ECE 661 (Fall 2016) - Computer Vision - HW 6

Debasmit Das

October 25, 2016

1 Otsu Segmentation

The whole process of carrying out the segmentation on the gray-scale image is based on a few steps-

- We construct a 256 (L) level histogram h of an image such that each level $h[i]$ contains the number of pixels n_i of the gray level i in the image
- We calculate the average grayscale value μ_T of the image such that $\mu_T = \sum_1^L ip_i$, $p_i = n_i/N$ and N is the total number of pixels in the image.
- For each level k in the histogram we calculate the zeroth order cumulative moment - $\omega(k) = \sum_1^k p_i$, the first order cumulative moment - $\omega(k) = \sum_1^k ip_i$ and then the between class variance - $\sigma_B^2(k) = [\mu_T\omega(k - \mu(k)]^2/[\omega(k)(1 - \omega(k))]$
- We choose the threshold k^* such that $\sigma_B^2(k^*)$ is maximum.
- Now masking is done such that pixels having gray-level value below k^* is 0 and those above k^* is 1. The mask will represent the foreground.
- the same process is repeated for a number of iterations. The subsequent iterations will not contain pixels below the threshold. The number of iterations are chosen manually.

2 RGB based Image segmentation

The implementation of this method takes the following steps -

- We separate the colored image into the 3 RGB channels
- We obtain the foreground mask of each channel by applying the Otsu's algorithm
- We next decide how to combine the mask of for each channel to obtain the overall mask. This will depend on the color distributions of the foreground-

  (a) For the lake image, the lake is mostly blue, while the background has other colors. So, we should have blue as the foreground mask, while the green and red are the background masks. Hence, the overall foreground mask should be a bitwise logical operation such that
  $\texttt{mask} = \texttt{mask}_b$ `AND` (`NOT` $\texttt{mask}_r$) `AND` (`NOT` $\texttt{mask}_g$) where $\texttt{mask}, \texttt{mask}_b, \texttt{mask}_r, \texttt{mask}_g$ are the overall mask, blue mask, red mask and green mask respectively.

  (b) For the leopard image, there is no dominant color. So all channels are treated equally such that

$\texttt{mask} = \texttt{mask}_b \quad \texttt{AND} \quad (\texttt{mask}_r) \quad \texttt{AND} \quad (\texttt{mask}_g)$

(c) For the brain image, there is no dominant color. So all channels are treated equally such that
$\texttt{mask} = \texttt{mask}_b \quad \texttt{AND} \quad (\texttt{mask}_r) \quad \texttt{AND} \quad (\texttt{mask}_g)$

3 Texture based Image segmentation

Given a color image, the texture-based implementation consists of the following steps

- Firstly, the image is converted to grayscale.
- Then we create a gray-scale image such that each pixel represents the variance of the gray-scale images of the $N \times N$ window around the corresponding pixels. This is done to for N =3,5,7 to create 3 channels of the image
- Otsu's algorithm is applied to each channel to obtain separate foreground masks
- To merge the foreground and the background mask, we need to find the variance properties in each of the images to apply it to for each image -

  (a) For the lake image, the pixels of the foreground (lake) do not have much variance. So we should unmask all the 3 channels Hence, the overall foreground mask should be a bitwise logical operation such that
  $\texttt{mask} = (\texttt{NOT} \quad \texttt{mask}_1) \quad \texttt{AND} \quad (\texttt{NOT} \quad \texttt{mask}_2) \quad \texttt{AND} \quad (\texttt{NOT} \quad \texttt{mask}_3)$ where $\texttt{mask}, \texttt{mask}_1, \texttt{mask}_2, \texttt{mask}_3$ are the overall mask, mask for 3×3 window, mask for 5×5 window and mask for 7×7 window respectively.

  (b) For the leopard image, there is variance in the foreground. So all channels are treated equally such that
  $\texttt{mask} = \texttt{mask}_1 \quad \texttt{AND} \quad (\texttt{mask}_2) \quad \texttt{AND} \quad (\texttt{mask}_3)$

  (c) For the brain image, there is variance in the foreground (white matter) and background (grey matter). that
  $\texttt{mask} = \texttt{mask}_1 \quad \texttt{AND} \quad (\texttt{mask}_2) \quad \texttt{AND} \quad (\texttt{mask}_3)$

4 Noise Elimination

The foreground and background might be very noisy. To eliminate noise, we do the following. Proper window size for dilation and erosion should be chosen -

- We carry out erosion followed by dilation to remove noise in the background.
- We carry out dilation followed by erosion to remove noise in the foreground.

5 Contour Extraction

Contour extraction is done after image segmentation to trace out the borders separating the foreground from the background. So, given the binary mask, after segmentation, the contour is

the set of foreground pixels that are 8-neighbours of the background pixels. For that we use the follwoing steps -

- Pixel value of 0 implies that it does not belong to the contour
- Only if the pixel value is 1 and any pixel in the 3×3 window is 0, will the pixel be classified as contour.

6 Comments

The following comments are made about the experiments -

- Texture based image segmentation produces better results especially with challenging images like the leopard and RGB based image segmentation produces better results especially with challenging images like the brain.
- RGB image segmentation is faster owing to the fact that it has less number of steps.
- The no. of iterations for the different channels of RGB based segmentation depend on the color of foreground and background
- The no. of iterations for the different channels of the texture based segmentation is 1 because the variance of the variance in texture based segmentation has no implied meaning
- As far as human supervision is concerned, RGB based segmentation requires knowledge of the color of the foreground such as that of the lake. On the other hand Texture based segmentation requires knowledge of the variance of the foreground. In that regard, both these methods require human supervision but Texture based method is robust to human supervision. For example, misleading the color of the foreground for RGB based segmentation will result in totally absurd result.

7 Experimental Results

7.1 Image 1(*Lake*)

Figure 1: Original Image

7.1.1 RGB based segmentation

No. of Otsu algorithm iterations for blue channel = 1
No. of Otsu algorithm iterations for green channel = 4
No. of Otsu algorithm iterations for red channel = 3
Size of filter for removing background noise = 13×13
Size of filter for removing foreground noise = 7×7

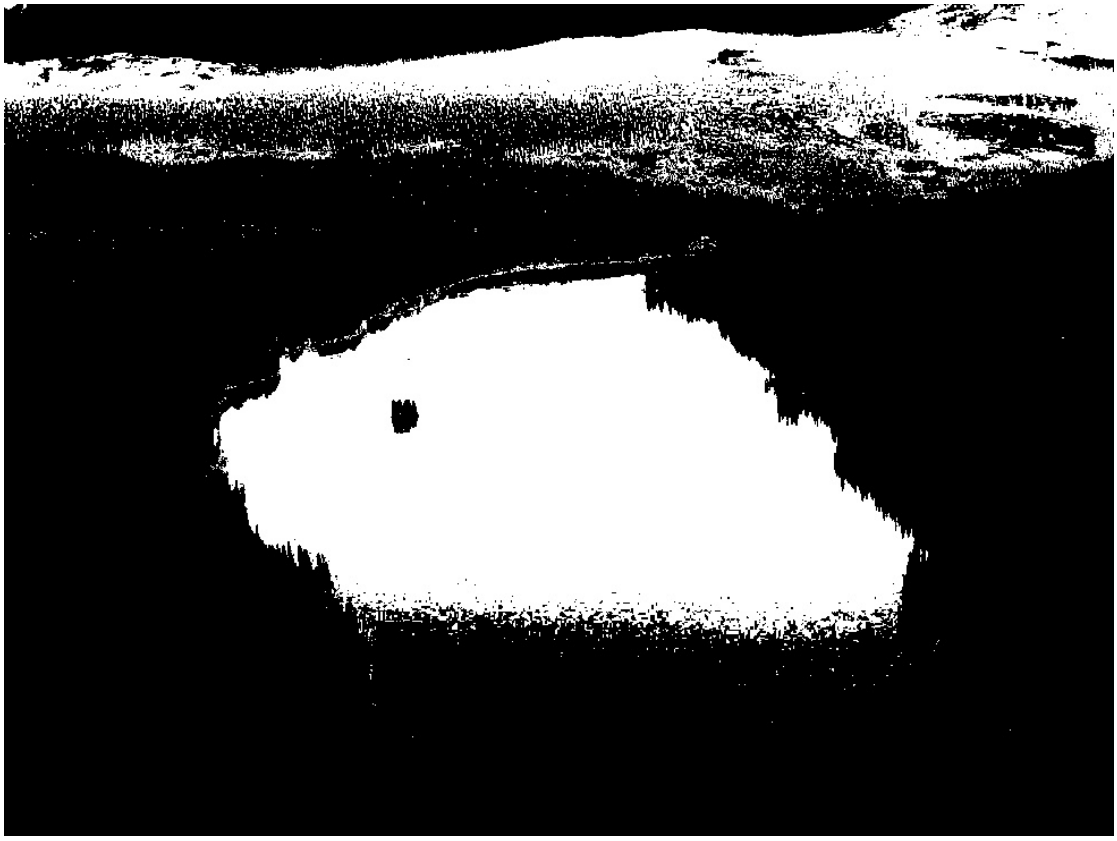

Figure 2: Segmentation with noise

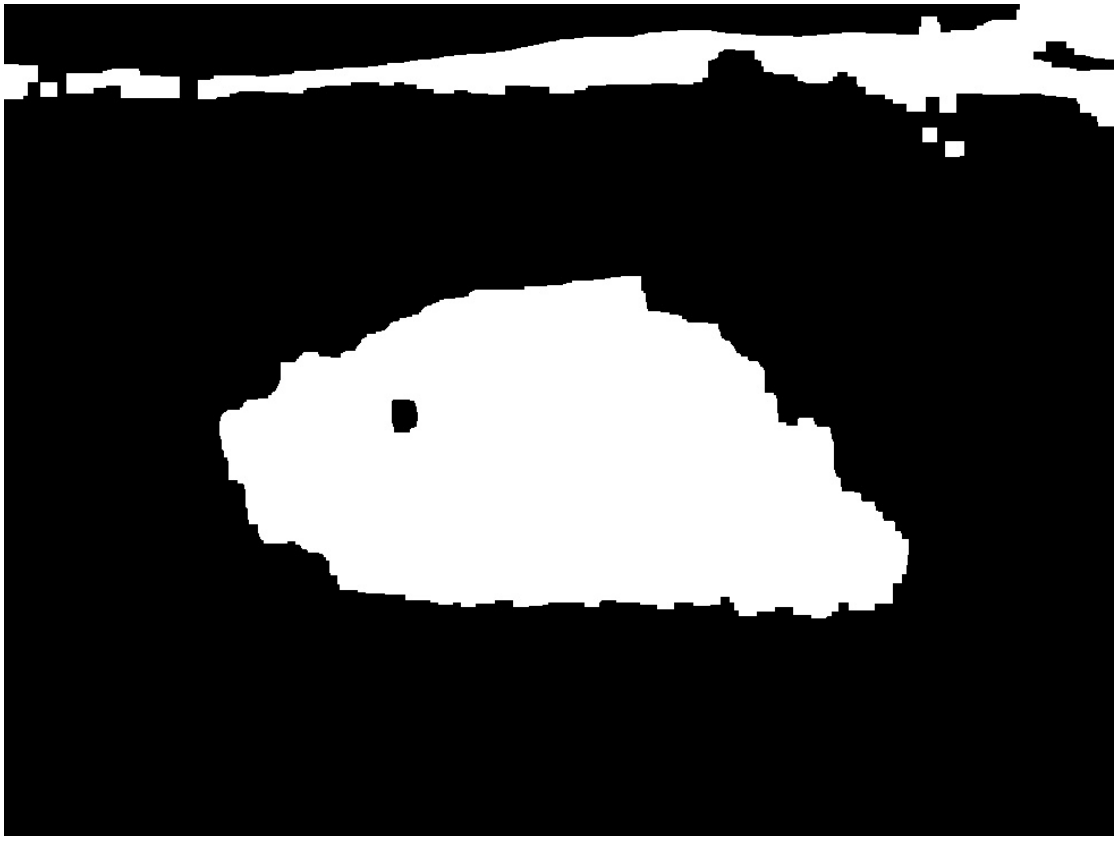

Figure 3: Segmentation without noise

Figure 4: Contour

7.1.2 Texture based segmentation

Size of filter for removing background noise = 31×31
Size of filter for removing foreground noise = 7×7

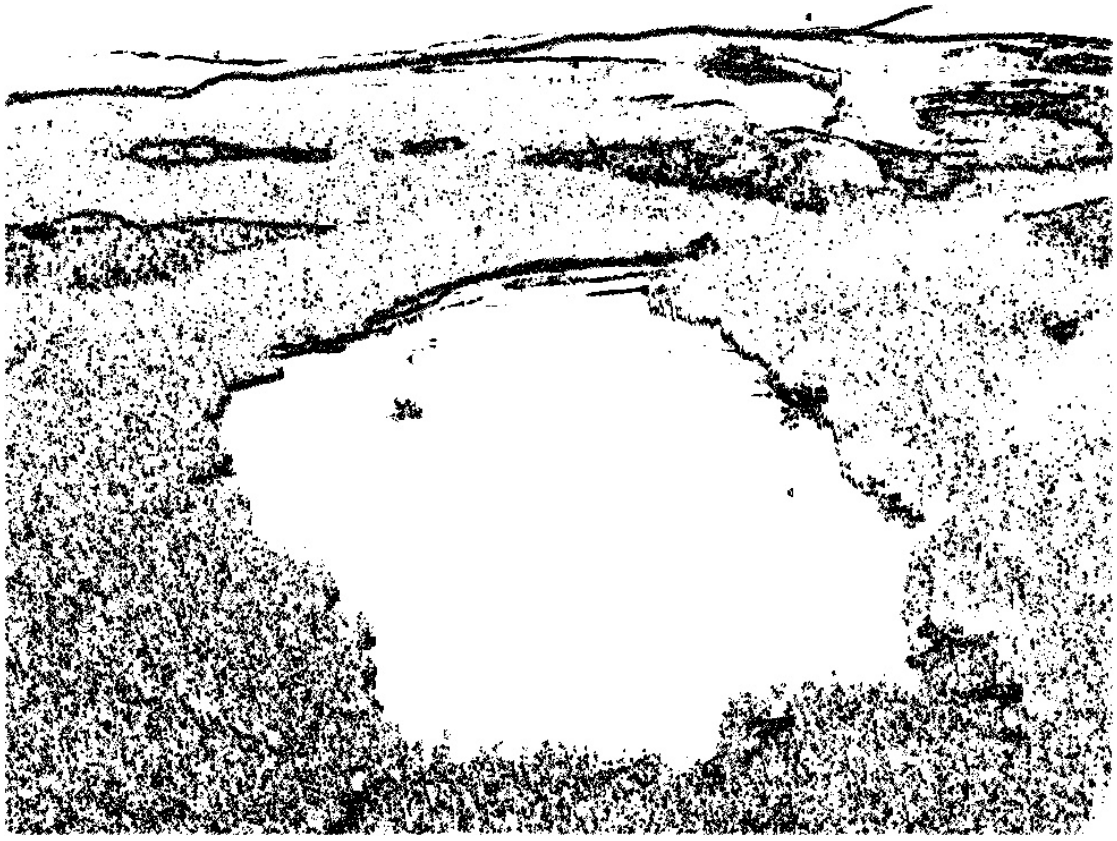

Figure 5: Segmentation with noise

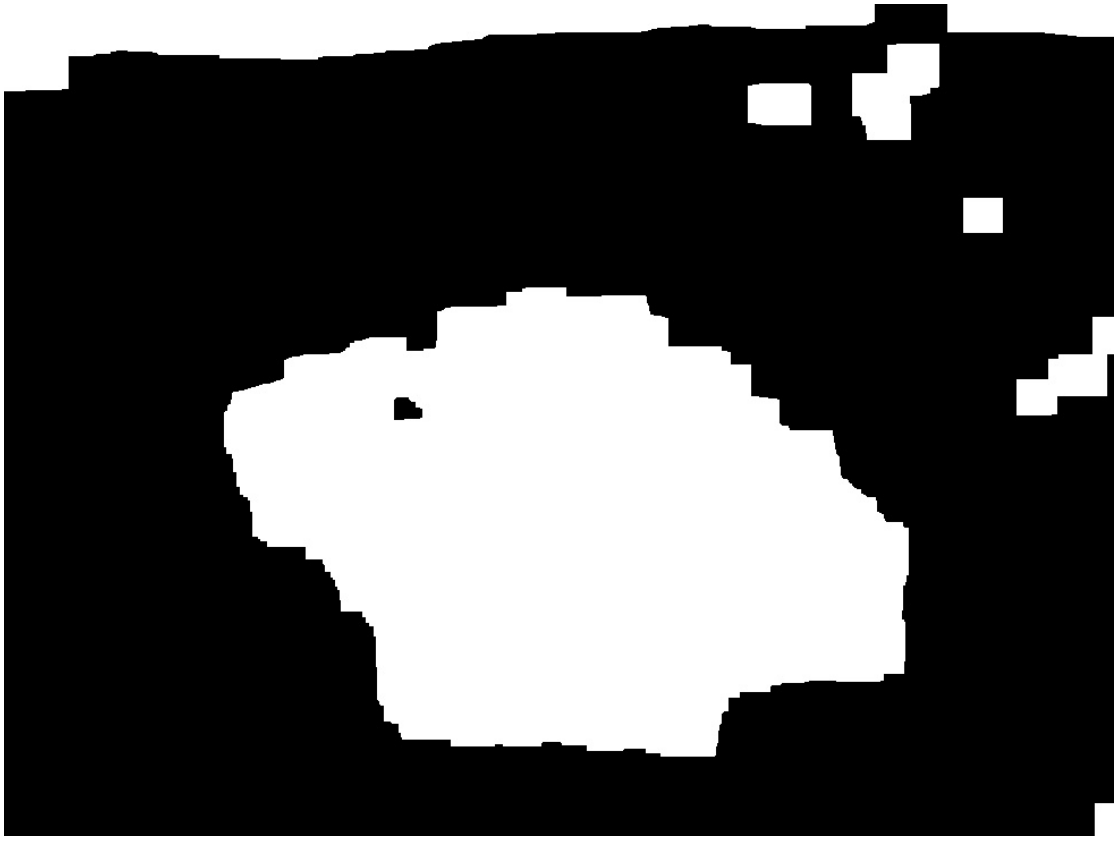

Figure 6: Segmentation without noise

Figure 7: Contour

7.2 Image 2(*Leopard*)

Figure 8: Original Image

7.2.1 RGB based segmentation

No. of Otsu algorithm iterations for blue channel = 1
No. of Otsu algorithm iterations for green channel = 1
No. of Otsu algorithm iterations for red channel = 1
Size of filter for removing background noise = 5×5
Size of filter for removing foreground noise = 7×7

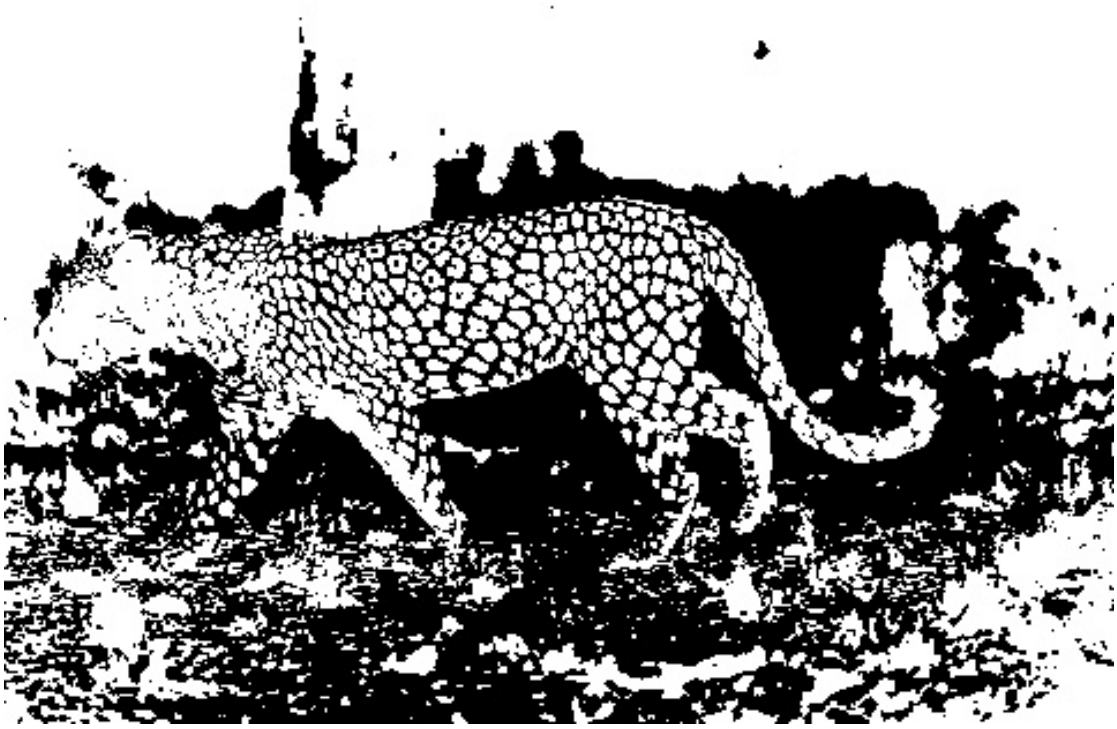

Figure 9: Segmentation with noise

Figure 10: Segmentation without noise

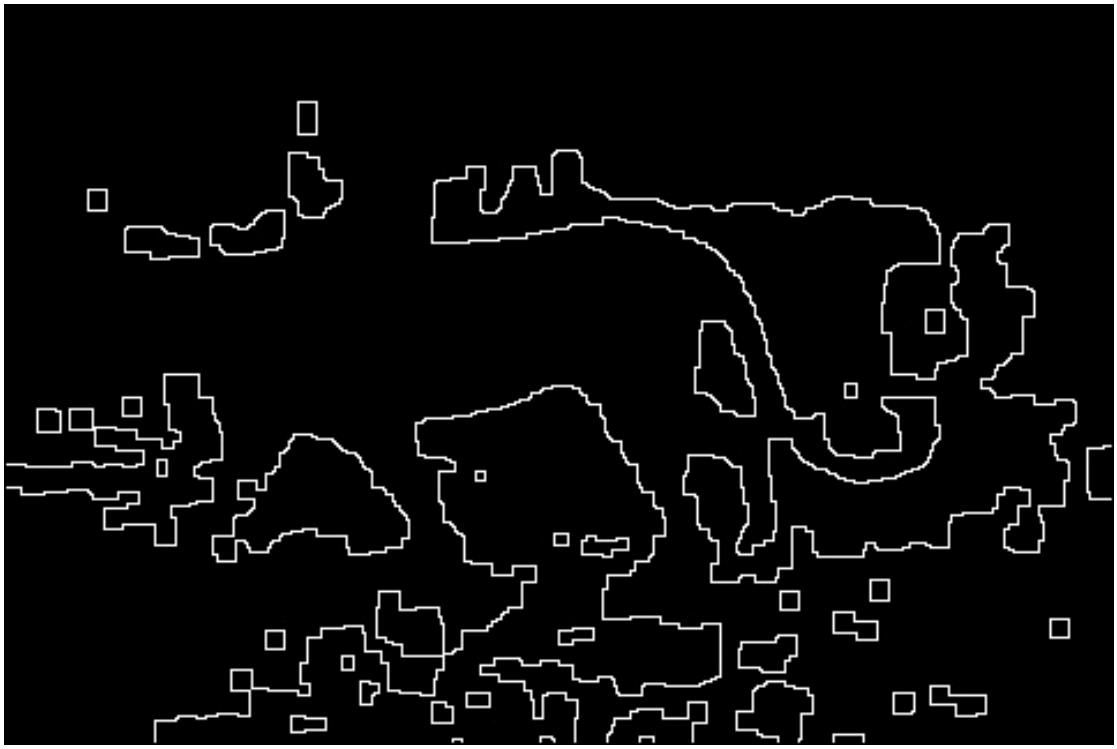

Figure 11: Contour

7.2.2 Texture based segmentation

Size of filter for removing background noise = 5×5
Size of filter for removing foreground noise = 5×5

Figure 12: Segmentation with noise

Figure 13: Segmentation without noise

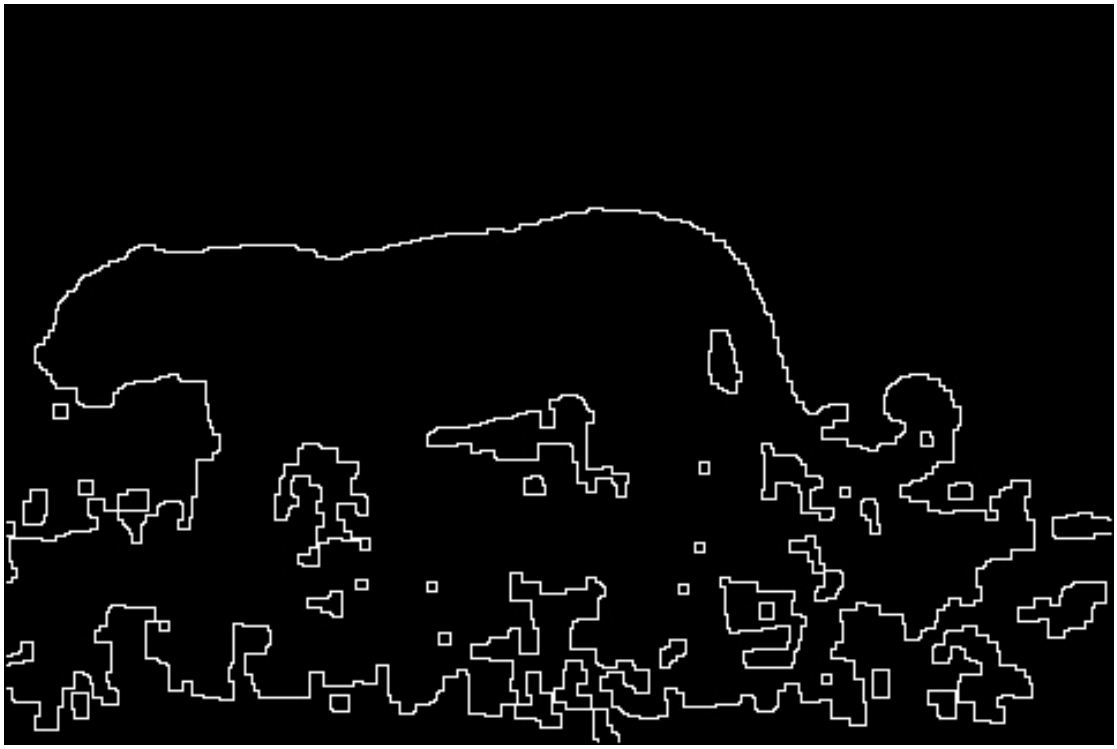

Figure 14: Contour

7.3 Image 3(*Brain*)

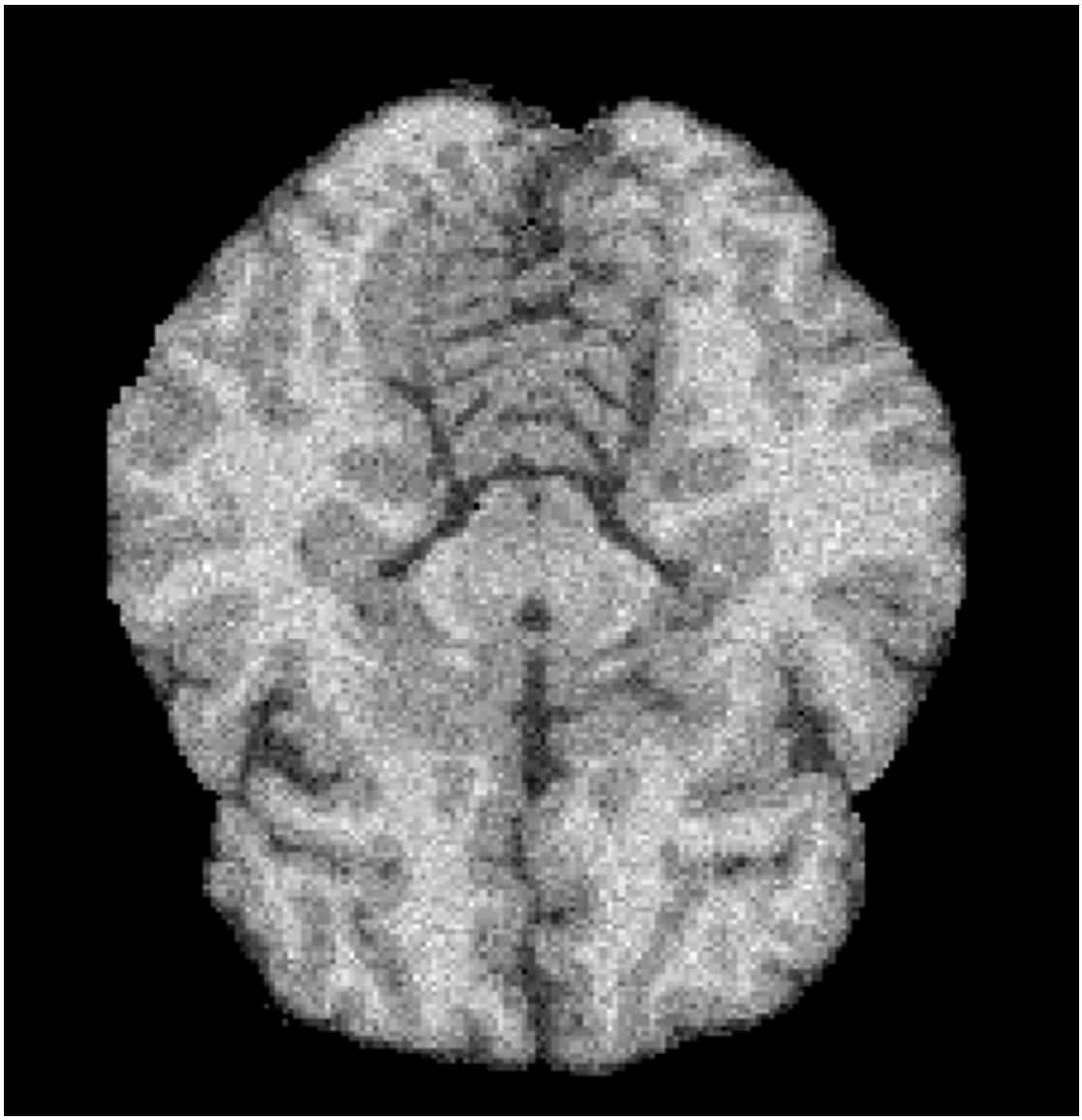

Figure 15: Original Image

7.3.1 RGB based segmentation

No. of Otsu algorithm iterations for blue channel = 1
No. of Otsu algorithm iterations for green channel = 1
No. of Otsu algorithm iterations for red channel = 1
Size of filter for removing background noise = 3×3
Size of filter for removing foreground noise = 13×13

Figure 16: Segmentation with noise

Figure 17: Segmentation without noise

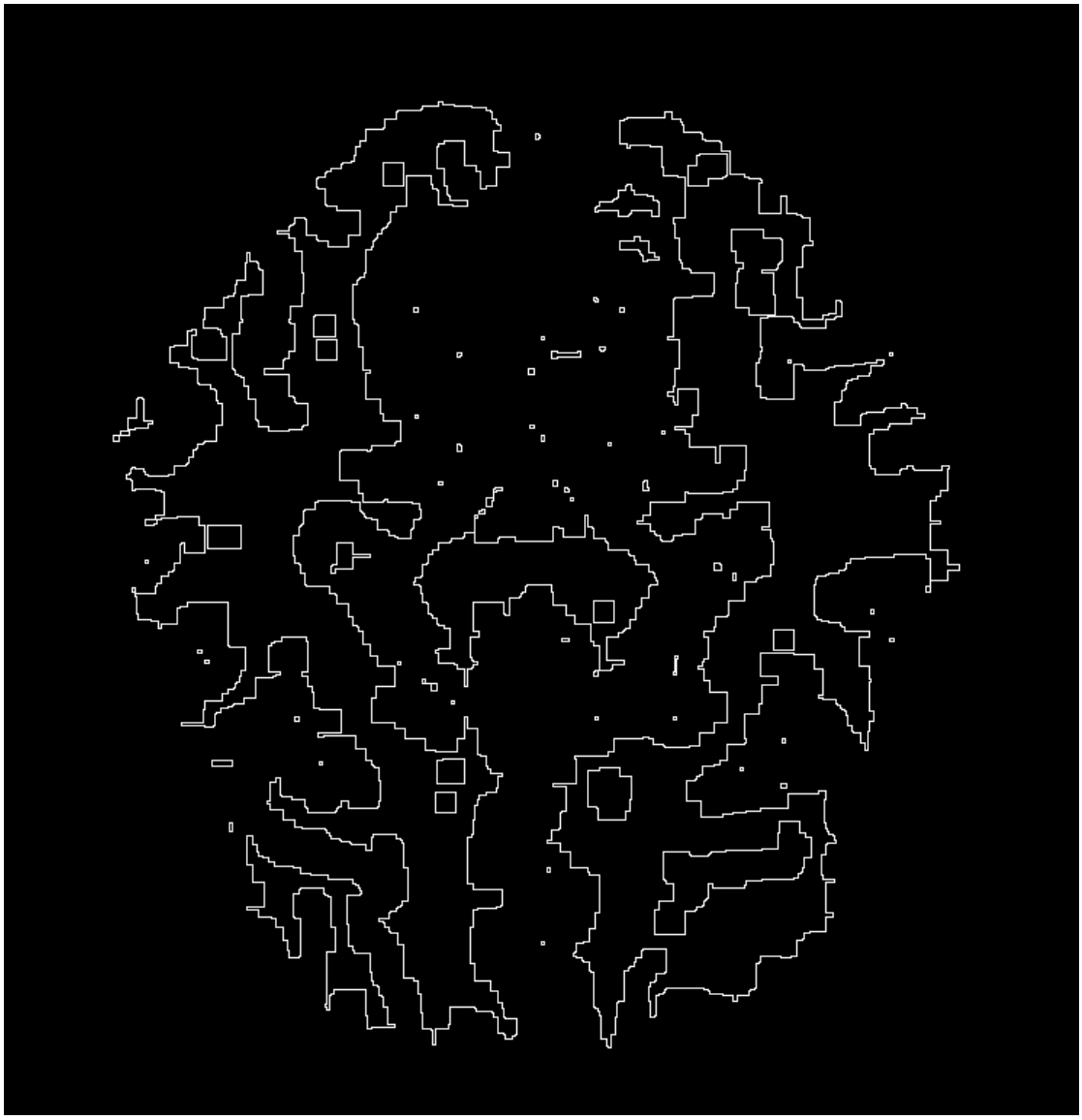

Figure 18: Contour

7.3.2 Texture based segmentation

Size of filter for removing background noise = 5×5
Size of filter for removing foreground noise = 5×5

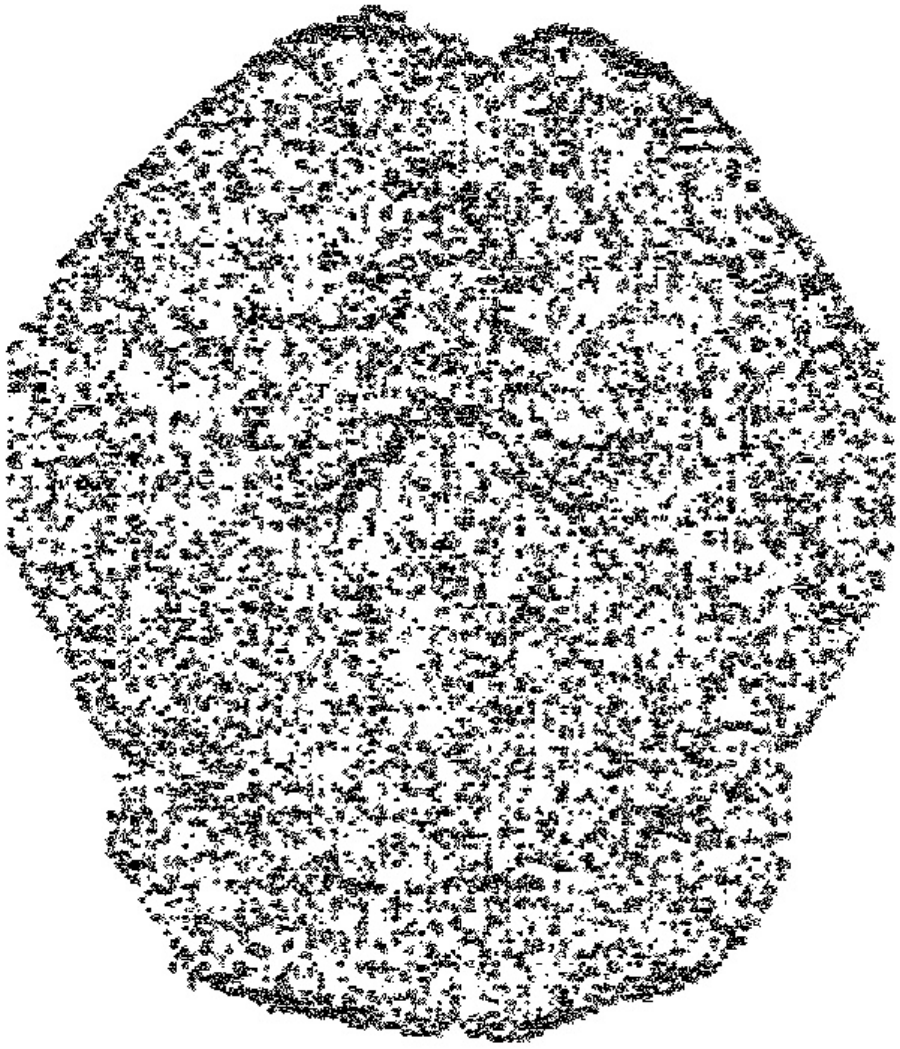

Figure 19: Segmentation with noise

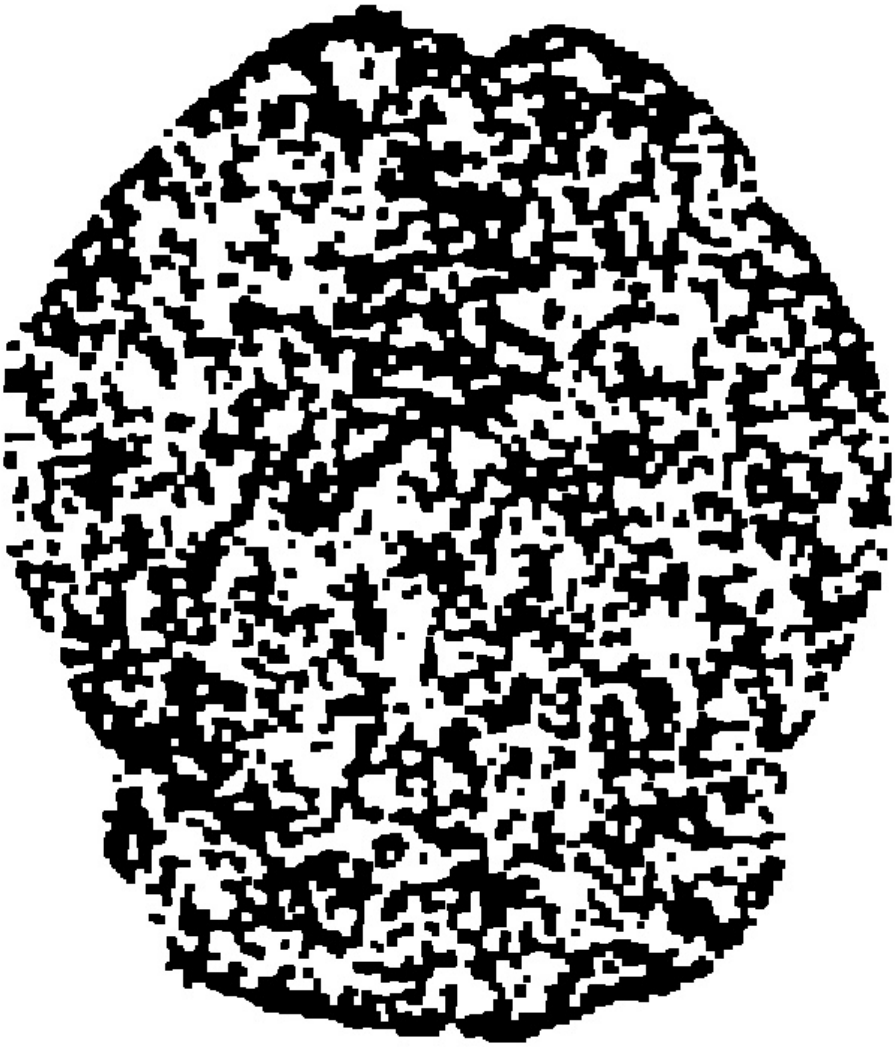

Figure 20: Segmentation without noise

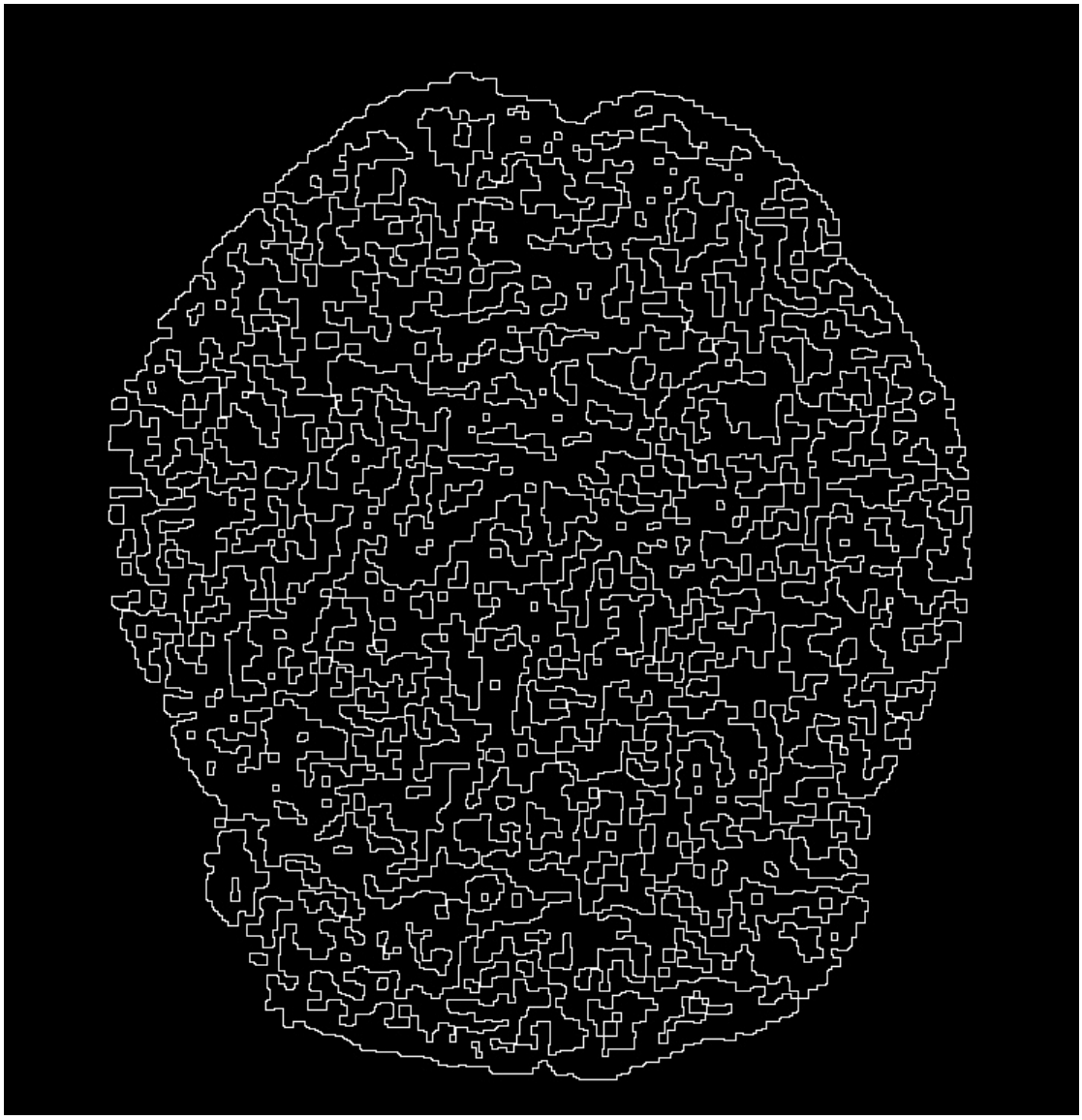

Figure 21: Contour

Code

The script is as follows and is self-explanatory

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 20 17:46:23 2016

@author: debasmit
"""

import numpy as np
```

```
import cv2



#Applying OTSU's algorithm for RGB images
def otsuRGB(img, maskinvert, iterations):
    #Mask invert is just for inverting and non-inverting channels
    output=np.zeros((img.shape[0],img.shape[1]),np.uint8)
    output.fill(255)

    #Iterating over all the three channels
    for c in range(0,3):

        #The mask of channel c
        mask=None

        #Applying Otsu's algorithm to each channel for a number of iterations
        for i in xrange(iterations[c]):
            mask=otsuGray(img[:,:,c],mask)

        print mask

        if maskinvert[c] == 1:
            output=cv2.bitwise_and(output,cv2.bitwise_not(mask))
        else:
            output=cv2.bitwise_and(output, mask)

    return output



#Applying otsu's algorithm of a grayscale image
def otsuGray(img, mask=None):

    hist=np.zeros((256,1))

    #Total number of pixels in the image
    npixels=0.
    mugray=0.;
    threshold=-1;
    maxsigmab=-1;

    for i in range(0, img.shape[0]):
        for j in range(0, img.shape[1]):
            if mask is None or mask[i][j] !=0:
                npixels=npixels+1;
                mugray=mugray+img[i,j]
                hist[img[i,j]] = hist[img[i,j]] + 1

    # The average grayscale for the entire image
    mugray=mugray/npixels;
```

```
    #The cumulative probability of pixels less than equal to k
    wi=0

    #The cumulative average grayscale value of pixels less than equal to k
    mui=0



    for i in range(0, 256):

        #number of pixels at a particular graylevel
        ni=hist[i]

        #probability of pixels at level i
        pi=ni/npixels

        #Update the cumulative probability of pixels  and also the cumulative average grayscale
        wi=wi+pi
        mui=mui+i*pi

        #To avoid the intial and final case this exception is provided so that the
        #between class sigma does not blow up
        if wi==0 or wi==1:
            continue



        #The between class variance if the threshold was i
        sigmabi=((mui*wi - mui)**2)/(wi*(1-wi))

        if sigmabi>maxsigmab:
            threshold=i
            maxsigmab=sigmabi

    output=np.zeros((img.shape[0],img.shape[1]),dtype='uint8')

        #In case the whole whole image was black
    if threshold ==-1:
        return output

        #Creating the output
    for i in range(0, img.shape[0]):
        for j in range(0, img.shape[1]):
            if img[i,j] > threshold:
                output[i,j]=255

    return output

#Texture based image representation is returned
def textureImage(img):
```

```
    #The image is converted to gray scale
    img1=img
    imgg=cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)

    #Initializing the texture based representation
    output=np.zeros_like(img)

    #Window sizes used for texture based representation
    wsize=[3,5,7]

    for k,w in enumerate(wsize):
        d=w/2

        for i in xrange(d, imgg.shape[0]-d):
            for j in xrange(d, imgg.shape[1]-d):
                output[i,j,k]=np.int(np.var(imgg[i-d:i+d+1,j-d:j-d+1]))

    return output



#The contour extraction algorithm. This is done after segmentation
def contourExtract(img):

    #The contour image output is initialised
    output=np.zeros((img.shape[0],img.shape[1]),dtype='uint8')

    #Do the contouring for each pixel in an image
    for i in xrange(1, img.shape[0]-1):
        for j in xrange(1, img.shape[1]-1):
            if img[i,j]!=0 and np.min(img[i-1:i+2,j-1:j+2])==0:
                output[i,j]=255


    return output


##########################################
# Main method starts here
##########################################

if __name__ == "__main__":

    img=cv2.imread('lake.jpg')

    texture=0; #Flag to select whether we use texture based image segmentation

    if texture==0:
        output=otsuRGB(img,[0,1,1],[1,4,3])
    else:
        outputi=textureImage(img)
        output=otsuRGB(outputi,[1,1,1],[1,1,1])
```

```
cv2.imwrite('lake_segment_withnoise.jpg', output)

#Steps to remove noise in the foreground
kern=np.ones((7,7),np.uint8)
output=cv2.dilate(output,kern)
output=cv2.erode(output,kern)


#Steps to remove noise from background
kern=np.ones((13,13),np.uint8)
output=cv2.erode(output,kern)
output=cv2.dilate(output,kern)



cv2.imwrite('lake_segment_withoutnoise.jpg', output)

#Extracting the contour
outputcontour=contourExtract(output)

cv2.imwrite('lake_contour.jpg', outputcontour)
```