CONTENTS

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCES,

PILANI HYDERABAD CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

PROJECT REPORT ON

Medical Image Processing

By- MEHUL JAIN

ID No: 2019A3PS1315H

Under the Guidance of

DR. JABEZ CHRISTOPHER

CSIS

Hyderabad, India

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my parents for providing me with the resources to study at such an esteemed institution and also for their constant support and motivation. I would like to thank Dr Jabez J Christopher sir and Gunjan Parihar Ma’am for letting me work on this project and for their continuous guidance and support.

DEPARTMENT OF COMPUTER SCIENCE

BIRLA INSTITUTE OF TECHNOLOGY

AND SCIENCE, PILANI

CERTIFICATE

This is to certify that the project report entitled “Medical Image Processing” submitted by Mr Mehul Jain (ID No. 2019A3PS1315H) in fulfillment of the requirements of the course CS F366, Lab Oriented Project Course, embodies the work done by him under my supervision and guidance.

Dr. Jabez Christopher
Department of CSE
BITS Pilani, Hyderabad Campus

ABSTRACT

The primary objective of the project is to study the algorithms used in the field of medicine to study patterns and different segmentation methods. Different techniques can result in different accuracies. Based on the results obtained, the most accurate result can be used in mobile applications for medical diagnosis.

Chapter 1

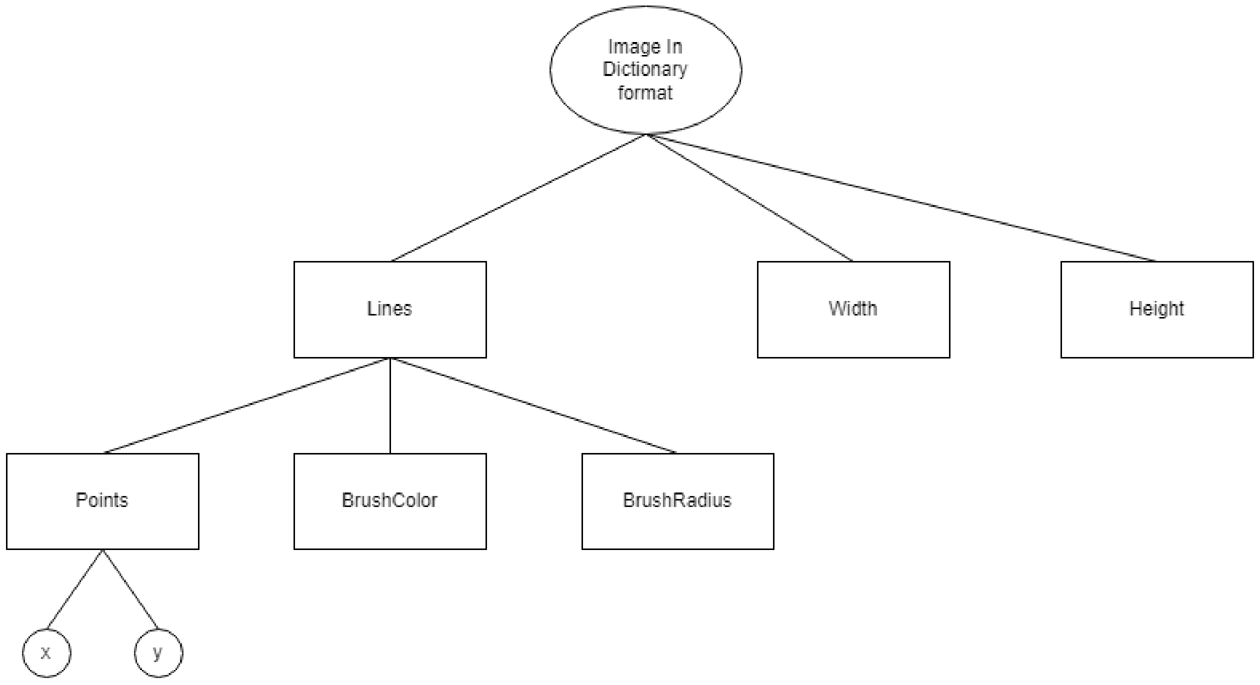
1. Algorithms for Segmentation techniques

1.1 Problem Statement

To find the image perimeter and area using different methods.

