# ECE 661 : HW6

Yellamraju Tarun (ytarun@purdue.edu)

October 24, 2016

## 1 Implementation of Otsu Algorithm

- First, we construct a $L = 256$ level histogram which counts the number of pixels with a given intensity level, i.e. $h[i] = n_i$ where $n_i$ denotes the number of pixels with intensity level $i$.
- Next, using the histogram generated above, we estimate the probability mass function of intensity levels by calculating $p_i = \frac{n_i}{N}$ for each intensity level $i$ where $N$ denotes the total number of pixels in the image.
- Next we calculate

$$\omega_0 = \sum_{i=1}^{k} p_i \quad and \quad \omega_1 = \sum_{i=k+1}^{L} p_i \tag{1}$$

$$\mu_0 = \sum_{i=1}^{k} \frac{ip_i}{\omega_0} \quad and \quad \mu_1 = \sum_{i=k+1}^{L} \frac{ip_i}{\omega_1} \tag{2}$$

$$\sigma_B^2 = \omega_0\omega_1(\mu_0 - \mu_1)^2 \tag{3}$$

  for every value of $k \in [0, L-1]$
- We find a value of $k = k^*$ such that the between class variance $\sigma_B^2$ is maximized.
- All pixels with intensity $> k^*$ are determined to be part of the foreground while the rest are part of the background

## 2 Intensity Based Segmentation

We treat each of the three color channels of an RGB image as a separate image and generate masks for each of them using the Otsu segmentation algorithm as described above. The composite mask is formed by taking the logical AND of the individual masks to form the final segmentation. We can manually modify the way the combination of the three masks is taken based on the image we are segmenting as follows

- For the lake image, we take the complements of R and G channels while taking the retaining the B channel segmentation as is. The Otsu algorithm is run iteratively in this case to produce better segmentation.
- For the leopard image, we retain all three channels as they are without modifying them. There was no need for recursively applying the segmentation algorithm in this case.
- For the brain image, we retain all three channels as they are without modifying them.The Otsu algorithm is run iteratively in this case to produce better segmentation.

# 3 Texture Based Segmentation

To form the texture images, we first convert the image to grayscale. Next, we place $N \times N$ windows around each pixel and calculate the variance of the pixels within this window and place that value in the location of the center pixel. We do this for $N = 3, 5, 7$ forming 3 texture images for each image. These 3 texture images are treated as RGB channels and we follow the same procedure for segmentation thereafter as mentioned for the intensity based segmentation approach.

- For the lake image, we take the complements of all three texture channels since the lake is a smooth region and subsequently has a very low texture response. There was no need for recursively applying the segmentation algorithm in this case.
- For the leopard image, we retain all three channels as they are without modifying them. There was no need for recursively applying the segmentation algorithm in this case.
- For the brain image, we retain all three channels as they are without modifying them. There was no need for recursively applying the segmentation algorithm in this case.

# 4 Noise Filtering in Segmentation Masks

It is often the case that there are many gaps or holes in the foreground segmentation mask and small patches of noise detected as foreground in the background. To fill in the gaps in the foreground segmentation, we can perform a dilation process followed by erosion with an appropriately sized operator (refer to source code for details). To remove background noise, we can first do an erosion process followed by dilation. We filter background noise only for the lake image since for the other images, this process will remove a large chunk of the foreground segmentation as well.

## 5 Extraction of Segmentation Contour

Once we have the segmentation mask, we can extract the contour of the segmented region by scanning the mask pixels in raster order. When we encounter a pixel with value 1 and at least one of its 8 neighbors has value 0, it is determined to be on the contour. If not, the pixel is not on the contour.

## 6 Observations

The Otsu algorithm works better with intensity based segmentation for some cases and with texture based segmentation for other cases. For example, for the brain image, since the image was in general a bit noisy, there was a significant texture content in not just the white matter but the gray matter as well because of which the segmentation algorithm could not distinguish between the two. For this case, intensity based segmentation was a lot more successful in segmenting the white matter from the rest of the image.

Another example is the leopard image. The leopard owing to its spots contains a lot of texture content which leads to good results with texture based segmentation. Intensity based segmentation is not very effective in this case. At the same time however, the grassy region at the bottom of the image also has a strong texture response and gets classified as foreground in this approach.

The lake image is an example where both methods work well. There is however a small issue of some of the background at the top of the image appearing similar in both intensity and texture to the foreground of the lake which erroneously gets classified as foreground. The errors seem to be less in the case of the texture based segmentation.

Further, the texture based method is computationally more intensive due to the formation of the texture images. In general scenarios of images with low noise, it seems that the texture based method performs better than the intensity based method.

# 7 Experimental Results

## 7.1 Lake Image

Figure 1: Original Image

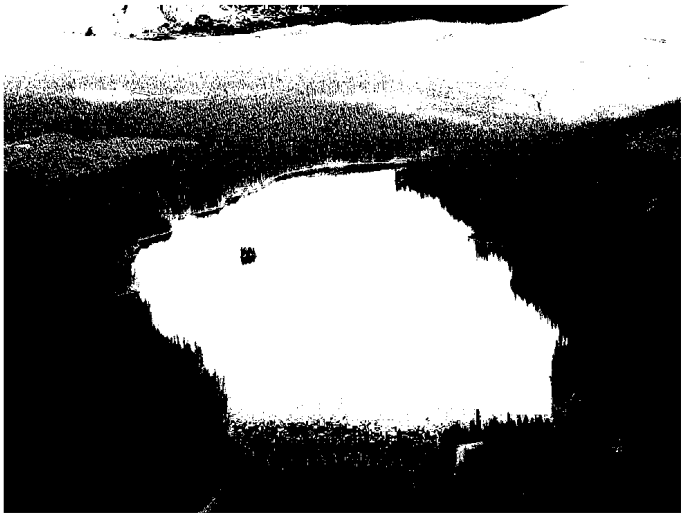

Figure 2: Segmentation mask using RGB values prior to noise filtering

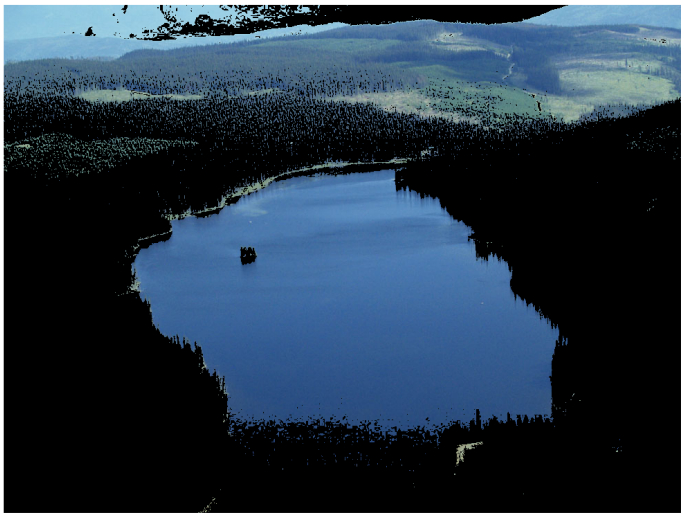

