Digital Image Processing Lab Report 1
CSE 4733

Abdullah Al Jubaer Gem
ID: 210041226
Section: 2B

November 24, 2025

1 Task 1: Grayscale Histogram Generation

1.1 Problem Statement

Develop a function to compute and display the histogram of a grayscale image.

1.2 Solution Approach

The histogram function computes the frequency of each pixel intensity level (0-255) in the grayscale image. It uses `numpy.histogram` to calculate the distribution and `matplotlib` to plot the bar chart.

1.3 Implementation

```
def my_histogram(image, title="Image Histogram"):

    if len(image.shape) == 3:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    hist, bins = np.histogram(image.flatten(), 256, [0, 256])

    plt.figure(figsize=(8, 4))
    plt.title(title)
    plt.xlabel("Pixel Value")
    plt.ylabel("Frequency")
    plt.bar(range(256), hist, width=1, color='black')
    plt.xlim([0, 255])
    plt.grid(alpha=0.3)
    plt.show()
```

1.4 Output

The following figure shows the histogram of the grayscale version of the input image.

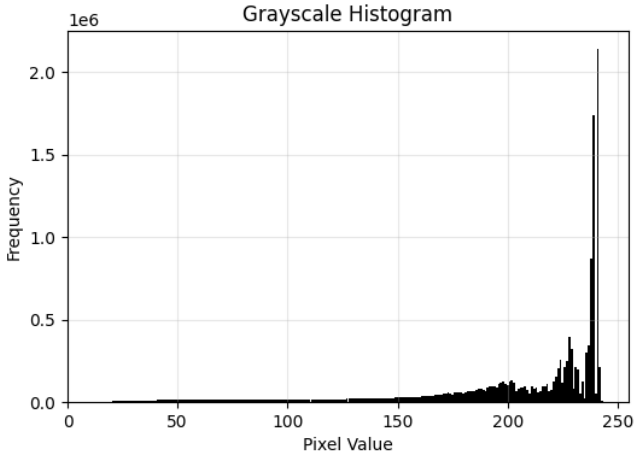

Figure 1: Grayscale Histogram

1.5 Observation

The histogram reveals the distribution of pixel intensities, showing whether the image is predominantly dark, bright, or well-balanced.

2 Task 2: Color Channel Separation and Display

2.1 Problem Statement

Extract and display the three color channels (Red, Green, Blue) of a color image, with each channel displayed in its respective color.

2.2 Solution Approach

The image is split into its Blue, Green, and Red components using OpenCV. To visualize each channel in its respective color, we create a 3-channel image where the other two channels are set to zero.

2.3 Implementation

```
def channel_show(color_image):
    b, g, r = cv2.split(color_image)

    zeros = np.zeros_like(b)

    blue_img  = cv2.merge([b, zeros, zeros])
    green_img = cv2.merge([zeros, g, zeros])
    red_img   = cv2.merge([zeros, zeros, r])

    cv2_imshow(blue_img);  print("Blue Channel")
    cv2_imshow(green_img); print("Green Channel")
    cv2_imshow(red_img);   print("Red Channel")

    return b, g, r
```

2.4 Output

The three separated color channels are shown below.

Figure 2: Separated Red, Green, and Blue Channels

2.5 Observation

Separating channels visualizes the contribution of Red, Green, and Blue components to the final image, with bright areas indicating high intensity of that specific color.

3 Task 3: Color Histogram Analysis

3.1 Problem Statement

Generate histograms for all three color channels of a color image to analyze the distribution of Red, Green, and Blue intensities.

3.2 Solution Approach

The `channel_show` function is used to separate the channels, and then `my_histogram` is called for each channel to plot its distribution.

3.3 Implementation

```
def color_histogram(color_image):
    b, g, r = channel_show(color_image)

    my_histogram(r, "Red Channel Histogram")
    my_histogram(g, "Green Channel Histogram")
    my_histogram(b, "Blue Channel Histogram")
```

3.4 Output

The histograms for the Red, Green, and Blue channels are displayed below.

