        }
    }
}
```

Figure 24: serial.cpp

```cpp
#include <iostream>
#include <algorithm>
#include <cmath>
#include "omp.h"
#define COLOR_RANGE 256
#define PAD 16

void _histEqThread(int *output, const int *input, int width, int height)
{
    int histogram[COLOR_RANGE] = {0};
    for (int i = 0; i < width * height; i++)
    {
        histogram[input[i]]++;
    }

    int cdf[COLOR_RANGE] = {0};
    cdf[0] = histogram[0];

    // Compute the CDF
    for (int i = 1; i < COLOR_RANGE; i++)
    {
        cdf[i] = cdf[i - 1] + histogram[i];
    }

    // Normalize the CDF
    float cdf_min = *std::min_element(cdf, cdf + COLOR_RANGE);
    for (int i = 0; i < COLOR_RANGE; i++)
    {
        cdf[i] = std::round((cdf[i] - cdf_min) * 255.0f / (width * height - cdf_min));
    }

    // Map the intensity values of the input with CDF
    for (int i = 0; i < width * height; i++)
    {
        output[i] = cdf[input[i]];
    }
}
```

Figure 25: thead.cpp

```python
import numpy as np
import cv2
import myHist
import pytest
import time
import matplotlib.pyplot as plt

filename = 'img/lake.jpg'

def test_histEq():
    myHist_output = openCV_output = cv2.imread(filename)

    myHist_output[:, :, 0] = myHist.histEq_thread(myHist_output[:, :, 0])
    openCV_output[:, :, 0] = cv2.equalizeHist(openCV_output[:, :, 0])

    assert np.array_equal(myHist_output, openCV_output)

def test_histEq_serial_performance():
    my_img = opencv_img = cv2.imread(filename)

    start_time = time.time()
    my_img[:, :, 0] = myHist.histEq_serial(my_img[:, :, 0])
    my_hist_time = time.time() - start_time

    start_time = time.time()
    opencv_img[:, :, 0] = cv2.equalizeHist(opencv_img[:, :, 0])
    opencv_hist_time = time.time() - start_time

    performance = my_hist_time / opencv_hist_time
    assert my_hist_time < opencv_hist_time

def test_histEq_thread_performance():
    my_img = opencv_img = cv2.imread(filename)

    start_time = time.time()
    my_img[:, :, 0] = myHist.histEq_thread(my_img[:, :, 0])
    my_hist_time = time.time() - start_time

    start_time = time.time()
    opencv_img[:, :, 0] = cv2.equalizeHist(opencv_img[:, :, 0])
    opencv_hist_time = time.time() - start_time

    performance = my_hist_time / opencv_hist_time
    assert my_hist_time < opencv_hist_time

if __name__ == '__main__':
    files = ['img/acne-face.jpg', 'img/berry.jpg', 'img/lake.jpg', 'img/gray.jpg', 'img/flower.jpg', 'img/dark-mountain.jpg', 'img/forest.jpg']

    for f in files:
        my_img_serial = cv2.imread(f)
        my_img_thread = my_img_serial.copy()
```

Figure 26: test.py

```
        plt.hist(ycbcr[:, :, 0].ravel(), 256, [0, 256], color='b', alpha=0.5)
        plt.savefig(f'output/{f.split("/")[1].split(".")[0]}-intensity.png')
        start_time = time.time()
        my_img_serial[:, :, 0] = myHist.histEq_serial(my_img_serial[:, :, 0])
        my_hist_serial_time = time.time() - start_time
        start_time = time.time()
        my_img_thread[:, :, 0] = myHist.histEq_thread(my_img_thread[:, :, 0])
        my_hist_thread_time = time.time() - start_time
        start_time = time.time()
        opencv_img[:, :, 0] = cv2.equalizeHist(opencv_img[:, :, 0])
        opencv_hist_time = time.time() - start_time
        if f == 'img/forest.jpg':
            plt.figure()
            plt.title(f'RGB')
            plt.xlabel('Intensity Value')
            plt.ylabel('Count')
            rgb = cv2.imread(f)
            plt.hist(rgb[:,:,0].ravel(), 256, [0, 256], color='orange', alpha=0.5, label='B')
            plt.hist(rgb[:,:,1].ravel(), 256, [0, 256], color='b', alpha=0.5, label='G')
            plt.hist(rgb[:,:,2].ravel(), 256, [0, 256], color='g', alpha=0.5, label='R')
            plt.legend()
            plt.savefig(f'output/{f.split("/")[1].split(".")[0]}-rgb.png')
            plt.figure()
            plt.title(f'YCbCr')
            plt.xlabel('Intensity Value')
            plt.ylabel('Count')
            ycbcr = cv2.cvtColor(rgb, cv2.COLOR_BGR2YCrCb)
            plt.hist(ycbcr[:,:,0].ravel(), 256, [0, 256], color='orange', alpha=0.5, label='Y')
            plt.hist(ycbcr[:,:,1].ravel(), 256, [0, 256], color='b', alpha=0.5, label='Cr')
            plt.hist(ycbcr[:,:,2].ravel(), 256, [0, 256], color='g', alpha=0.5, label='Cb')
            plt.legend()
            plt.savefig(f'output/{f.split("/")[1].split(".")[0]}-ycbcr.png')
        print(f'my_serial_histogram_time: {my_hist_serial_time*1000:.3f} ms')
        print(f'my_thread_histogram_time: {my_hist_thread_time*1000:.3f} ms')
        print(f'opencv_histogram_time: {opencv_hist_time*1000:.3f} ms')
        print('================================================')
    exec_time = [[0.074, 0.435, 3.243, 1.148], [0.053, 0.391, 2.748, 1.055], [0.042, 0.262, 5.527, 2.526]]
    plt.figure()
    plt.title('Performance Comparison')
    plt.xlabel('Execution Time (ms)')
    plt.ylabel('Image')
    height = 0.3
    x = np.arange(len(exec_time[0]))
    plt.barh(x, exec_time[0], height, color='orange', label='serial')
    plt.barh(x + height, exec_time[1], height, color='green', label='openmp')
    plt.barh(x + height*2, exec_time[2], height, color='royalblue', label='opencv')
    plt.yticks(x + height, ['gray', 'flower', 'acne-face', 'forest'])
    plt.legend()
    plt.savefig('output/performance.png')

```

Figure 27: test.py

5 CONCLUSION

5.1 Future improvements

This essay already finished all the assignment's requirements. There will absolutely be improvement of the experiments for future use.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital image processing, 3rd ed.* 2007.

[2] M. J. Horry, S. Chakraborty, B. Pradhan, M. Fallahpoor, C. Hossein, and M. Paul, *Systematic investigation into generalization of covid-19 ct deep learning models with gabor ensemble for lung involvement scoring*, 2021. arXiv: 2105.15094 [cs.CV].

[3] K. A. Ramírez-Gutiérrez, D. Cruz-Pérez, J. Olivares-Mercado, M. Nakano-Miyatake, and H. Perez-Meana, "A face recognition algorithm using eigenphases and histogram equalization."

[4] X. Wu, T. Kawanishi, and K. Kashino, *Reflectance-guided, contrast-accumulated histogram equalization*, 2022. arXiv: 2209.06405 [cs.CV].

[5] Z. Yin, S. Zhu, H. Su, J. Peng, W. Lyu, and B. Luo, *Adversarial examples detection with enhanced image difference features based on local histogram equalization*, 2023. arXiv: 2305.04436 [cs.CV].