1.2 Input format

The input to the problem given is an image in the form of a dictionary which is stored in “JSON”



The values of each coordinate are rounded to the nearest value.

```
x_values = [];
y_values = [];
for value in json['lines']:
  for val in value["points"]:
    x_values.append(round(val['x']))
    y_values.append(round(val['y']))
```

To get the coordinates as a tuple, zip the x and y coordinates.

```
x_y_values = list(zip(x_values, y_values))
```

To get the updated values of x and y, unzip the modified x_y_values.

1.3 Methods to get the perimeter and area of the image

As seen above, preprocessing is done. The final values of x coordinates are stored in x_values. The final values for y coordinates are stored in y_values. The list of tuples, x_y_values contains the final coordinates where c_i is the i^{th} coordinate whose coordinates are (x_i, y_i).

1.3.1 Manual method

Filename: single_manual.ipynb

Algorithm:

a. Converting from list to numpy arrays

1. After getting the relevant x_values, y_values, x_y_values, initialize these to a numpy array.

```
x_values = np.array(x_values)
y_values = np.array(y_values)
x_y_values = np.array(x_y_values)
```

2. Get the maximum values of the x and y coordinates.This gives the maximum or the top-right coordinates of the input image.

```
max_x = max(x_values)
max_y = max(y_values)
```

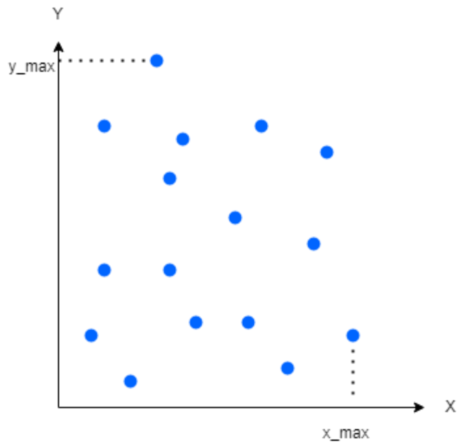
The output is as follows

```
max_x            max_y

157              141
```

Why to choose maximum values of x and y?

The following are the reasons:

- The image is given as an input of (x, y) values. The maximum value of x gives the coordinate which is the most extreme an image is toward the x-axis and the same along the y axis. For example, if given below are the points for an image

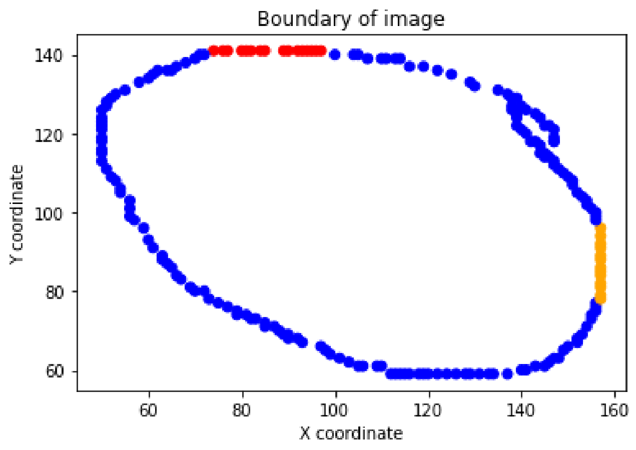


then y_max is the maximum value of y and x_max is the maximum value of x. Therefore, (x_max, y_max) is the top-right most coordinate of the image.

- This helps to create a numpy ndarray which is used to convert the input image to a matrix. This is an important step as image being a non structural data type is now converted to an ndarray structural data type.

b. Plotting to get an inference

1. Plotting an image using scatter plot is to get an idea about the input image. On plotting the input data,



This tells that the input image given is of a boundary of an image.

```
plt.figure()
x, y = zip(*x_y_values)
plt.scatter(x, y, c = colors)
plt.xlabel('X coordinate')
plt.ylabel('Y coordinate')
plt.title('Boundary of image')
```

The boundary is defined by the following

```
colors = []
for i in x_y_values:
  if i[0] == max_x:
    colors.append('orange')
  elif i[1] == max_y:
    colors.append('red')
  else:
    colors.append('blue')
```

where the orange data points represent the data points with the maximum value of the image along the x axis and the red data points represent the data points with the maximum value of the image along y axis.

c. Converting the image to a numpy ndarray

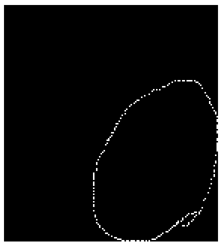
1. Firstly, initialize a zero numpy array with the size (max_x) by (max_y).

```
convert_matrix = np.zeros(shape=(max_x+1,max_y+1))
```

2. Each coordinate in the data point is represented by a ‘1’ in the numpy array. The other points are represented as ‘0’.

```
for value in x_y_values:
  convert_matrix[value[0]][value[1]] = 1
```

This is represented as an image



The perimeter using this is given by the number of non zero values in the matrix

```
np.count_nonzero(convert_matrix)
```

```
193
```

3. Create a copy of the given image data points ndarray as below

```
convert_matrix_copy = convert_matrix.copy()
```

4. Using the given numpy ndarray, now find the starting and ending index of each vector starting and ending in 1.

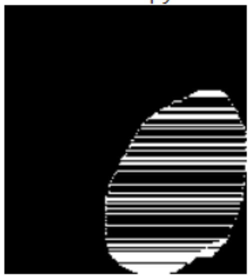
   For filling along the row, use the following

```
for row_index in range(len(convert_matrix_copy)):
  count_one = 0
  index_one = []
  for element_index in range(len(convert_matrix_copy[row_index])):
    if(convert_matrix_copy[row_index][element_index] == 1):
      index_one.append(element_index)
      count_one += 1
    if(count_one > 1):
      start_index = index_one[0]
      end_index = index_one[len(index_one)-1]
      for i in range(start_index,end_index):
        convert_matrix_copy[row_index][i] = 1
```

   The given figure after converting array to image is

```
print(type(convert_matrix))
cmc = Image.fromarray(convert_matrix_copy * 255)
cmc = cmc.convert('RGB')
cmc
```

```
<class 'numpy.ndarray'>
```



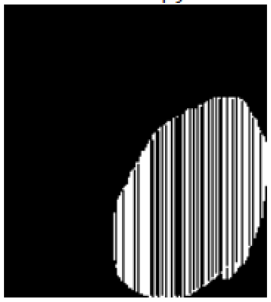
   For filling along the column, use the following

```
for column_index in range(convert_matrix_copy.shape[1]):
  count_one = 0
  index_one = []
  for row_index in range(len(convert_matrix_copy)):
    if(convert_matrix_copy[row_index][column_index] == 1):
      index_one.append(row_index)
      count_one += 1
    if(count_one > 1):
      start_index = index_one[0]
      end_index = index_one[len(index_one)-1]
      for i in range(start_index,end_index):
        convert_matrix_copy[i][column_index] = 1
```

The given figure after converting image to array is

```
print(type(convert_matrix))
cmc = Image.fromarray(convert_matrix_copy * 255)
cmc = cmc.convert('RGB')
cmc
```

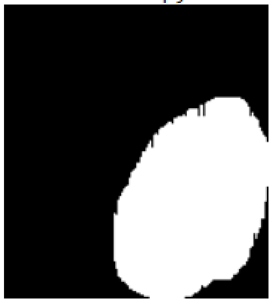
```
<class 'numpy.ndarray'>
```



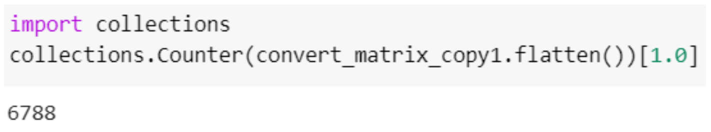
Combining the two results

```
print(type(convert_matrix_copy1))
cmc = Image.fromarray(convert_matrix_copy1 * 255)
cmc = cmc.convert('RGB')
cmc
```

```
<class 'numpy.ndarray'>
```



5. To find the area of the given figure, it is the number of ones in the final output



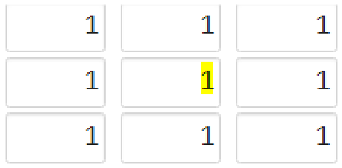
```
import collections
collections.Counter(convert_matrix_copy1.flatten())[1.0]
```

```
6788
```

where the final image is stored in convert_matrix_copy1.

6. To find the perimeter of the image,

   - A pixel is not at the boundary if it is surrounded by the value 1 in all directions.

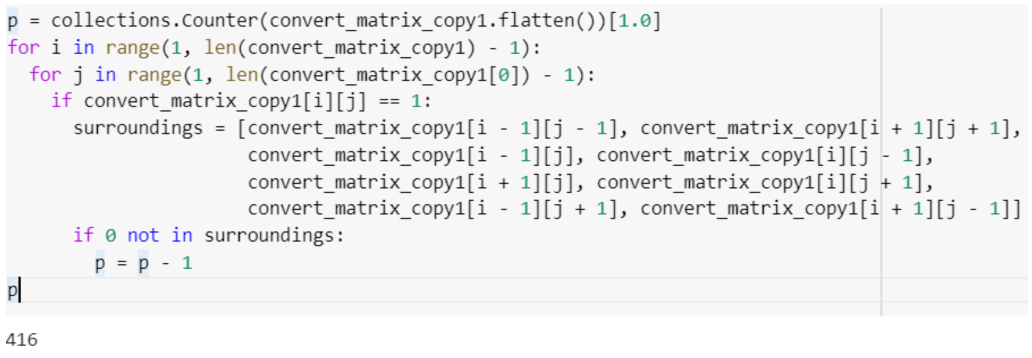


1	1	1
1	1	1
1	1	1

   The highlighted cell is surrounded by 1 in all directions which means that it can not be in the boundary.

   - This means that the total number ones which do not have any zeros in the surrounding area are to be subtracted from the total ones to get the perimeter.

   The code implementation of the above gives the perimeter as



```
p = collections.Counter(convert_matrix_copy1.flatten())[1.0]
for i in range(1, len(convert_matrix_copy1) - 1):
  for j in range(1, len(convert_matrix_copy1[0]) - 1):
    if convert_matrix_copy1[i][j] == 1:
      surroundings = [convert_matrix_copy1[i - 1][j - 1], convert_matrix_copy1[i + 1][j + 1],
                      convert_matrix_copy1[i - 1][j], convert_matrix_copy1[i][j - 1],
                      convert_matrix_copy1[i + 1][j], convert_matrix_copy1[i][j + 1],
                      convert_matrix_copy1[i - 1][j + 1], convert_matrix_copy1[i + 1][j - 1]]
      if 0 not in surroundings:
        p = p - 1
p
```

```
416
```

NOTE: This is the manual method to get the perimeter which is not very accurate.

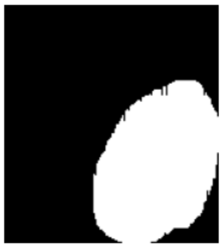
d. Using skimage

1. Use skimage.measure.label on the final image, convert_matrix_copy1. The property of the label is used to connect two points with the same value.

```
label_img = skimage.measure.label(convert_matrix_copy1)
```

The output is

```
li = Image.fromarray((label_img * 255).astype(np.uint8))
li = li.convert('RGB')
li
```



After making all connections using label, now get region properties using

```
regions = skimage.measure.regionprops(label_img)
for props in regions:
  print (props.area, vdk_perimeter(props.convex_image))
```

```
6788 284
```

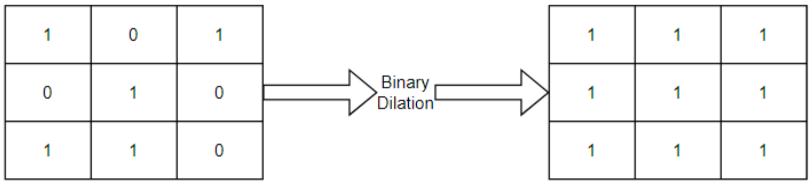
This has many properties like area, centroid and many more.

2. The vdk_perimter function:

    - This function is applied to all regions having boolean values. Using convex_image, convert all non zero values to 1 else assign all zeros to 0. It is to provide a convex binary hull.

    - The original size of the image is padded with zeros in all directions. This convex hull is combined with the initialized padded matrix and is then passed into skimage.morphology.binary_dilation function which dilates the image. Initial number of iterations are 1.

    - The binary_dilation is to enlarge bright regions and shrink dark regions. This is done by taking the maximum value of the surrounding cells and giving all of them the maximum value



where one represents true and zero represents false.

- Convert all false values to 0 and true values to 1. The perimeter can be calculated as the number of cells of value 1 surrounded by a 0 at least in one direction. After getting all the perimeter cells, count the number of non zero values.

```python
def vdk_perimeter(convert_matrix_copy1):
    (w, h) = convert_matrix_copy1.shape
    data = np.zeros((w + 2, h + 2), dtype=convert_matrix_copy1.dtype)
    data[1:-1, 1:-1] = convert_matrix_copy1
    data = skimage.morphology.binary_dilation(data)
    newdata = np.copy(data)
    for i in range(1, w + 1):
        for j in range(1, h + 1):
            cond = data[i, j] == data[i, j + 1] and \
                   data[i, j] == data[i, j - 1] and \
                   data[i, j] == data[i + 1, j] and \
                   data[i, j] == data[i - 1, j]

            if cond:
                newdata[i, j] = 0
    return np.count_nonzero(newdata)
```

1.3.2 Otsu method

Filename: single_otsu.ipynb

Algorithm:

a. Equalize an image using the equalHist property of OpenCV.

   1. Using this property, an image in grayscale is converted to an image with better intensity. The histogram of grayscale image is made and its intensity is distributed in such a way that the lower contrast areas gain a higher contrast.

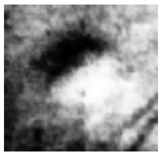
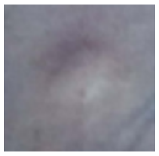


Image after applying equalHist


Input Image

NOTE: To take the grayscale of the input image, use

$$X = 0.2125R + 0.7154G + 0.0721B$$

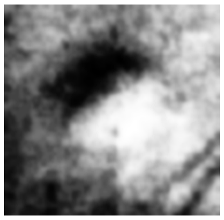
instead of taking the conventional average of R, G, B values where R, G, B stands for Red, Green, and Blue respectively.

b. Removing noise from the image.

1. A Gaussian kernel is used to remove the noise from an image. A gaussian kernel is defined as the odd matrix kernel where each axis has its own deviation. A 2D Gaussian kernel will have a standard deviation along the X axis and Y axis. Here standard deviation is taken as 0 and the matrix is a 5 by 5 kernel.

```
blur = cv2.GaussianBlur(image_equalized,(5,5),0)
```

where the output looks like after applying GaussianBlur()



2. Using the threshold function, find the optimal threshold by using the cv2.THRESH_OTSU flag along with cv2.THRESH_BINARY parameters.

   - Threshold for binary is the value which is the deciding point of a pixel. If a pixel has a value, v_t and the threshold value is given as t then

     threshold(v_t) = 0 if $v_t > t$

     threshold(v_t) = 255 if $v_t \leq t$

   The output, t is

```
ret3
```

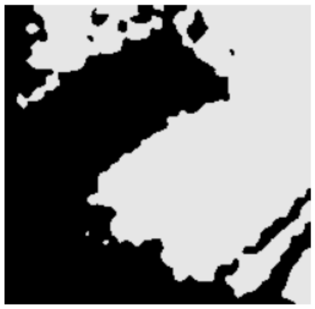
```
130.0
```

- Threshold for Otsu is a flag which is used to generate the threshold automatically. If the image is a bimodal (an image whose histogram has two peaks) then it tries to generate a threshold in between the two peaks. This principle is followed in multiple peaks as well. The first value generated is the threshold value and the second is the output image after applying binary threshold at otsu threshold flag value.

```
ret3,th3 = cv2.threshold(blur,150,230,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

and the output generated is

```
from PIL import Image
cm = Image.fromarray(th3)
cm = cm.convert('RGB')
cm
```



c. Applying skimage

1. After applying the binary threshold and otsu flag, label the image and get the properties of each region.

```
label_img = skimage.measure.label(th3)
regions = skimage.measure.regionprops(label_img)
```

2. Apply vdk_perimeter function for each region

```
label_img = skimage.measure.label(th3)
regions = skimage.measure.regionprops(label_img)

for props in regions:
    print (props.area, vdk_perimeter(props.convex_image))
```

```
2538 249
22 20
16755 515
44 22
5 9
9 11
462 91
```

1.3.3 Watershed method

Filename: single_watershed.ipynb

Algorithm:

a. Applying equalizeHist on input image with markers

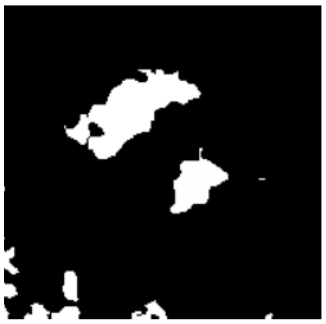
1. Applying equalizeHist distributes the image intensity. Markers are to provide local elevations. On applying markers, move away from the marker.

2. Markers are applied in ranges.

```
image_equalized = cv2.equalizeHist(image)
markers = np.zeros_like(image_equalized)
markers[image_equalized < 20] = 1
markers[image_equalized > 250] = 2
```

and the output is

```
from PIL import Image
cm = Image.fromarray(markers * 255)
cm = cm.convert('RGB')
cm
```



b. Applying sobel to image

1. Two sobel kernels are chosen and are applied in the x and y direction by convolution.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * I$$

   where I is the image and G_x and G_y is in the convolution image of image, I.
2. After getting the output, gradient needs to be calculated as follows

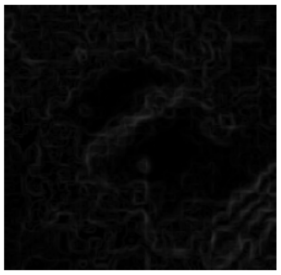
$$G = \sqrt{G_x^2 + G_y^2}$$

The application is

```
elevation_map = sobel(image_equalized)
```

and the output as gradient of the image intensity is

```
from PIL import Image
cm = Image.fromarray(elevation_map * 255)
cm = cm.convert('RGB')
cm
```



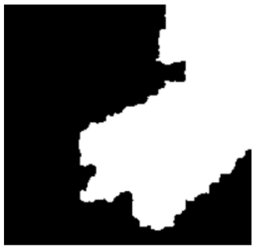
c. Finding boundaries

1. Using watershed function, the boundaries can be found from the markers. This can be further improved by using binary_fill_holes from the scipy library.

