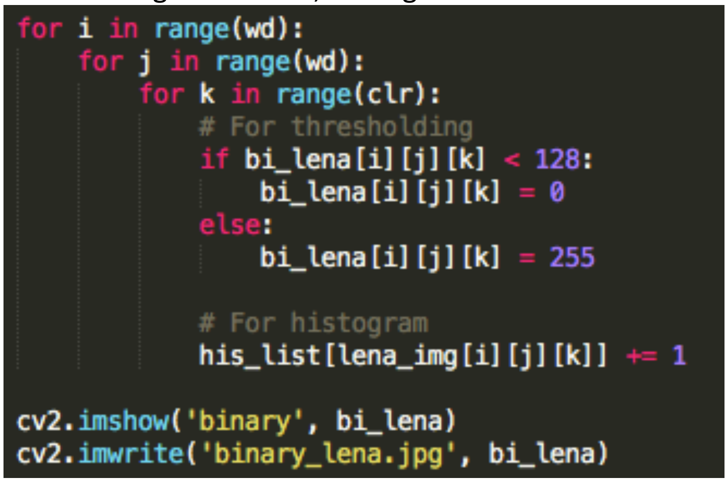
R07922141
張緣彩
22/09/2018

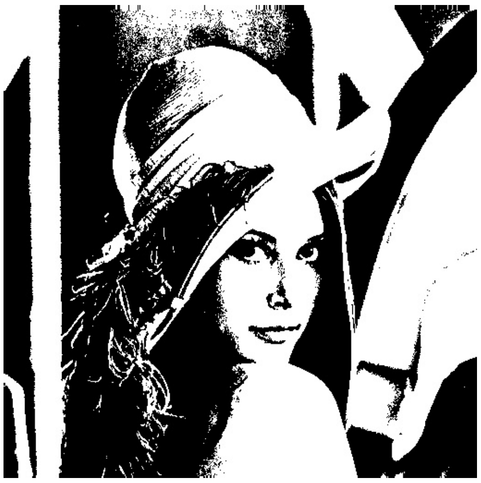
Computer Vision Hw2 Report

Part I. Binarization

Using a for loop to loop through width, height and color channel of the image, then thresholding each value, the algorithm is as below.

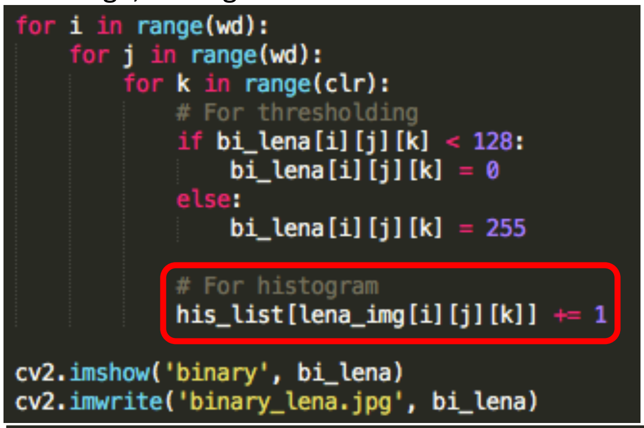


```
for i in range(wd):
    for j in range(wd):
        for k in range(clr):
            # For thresholding
            if bi_lena[i][j][k] < 128:
                bi_lena[i][j][k] = 0
            else:
                bi_lena[i][j][k] = 255

            # For histogram
            his_list[lena_img[i][j][k]] += 1

cv2.imshow('binary', bi_lena)
cv2.imwrite('binary_lena.jpg', bi_lena)
```

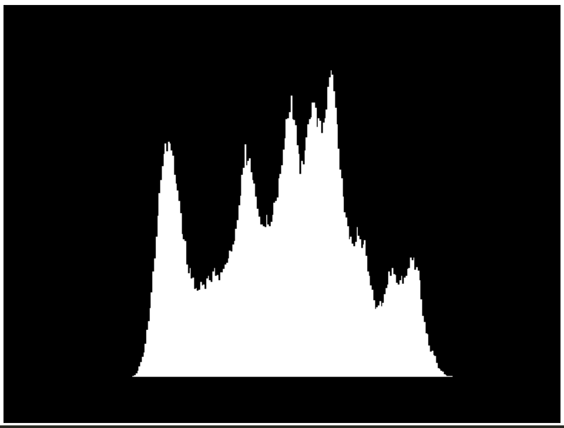


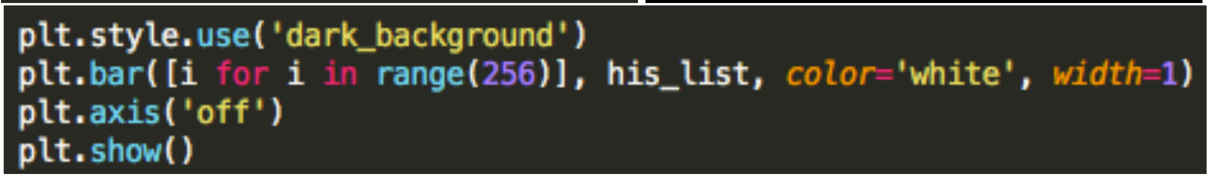
Part II. Histogram

Calculate each value (0~255) frequency in the image, then use bar chart to plot histogram of the image, the algorithm is as below.



```
for i in range(wd):
    for j in range(wd):
        for k in range(clr):
            # For thresholding
            if bi_lena[i][j][k] < 128:
                bi_lena[i][j][k] = 0
            else:
                bi_lena[i][j][k] = 255

            # For histogram
            his_list[lena_img[i][j][k]] += 1

cv2.imshow('binary', bi_lena)
cv2.imwrite('binary_lena.jpg', bi_lena)
```

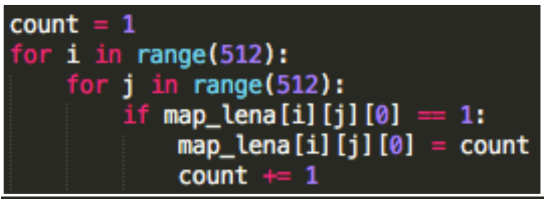




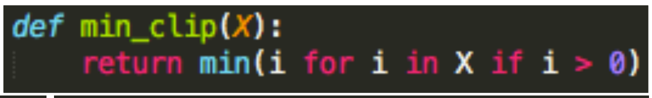
```
plt.style.use('dark_background')
plt.bar([i for i in range(256)], his_list, color='white', width=1)
plt.axis('off')
plt.show()
```

Part III. Connected Component and Bounding Box

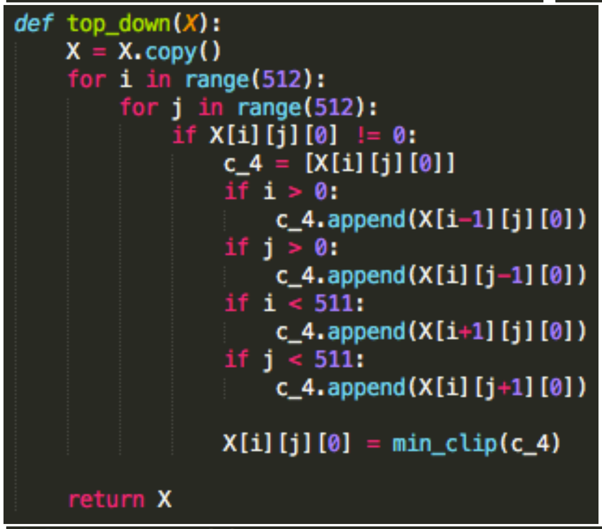
I'm using 4-connected with iterative algorithm. In the iterative algorithm, first we need a to give each non-zero pixel an unique index. Then, we need top-down and bottom-up function to keep cycling the whole image several time. In each cycle, the function loop through every pixel and giving that pixel a smallest value among its neighbor (4 neighbors). Finally, we keep iterate it until the sum of the value of the whole image stopped decrease. By the way, I have removed the small connected component as well. As result, we need to iterate roughly 20 times with 37 seconds. The algorithm is as below.



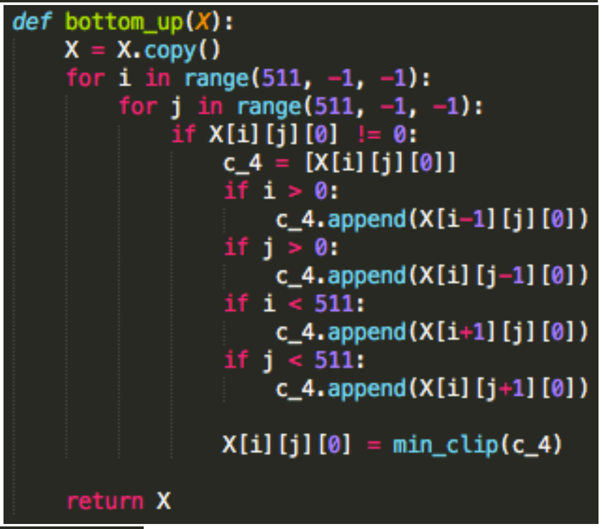
```
count = 1
for i in range(512):
    for j in range(512):
        if map_lena[i][j][0] == 1:
            map_lena[i][j][0] = count
            count += 1
```



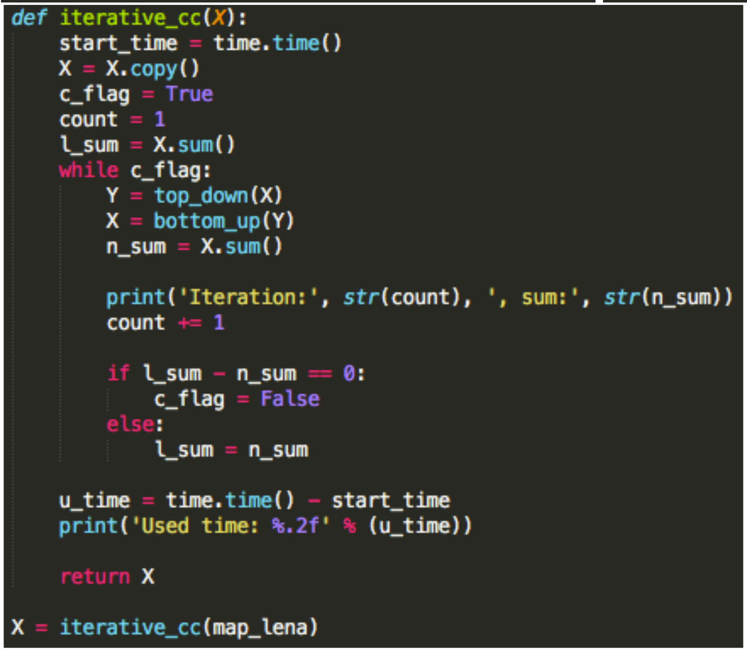
```
def min_clip(X):
    return min(i for i in X if i > 0)
```



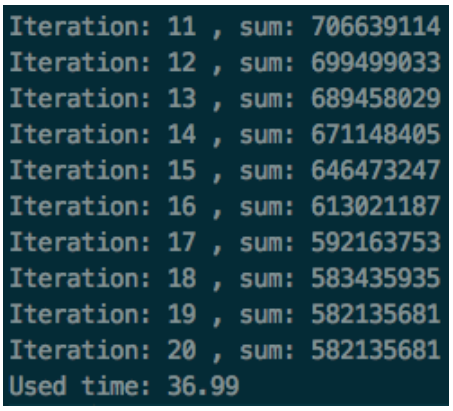
```
def top_down(X):
    X = X.copy()
    for i in range(512):
        for j in range(512):
            if X[i][j][0] != 0:
                c_4 = [X[i][j][0]]
                if i > 0:
                    c_4.append(X[i-1][j][0])
                if j > 0:
                    c_4.append(X[i][j-1][0])
                if i < 511:
                    c_4.append(X[i+1][j][0])
                if j < 511:
                    c_4.append(X[i][j+1][0])

                X[i][j][0] = min_clip(c_4)

    return X
```



```
def bottom_up(X):
    X = X.copy()
    for i in range(511, -1, -1):
        for j in range(511, -1, -1):
            if X[i][j][0] != 0:
                c_4 = [X[i][j][0]]
                if i > 0:
                    c_4.append(X[i-1][j][0])
                if j > 0:
                    c_4.append(X[i][j-1][0])
                if i < 511:
                    c_4.append(X[i+1][j][0])
                if j < 511:
                    c_4.append(X[i][j+1][0])

                X[i][j][0] = min_clip(c_4)

    return X
```

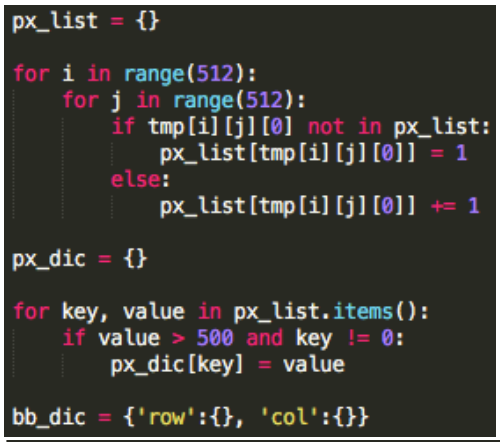


```
def iterative_cc(X):
    start_time = time.time()
    X = X.copy()
    c_flag = True
    count = 1
    l_sum = X.sum()
    while c_flag:
        Y = top_down(X)
        X = bottom_up(Y)
        n_sum = X.sum()

        print('Iteration:', str(count), ', sum:', str(n_sum))
        count += 1

        if l_sum - n_sum == 0:
            c_flag = False
        else:
            l_sum = n_sum

    u_time = time.time() - start_time
    print('Used time: %.2f' % (u_time))

    return X

X = iterative_cc(map_lena)
```

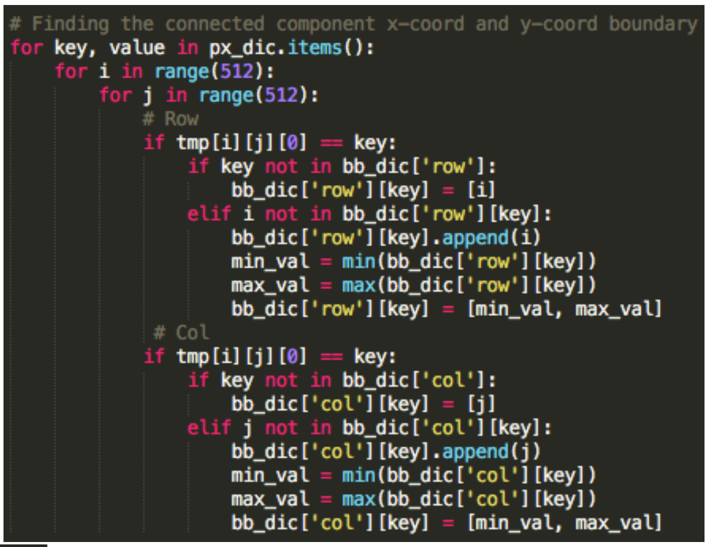


```
Iteration: 11 , sum: 706639114
Iteration: 12 , sum: 699499033
Iteration: 13 , sum: 689458029
Iteration: 14 , sum: 671148405
Iteration: 15 , sum: 646473247
Iteration: 16 , sum: 613021187
Iteration: 17 , sum: 592163753
Iteration: 18 , sum: 583435935
Iteration: 19 , sum: 582135681
Iteration: 20 , sum: 582135681
Used time: 36.99
```

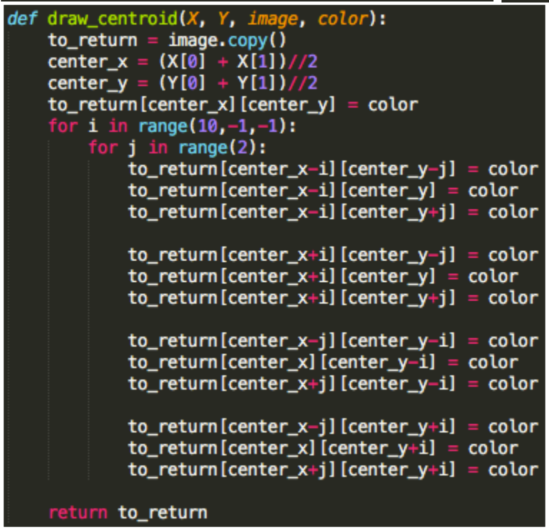
For the bounding box and centroid, we need to find the minimum and maximum position of X-coordinate and Y-coordinate. Then, use the X-coordinate and Y-coordinate to write the bounding box and centroid. For centroid, we draw a horizontal and a vertical rectangle. For the bounding box, we draw two horizontal lines and two vertical lines. The algorithm is as below.



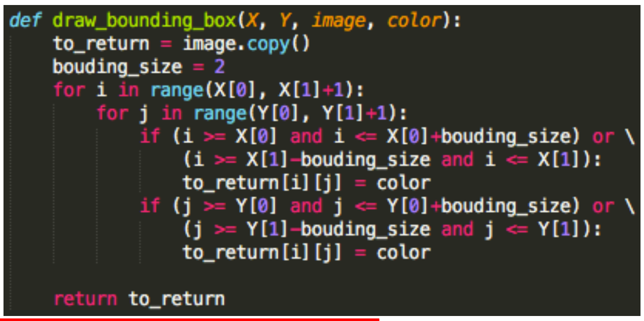
```python
px_list = {}

for i in range(512):
    for j in range(512):
        if tmp[i][j][0] not in px_list:
            px_list[tmp[i][j][0]] = 1
        else:
            px_list[tmp[i][j][0]] += 1

px_dic = {}

for key, value in px_list.items():
    if value > 500 and key != 0:
        px_dic[key] = value

bb_dic = {'row':{}, 'col':{}}
```



```python
# Finding the connected component x-coord and y-coord boundary
for key, value in px_dic.items():
    for i in range(512):
        for j in range(512):
            # Row
            if tmp[i][j][0] == key:
                if key not in bb_dic['row']:
                    bb_dic['row'][key] = [i]
                elif i not in bb_dic['row'][key]:
                    bb_dic['row'][key].append(i)
                    min_val = min(bb_dic['row'][key])
                    max_val = max(bb_dic['row'][key])
                    bb_dic['row'][key] = [min_val, max_val]
             # Col
            if tmp[i][j][0] == key:
                if key not in bb_dic['col']:
                    bb_dic['col'][key] = [j]
                elif j not in bb_dic['col'][key]:
                    bb_dic['col'][key].append(j)
                    min_val = min(bb_dic['col'][key])
                    max_val = max(bb_dic['col'][key])
                    bb_dic['col'][key] = [min_val, max_val]
```



```python
def draw_centroid(X, Y, image, color):
    to_return = image.copy()
    center_x = (X[0] + X[1])//2
    center_y = (Y[0] + Y[1])//2
    to_return[center_x][center_y] = color
    for i in range(10,-1,-1):
        for j in range(2):
            to_return[center_x-i][center_y-j] = color
            to_return[center_x-i][center_y] = color
            to_return[center_x-i][center_y+j] = color

            to_return[center_x+i][center_y-j] = color
            to_return[center_x+i][center_y] = color
            to_return[center_x+i][center_y+j] = color

            to_return[center_x-j][center_y-i] = color
            to_return[center_x][center_y-i] = color
            to_return[center_x+j][center_y-i] = color

            to_return[center_x-j][center_y+i] = color
            to_return[center_x][center_y+i] = color
            to_return[center_x+j][center_y+i] = color

    return to_return
```



```python
def draw_bounding_box(X, Y, image, color):
    to_return = image.copy()
    bouding_size = 2
    for i in range(X[0], X[1]+1):
        for j in range(Y[0], Y[1]+1):
            if (i >= X[0] and i <= X[0]+bouding_size) or \
               (i >= X[1]-bouding_size and i <= X[1]):
                to_return[i][j] = color
            if (j >= Y[0] and j <= Y[0]+bouding_size) or \
               (j >= Y[1]-bouding_size and j <= Y[1]):
                to_return[i][j] = color

    return to_return
```