Computer Vision HW2 Report R06922030 傅敏桓

Environment: Python 3.6 with cv2, numpy

* The cv2 module is used for image I/O for all tasks, and is used to draw rectangles for task 3.

Benchmark: lena.bmp

Usage: $> python3 hw2.py [image_path]

Task 1: Binarization, threshold at 128

Method description: Each pixel is iterated and assigned 0 (black) if its intensity is lower than the threshold, or assigned 255 (white) if higher than the threshold.

Principal code fragment:

```
def binarize(img):
    new_img = np.zeros(img.shape, np.int)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            new_img[i, j] = 255 if img[i, j] >= 128 else 0
    return new_img
```

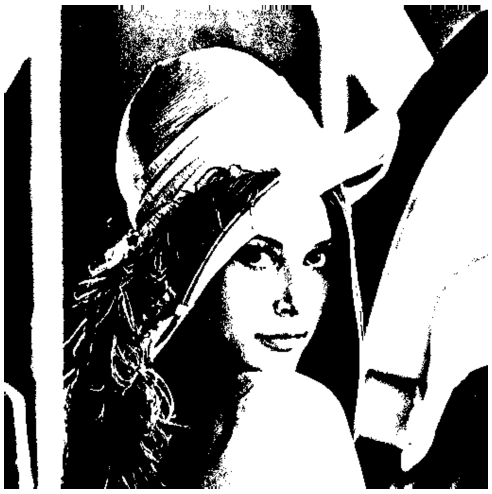
Task 2: Histogram

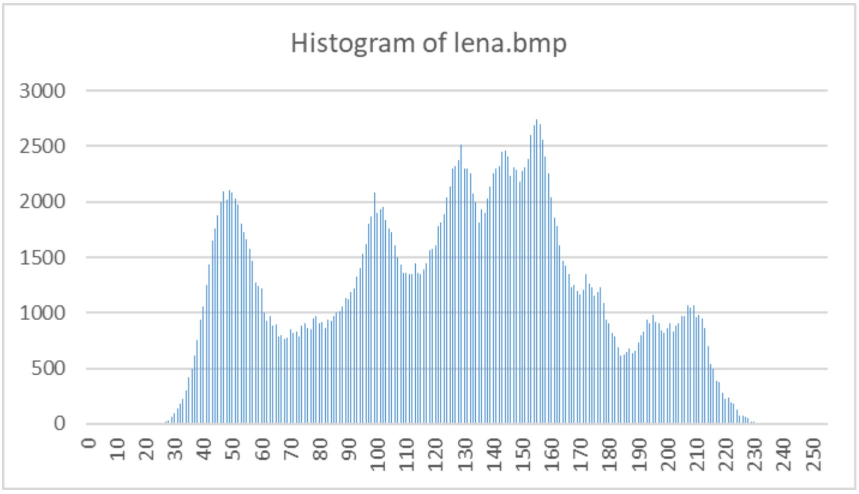
Method description: Each pixel is iterated and counted by its intensity. The program outputs an array sized 256, in which each entry represents the number of pixels with the intensity value. The output is further processed using Microsoft Excel into visualized histogram.

Principal code fragment:

```
def count_bins(img):
    count = np.zeros(MAX_INTENSITY+1, np.int)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            count[img[i, j]] += 1
    return count
```

Results for task 1 & 2:


(a) Binarization



(b) Histogram

Task 3: Connected components

Method Description:

The program detects connected components using a two-pass algorithm with Union-find data structure. For this task, 4-connected neighborhood is adopted. The image is binarized in advance.

The first pass iterates each pixel in the image. For each non-zero pixel in the image, we check its upper and left neighboring pixels and assign a new label to it if there is no non-zero neighboring pixel; otherwise, we assign the minimal label of its neighbor to it. Besides, if both neighboring pixels are not zero and have different labels, we associate the two labels to the same equivalent class (disjoint set). The second pass reassigns the label of each pixel to the minimal label of the equivalent class it belongs to.

We choose the connected components with at least 500 pixels and visualize the regions by bounding boxes (the cross at the centroid are omitted).

Principal code fragment:

```python
# First pass
for i in range(bin_img.shape[0]):
    for j in range(bin_img.shape[1]):
        if bin_img[i, j] != 0:
            neighbors = []
            if i > 0 and bin_img[i, j] == bin_img[i-1, j]:
                neighbors.append(label_map[i-1, j])
            if j > 0 and bin_img[i, j] == bin_img[i, j-1]:
                neighbors.append(label_map[i, j-1])

            if len(neighbors) == 0:
                cur_label += 1
                label_map[i, j] = cur_label
            else:
                label_map[i, j] = min(neighbors)
                if(len(neighbors) == 2):
                    union_sets.union(neighbors[0], neighbors[1])
area_count = np.zeros(cur_label+1, np.int)
```

```python
# Second pass
for i in range(bin_img.shape[0]):
    for j in range(bin_img.shape[1]):
        if bin_img[i, j] != 0:
            label_map[i, j] = union_sets.find(label_map[i, j])
            area_count[label_map[i, j]] += 1
```

Result of task 3: