**Computer Vision**
CA1

Fall 2024
**Parsa Darban 810100141**
This file contains the report and results of the simulations conducted.

## Abstract

In this project, we will become familiar with the basic concepts of image processing. Tasks such as converting colored images to binary and grayscale, adjusting image contrast, and etc. We will then examine each of these processes in detail.

Next, we will explore image similarity (template matching).

## Part 1

### Q1

In this section, we will first convert color images to grayscale and binary. As we know, color images are three-dimensional matrices. The first two dimensions determine their size (pixels), and the third dimension specifies their color channels.

Since we are loading images using the OpenCV library, our color channel order is blue, green, and red.

Using the following formula and the function gray_image_func, we convert the image to grayscale:

$$gray\ scale = 0.2989 * red + 0.5870 * green + 0.1140 * blue$$

This transformation changes our image from three dimensions to two dimensions, with each value ranging from 0 to 255, indicating the grayscale intensity of each pixel.

Next, we use the function binary_image_func to convert our image into a binary matrix. For this, we first convert the image to grayscale, then apply a threshold. If the pixel value is greater than the threshold, it is set to 255; otherwise, it is set to 0.

The results of these functions can be found in the Part1/output/Q1 folder.

### Q2

Now, we will discuss changing the contrast of an image. Contrast refers to the difference in brightness or intensity between the darker and lighter parts of an image. High-contrast images have a wide range of brightness levels, making the image appear sharp and distinct, while low-contrast images have similar brightness levels across all pixels, making the image appear flat.

Here, two functions are defined to change the contrast for color and grayscale images, and the result of each can be seen in Part1/output/Q2.

### Q3

In this section, we add twenty units to the grayscale image. The only point to consider in this part of the code is that we should initially treat the grayscale range as 16-bit, shift it, and then convert it back to 8-bit. For example, if this is not done, pixels with a value of 255 would add 20, but the result would wrap around to 19, completely altering the color.

Adding a constant value to each pixel in a grayscale image increases brightness, enhancing visibility and revealing details in darker areas. This method is simple and computationally efficient, brightening the image uniformly. However, it may lead to saturation in very bright areas, causing a loss of detail, and can reduce contrast if the image already has high-intensity values.

The result of this can be seen in Part1/output/Q3.

## Q4

In this section, we need to add salt-and-pepper noise to the image. This noise resembles the static on old televisions and randomly appears as black and white spots scattered across the image.

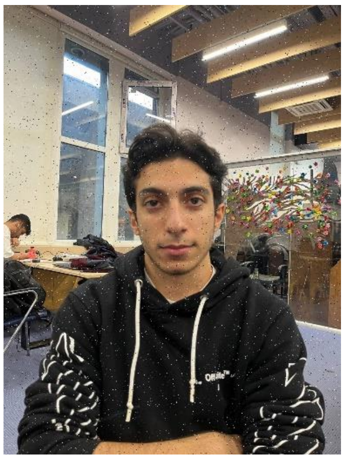


Figure 1 (Peper-Salt noise)

## Part 2

### Q1

Here, we first convert the image into three different color models and decompose it into their components. We choose BGR, HSV, and YUV. The decomposed images are available in file Part2/output/Q1.

BGR

In this decomposition we see the result as Blue, Green and Red.

HSV

HSV, which stands for Hue, Saturation, and Value, is a color model that represents colors in a way that is more aligned with human perception than traditional RGB (Red, Green, Blue) models. In the HSV model, Hue refers to the type of color (e.g., red, green, blue) and is represented as a degree on a color wheel, typically ranging from 0° to 360°. Saturation measures the intensity or purity of the color, with 0% being completely unsaturated (gray) and 100% being fully saturated (the pure color). Value, also known as brightness, represents the lightness or darkness of the color, ranging from 0% (black) to 100% (full brightness).

YUV:

YUV is a color space commonly used in video compression and broadcasting, which separates image luminance (Y) from chrominance (U and V) components. The Y channel represents the brightness or intensity of the image, allowing for the preservation of detail in grayscale. In contrast, the U and V channels carry the color information, where U represents the blue projection and V represents the red projection.

