Computer Vision I

Homework 2 - Basic Image Manipulation B04902090 資工四 施長元

Usage of the full code:

```
python main.py [Image_path] [Task]
Tasks: binary, histogram, connected_components
```

After the code exit, the output file will be in the same directory with the main.py

Environment: Python3.6 on Windows Linux Subsystem (Ubuntu 16.04)

Contents:

Write a program to generate:

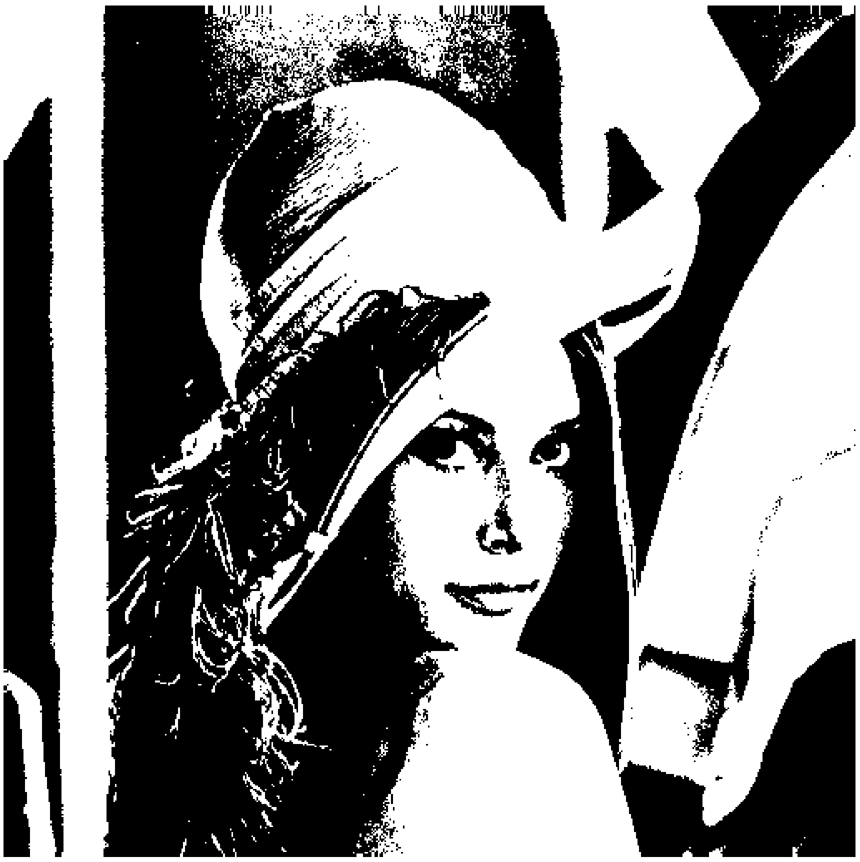
1. a binary image (threshold at 128)

Sample usage: `python main.py lena.bmp binary`

```python
def binary():
    c_label = 0
    for i in range(img.shape[0]):
        # ROW
        for j in range(img.shape[1]):
            b_img[i, j] = (1 if img[i, j] > 128 else 0)
```

Algorithm: 兩層迴圈，倘若像素值>128: 將其改成 1，否則改成 0
把結果乘上 255 後輸出

Result:



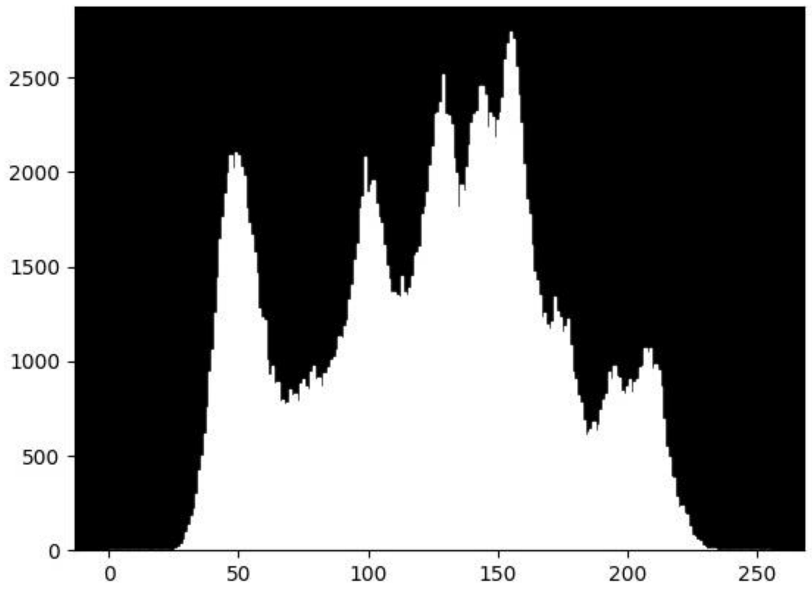
2. a histogram

Sample usage: `python main.py lena.bmp histogram`

```
elif sys.argv[2] == "histogram":
    result, x = np.zeros(256, dtype=np.int32), list(range(256))
    for i in range(img.shape[0]):
        for j in range(img.shape[1]): result[img[i, j]] += 1
    plt.rcParams['axes.facecolor'] = 'black'
    plt.bar(x, result, color='white', edgecolor='white')
    plt.savefig("hw2_histogram.jpg")
```

