National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Computing

EE 433: Digital Image Processing

Name: Amal Saqib

CMS: 282496

Class: BSCS 9C

Lab 5: Connected Component Labeling

Date: 11th October 2021

Time: 2.00Pm to 5.00Pm

Instructor: Dr. Imran Malik

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Lab 5

Connected Components Labelling

Introduction

This lab is to introduce connected components labeling.

Objectives

This lab will provide the concepts of connected components and its significance in image processing.

Tools/Software Requirement

Python 3.X

Description

Connected components labeling scans an image and groups its pixels into components based on pixel connectivity, *i.e.* all pixels in a connected component share similar pixel intensity values and are in some way connected with each other. Once all groups have been determined, each pixel is labeled with a Gray-level or a color (color labeling) according to the component it was assigned to.

Extracting and labeling of various disjoint and connected components in an image is central to many automated image analysis applications.

Lab Tasks

Implement the connected component labeling algorithm discussed in class on the image given in the lab.

1. Use 4-connectivity

CODE

```
from PIL import Image
import matplotlib.pyplot as plt

def binarization(img):
    # decide the threshold
    threshold = 200
```

```
    # load the pixels of the image
    pix = img.load()
    # get width and height of the input image
    width, height = img.size
    # iterate through all the pixels
    for x in range(width):
        for y in range(height):
            # set background to black and the objects to white
            if pix[x, y] > threshold:
                pix[x, y] = 0
            else:
                pix[x, y] = 255
    return img

def connectedComponent(img):
    # load the pixels of the image
    pix = img.load()
    # get width and height of the input image
    width, height = img.size

    # label array is the size of the image and contains the label for each pixel
    label_array = [[0 for i in range(height)]  for i in range(width)]
    # the labels
    label = 0
    # keeps track of the parent labels
    parent = []

    # iterate through all the pixels
    for x in range(width):
        for y in range(height):
            # if pixel is white
            if pix[x,y] == 255:
                # if there are no pixels on the top and left, make new label
                if (label_array[x-1][y] == 0 and label_array[x][y-1] == 0):
                    label_array[x][y] = label + 1
                    label += 1

                # if there is a label on either the top or the left pixel, label the new pixel as that
one
                elif (label_array[x-1][y] == 0 or label_array[x][y-1] == 0):
                    label_array[x][y] = max(label_array[x - 1][y], label_array[x][y-1])

                # if there are labels on both the top and the left pixels, give the new pixel the
```

```
smaller of the labels and keep track of the parent (smaller) label and child (larger) label
            else:
                label_array[x][y] = min(label_array[x - 1][y], label_array[x][y-1])
                if label_array[x-1][y] != label_array[x][y-1]:
                    parent.append([min(label_array[x - 1][y], label_array[x][y-1]),
max(label_array[x - 1][y], label_array[x][y-1])])

    # second pass
    for x in range(width):
        for y in range(height):
            # change all the child labels to parent labels
            for i in range(len(parent)):
                if label_array[x][y] == parent[i][1]:
                    label_array[x][y] = parent[i][0]

    # display the image
    plt.imshow(label_array, cmap = 'nipy_spectral')
    plt.show()

# open image
image = Image.open("Lab5-image.png").convert('L')

# binarize image
binarized = binarization(image)

# apply the algorithm
connectedComponent(binarized)
```