```
markers = np.zeros_like(image_equalized)
markers[image_equalized < 20] = 1
markers[image_equalized > 250] = 2
segmentation = watershed(elevation_map, markers)
```

2. The segmentation output is

```
from PIL import Image
cm = Image.fromarray(segmentation)
cm = cm.convert('RGB')
cm
```



and upon applying the binary_fill_holes on segmentation image

```
segmentation = ndi.binary_fill_holes(segmentation - 1)
```

which gives

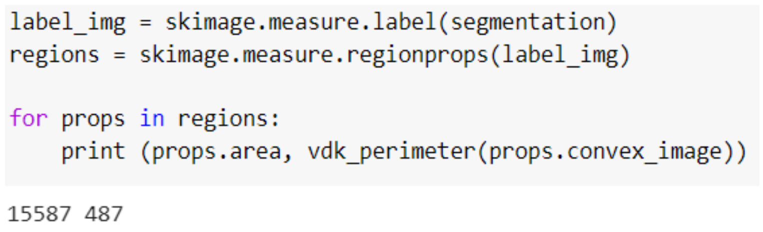
- The segmentation is subtracted by 1 because it has to be made sure that the background is at 0 and it also makes the boundary values -1.

- The binary_fill_holes function makes sure that any zero surrounded by ones is made as 1 to make the region connected so as to make clear boundaries.

d. Getting the area and perimeter

   1. The vdk_function after labeling can be used to find the perimeter. The area is found by regionprops function is used to get the area of the bounded figure.

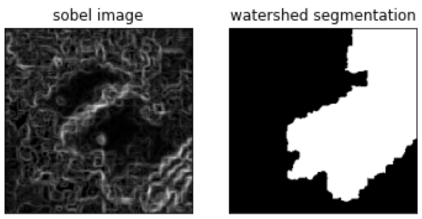
The final outputs are

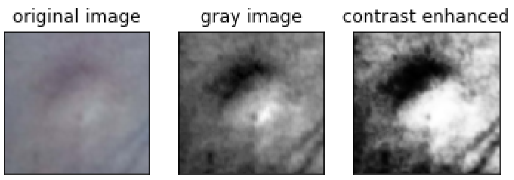


```
label_img = skimage.measure.label(segmentation)
regions = skimage.measure.regionprops(label_img)

for props in regions:
    print (props.area, vdk_perimeter(props.convex_image))
```

```
15587 487
```

and





1.3.4 Region method

Filename: single_region.ipynb

Algorithm:

a. Applying equalizeHist on input image and generating seeds.

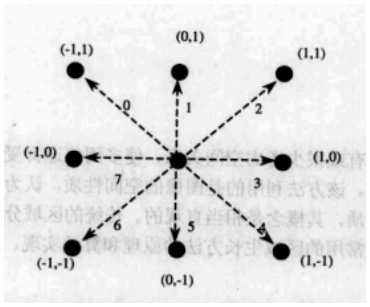
1. After applying equalizeHist, seeds are generated. The seeds are a list of Point class objects.

```
class Point(object):
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def getX(self):
        return self.x
    def getY(self):
        return self.y
```

2. Pass the equalized image to the region grow function for the region algorithm.

```
def regionGrow(img,seeds,thresh,p = 1):
    height, width = img.shape
    seedMark = np.zeros(img.shape)
    seedList = []
    for seed in seeds:
        seedList.append(seed)
    label = 1
    connects = selectConnects(p)
    while(len(seedList)>0):
        currentPoint = seedList.pop(0)

        seedMark[currentPoint.x,currentPoint.y] = label
        for i in range(8):
            tmpX = currentPoint.x + connects[i].x
            tmpY = currentPoint.y + connects[i].y
            if tmpX < 0 or tmpY < 0 or tmpX >= height or tmpY >= width:
                continue
            grayDiff = getGrayDiff(img,currentPoint,Point(tmpX,tmpY))
            if grayDiff < thresh and seedMark[tmpX,tmpY] == 0:
                seedMark[tmpX,tmpY] = label
                seedList.append(Point(tmpX,tmpY))
    return seedMark
```

- The list of Points in seeds are labeled initially.
- The points are connected in all directions as shown in the figure