### Q2

We want to plot the histogram of a color and grayscale image. A histogram shows the number of pixels for different color intensity levels. For example, for a BGR image, we have:

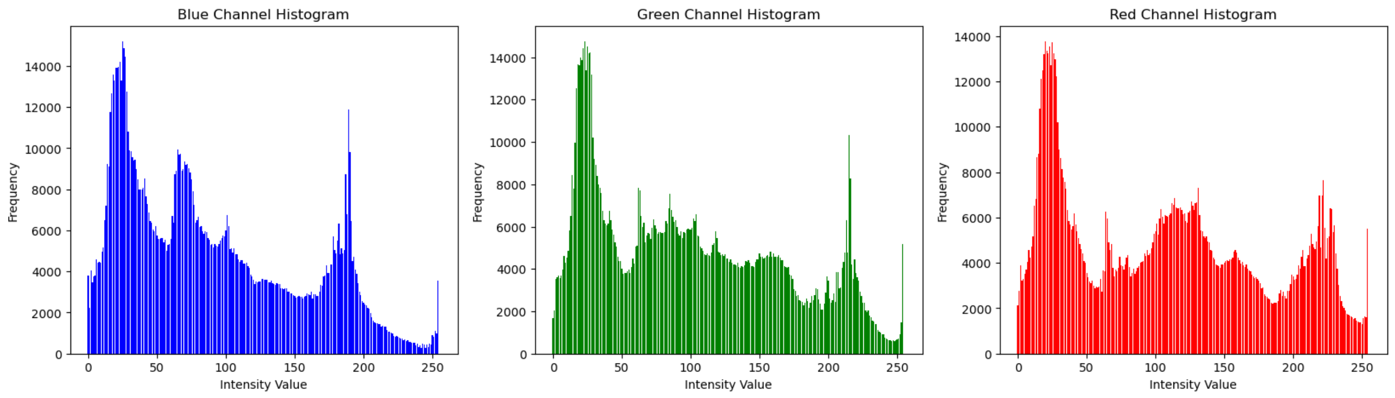


Figure 2 (BGR channel histogram)

Also, we have for the grayscale image:

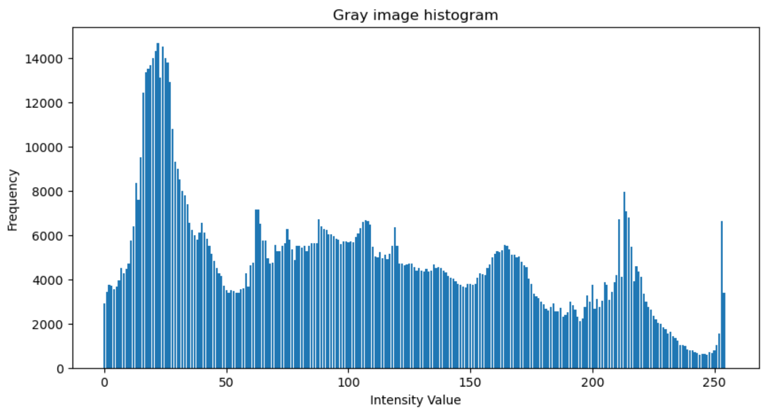


Figure 3 (Grayscale histogram)

As mentioned, contrast is the difference between the brightness and darkness of an image. Therefore, if we want to have a high-contrast image, the pixel values in the image should be spread over a wider range. However, the more this spectrum is spread over a smaller range, the lower the contrast of our image will be.

We can say that our image, whether in color or grayscale, has high contrast. However, if we plot the histogram for an image whose contrast we have adjusted in the previous sections, the result will be as shown below, confirming the explanations provided.

