Lab1: Experiment Report on Image Analysis using OpenCV

Name: Wu Han
Class: SEIEE 2301

November 21, 2024

1 Experiment Overview

In this experiment, we studied basic image processing concepts using OpenCV. We worked with grayscale and color images. Grayscale images are represented by a $W \times H$ matrix of intensity values ranging from 0 to 255, where 0 represents black and 255 represents white. Color images, on the other hand, are represented by a $W \times H \times 3$ matrix consisting of red (R), green (G), and blue (B) components.

The conversion formula from RGB to grayscale is as follows:

$$\text{Gray} = 0.299R + 0.587G + 0.114B \tag{1}$$

We analyzed images using histograms, including grayscale histograms, gradient histograms, and color histograms. The color histogram reflects the dominant colors, the grayscale histogram shows the distribution of intensity values, and the gradient histogram represents the image's texture.

OpenCV is a cross-platform image processing library with various interfaces like Python, Ruby, and MATLAB, which implements many common image processing and computer vision algorithms. In Lab1, we use python and OpenCV-python library to implement this task.

2 Experiment Environment

The experiment environment was set up as follows:

- Python3 (using VSCode)
- NumPy extension
- OpenCV library

3 Drawing Color Histograms

3.1 Problem Statement and Code Description

A color image is represented by a $W \times H \times 3$ matrix. A color histogram is used to measure the proportion of each color channel in the image.

The total energy of a color $E(c)$ is given by:

$$E(c) = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} I(x, y, c) \tag{2}$$

The proportion of each color channel is then calculated as follows:

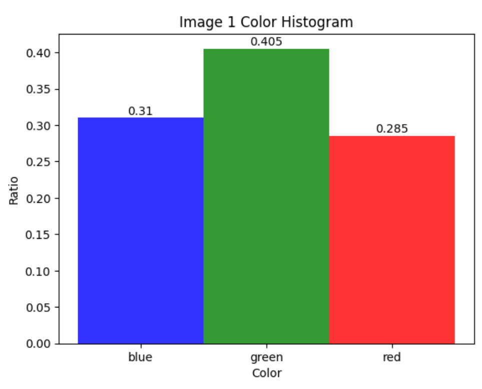
$$H(c) = \frac{E(c)}{\sum_{i=0}^{2} E(i)} \tag{3}$$

The color histogram is plotted using the `bar` function from Matplotlib. The key code is shown as follows:

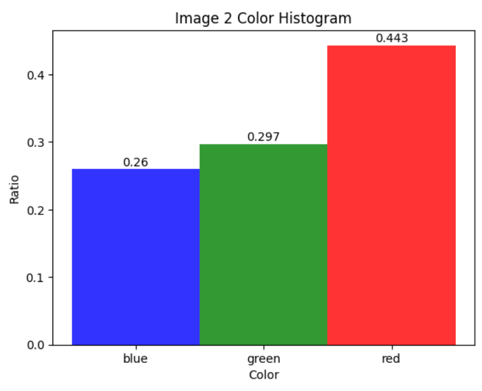
```
img = cv2.imread(f'images/img{index}.jpg', cv2.IMREAD_COLOR)
    if img is None:
        print(f"Error:␣Unable␣to␣read␣images/img{index}.jpg")
        return

    # Initialize RGB sum values
    blue, green, red = 0, 0, 0

    # Calculate the sum of the blue, green, and red channel values
    blue = np.sum(img[:, :, 0])
    green = np.sum(img[:, :, 1])
    red = np.sum(img[:, :, 2])

    # Calculate the total sum and normalize each color channel ratio
    total_sum = blue + green + red
    if total_sum == 0:
        print(f"Warning:␣Zero␣intensity␣found␣in␣images/img{index}.jpg")
        return

    color_ratios = [round(blue / total_sum, 3), round(green / total_sum, 3)

    # Plotting the color histogram
    colors = ['blue', 'green', 'red']
    plt.bar(colors, color_ratios, color=colors, alpha=0.8, width=1.0)
# Set bar color and transparency
    plt.xlabel('Color')
    plt.ylabel('Ratio')
    plt.title(f'Image␣{index}␣Color␣Histogram')

    # Add labels for each bar
    for color, ratio in zip(colors, color_ratios):
        plt.text(color, ratio + 0.001, f'{ratio}', ha='center', va='bottom')
```

