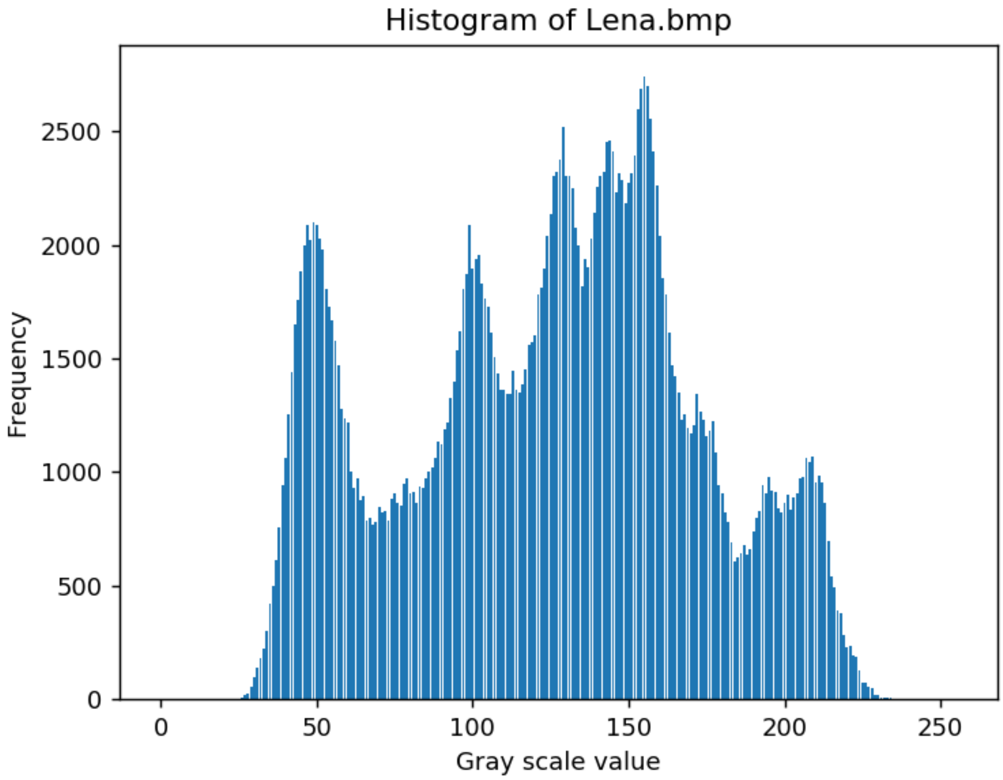
Computer Vision Homework #2

資工四 b05902115 陳建丞

- ## Result

  1. **Binarize at 128**

  2. **Histogram**



3. **Connected component labeling**

- # Implementation

  - ## Binarize at 128

    In this homework, I use `skimage` toolkit to deal with the io of the images. To generate a binary image, I iterate over the orignal image to check if the pixel gray scale value is greater of less than 128. If the pixel value \geq 128, set the pixel value to 255, otherwise, set the pixel value to 0.

    0. Preprocess

    ```
    from skimage import io

    lena = io.read('lena.bmp')
    ```

    1. Iterate over the image

```python
lena_binarized = lena.copy()

for i in range(len(lena_binarized)):
    for j in range(len(lena_binarized)):
        if lena_binarized[i][j] >= 128:
            lena_binarized[i][j] = 255
        else:
            lena_binarized[i][j] = 0
```

- ### Histogram

  In this part, I just simply iterate over the image and accumulate the frequency of each pixel values. And to show the result, I adopt `matplotlib` to generate the histogram.

```python
import matplotlib.pyplot as plt
import numpy as np

pixels = np.zeros((256), dtype = int)

# Accumulate the frequency of each pixel values
for row in lena:
    for i in row:
      pixels[i] += 1

# Generate the corresponding histogram
plt.bar(range(len(pixels)), pixels)
plt.title('Histogram of Lena.bmp')
plt.xlabel('Gray scale value')
plt.ylabel('Frequency')
plt.save('histogram.png')
```

- ### Connected component labeling

  In this problem, I adopt **the classical algorithm** with **4-connected neighborhood detection** taught in the lecture. My algorithm progresses as original classical algorithm, except it do not hold the information of equivalent labels. Instead, when encountering a conflict (the lefter label and upper label has different value so these two values shall be equivalent), it fixes the conflict immediately. In detail, it looks up `labels` to re-label all pixels in one label to another. This modification improves the complexity in development that the original image is iterated once.

  And to plot the bounding rectangles, I apply the `cv2` toolkit which can be easily used to draw a rectangle with function `cv2.rectangle()` and the centroid + with function `cv2.line()`

  Find connected component

```
pixels = lena_binarized.copy()
labels = []
pixels_label = [[-1] * len(lena_binarized) for i in
range(len(lena_binarized))]

for i in range(len(lena_binarized)):
    for j in range(len(lena_biarized)):
        if pixels[i][j] == 0:
            continue
        left_label = -1
        if j > 0 and pixels_label[i][j-1] != -1:
            left_label = pixels_label[i][j-1]
        if i > 0 and pixels_label[i-1][j] != -1:
            top_label = pixels_label[i-1][j]

            if left_label != -1 and left_label != top_label:
                for x, y in labels[left_labels]:
                    pixels_label[x][y] = top_label
                labels[top_label] += labels[left_label]
                labels[left_label] = None

            left_label = top_label

        if left_label == -1:
            left_label = len(labels)
            labels.append([(i, j)])
        else:
            labels[left_label].append((i, j))
```

Bounding box and centroid

```
import cv2

lena_connected = lena_binarized.copy()
for component in labels:

    (left, top), (right, bottom) = component[0], component[0]
    cx, y = 0, 0
    for y, x in component:
        if x < left:
            left = x
        if x > right:
            right = x
        if y < top:
            top = y
        if y > bottom:

        cx += x
```

```
        cy += y

    cx = round(cx/len(component))
    cy = round(cy/len(component))

    cv2.rectangle(lena_connected, (left, top), (right, bottom), (255,
0 ,0), 1)
    cv2.line(lena_connected, (cx-10, cy), (cx+10, cy), (255, 0, 0), 5)
    cv2.line(lena_connected, (cx, cy-10), (cx, cy+10), (255, 0, 0), 5)
```