Algorithm: 兩層迴圈，把每個像素的值累計到一個大小 256 的陣列中，並使用 `matplotlib.pyplot` 輸出結果

Result:



3. connected components (+ at centroid, bounding box)

Sample usage: `python main.py lena.bmp connected_components`

Algorithm: 使用投影片中的 An Efficient Run-Length Implementation of the Local Table Method，我使用 **4-connected** neighborhood detection

● 用 1. 的結果做 binarize，同時將每行各自連在一起的部分放入 table 中；
  Table(c_all)的資料結構:
  `[row][row 中的 parts][label, start, end]`

```
# Connect components
k, c_row = 0, []
while k < img.shape[1]:
    if b_img[i, k] == 1:
        c_dot = [c_label, k]
        while k < img.shape[1] and b_img[i, k] == 1: k += 1
        c_dot.append(k-1)
        # Update the dot in row
        c_row.append(c_dot)
        c_label += 1
    else: k += 1
# Update the row in all
c_all.append(c_row)
```

- 接著從左上掃到右下，再從右下掃到左上，跑三層迴圈；倘若上下重疊(call 下面提到的 function)，就把上下兩個 label 較大的換成比較小的那一個，並把判斷有變動的 indicator 設定成 True；不斷重複這動作直到再也沒有任何變動為止。

```
update = True
while update:
    update = False
    # Top-left to Bottom-right
    for i in range(1, len(c_all)):
        for j in range(len(c_all[i])):
            for k in range(len(c_all[i-1])):
                if overlap(c_all[i][j], c_all[i-1][k]):
                    c_all[i][j][0] = c_all[i-1][k][0] = min(c_all[i][j][0],
                        c_all[i-1][k][0])
                    update = True

    # Bottom-right to Top-left
    for i in reversed(range(1, len(c_all))):
        for j in reversed(range(len(c_all[i]))):
            for k in reversed(range(len(c_all[i-1]))):
                if overlap(c_all[i][j], c_all[i-1][k]):
                    c_all[i][j][0] = c_all[i-1][k][0] = min(c_all[i][j][0],
                        c_all[i-1][k][0])
                    update = True
```

判斷上下區塊是否重疊的 Code:

```
def overlap(a, b):
    if a[0] == b[0]: return False
    elif a[1] < b[1] and a[2] < b[1]: return False
    elif a[1] > b[2] and a[2] > b[2]: return False
    else : return True
```

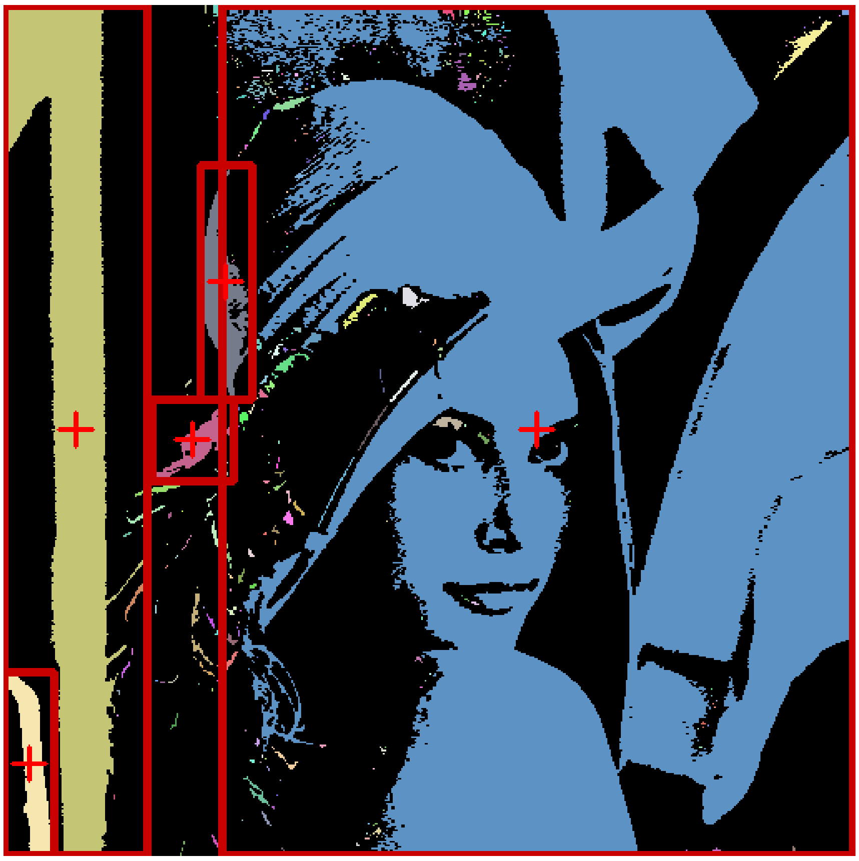
除了上下的 label 一樣，或是上面的區域完全在下面的左邊或右邊以外，其餘則是有重疊且目前分在不同 label，就 Return True。

- 然後標 bounding boxes，畫圖~

個人把不同 group 的 pixel 上了隨機的顏色，方便辨別；但還是附上正常版的黑白版本。

加上上色的 code 之後，還是把包含註解的總行數壓在 120 行內，大心♥

Result:



黑白的版本: