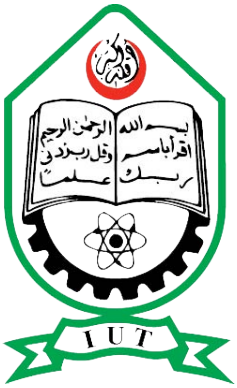
Islamic University of Technology



Digital Image Processing Lab

CSE 4734

LAB 1 REPORT

Submitted by:

Tanjil Hasan Khan
Student ID: 210041246
Section: 2B
BSc in CSE, Dept of Computer Science and Engineering
Islamic University of Technology (IUT)

November 24, 2025

Contents

1 Introduction **3**

2 Task 1: Grayscale Histogram Generation **3**
- 2.1 Objective . . . 3
- 2.2 How I Implemented It . . . 4
- 2.3 Code Implementation . . . 4
- 2.4 Output Image . . . 5
- 2.5 Understanding the Output . . . 5

3 Task 2: Separating RGB Channels **5**
- 3.1 Objective . . . 5
- 3.2 How I Implemented It . . . 6
- 3.3 Code Implementation . . . 6
- 3.4 Output Images . . . 7
- 3.5 Understanding the Output . . . 7

4 Task 3: Histogram of Color Image and Dominant Color Analysis **8**
- 4.1 Objective . . . 8
- 4.2 How I Implemented It . . . 8
- 4.3 Code Implementation . . . 8
- 4.4 Output Image . . . 10
- 4.5 Understanding the Output . . . 10

5 Task 4: Color Channel Enhancement **10**
- 5.1 Objective . . . 10
- 5.2 How I Implemented It . . . 11
- 5.3 Code Implementation . . . 11
- 5.4 Output Image . . . 12
- 5.5 Understanding the Output . . . 12

6 Task 5: Negative of a Grayscale Image **13**
- 6.1 Objective . . . 13
- 6.2 How I Implemented It . . . 13
- 6.3 Code Implementation . . . 13
- 6.4 Output Image . . . 14
- 6.5 Understanding the Output . . . 14

7 Task 6: Negative Transformation for Color Images **15**
7.1 Objective 15
7.2 How I Implemented It 15
7.3 Code Implementation 15
7.4 Output Image 16
7.5 Understanding the Output 16

1 Introduction

Digital Image Processing (DIP) deals with the manipulation and analysis of images using computational techniques. Since images are represented as matrices of pixel intensities, various operations can be applied to extract information, improve visual quality, or transform images for further processing. The purpose of this lab is to implement several basic DIP functions manually to better understand how pixel-level operations work.

This lab consists of six tasks:

- **Task 1:** Generate a histogram of a grayscale image to observe intensity distribution.
- **Task 2:** Separate a color image into its Red, Green, and Blue channels and display each one.
- **Task 3:** Use the histogram function to plot RGB histograms and analyze color dominance.
- **Task 4:** Enhance a selected color channel by a given factor and display the enhanced image.
- **Task 5:** Generate the negative of a grayscale image using intensity inversion.
- **Task 6:** Extend the negative transformation to color images.

Each task includes the implemented code, a brief explanation of the approach, and the interpretation of the resulting output.

2 Task 1: Grayscale Histogram Generation

2.1 Objective

The goal of this task is to write a function that takes a grayscale image as input and generates its histogram. A histogram shows how frequently each intensity value (0–255) appears in the image. This helps us understand the brightness distribution and contrast characteristics of the image.

2.2 How I Implemented It

First, I loaded a grayscale image and converted it into a NumPy array. Then I flattened the array into a 1D list of pixel values. Using `numpy.histogram`, I calculated how many times each intensity value occurs. Finally, I plotted the histogram using `matplotlib`.

2.3 Code Implementation

```
def my_histogram(gray_image):
  #implement this function

  img = Image.open(gray_image).convert('L')
  img_array = np.array(img)

  hist, bins = np.histogram(img_array.flatten(), bins=256,
   range=(0,255))

  plt.figure(figsize=(12, 5))

  plt.subplot(1, 2, 1)
  plt.imshow(img_array, cmap='gray')
  plt.title("Grayscale Image")
  plt.axis('off')

  plt.subplot(1, 2, 2)
  plt.bar(range(256), hist, width=1.0)
  plt.title('Grayscale Histogram')
  plt.xlabel('Gray level (0-255)')
  plt.ylabel('Frequency')
  plt.xlim([0, 255])

  plt.tight_layout()
  plt.show()
```

Listing 1: Code Implementation for Task 1

2.4 Output Image

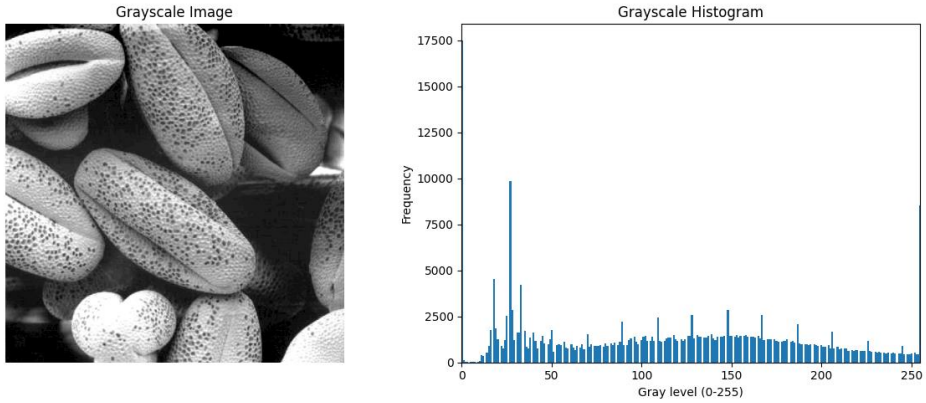


Figure 1: Grayscale Histogram

2.5 Understanding the Output

From the resulting histogram plot, I observed how the intensity values are spread across the image:

- Peaks in the lower intensity range indicate darker regions.
- Peaks in the higher intensity range indicate brighter regions.
- A narrow distribution means low contrast, whereas a wide spread means higher contrast.

Overall, the histogram helped me visualize the brightness characteristics of the grayscale image and understand how pixel intensities are distributed.

3 Task 2: Separating RGB Channels

3.1 Objective

The goal of this task is to take a color image and split it into its three individual channels: Red, Green, and Blue. Each channel should be displayed in its own color, meaning the red channel produces a red-tinted image, the green channel produces a green-tinted image, and similarly for blue. This helps us understand how each channel contributes to the final full-color image.

3.2 How I Implemented It

First, I loaded the input color image using PIL and converted it into an RGB NumPy array. Then I extracted the Red, Green, and Blue channels separately by indexing the third dimension of the array. To display each channel in its respective color, I kept the target channel unchanged and replaced the other two channels with zero matrices. Finally, I displayed the three channel images side by side using `matplotlib`.

3.3 Code Implementation

```
def channel_show(color_image):
    img = Image.open(color_image).convert('RGB')
    img_array = np.array(img)

    R = img_array[:, :, 0]
    G = img_array[:, :, 1]
    B = img_array[:, :, 2]

    zeros = np.zeros_like(R)

    red_img = np.stack([R, zeros, zeros], axis=2)
    green_img = np.stack([zeros, G, zeros], axis=2)
    blue_img = np.stack([zeros, zeros, B], axis=2)

    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.imshow(red_img)
    plt.title("Red Channel")
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.imshow(green_img)
    plt.title("Green Channel")
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.imshow(blue_img)
    plt.title("Blue Channel")
    plt.axis('off')

    plt.tight_layout()
```

```
    plt.show()


channel_show('Fig0630(01)(strawberries_fullcolor).tif')
```

Listing 2: Code Implementation for Task 2

3.4 Output Images

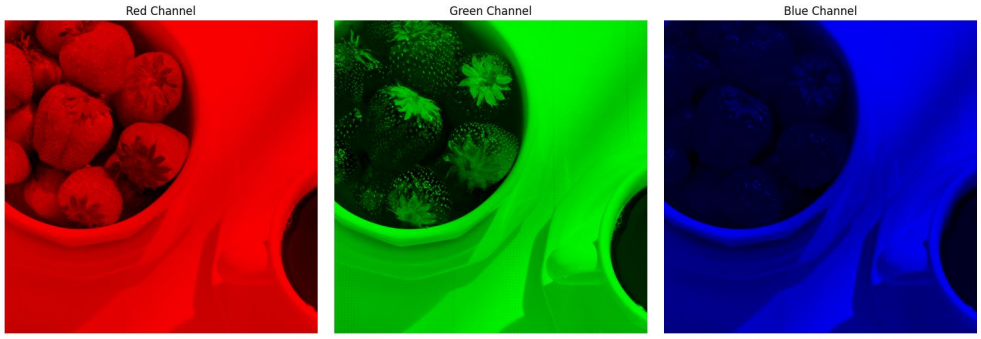


Figure 2: Separated Red, Green, and Blue Channels

3.5 Understanding the Output

From the channel-separated images, I observed the following:

- The **red channel** highlights areas where the original image contains strong red components (e.g., strawberries appear bright).
- The **green channel** emphasizes regions with more green intensity, often showing moderate brightness depending on image content.
- The **blue channel** displays brightness only in areas where blue is present; in many natural images, it tends to appear darker.

By visualizing each channel separately, I gained a better understanding of how RGB channels combine to form a full-color image.

4 Task 3: Histogram of Color Image and Dominant Color Analysis

4.1 Objective

The goal of this task is to generate individual histograms for the Red, Green, and Blue channels of a color image and determine which color is dominant. By comparing the intensity distributions of each channel, we can analyze how different colors contribute to the image's appearance.

4.2 How I Implemented It

I reused the histogram approach from Task 1 but applied it separately to the R, G, and B channels. After splitting the color image into its three components, I computed a histogram for each channel using `numpy.histogram`. Then, I calculated the mean intensity of each channel to identify which color is dominant. The channel with the highest average intensity is considered the most prominent in the image.

4.3 Code Implementation

```
def color_histogram(color_image):
    img = Image.open(color_image).convert('RGB')
    img_array = np.array(img)

    R = img_array[:, :, 0]
    G = img_array[:, :, 1]
    B = img_array[:, :, 2]

    hist_R, _ = np.histogram(R.flatten(), bins=256, range
   =(0,255))
    hist_G, _ = np.histogram(G.flatten(), bins=256, range
   =(0,255))
    hist_B, _ = np.histogram(B.flatten(), bins=256, range
   =(0,255))

    plt.figure(figsize=(15, 5))

    plt.subplot(1, 3, 1)
    plt.bar(range(256), hist_R, color='red')
    plt.title('Red Channel Histogram')
```

```
    plt.xlabel('Intensity (0-255)')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 2)
    plt.bar(range(256), hist_G, color='green')
    plt.title('Green Channel Histogram')
    plt.xlabel('Intensity (0-255)')
    plt.ylabel('Frequency')

    plt.subplot(1, 3, 3)
    plt.bar(range(256), hist_B, color='blue')
    plt.title('Blue Channel Histogram')
    plt.xlabel('Intensity (0-255)')
    plt.ylabel('Frequency')

    plt.tight_layout()
    plt.show()

    dom_R = R.mean()
    dom_G = G.mean()
    dom_B = B.mean()

    print("\nAverage Intensity Per Channel:")
    print(f"Red: {dom_R}")
    print(f"Green: {dom_G}")
    print(f"Blue: {dom_B}")

    dominance = {"Red": dom_R, "Green": dom_G, "Blue": dom_B}
    dominant_color = max(dominance, key=dominance.get)
    print(f"\nDominant color in the image: {dominant_color}")
```

Listing 3: Code Implementation for Task 3

4.4 Output Image

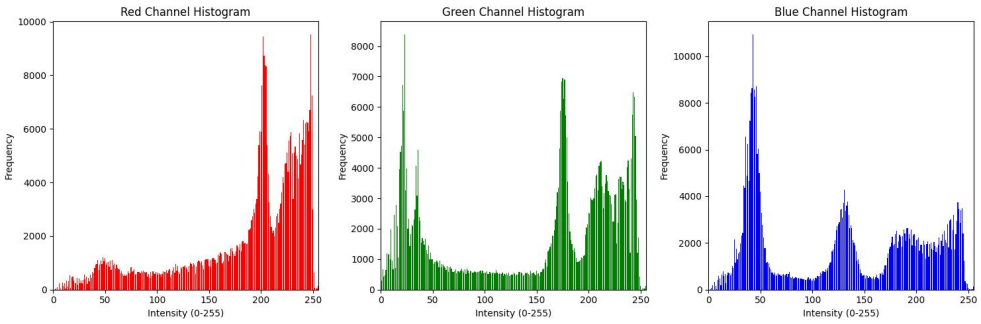


Figure 3: RGB Histograms and Dominant Color

4.5 Understanding the Output

From my printed output:

- Red channel mean intensity: **186.29**
- Green channel mean intensity: **145.85**
- Blue channel mean intensity: **126.47**

Based on these values, the **Red channel** has the highest average intensity. This indicates that the chosen image contains more red tones compared to green and blue.

The histogram visualizations also support this, as the red histogram shows higher frequency peaks. Through this task, I learned how RGB channels contribute individually to the overall color composition and how histograms help analyze color dominance in an image.

5 Task 4: Color Channel Enhancement

5.1 Objective

The goal of this task is to enhance a specific color channel (Red, Green, or Blue) of an image by a given enhancement factor. This helps us understand how increasing the intensity of one color affects the overall appearance of the image.

5.2 How I Implemented It

I loaded the color image and converted it into a NumPy array. To prevent overflow during multiplication, I converted the pixel values to `float32`. Then I increased the selected channel by multiplying it with (1+enhancement_val). After enhancement, I clipped the values to the valid range $[0, 255]$ and converted the image back to `uint8` for display.

5.3 Code Implementation

```
def color_enhance(color_image, channel_number,
    enhancement_val):
    img = Image.open(color_image).convert('RGB')
    img_array = np.array(img).astype(np.float32)

    img_array[:, :, channel_number] = (
        img_array[:, :, channel_number] * (1 +
    enhancement_val)
    )

    img_array = np.clip(img_array, 0, 255)
    enhanced_image = img_array.astype(np.uint8)

    return enhanced_image
```

Listing 4: Code Implementation for Task 4

5.4 Output Image

Figure 4: Enhanced Image (Example: R, G, B Channel Enhanced)

5.5 Understanding the Output

- Enhancing a channel increases the brightness of that particular color across the entire image.
- A higher enhancement value makes the selected color more dominant.
- For example, enhancing the red channel makes warm regions appear brighter and more intense.

This task helped me understand how individual color channels contribute to the visual appearance of an image, and how modifying a single channel can significantly change the overall color balance.

6 Task 5: Negative of a Grayscale Image

6.1 Objective

The goal of this task is to generate the negative of a grayscale image. For an 8-bit grayscale image, the negative transformation is defined as:

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

where $L_{\max} = 255$ and $f(x, y)$ is the intensity of the original image. This operation inverts the brightness values, turning dark regions into light and vice versa.

6.2 How I Implemented It

I first loaded the input image in grayscale mode and converted it into a NumPy array. Then I applied the intensity inversion formula by subtracting each pixel value from 255. Finally, I displayed both the original and its negative side-by-side for comparison.

6.3 Code Implementation

```
def invert_gray(gray_image):
    img = Image.open(gray_image).convert('L')
    img_array = np.array(img)

    L_max = 255
    negative = L_max - img_array

    negative_img = negative.astype(np.uint8)

    plt.figure(figsize=(10, 4))

    plt.subplot(1, 2, 1)
    plt.imshow(img_array, cmap='gray')
    plt.title("Original Grayscale Image")
```

```
    plt.axis("off")

    plt.subplot(1, 2, 2)
    plt.imshow(negative_img, cmap='gray')
    plt.title("Negative Image")
    plt.axis("off")

    plt.tight_layout()
    plt.show()
```