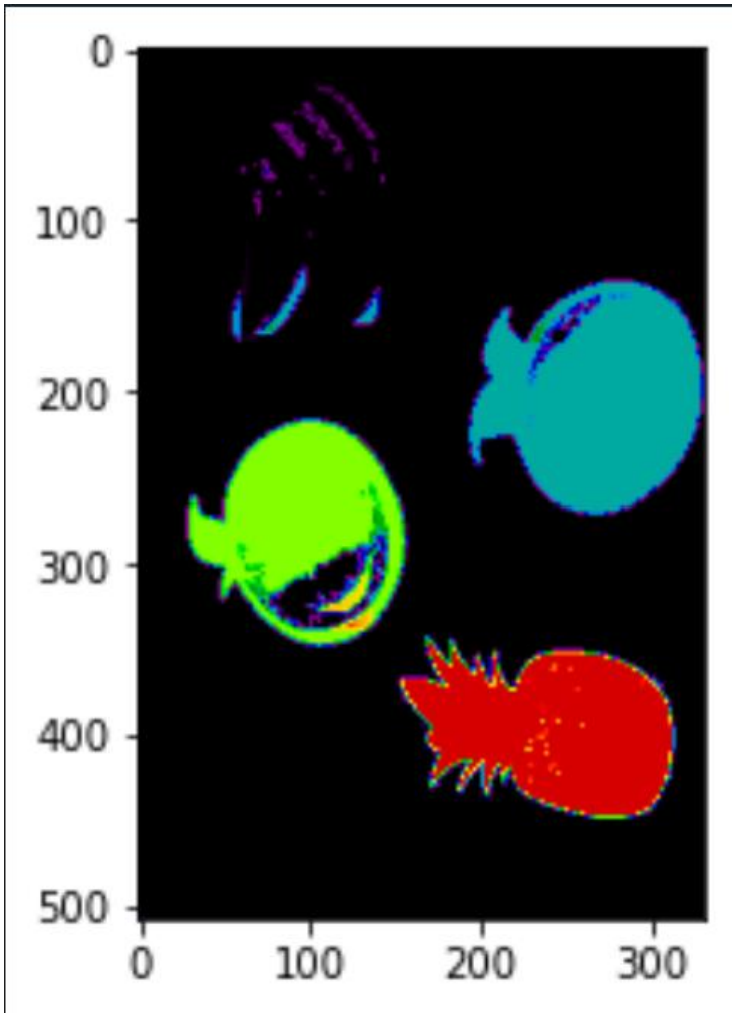
```python
from PIL import Image
import matplotlib.pyplot as plt

def binarization(img):
    # decide the threshold
    threshold = 200

    # load the pixels of the image
    pix = img.load()
    # get width and height of the input image
    width, height = img.size
    # iterate through all the pixels
    for x in range(width):
        for y in range(height):
            # set background to black and the objects to white
            if pix[x, y] > threshold:
                pix[x, y] = 0
            else:
                pix[x, y] = 255
    return img
```

```
21
22  def connectedComponent(img):
23      # load the pixels of the image
24      pix = img.load()
25      # get width and height of the input image
26      width, height = img.size
27
28      # label array is the size of the image and contains the label for each pixel
29      label_array = [[0 for i in range(height)]  for i in range(width)]
30      # the labels
31      label = 0
32      # keeps track of the parent labels
33      parent = []
34
35      # iterate through all the pixels
36      for x in range(width):
37          for y in range(height):
38              # if pixel is white
39              if pix[x,y] == 255:
40                  # if there are no pixels on the top and left, make new label
41                  if (label_array[x-1][y] == 0 and label_array[x][y-1] == 0):
42                      label_array[x][y] = label + 1
43                      label += 1
```

```
44
45                  # if there is a label on either the top or the left pixel, label the new pixel as that one
46                  elif (label_array[x-1][y] == 0 or label_array[x][y-1] == 0):
47                      label_array[x][y] = max(label_array[x - 1][y], label_array[x][y-1])
48
49                  # if there are labels on both the top and the left pixels
50                  # give the new pixel the smaller of the labels and keep track of the parent (smaller) label and child (larger) label
51                  else:
52                      label_array[x][y] = min(label_array[x - 1][y], label_array[x][y-1])
53                      if label_array[x-1][y] != label_array[x][y-1]:
54                          parent.append([min(label_array[x - 1][y], label_array[x][y-1]), max(label_array[x - 1][y], label_array[x][y-1])])
55
56      # second pass
57      for x in range(width):
58          for y in range(height):
59              # change all the child labels to parent labels
60              for i in range(len(parent)):
61                  if label_array[x][y] == parent[i][1]:
62                      label_array[x][y] = parent[i][0]
63
64      # display the image
65      plt.imshow(label_array, cmap = 'nipy_spectral')
66      plt.show()
```

```
67
68      # open image
69      image = Image.open("Lab5-image.png").convert('L')
70
71      # binarize image
72      binarized = binarization(image)
73
74      # apply the algorithm
75      connectedComponent(binarized)
76
77
```