```python
def selectConnects(p):
    if p != 0:
        connects = [Point(-1, -1), Point(0, -1), Point(1, -1), Point(1, 0), Point(1, 1),  Point(0, 1), Point(-1, 1), Point(-1, 0)]
    else:
        connects = [ Point(0, -1),  Point(1, 0),Point(0, 1), Point(-1, 0)]
    return connects
```

- A threshold value is chosen which is compared with the difference between the grayscale value of the current point from seed and the temporary point after connections in all directions.

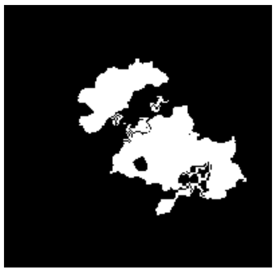
```python
def getGrayDiff(img,currentPoint,tmpPoint):
    return abs(int(img[currentPoint.x,currentPoint.y]) - int(img[tmpPoint.x,tmpPoint.y]))
```

  If the difference is less than the threshold value and the temporary point is unlabeled then the temporary point is considered as a seed. This is now used to get deeper connections of the image.

- This is repeated until all the points are connected.

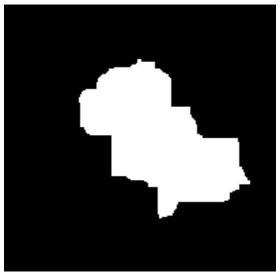
```python
binaryImg = regionGrow(img_eq,seeds,4)
```

  The binary image after the region grow algorithm is



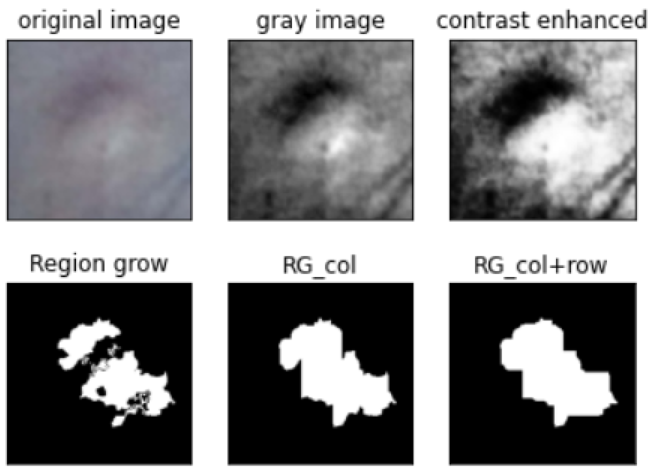
b. The area and perimeter can be found out by filling the regions

   1. Using the row wise and column wise fill method, the gaps can be filled giving



2. The perimeter can be found out by using the vdk_perimeter function.

c. The final results of the algorithm are



```
7597  313
```

Chapter 2

2. Conclusion and References

2.1 Conclusion

Taking into consideration the different methods and their outputs, it is positive to say that different methods give different accuracies based on the types of inputs provided. The results can differ based on the lighting, camera calibration and many other attributes which can change the image dependencies. The results can be applied on the medical diagnosis with proper guidance under professionals.

2.2 References

1. Research paper reference: Gunjan Parihar, Jabez Christopher, Y S Joycey, Y. B Lazarusz, and J Dayalan, " Measurement of Skin Test Wheals using Image Segmentation Approaches," International Conference on Computing, Communication and Automation, December 2021.

2. Code references: Github Repository

3. Image processing code by Gunjan Parhar.