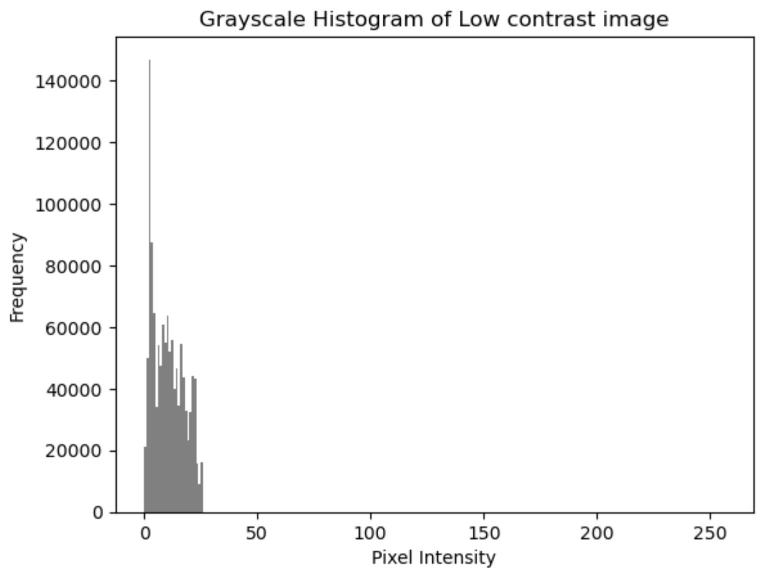


Figure 4 (low contrast image)

## Q3

In this section, we focus on image histogram equalization. Histogram equalization is an effective technique for enhancing image quality by redistributing pixel intensities, thereby increasing contrast and revealing hidden details. This method helps balance brightness and improves the clarity of textures and edges.

For example, the image below shows before and after equalization, demonstrating the increase in contrast.

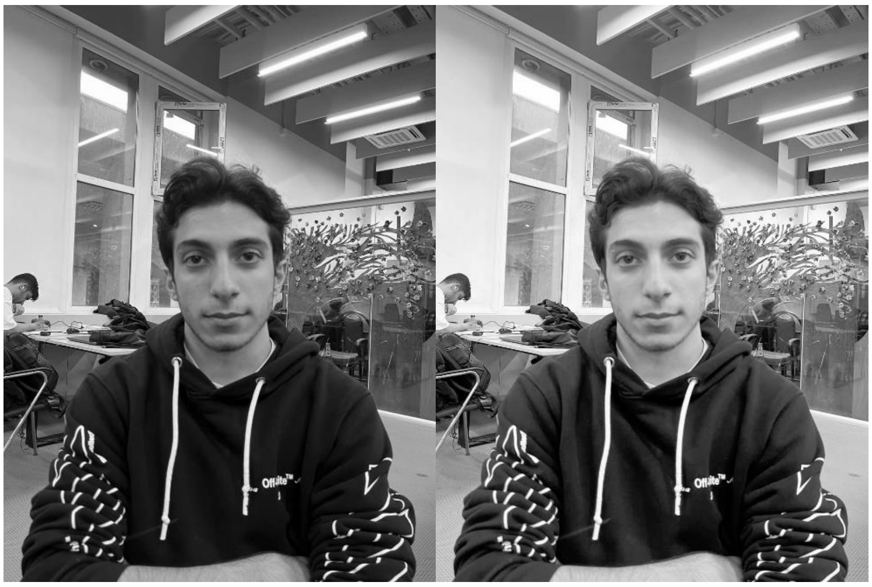


Figure 5 (Before and after equalizing (left to right))

The result of histogram is:

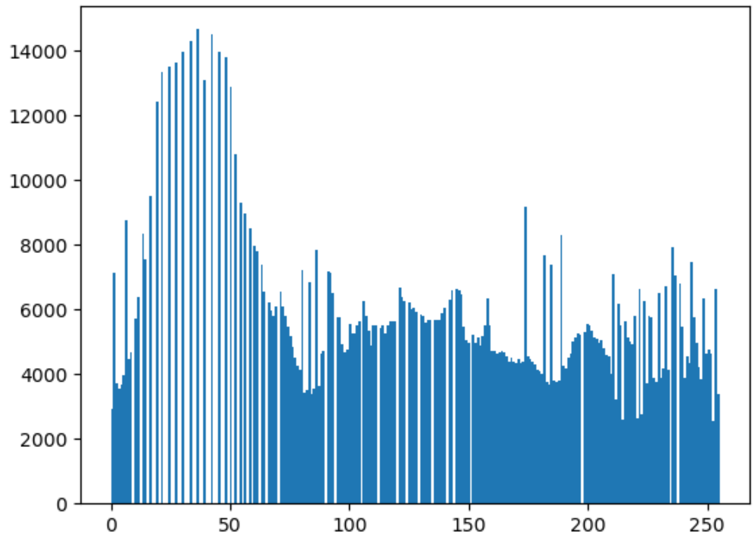


Figure 6 (Equalized histogram)

The mathematical logic behind histogram equalization involves calculating the CDF (Cumulative Distribution Function) for each intensity level.

