HW2 Report

R10922067 林云雲

Introduction

This program consists of three parts:

Part 1. Binarized the input source image.

Part 2. Generate a histogram of the binary image.

Part 3. Draw the connected components on the source image

Usage

Place the source image and main.py under the same directory. Run the following command in the terminal.

```
python3 main.py -s <source> -b <binary_threshold> -c <component_threshold>
```

Parameters

`-s <source>` : the file path of source image, default = `lena.bmp`
`-b <binary_threshold>` : binary threshold, default = `128`
`-c <component_threshold>` : minimum count of pixels for a connected component, default = `500`

Part 1. Binarize image

1.1 Method

First, source image is converted to grayscale mode (only one channel).

For each pixel, if its intensity is greater or equal to the binary threshold, its value in the binary image is 1, and otherwise 0.

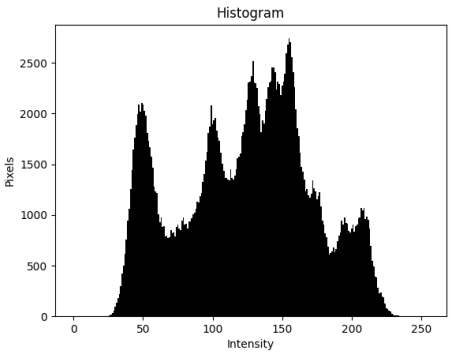
1.2 Result

The result image is saved as `binary.bmp` as shown on the right.

Part 2. Histogram

1.1 Method

Scan the grayscale image mentioned in Part 1 pixelwise. For each pixel, increment the pixel count of the its corresponding intensity in the histogram.



1.2 Result

The result image is saved as `histogram.png` as shown on the right.

Part 3. Connected Components

1.1 Source Code

`class LabelClass`

This class is a representation of a label. In the end of the computation, the pixels with the same label form a connected component. It contains information for a label including the equivalent label value, the number of pixels of the component, the weight and boundary of the component's

bounding box. (The weight is used to calculate the centroid of the component.)

`class ConnectedComponent`

This class is to find connected components using *4-adjacency* in an image. A space-efficient two-pass algorithm is implemented by using a *local equivalence table* to label the image.

1.2 Method

The binary image generated in Part 1 serves as the input for the algorithm, and only the pixels with a value of 1 will be labeled. The algorithm goes through a top-down process and then a bottom-up process.

The top-down process starts from the top row of the image array, and go down row by row. For each row, it scans through every pixel from left to right for two times. In the first scan, we assign a label to each pixel by choosing the smallest value among its 4-adjacency neighbors' labels. (If none of its neighbors has a label, then it will be assigned a new label.) Since the neighbors will be joint by the current pixel, they should be considered to belong to the same component with the same label. Hence different labels of the neighbors will be considered equivalent and stored into the local equivalence table. When the first scan is done, we do a second scan to relabel the pixels according to the equivalence table.

After the top-down process is done, the bottom-up process is then proceeded. It is almost the same as the top-down process, but done in the opposite direction by processing from bottom row to the top and scanning pixels from the right to the left. The result of the labeled image is shown on the right, with different colors representing different labels. In addition, during the second scan, since each pixel's label will be confirmed for the last time and will never be changed anymore, we can decide the label's pixel count, weight, and boundary box at the same time. This will avoid iterating over the whole image again to collect the connected components information.

1.3 Result

The result is saved as `connected_components.png` as shown on the right, with connected components that contain at least 500 pixels. The blue rectangles are the bounding boxes, and the red spot in each box shows the centroid.