This code reads an image using OpenCV and analyzes its color distribution. It calculates the total intensity of the blue, green, and red channels by summing their pixel values using NumPy.It then uses Matplotlib to create a bar chart showing the color distribution, with each bar representing the ratio of blue, green, or red in the image.

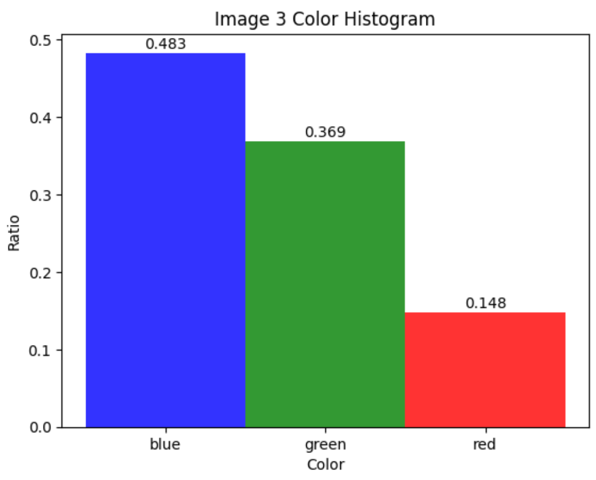
And the results of the code is shown in Figure1.



(a) Image 1



(b) Image 2



(c) Image 3

Figure 1: Color Histograms

3.2 Discussion And Conclusion

The `bar` function is used instead of the `hist` function because it allows better color differentiation.

As it's shown in Figure1, the resulting image is a bar chart that represents the color distribution of the analyzed image. Each bar corresponds to one of the primary color channels—blue, green, and red—and their heights indicate the normalized ratio of each color's intensity relative to the total intensity of the image. This visualization helps identify the dominant color in the image and provides insights into its overall color composition. For example, a taller red bar suggests a higher proportion of red in the image compared to blue and green.

4 Drawing Grayscale Histograms

4.1 Problem Statement and Code Description

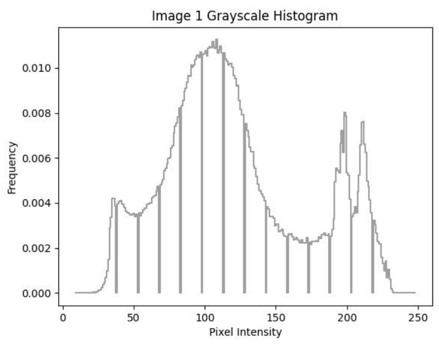
The grayscale histogram represents the relative proportion of pixel intensities in a grayscale image. Using NumPy's `ravel()` function, we can flatten the two-dimensional image matrix into a one-dimensional array.Uses NumPy's np.histogram to calculate the histogram of pixel intensities.

The key code are shown as follows:

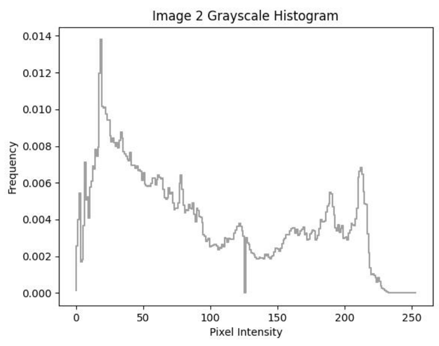
```
img = cv2.imread(f'images/img{index}.jpg', cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Error: Unable to read images/img{index}.jpg")
        return

    # Flatten the grayscale image to a 1D array
    ravel_img = img.ravel()

    # Calculate the histogram data
    hist, bins = np.histogram(ravel_img, bins=256, density=True)

    # Use plt.step() to plot the histogram as a step function
    plt.step(bins[:-1], hist, color='gray', where='mid', alpha=0.75)
    plt.title(f'Image {index} Grayscale Histogram')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
```

This code reads a grayscale image using OpenCV and calculates its histogram. First, it flattens the 2D image array into a 1D array using NumPy's `ravel()` function. Then, it computes the histogram using `np.histogram`, dividing pixel intensities into 256 bins and normalizing the frequencies. The histogram is plotted as a step function with `plt.step`, where the x-axis represents pixel intensity values and the y-axis shows their frequency.