## Q4

In this section, we perform the Fourier transform of the image. The Fourier transform shifts the image from the spatial domain to the frequency domain, providing essential information about the frequency content. This transform reveals high and low frequencies, enabling us to detect edges, details, noise, and repetitive patterns. It also helps us apply suitable filters to reduce noise or enhance specific details.

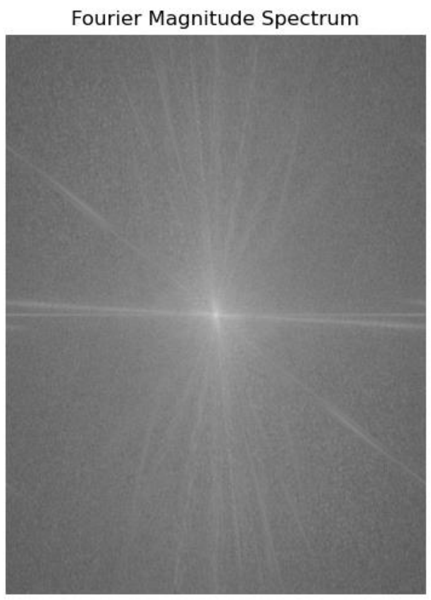
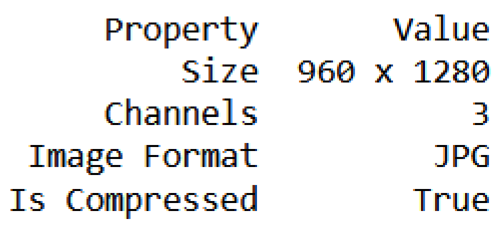


Figure 7 (Fourier transform of image)

In the first step, we used the logarithm of the frequency response for clarity in the results. The center point of the image indicates the low-frequency content, and the absence of brightness away from the center shows that the image lacks high-frequency information. Additionally, diagonal, vertical, and horizontal lines provide information such as repetitive patterns in the image or orientation at specific angles.

## Part 3

The result is:



```
        Property        Value
            Size   960 x 1280
        Channels            3
    Image Format          JPG
   Is Compressed         True
```

## Part 4

In this section, we aim to determine the similarity between images of a person based on four specific conditions:

1. First image: a front-facing view.

2. Second image: another image of the same person (looking at the camera with the exact same angle) but positioned closer to the camera.

3. Third image: with the face at a 45-degree angle to the camera.

4. Fourth image: with part of the face covered by a hand or mask.

Various algorithms can be used to solve this, and here we will apply two of these algorithms to grayscale images and one approach for binary images.

### Histogram matching

Histogram matching in template matching is a technique used to compare two images based on the similarity of their histograms. As we know, histogram represents the distribution of pixel intensities in an image, and matching histograms can help identify if two images share similar lighting, contrast, and overall tone, which is useful for various image comparison tasks.

We calculate histogram from function that we write and the use compareHist syntax to find the similarity.

Also we normalize the histogram because we want to fit within a defined range, between 0 and 1. We use NORM_MINMAX that it’s like we divided the value of data by maximum.

