Computer Vision HW01 旁聽生 陳又蕾 Brief description:

Brief Description: Used Python, OpenCV, numpy to generate a binary image, and the histogram and the connected components of that image. The connected components are surrounded by blue bounding boxes and a red cross inside to indicate where the centroid is in each box.

Binary Image

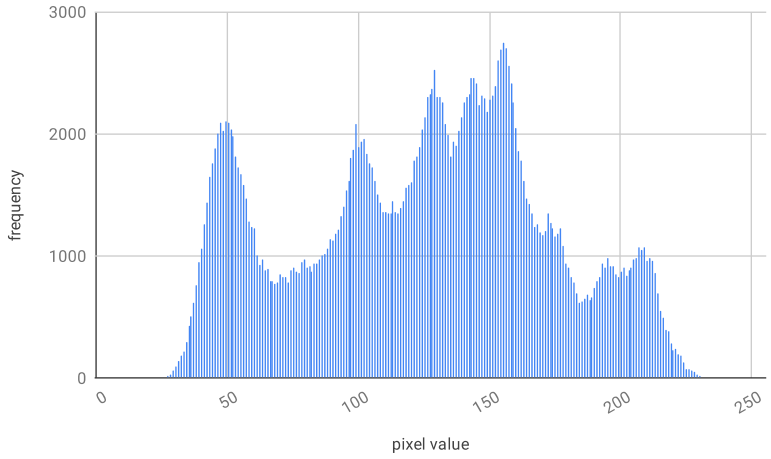
```
def binary(img):
    midpoint = 128
    res = np.zeros((img.shape[0], img.shape[1]), np.int)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i][j] > midpoint:
                res[i][j] = 255
            else:
                res[i][j] = 0
    return res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

I looped through all the pixels of the original image and set those that have a value above 128 to 255 and those that have a value below 128 to 0 on the respective pixel of the image copy.

Histogram

```
def histogram(img):
    count = np.zeros(256, np.int)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
```

```
            val = img[i][j]
            count[val] += 1
    return count;
```



I create an array of zeros with length 256 (length of pixel values 0-255) first, so that when I loop through the binary image, I can constantly update the number of times that a certain pixel value appears in that array. The resulting array values is then saved to a csv file using np.savetxt in order to import into Excel to produce a histogram.

Connected Components

```
def connected_components(img):
    temp = np.zeros(img.shape, np.int)
    binarized_img = binary(img)
    k = 1
    for i in range(binarized_img.shape[0]):
        for j in range(binarized_img.shape[1]):
            if binarized_img[i][j] != 0:
                temp[i][j] = k
                k += 1

    temp_copy = copy.deepcopy(temp)

    while True:
        temp = top_down(temp)
        temp = bottom_up(temp)
        comparison = (temp_copy == temp)
```

```
        if(comparison.all()):
            break
        temp_copy = temp

    components = omit_region_by_threshold(temp, 500)
    binarized_img = cv2.cvtColor(np.float32(binarized_img), cv2.COLOR_GRAY2BGR)
    binarized_img = draw_bounding_boxes(binarized_img, components)
    binarized_img = draw_centroid(binarized_img, components)

    return binarized_img;

def top_down(img):
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            if(img[row][col] > 0):
                img[row][col] = get_min_val(img, row, col)
    return img

def bottom_up(img):
    for row in range(img.shape[0] - 1, -1, -1):
        for col in range(img.shape[1]- 1, -1, -1):
            if(img[row][col] > 0):
                img[row][col] = get_min_val(img, row, col)
    return img

def get_min_val(img, row, col):
    min_val = img[row][col]
    for drow in [-1, 0, 1]:
        for dcol in [-1, 0, 1]:
            if drow == dcol == 0:
                continue
            cur_row = row + drow
            cur_col = col + dcol
            if(cur_row < 0 or cur_row >= img.shape[0] or
                cur_col < 0 or cur_col >= img.shape[1]):
                cur_val = None
            else:
```

```
            cur_val = img[cur_row][cur_col]
        if(cur_val != None and cur_val< min_val and cur_val > 0):
            min_val = cur_val
    return min_val

def omit_region_by_threshold(img, threshold):
    res = {}
    count = np.zeros(np.max(img) + 1, np.int)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i][j] > 0:
                count[img[i][j]] += 1
    for row in range(img.shape[0]):
        for col in range(img.shape[1]):
            if count[img[row][col]] > threshold:
                if(not(count[img[row][col]] in res)):
                    res[count[img[row][col]]] = [(row, col)]
                else:
                    res[count[img[row][col]]].append((row, col))
    return res

def draw_bounding_boxes(img, components):
    for values in components.values():
        left = values[0][0]
        top = values[0][1]
        right = values[0][0]
        bottom = values[0][1]
        for (r, c) in values:
            if r < left:
                left = r
            if r > right:
                right = r
            if c < top:
                top = c
            if c > bottom:
                bottom = c
        cv2.rectangle(img, (top, left), (bottom, right), (255, 0, 0))
```

```
    return img


def draw_centroid(img, components):
    for coords in components.values():
        center_y = int(sum([coord[0] for coord in coords])/len(coords))
        center_x = int(sum([coord[1] for coord in coords])/len(coords))
        cv2.drawMarker(img, (center_x, center_y), (0, 0, 255), cv2.MARKER_CROSS)
    return img
```

I used the iterative algorithm to find the connected components:

1. Give each pixel a unique label
2. Loop through the pixels from top to bottom, left to right and then bottom to top, right to left. Whenever the current pixel has a neighbor with a smaller non-zero label, the current pixel will be assigned to that smaller label.
3. Repeats step 2 until no pixels are assigned to a new value
4. Create a dictionary to store all connected components that is above the 500 threshold
5. Draw bounding boxes using cv2.rectangles using the dictionary
6. Calculate the centroid ((sum of pixel rows + sum of pixel columns) / (amount of pixels)) and use cv2.drawMarker to draw it