Figure 3: Segmentation using RGB values prior to noise filtering

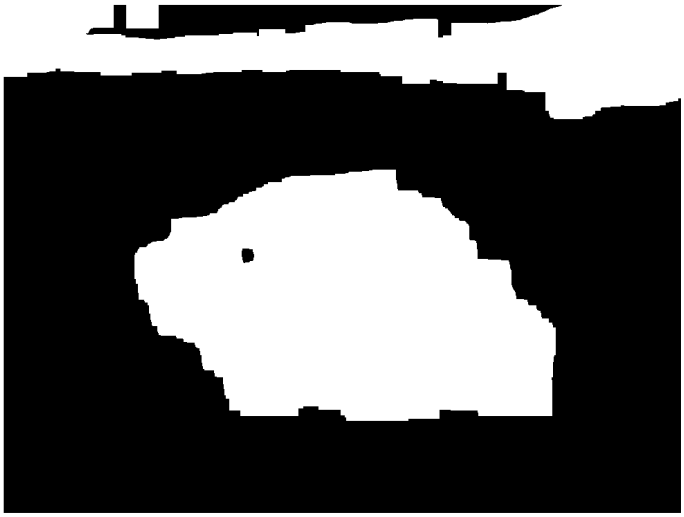

Figure 4: Segmentation mask using RGB values post noise filtering

Figure 5: Segmentation using RGB values post noise filtering

Figure 6: Contour based on RGB Segmentation

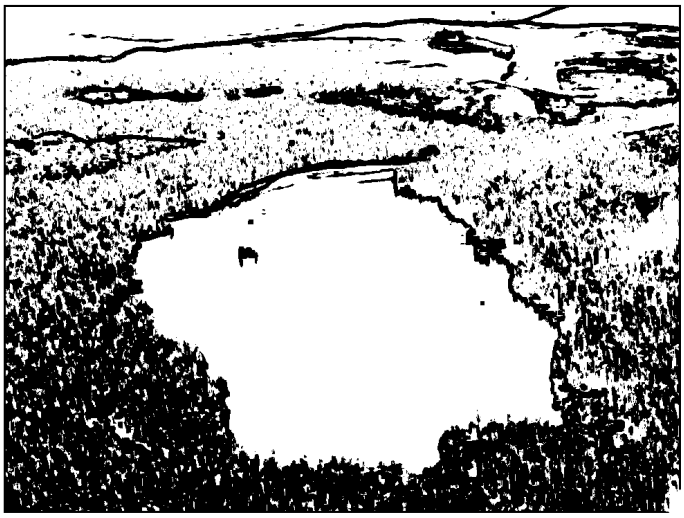

Figure 7: Segmentation mask using texture values prior to noise filtering

Figure 8: Segmentation using texture values prior to noise filtering