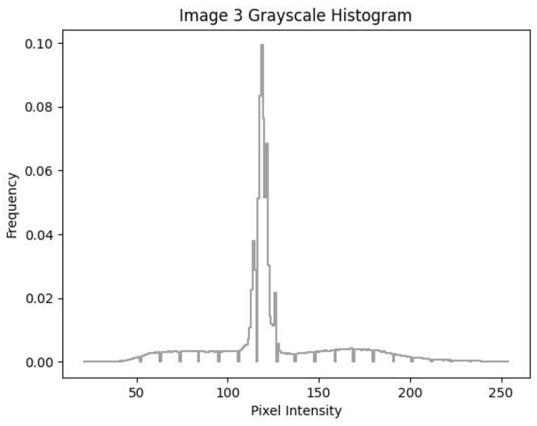
The result of running the code is shown in Figure2.



(a) Image 1



(b) Image 2



(c) Image 3

Figure 2: Grayscale Histograms

4.2 Discussion And Conclusion

The result of this code is a grayscale histogram that visually represents the distribution of pixel intensities in the image. The x-axis corresponds to pixel intensity values ranging from 0 (black) to 255 (white), while the y-axis represents the normalized frequency of each intensity level in the image. Peaks in the histogram indicate intensity values that appear more frequently, providing insights into the image's brightness and contrast. For example, a peak near 0 suggests a darker image, while a peak near 255 indicates a lighter image.

5 Drawing Grayscale Gradient Histograms

5.1 Problem Statement and Code Description

The gradient histogram represents the distribution of gradient intensities in a grayscale image, reflecting the texture complexity. The gradient at point (x, y) is defined as:

$$\nabla I(x,y) = \left(\frac{\partial I(x,y)}{\partial x}, \frac{\partial I(x,y)}{\partial y} \right) \tag{4}$$

The magnitude of the gradient is given by:

$$M(x, y) = \sqrt{I_x^2 + I_y^2} \tag{5}$$

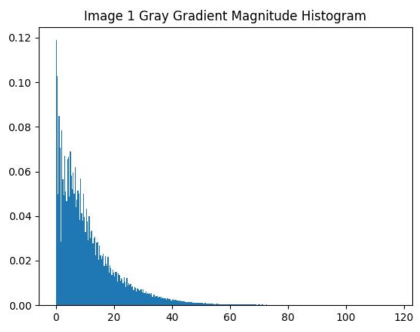
The gradient is calculated using NumPy's `gradient()` function. The histogram is generated similarly to the grayscale histogram.

The key codes are as follows:

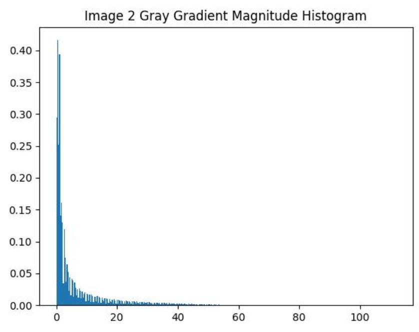
```
img = cv2.imread('images/img{}.jpg'.format(image_index), cv2.IMREAD_GRAYSCAL

    # Calculate the gradients in X and Y directions using numpy gradient fu
    img_gradient_x, img_gradient_y = np.gradient(img)

    # Initialize an empty array to store the gradient magnitude values
    img_gradient_magnitude = np.zeros(img.shape)

    # Calculate the gradient magnitude for each pixel using the formula sqr
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            img_gradient_magnitude[i][j] = math.sqrt(pow(img_gradient_x[i][
    # Display the gradient magnitude histogram
    plt.hist(img_gradient_magnitude.ravel(), bins=360, density=True)
    plt.title('Image␣{}␣Gray␣Gradient␣Magnitude␣Histogram'.format(image_inde
```