OUTPUT



2. Use 8-connectivity

For 8 connectivity please visit the following link :-

https://en.wikipedia.org/wiki/Connected-component_labeling

CODE

```
from PIL import Image
import matplotlib.pyplot as plt

def binarization(img):
    # decide the threshold
    threshold = 200

    # load the pixels of the image
    pix = img.load()
    # get width and height of the input image
```

```
    width, height = img.size
    # iterate through all the pixels
    for x in range(width):
        for y in range(height):
            # set background to black and the objects to white
            if pix[x, y] > threshold:
                pix[x, y] = 0
            else:
                pix[x, y] = 255
    return img

def connectedComponent(img):
    # load the pixels of the image
    pix = img.load()
    # get width and height of the input image
    width, height = img.size

    # label array is the size of the image and contains the label for each pixel
    label_array = [[0 for i in range(height)]  for i in range(width)]
    # the labels
    label = 0
    # keeps track of the parent labels
    parent = []

    # iterate through all the pixels
    for x in range(width):
        for y in range(height):
            # if pixel is while
            if pix[x,y] == 255:
                top_left = 0
                top = 0
                top_right = 0
                left = 0

                ''' checks to prevent if out of bounds '''
                if x > 0 and y > 0:
                    top_left = label_array[x-1][y-1]

                if x > 0:
                    top = label_array[x-1][y]

                if x > 0 and y + 1 < height:
                    top_right = label_array[x-1][y+1]
```

```
            if y > 0:
                left = label_array[x][y-1]

            neighbours = [top_left, top, top_right, left]
            non_zero = []

            # appends all the non zero neighbours to the non_zero array
            for i in range(len(neighbours)):
                if neighbours[i] != 0:
                    non_zero.append(neighbours[i])

            # if all the neighbours are 0, assign new label to new pixel
            if len(non_zero) == 0:
                label_array[x][y] = label + 1
                label += 1

            # if only one neighbour has label, assign that label to the new pixel
            elif len(non_zero) == 1:
                label_array[x][y] = non_zero[0]

            # if there are 2 or more non zero neighbours, assign the least label to the new
pixel
            if len(non_zero) > 1:
                label_array[x][y] = min(non_zero)
                newList=[]
                for value in non_zero:
                    if value not in newList:
                        newList.append(value)
                non_zero = newList
                if (len(non_zero) == 1):
                    continue
                non_zero.insert(0, min(non_zero))
                if (parent.count(non_zero) == 0):
                    parent.append(non_zero)

    # second pass
    for x in range(width):
        for y in range(height):
            # change all the child labels to parent labels
            for val in parent:
                if(val.count(label_array[x][y])>0):
                    label_array[x][y]=val[0]

        print(label_array[x])
```

```
    # display the image
    plt.imshow(label_array, cmap = 'nipy_spectral')
    plt.show()

# open image
image = Image.open("Lab5-image.png").convert('L')

# binarize image
binarized = binarization(image)

# apply the algorithm
connectedComponent(binarized)
```

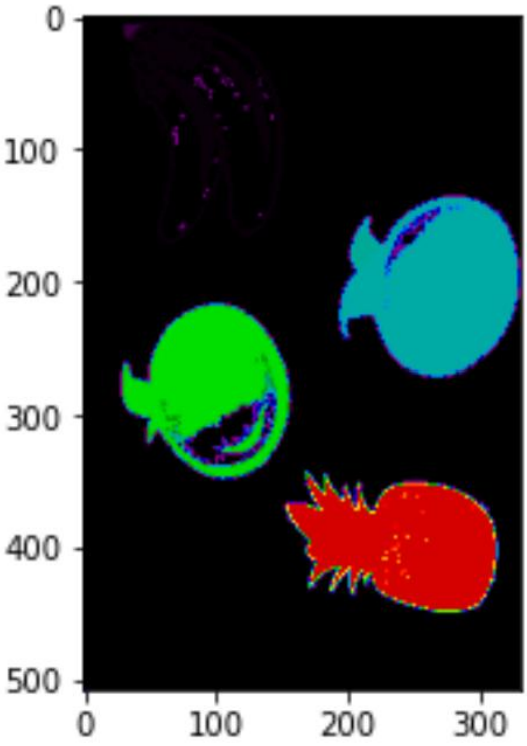
```python
from PIL import Image
import matplotlib.pyplot as plt

def binarization(img):
    # decide the threshold
    threshold = 200

    # load the pixels of the image
    pix = img.load()
    # get width and height of the input image
    width, height = img.size
    # iterate through all the pixels
    for x in range(width):
        for y in range(height):
            # set background to black and the objects to white
            if pix[x, y] > threshold:
                pix[x, y] = 0
            else:
                pix[x, y] = 255
    return img
```

```
21
22  def connectedComponent(img):
23      # load the pixels of the image
24      pix = img.load()
25      # get width and height of the input image
26      width, height = img.size
27
28      # label array is the size of the image and contains the label for each pixel
29      label_array = [[0 for i in range(height)]  for i in range(width)]
30      # the labels
31      label = 0
32      # keeps track of the parent labels
33      parent = []
34
35      # iterate through all the pixels
36      for x in range(width):
37          for y in range(height):
38              # if pixel is while
39              if pix[x,y] == 255:
40                  top_left = 0
41                  top = 0
42                  top_right = 0
43                  left = 0
```

```
44
45                  ''' checks to prevent if out of bounds '''
46                  if x > 0 and y > 0:
47                      top_left = label_array[x-1][y-1]
48
49                  if x > 0:
50                      top = label_array[x-1][y]
51
52                  if x > 0 and y + 1 < height:
53                      top_right = label_array[x-1][y+1]
54
55                  if y > 0:
56                      left = label_array[x][y-1]
57
58                  neighbours = [top_left, top, top_right, left]
59                  non_zero = []
60
61                  # appends all the non zero neighbours to the non_zero array
62                  for i in range(len(neighbours)):
63                      if neighbours[i] != 0:
64                          non_zero.append(neighbours[i])
65
66                  # if all the neighbours are 0, assign new label to new pixel
67                  if len(non_zero) == 0:
68                      label_array[x][y] = label + 1
69                      label += 1
70
71                  # if only one neighbour has label, assign that label to the new pixel
72                  elif len(non_zero) == 1:
73                      label_array[x][y] = non_zero[0]
```

```
                # if there are 2 or more non zero neighbours, assign the least label to the new pixel
                if len(non_zero) > 1:
                    label_array[x][y] = min(non_zero)
                    newList=[]
                    for value in non_zero:
                        if value not in newList:
                            newList.append(value)
                    non_zero = newList
                    if (len(non_zero) == 1):
                        continue
                    non_zero.insert(0, min(non_zero))
                    if (parent.count(non_zero) == 0):
                        parent.append(non_zero)

    # second pass
    for x in range(width):
        for y in range(height):
            # change all the child labels to parent labels
            for val in parent:
                if(val.count(label_array[x][y])>0):
                    label_array[x][y]=val[0]

        print(label_array[x])

    # display the image
    plt.imshow(label_array, cmap = 'nipy_spectral')
    plt.show()

# open image
image = Image.open("Lab5-image.png").convert('L')

# binarize image
binarized = binarization(image)

# apply the algorithm
connectedComponent(binarized)
```

OUTPUT



Deliverable

Hand in the source code from this lab at the appropriate location suggested by the Lab incharge.