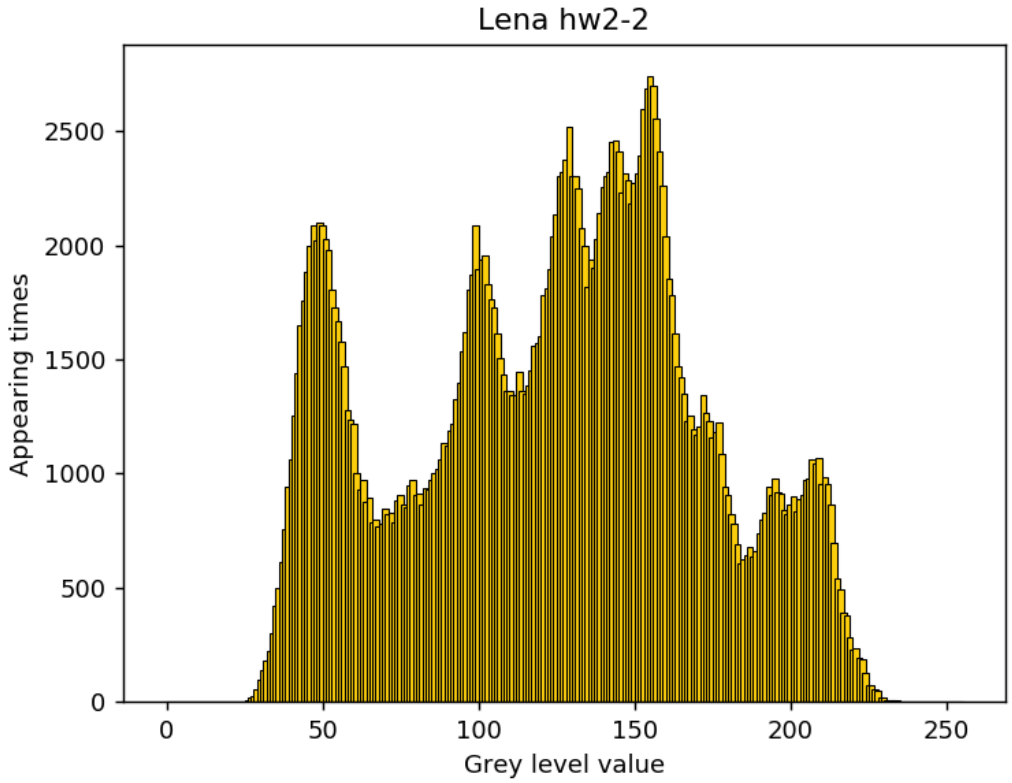
Computer Vision 1 HW#2

The first part: Binarizing the image

I used the python module `skimage` to read `lena.bmp` as a two-dimensional array, and then I changed its grey level value according to the threshold, gengerating the below picture `binarized.png`.

The second part: Calculating the histogram

Similarily, I used the python module `skimage` to access `lena.bmp` as a 2-D array. Then I traversed the whole array to calculate the numbers of each grey level value appearance, recording it in a list. After that, I ploted the following bar chart(`bar.png`) with `matplotlib`.

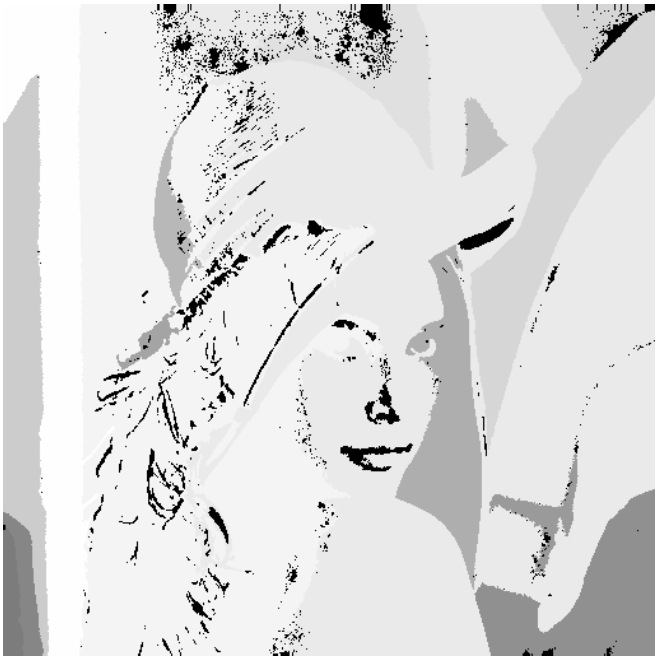


The third part: Finding the connected components

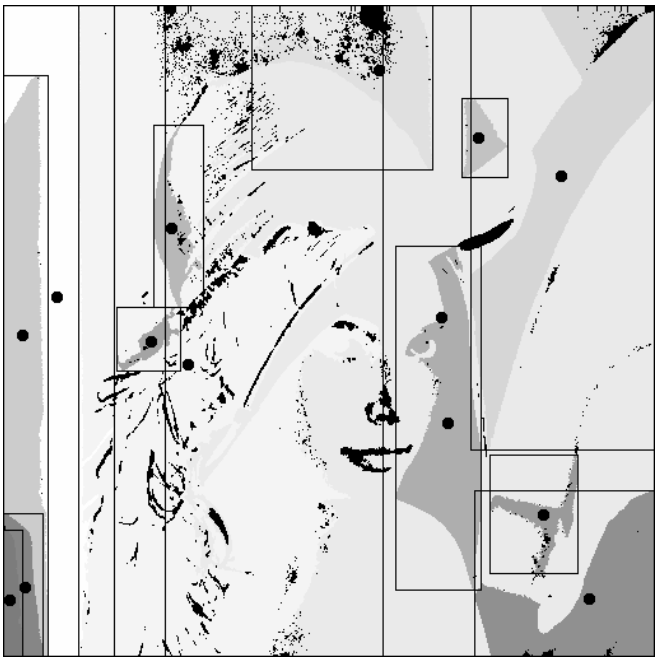
At this part, I was using 4-connected neighborhood detection.

However, I found the connected components by a BFS algorithm, which is adapted from the `recursive algorithm` but using breadth-firth-search instead of depth-first-search. In other words, I didn't really do the recursion; I traversed the whole binarized photo and determined the connected pixels with same color by BFS its all neighborhood, and writing down the area, centroids, and bounds into a list at the same time.

After that, I labelled each component with different grey level values and get the below image `connected_component.png`. (The black areas of the photo is the omitted components which are smaller than 500 px)



Then, I drawed the bounding boxes(the black rectangles) and centroids(the black dots) due to the previous generated list, generating the below image `bounding_box_with_centroids.png`.



my BFS algorithm:

```
def bfs(img, i, j):
    global color, decision_map
    queue, count = [(i, j)], 0
    upper, lower, left, right = 512, -1, 512, -1
    centroid_x, centroid_y = 0, 0
    delta_x, delta_y = [1, 0, -1, 0], [0, 1, 0, -1]
    decision_map[i][j] = color
    while(len(queue) != 0):
        # print(queue)
        # print(decision_map)
        temp = queue.pop()
        x, y = temp[0], temp[1]
        centroid_x += x
        centroid_y += y
        upper, lower, left, right = (
            x if(x < upper) else upper, x if(x > lower) else lower,
            y if(y < left) else left, y if(y > right) else right
        )

        for d in range(4):
            if((valid_range(x + delta_x[d], y + delta_y[d]))
            and (decision_map[x + delta_x[d]][y + delta_y[d]] == 0)
            and (img[x + delta_x[d]][y + delta_y[d]] == img[x][y])):
                queue = [(x + delta_x[d], y + delta_y[d])] + queue
                decision_map[x + delta_x[d]][y + delta_y[d]] = color
        count += 1
    if(count < threshold):
        decision_map[decision_map == color] = -1
        return False
    else:
        color += 10
    return upper, lower, left, right, int(centroid_x / count),
int(centroid_y / count)
```