Listing 5: Code Implementation for Task 5

6.4 Output Image

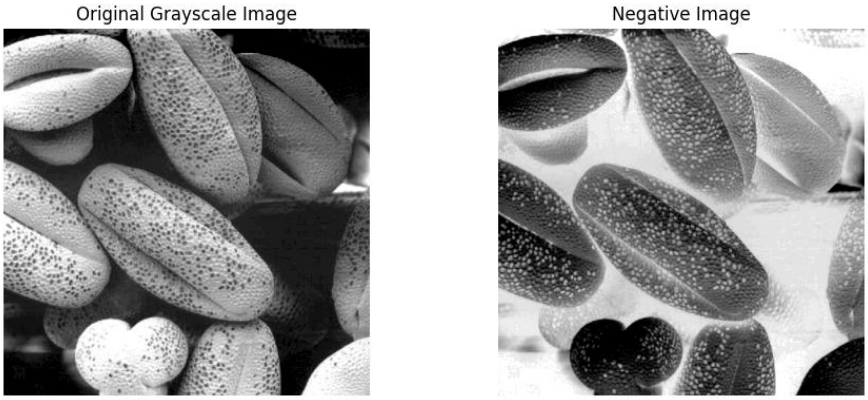


Figure 5: Original Grayscale Image and Its Negative

6.5 Understanding the Output

From the output, I observed the following:

- Dark regions in the original image become light in the negative.
- Light regions become dark, exactly as expected from the inversion equation.
- The overall structure and shapes remain the same, but the brightness interpretation is reversed.

This transformation is commonly used in photographic film processing and is helpful in certain image enhancement applications.

7 Task 6: Negative Transformation for Color Images

7.1 Objective

The goal of this task is to extend the negative transformation from grayscale images (Task 5) to full RGB color images. Instead of working on a single intensity channel, we apply the inversion formula to all three color channels (R, G, and B).

7.2 How I Implemented It

First, I loaded the input image as an RGB image and converted it into a NumPy array. Since each pixel has three values (one for each channel), I applied the negative transformation to the whole array using the formula:

$$g(x, y) = 255 - f(x, y)$$

This operation inverts every color channel, producing the complementary color. Finally, I displayed both the original and inverted color images side by side using `matplotlib`.

7.3 Code Implementation

```
def invert_color(color_image):
    img = Image.open(color_image).convert('RGB')
    img_array = np.array(img).astype(np.uint8)

    inverted = 255 - img_array

    plt.figure(figsize=(10, 4))

    plt.subplot(1, 2, 1)
    plt.imshow(img_array)
    plt.title("Original Color")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(inverted)
    plt.title("Inverted Color")
    plt.axis('off')
```

```
18
19      plt.tight_layout()
20      plt.show()
```

Listing 6: Code Implementation for Task 6

7.4 Output Image

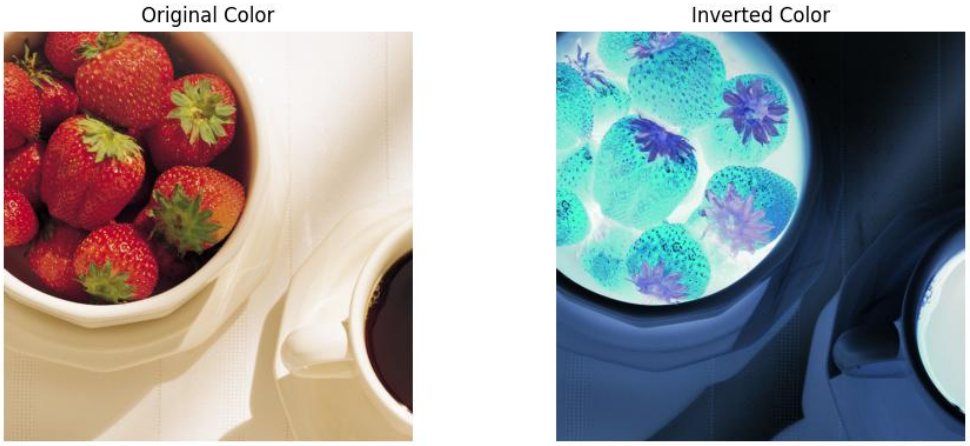


Figure 6: Original vs Inverted Color Image

7.5 Understanding the Output

The inverted color image displays the complementary colors of the original RGB image:

- High-intensity colors become dark, and dark colors become bright.
- Reds become cyan, greens become magenta, and blues become yellow.
- The transformation visually emphasizes regions with strong color presence.

This task helped me understand how pixel-wise transformations operate on multi-channel images and how color inversion works across the RGB spectrum.