This code computes the gradient magnitude of a grayscale image. It calculates gradients in the x and y directions using NumPy's `gradient` function, then computes the gradient magnitude for each pixel using the formula $\sqrt{I_x^2 + I_y^2}$. Finally, it displays a histogram of the gradient magnitudes, showing their frequency distribution.

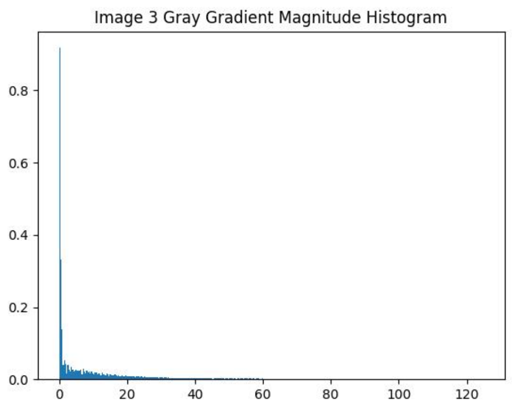
The result of running the code is shown in Figure3.



(a) Image 1



(b) Image 2



(c) Image 3

Figure 3: Grayscale Gradient Histograms

5.2 Discussion And Conclusion

The output is a histogram representing the distribution of gradient magnitudes in the grayscale image. The x-axis corresponds to the gradient magnitude values, which measure the intensity of changes in pixel values (edges or transitions) across the image. The y-axis shows the normalized frequency of these values. Peaks in the histogram indicate dominant gradient strengths, helping identify regions with strong edges or texture variations. For example,in the first image, the color variation is stronger, resulting in a histogram with higher frequencies for larger gradient magnitudes, indicating prominent edges or transitions. In the third image, the color variation is weaker, with most gradient magnitudes concentrated near lower values, reflecting smoother transitions and fewer sharp edges.

6 Extended Considerations

6.1 What's the function of `cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)` in the example code

OpenCV reads images in BGR format, while most image processing libraries use RGB format. Therefore, the `cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)` function is used to convert the image format to prevent the red and blue channels from being swapped.

6.2 How to correctly display the colors of a grayscale image in pyplot?

By default, Matplotlib uses the green channel to display grayscale images, resulting in a greenish appearance. To ensure proper display, we use the `cmap='gray'` parameter in the `imshow` function.

6.3 Further Improvements to Color Histograms

In this experiment, we calculated the proportion of each color channel. To obtain more detailed information, we can use the `split()` function to separate the three channels and generate individual histograms for each channel.

6.4 Conversion of Color Images to Grayscale

The conversion formula used in Chapter 1 is based on the human perception of brightness, which is currently the most widely used standardized formula.

6.5 LaTeX Challenges

In this task, I face many difficulties while using LaTeX, but in the end, I've solved it by turning for my classmates for help.Difficulties include:

1. **Formatting**: Achieving precise alignment (e.g., images in a triangular layout) required careful use of `subfigure`, `hspace`, and `vspace`.
2. **Background Color**: Ensuring code listings' background color fully covered the content involved tweaking `xleftmargin`, `xrightmargin`, and `lineskip`.
3. **Histograms**: Representing plots and figures with consistent captions and alignment in LaTeX was non-trivial, especially for complex layouts.
4. **Compilation Issues**: Some configurations required specific compilers (e.g., `xelatex`) for compatibility with packages like `minted`.

6.6 Summary

The lab focuses on image processing and visualization, complemented by LaTeX for documentation. Tasks include color ratio analysis, grayscale histograms, and gradient magnitude visualization, all aimed at extracting meaningful insights from images.