Figure 9: Segmentation mask using texture values post noise filtering

Figure 10: Segmentation using texture values post noise filtering

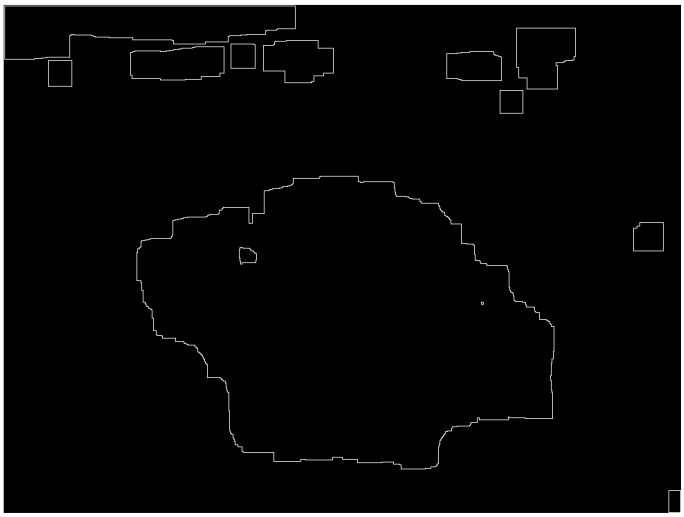

Figure 11: Contour based on texture Segmentation

### 7.2 Leopard Image

Figure 12: Original Image

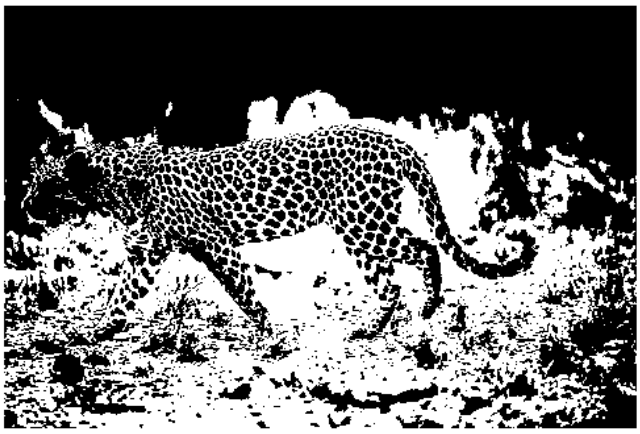

Figure 13: Segmentation mask using RGB values prior to noise filtering

Figure 14: Segmentation using RGB values prior to noise filtering

Figure 15: Segmentation mask using RGB values post noise filtering

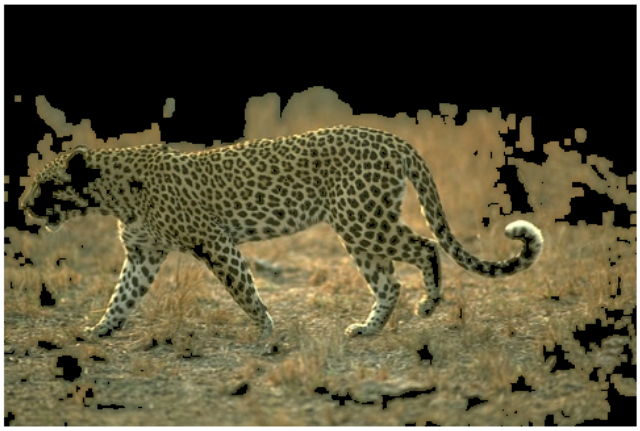

Figure 16: Segmentation using RGB values post noise filtering

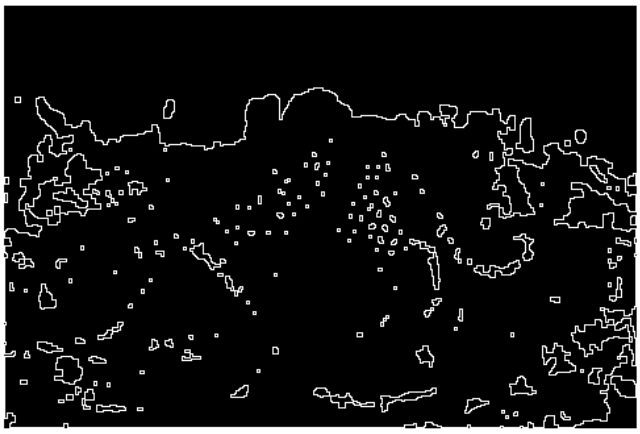

Figure 17: Contour based on RGB Segmentation

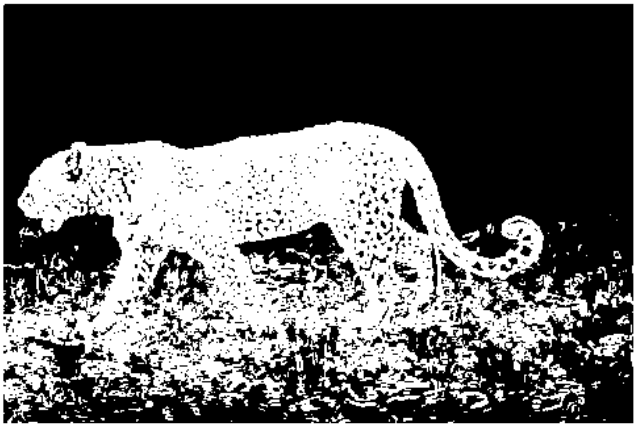

Figure 18: Segmentation mask using texture values prior to noise filtering

Figure 19: Segmentation using texture values prior to noise filtering

Figure 20: Segmentation mask using texture values post noise filtering