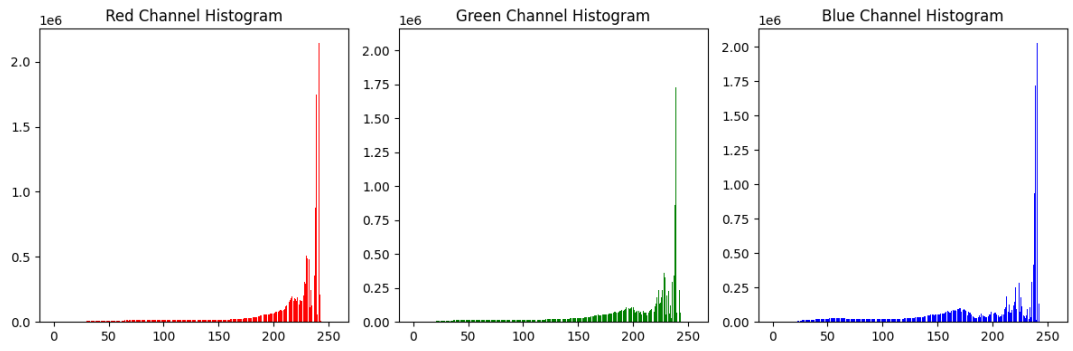

Figure 3: RGB Histograms

3.5 Observation

Comparing RGB histograms helps identify the dominant color channel and the overall color balance of the image.

4 Task 4: Color Channel Enhancement

4.1 Problem Statement

Implement a function to enhance a specific color channel by a given enhancement value between 0 and 1.

4.2 Solution Approach

The selected channel is multiplied by $(1 + \text{enhancement\_val})$. The result is clipped to the range [0, 255] to ensure valid pixel values and then merged back with the other channels.

4.3 Implementation

```python
def color_enhance(color_image, channel_number, enhancement_val):
    channel_map = {0: 2, 1: 1, 2: 0}
    ocv_channel = channel_map[channel_number]

    # Split into BGR
    b, g, r = cv2.split(color_image)
    channels = [b, g, r]

    factor = 1 + enhancement_val

    enhanced_channel = np.clip(channels[ocv_channel] * factor,
                               0, 255).astype("uint8")
    channels[ocv_channel] = enhanced_channel

    enhanced_img = cv2.merge(channels)

    cv2_imshow(enhanced_img)
    print(f"Enhanced channel: {['Red','Green','Blue'][channel_number]}")

    return enhanced_img
```

4.4 Output

The image below shows the result of enhancing the Red channel by 50%.

Figure 4: Enhanced Image (Red Channel +50%)

4.5 Observation

Enhancing a specific channel increases its intensity relative to others, giving the image a tint of that color.

5 Task 5: Grayscale Image Negation

5.1 Problem Statement

Create a function to compute the negative of a grayscale image using the formula:

$$g(x, y) = L_{max} - f(x, y)$$

where $L_{max} = 255$ for 8-bit images.

5.2 Solution Approach

Each pixel intensity is subtracted from 255. This inverts the brightness, making dark areas light and light areas dark.

5.3 Implementation

```
def grayscale_negative(gray_img):
    Lmax = 255
    negative_img = Lmax - gray_img
    cv2_imshow(negative_img)
    return negative_img
```

5.4 Output

The negative of the grayscale image is shown below.

Figure 5: Negative Grayscale Image

5.5 Observation

Inverting pixel values transforms light regions to dark and vice versa, effectively creating a photographic negative.

6 Task 6: Color Image Negation

6.1 Problem Statement

Extend the grayscale negation function to work with color images.

6.2 Solution Approach

The negation operation is applied to the color image. Since NumPy operations are element-wise, subtracting the image array from 255 inverts all three channels (R, G, B) simultaneously.

6.3 Implementation

```
def invert_color(color_image):
    negative_img = 255 - color_image
    cv2_imshow(negative_img)
    return negative_img
```

6.4 Output

The negative of the color image is shown below.

Figure 6: Negative Color Image

6.5 Observation

Inverting color channels produces complementary colors, where red becomes cyan, green becomes magenta, and blue becomes yellow.