Computer Vision

Home Work 2

Name – ROHIT DAS Student ID - 61047086s

Project – Basic Image Manipulation.

Language and library used: Python, OpenCV, Numpy, Matplotlib

Description: This program will perform the following functions while executing lena.bmp image file:

1. Binarize lena.bmp(threshold at 128)
2. Histogram of Original Image
3. Connected Components of Binary image (regions with + at centre, bounding box)

Parameters: None. Please Copy-paste the image path inside the program.

Algorithms Used –

Part 1: Binarize lena.bmp (threshold at 128) – This is a simple code for binarization in Python. Check if the pixel value is above 127(0x7f). If yes, put 255(0xff) as the value. If not put 0 as pixel value.

Principal Code:

```
def img_binarize(img_in):
    image_pixel_check = (img_in > 0x7f) * 0xff
    image_binary = (image_pixel_check == 0xff) * 1
    return (image_binary)
```

Part 1: Histogram of Binarized Image- Iterate through each pixel intensity and show the values statistically using matplotlib

Principal Code:

```
def show_histogram(image):
    image_histogram = np.zeros([256], np.int32)
    for h_row in range(0, row):
        for w_col in range(0, col):
            image_histogram[image[h_row, w_col]] += 1
    # Creating histogram
    plt.plot(image_histogram)
    plt.title("Original Histogram")
    plt.xlabel("Intensity")
    plt.ylabel("Pixels")
    plt.show()
```

Part 2: Connected Components of Binary image (regions with + at centre, bounding box) Using 4 connect pixels (Iterative Algorithm) -

1. Iterate over every pixel and label them separately.
2. Then using Union Find algorithm to find the parent child relationship of the components.
3. Find the connected component of the images above label threshold 500 pixel
4. Draw rectangle and centroid using rectangle and line function.
5. End

Principal Code –

- Connected Components:

```
def connected_components():
    global op_cnt

    # set parent label
    row, col = cc_img.shape
    for pixels in range(row * col):
        parent_label.append(pixels)

    # do connected components using 4 connected neighbours (left and
up)
    label = 2
    for h_row in range(row):
        for w_col in range(col):
            ok1 = 0
            ok2 = 0
            op_cnt += 1
            if cc_img[h_row, w_col] == 1:
                if w_col - 1 >= 0 and cc_img[h_row, w_col - 1] > 1:
# left has already labeled
                    cc_img[h_row, w_col] = union_find(cc_img[h_row,
w_col - 1])
                    ok1 = 1

                if h_row - 1 >= 0 and cc_img[h_row - 1, w_col] > 1:
# up has already labeled
                    if ok1:
# set the connected component to make left = up as the same group
                        parent_label[cc_img[h_row, w_col]] =
union_find(cc_img[h_row - 1, w_col])
                    else:
                        cc_img[h_row, w_col] = cc_img[h_row - 1,
w_col]

                    ok2 = 1

                if ok2 == 0 and ok1 == 0:
                    cc_img[h_row, w_col] = label
                    label += 1

    # union and find merging
    for h_row in range(row):
        for w_col in range(col):
            op_cnt += 1
            if cc_img[h_row, w_col] > 1:
```

```
                print(cc_img[h_row, w_col])
                cc_img[h_row, w_col] = union_find(cc_img[h_row,
w_col])
                print(cc_img[h_row, w_col])
```

- Union Find algorithm

```
def union_find(label):
    original_label = label
    cnt = 0
    row,col = image.shape
    global op_cnt
    while label != parent_label[label] and cnt < row * col:
        op_cnt += 1
        label = parent_label[label]
        cnt += 1
    return label
```

- Connected components above 500 pixel

```
# statistical data for label threshold > 500
mymap = [0 for pixel in range(row * col)]
for h_row in range(0, row):
    for w_col in range(0, col):
        mymap[cc_img[h_row,w_col]] += 1
cc_pos = {}
cc_value = []
for h_row in range(0, row):
    for w_col in range(0, col):
        if cc_img[h_row,w_col] and cc_img[h_row,w_col] not in
cc_value and mymap[cc_img[h_row,w_col]] > LABEL_THRESHOLD:
            cc_value.append(cc_img[h_row,w_col])
for i in cc_value:
    cc_pos[i] = []
for h_row in range(0, row):
    for w_col in range(0, col):
        if cc_img[h_row,w_col] and mymap[cc_img[h_row, w_col]] >
LABEL_THRESHOLD:
            cc_pos[cc_img[h_row, w_col]].append((h_row, w_col))
```