Figure 21: Segmentation using texture values post noise filtering

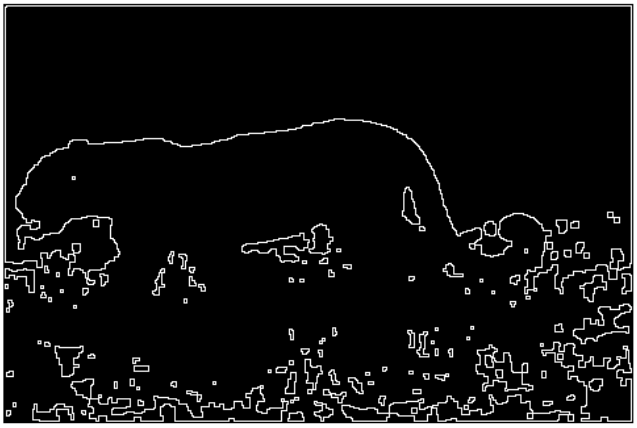

Figure 22: Contour based on texture Segmentation

### 7.3 Brain Image

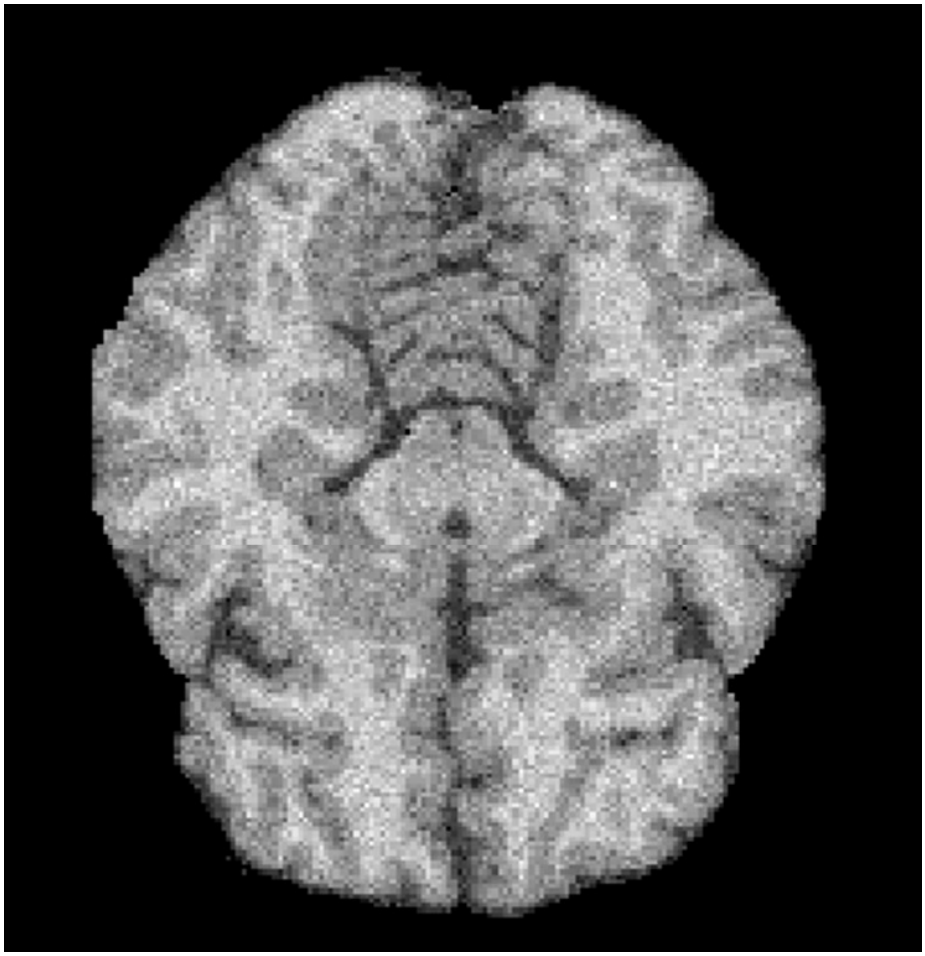

Figure 23: Original Image

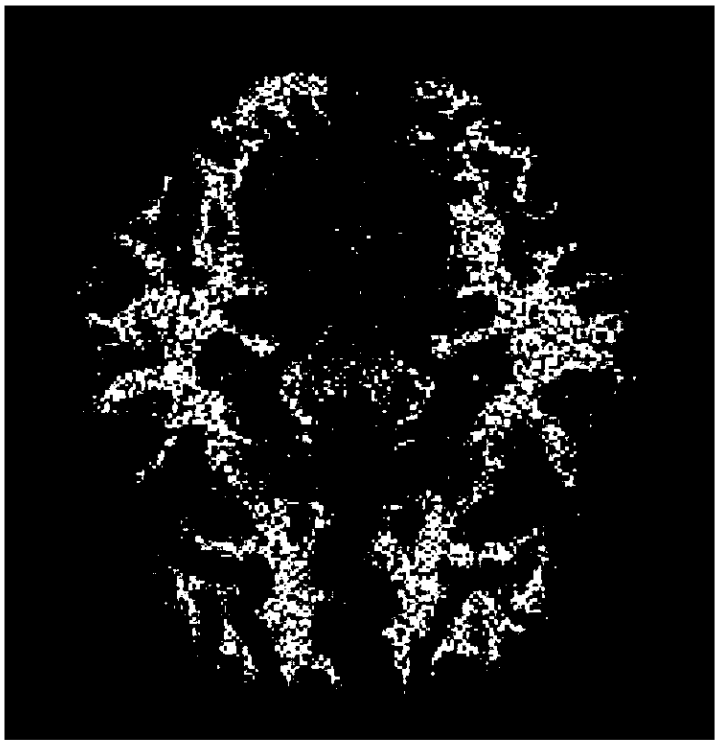

Figure 24: Segmentation mask using RGB values prior to noise filtering

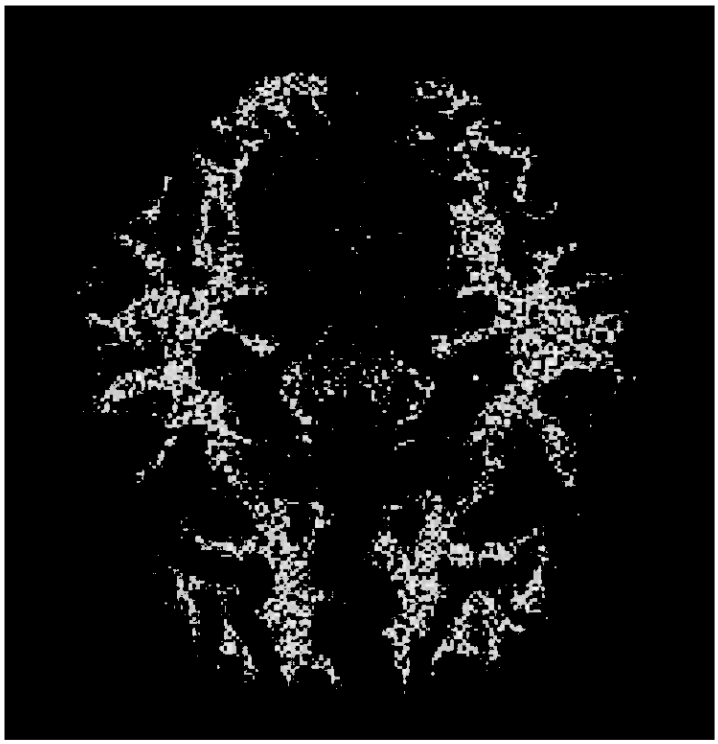

Figure 25: Segmentation using RGB values prior to noise filtering