The result is:

```
Similarity score between image 1 and image 2: 0.6905910553631619
Similarity score between image 1 and image 3: 0.6149220242099003
Similarity score between image 1 and image 4: 0.8682669191286333
```

### NCC (normalized cross correlation)

As we know cross correlation can find the similarity of two signals. The cons of this mathematic way are sensitivity to the value of signal even it doesn’t have similarity compare with another one. For solving this problem, we normalized the cross correlation. Results are invariant to the global brightness changes, i.e. consistent brightening or darkening of either image has no effect on the result. The formula of this algorithm is:

$$R(x,y) = \frac{\sum_{x',y'} (T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

By implementing this algorithm according to the formula, we have:

```
NCC similarity: 0.3621893957535066
NCC similarity: 0.45671448100813383
NCC similarity: 0.43127906615617495
```

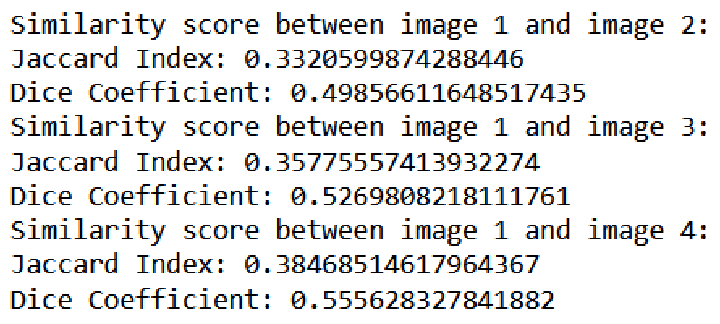
Please note that the NCCOEFF algorithm has also been implemented but was commented out due to its similar response to NCC. Both algorithms are alike, differing only in their output structure.

## Binary image matching

Another approach to examining this is to assess the similarity of the images in binary form. For this purpose, we use two indices: Jaccard and Dice. First, we binarize the images once using the custom function and once using a library function. The reason for using the library function is to apply a thresholding method called Otsu. This method automatically calculates the optimal threshold value based on the image histogram to minimize intra-class variance, making it suitable for images with two distinct brightness classes, such as images with backgrounds.

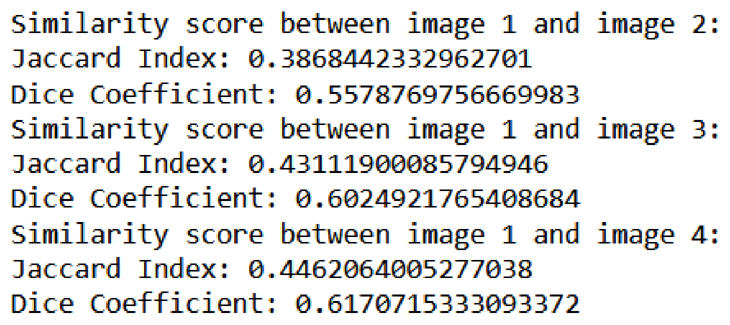
Next, we normalize the images to convert them to binary values (0 and 1). Then we apply the two mentioned indices. The Dice coefficient and Jaccard index both measure the overlap between two regions or images. The Dice coefficient is especially useful for applications where even small areas of overlap are important, such as in medical images where small regions like tumors are significant. The Jaccard index, on the other hand, is more commonly used for applications like object detection in images because it offers a stricter comparison. If there is a small difference in the region, the Jaccard index reflects this more strongly. In the output, the weighted Dice coefficient also shows the weighting for each matching.

The manual threshold result is:



```
Similarity score between image 1 and image 2:
Jaccard Index: 0.3320599874288446
Dice Coefficient: 0.49856611648517435
Similarity score between image 1 and image 3:
Jaccard Index: 0.35775557413932274
Dice Coefficient: 0.5269808218111761
Similarity score between image 1 and image 4:
Jaccard Index: 0.38468514617964367
Dice Coefficient: 0.555628327841882
```

And the OTSU threshold result is:



```
Similarity score between image 1 and image 2:
Jaccard Index: 0.3868442332962701
Dice Coefficient: 0.5578769756669983
Similarity score between image 1 and image 3:
Jaccard Index: 0.43111900085794946
Dice Coefficient: 0.6024921765408684
Similarity score between image 1 and image 4:
Jaccard Index: 0.4462064005277038
Dice Coefficient: 0.6170715333093372
```

In general, if your image is simple and has high contrast, the binary method is suitable for template matching. However, in more complex cases, you may need to use more advanced methods.