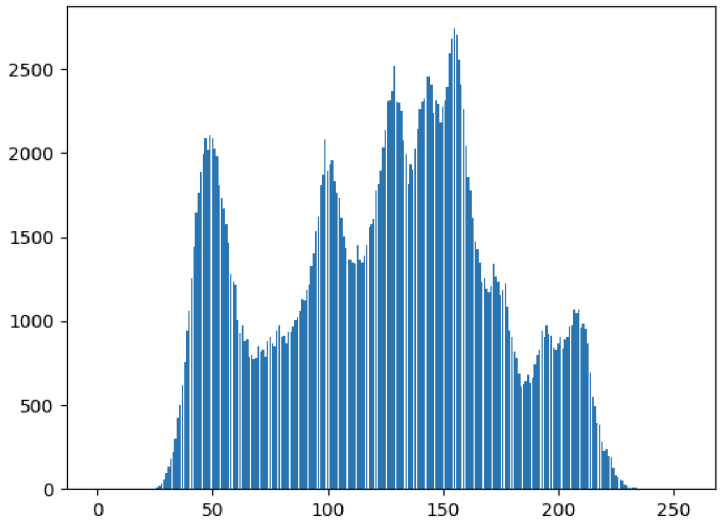
Computer Vision Homework #2

Student-ID: r10944020; Name: 林顥倫; Department: GINM

[Results]

(a) A Binary Image

(b) A histogram



(c) connected component (with center regions with + at centroid, bounding box)
Left: 4-connected neighboring (2+ overlap, a little bit hard to distinguish)
Right: 8-connected neighboring

[Code Fragment]
Part-1 and Part-2 Functions

```python
import numpy as np
from pprint import pprint
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw

def binarize(img):
    imageW, imageH = img.width, img.height
    new_img = img.copy()
    new_img_pixel = new_img.load()
    for x in range(imageW):
        for y in range(imageH):
            new_img_pixel[x, y] = 255 if img.getpixel((x,y)) >= 128 else 0
    new_img.save('./binarize_lena.bmp')

def img_2_histogram(img):
    hist = [0 for i in range(256)]
    imageW, imageH = img.width, img.height
    for x in range(imageW):
        for y in range(imageH):
            hist[img.getpixel((x,y))] += 1
    plt.bar(range(0, 256), hist)
    plt.savefig('./histogram_lena.png')
```

Part-3 Main Function

```python
def connected_components(img, NEIGHBOR):
    THRESHOLD, OBJECTNUM = 500, 1
    binarized_img = Image.open('./binarize_lena.bmp')
    imageW, imageH = binarized_img.width, binarized_img.height
    # traversal = for DFS recording the location traversal or not; img_objects = for recording objects
    traversal, img_objects = np.zeros((imageW, imageH)), np.zeros((imageW, imageH))
    # record how many pixels in each region
    object_count_map = np.zeros(imageW*imageH+1)
    # create new image and process image pixel values and convert binary to RGB format
    connected_components_img = Image.new('RGB', (imageW, imageH))
    connected_components_img_pixel = connected_components_img.load()
    for x in range(imageW):
        for y in range(imageH):
            if binarized_img.getpixel((x, y)) == 0: connected_components_img_pixel[x, y] = (0, 0, 0)
            else: connected_components_img_pixel[x, y] = (255, 255, 255)
    # mark objects
    for x in range(imageW):
        for y in range(imageH):
            # no need to deal with 0 value pixel
            if binarized_img.getpixel((x, y)) == 0: traversal[x, y] = 1
            elif binarized_img.getpixel((x, y)) == 255 and traversal[x, y] == 0:
                # start DFS traversal for marking region
                stk = [(x, y)] # stack for DFS traversal
                while len(stk):
                    (c, r) = stk.pop()
                    # print((c,r))
                    # if the node is visited, continue;
                    # if not, mark as visited & give label & update count & DFS traversal
                    if traversal[c, r] == 1: continue
                    traversal[c, r] = 1
                    img_objects[c, r] = OBJECTNUM
                    object_count_map[OBJECTNUM] += 1
                    # YOU CAN CHANGE NUMBER IN FIRST PARAMETER (4 or 8)
                    stk = neighbor_dfs_traversal(NEIGHBOR, c, r, imageW, imageH, traversal, binarized_img, stk)
                OBJECTNUM += 1 # print(OBJECTNUM)
    # get bounding boxes
    bounding_boxes = create_bounding_boxes(object_count_map, img_objects, imageW, imageH, THRESHOLD)
    # draw bounding boxes and box centers
    draw = ImageDraw.Draw(connected_components_img)
    for bounding_box in bounding_boxes:
        (left, right, up, down, object_id) = bounding_box[0], bounding_box[1], bounding_box[2], bounding_box[3], bounding_box[4]
        centerX, centerY = getCentroid(object_id, img_objects, object_count_map[object_id], imageW, imageH)
        draw.rectangle(((left, up), (right, down)), outline='blue')
        draw.line(((centerX-6, centerY), (centerX+6, centerY)), fill='red', width=3) # horizontal
        draw.line(((centerX, centerY-6), (centerX, centerY+6)), fill='red', width=3) # vertical
    connected_components_img.save('./connected_components_lena_' + str(NEIGHBOR) + '.bmp')
```

Part-3 Auxiliary Functions

```python
def create_bounding_boxes(object_count_map, img_objects, imageW, imageH, THRESHOLD):
    boxes = []
    for object_id, count in enumerate(object_count_map):
        if count >= THRESHOLD:
            left, right, up, down = imageW, 0, imageH, 0
            for x in range(imageW):
                for y in range(imageH):
                    if img_objects[x, y] == object_id:
                        left = x if x < left else left
                        right = x if x > right else right
                        up = y if y < up else up
                        down = y if y > down else down
            boxes.append((left, right, up, down, object_id))
    return boxes

def neighbor_dfs_traversal(neighbors, x, y, imageW, imageH, traversal, img, stk):
    if neighbors == 8:
        for px in range(x-1, x+2):
            for py in range(y-1, y+2):
                if px >= 0 and px < imageW and py >= 0 and py < imageH:
                    if traversal[px, py] == 0 and img.getpixel((px,py)) != 0:
                        stk.append((px, py))
    if neighbors == 4:
        for (px, py) in [(x-1, y-1), (x+1, y+1), (x-1, y+1), (x+1, y-1)]:
            if px >= 0 and px < imageW and py >= 0 and py < imageH:
                if traversal[px, py] == 0 and img.getpixel((px,py)) != 0:
                    stk.append((px, py))
    return stk

def getCentroid(object_id, img_objects, amount_of_pixel, imageW, imageH):
    centerX, centerY = 0, 0
    for x in range(imageW):
        for y in range(imageH):
            if img_objects[x, y] == object_id:
                centerX += x
                centerY += y
    return int(centerX/amount_of_pixel), int(centerY/amount_of_pixel)
```

[Code Explanation]

Part-1 Binarized Image

1-1. Create a new image which is copy from original image.

1-2. Check if the pixel value is greater or equal to 128. If true, set value to 255; otherwise, 0.

1-3. Save binarized lena image.

Part-2 Image to Histogram

2-1. Create a hash vector with dimension 256 to record the pixel value’s occurrence.

2-2. Traverse all pixels to record their value occurrence.

2-3. Save histogram image.

Part-3 Connected Components

3-1. Create a new image which is in RGB format (since we are going to draw boxes and center)

3-2. Use DFS traversal to record how many objects in the image (why using DFS? inspired from leetcode #200 number of islands). At the same time, maintain a hash vector to store the object id and its occurrence.

3-3. Choose 4 or 8 connected traversal in neighbor_dfs_traversal auxiliary function.

3-4. After marking objects, let’s create bounding boxes. We only deal with whose object id occurs >= 500 times.

3-5. Draw bounding boxes and corresponding centroids.

3-6. Save connected components images.