- Rectangle and Centroid Function

```
# draw the rectangles and centroid

for each_cc_value in cc_value:
    up = min(cc_pos[each_cc_value], key=lambda u: u[0])[0]
    down = max(cc_pos[each_cc_value], key=lambda d: d[0])[0]
    left = min(cc_pos[each_cc_value], key=lambda l: l[1])[1]
    right = max(cc_pos[each_cc_value], key=lambda r: r[1])[1]
# Centroid = average of cc pixels
    cen_i, cen_j = [sum(i) / len(i) for i in
zip(*cc_pos[each_cc_value])]
    cen_i = int(cen_i)
    cen_j = int(cen_j)

    draw_rect(up, down, left, right)
    draw_cent(cen_i, cen_j)
  ############### draw the result rectangle ######
def draw_rect(up, down, left, right):
```

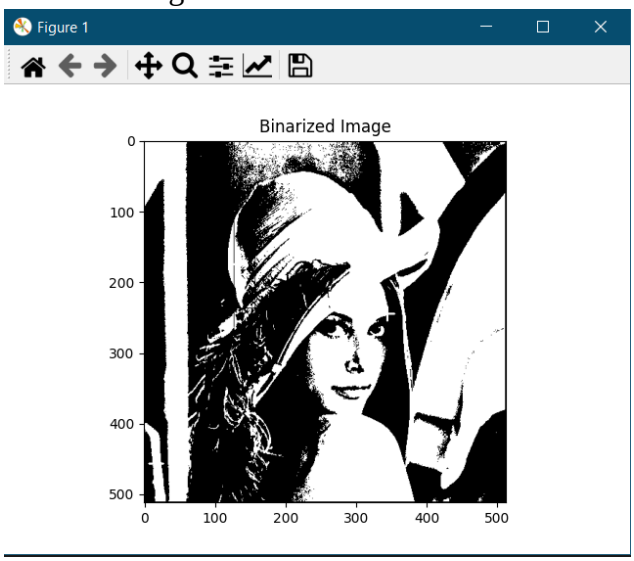
```
    cv2.rectangle(image, (left, up), (right, down), (0,0,255), 1)

############## draw the result centroid #######
def draw_cent(cen_i, cen_j):
    SHIFT = 10
    cv2.line(image, (cen_j - SHIFT, cen_i), (cen_j + SHIFT, cen_i),
(255,255,255), 2)
    cv2.line(image, (cen_j, cen_i - SHIFT), (cen_j, cen_i + SHIFT),
(255,255,255), 2)
```

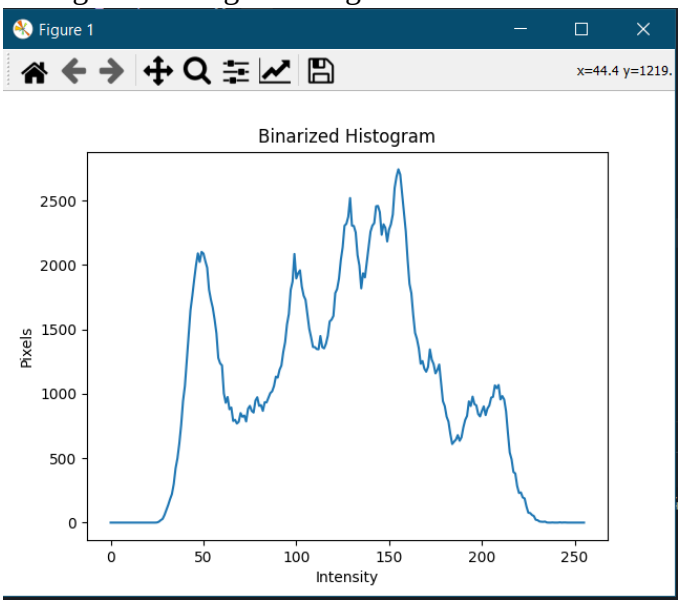
Example:

- Original image

- Binarize image



- Histogram of Original Image



- Connected Components of image