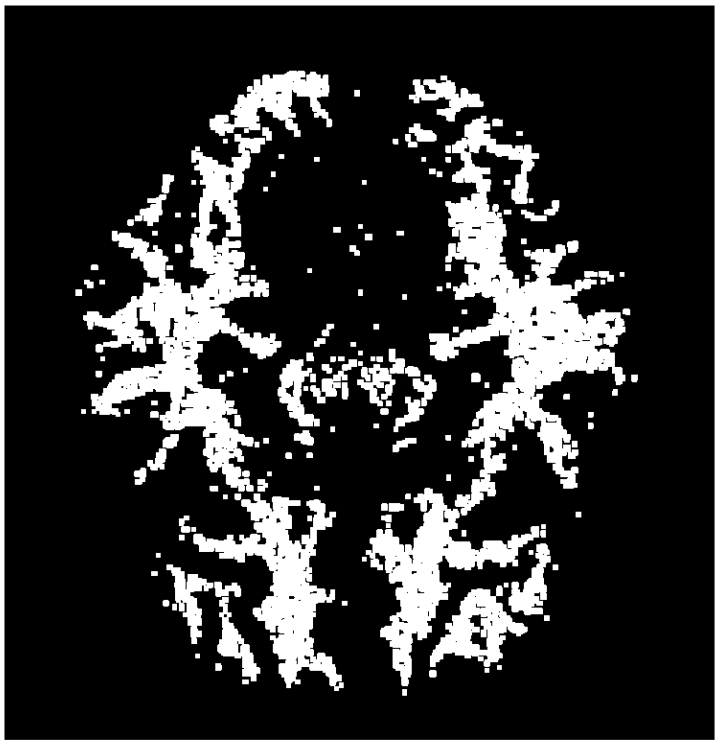

Figure 26: Segmentation mask using RGB values post noise filtering

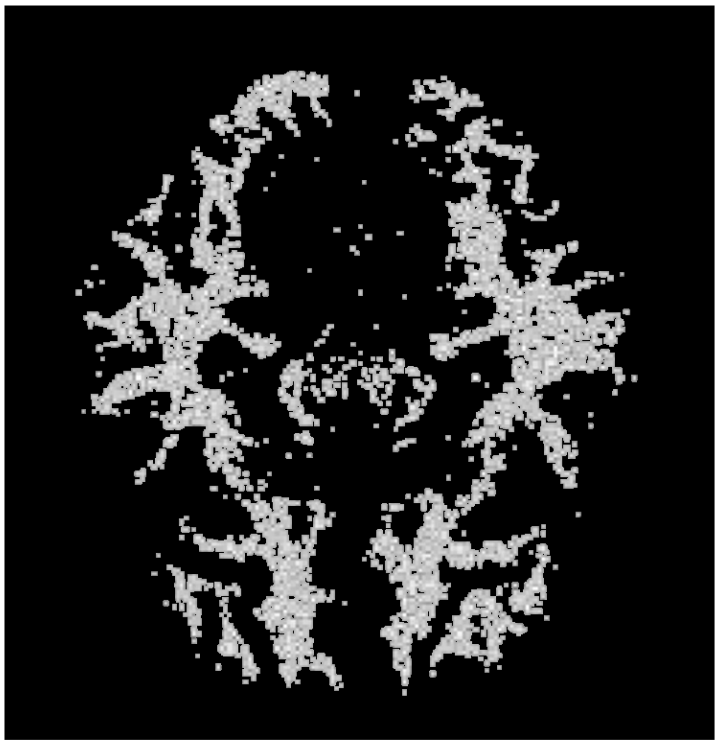

Figure 27: Segmentation using RGB values post noise filtering

Figure 28: Contour based on RGB Segmentation

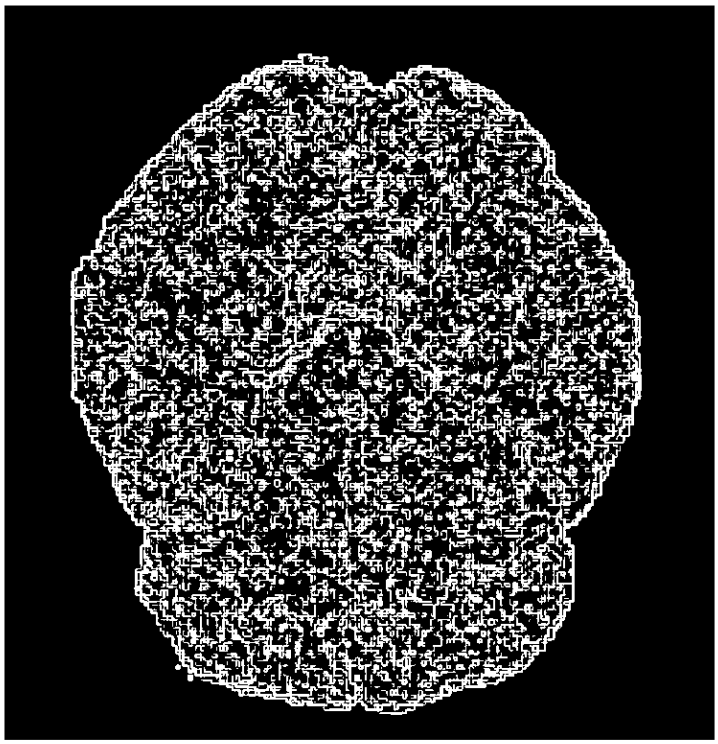

Figure 29: Segmentation mask using texture values prior to noise filtering

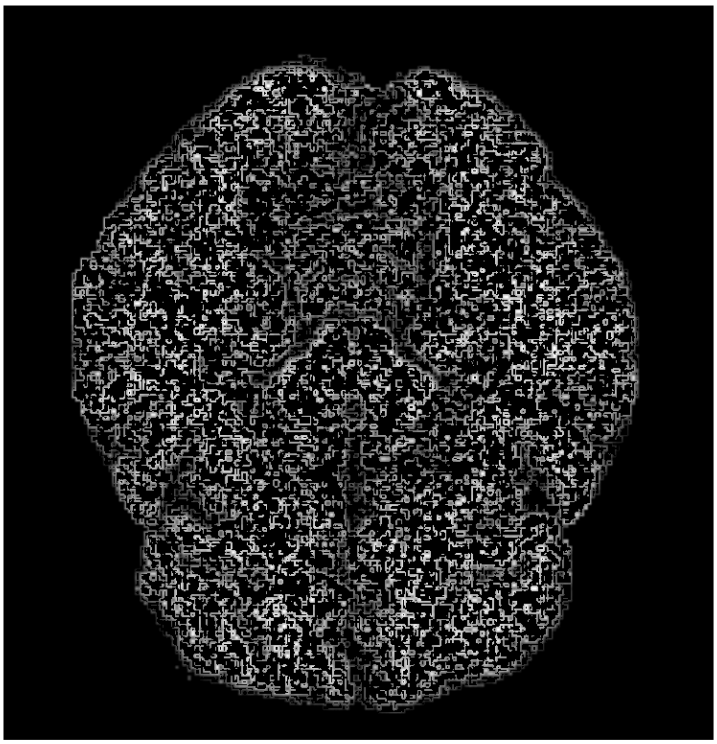

Figure 30: Segmentation using texture values prior to noise filtering

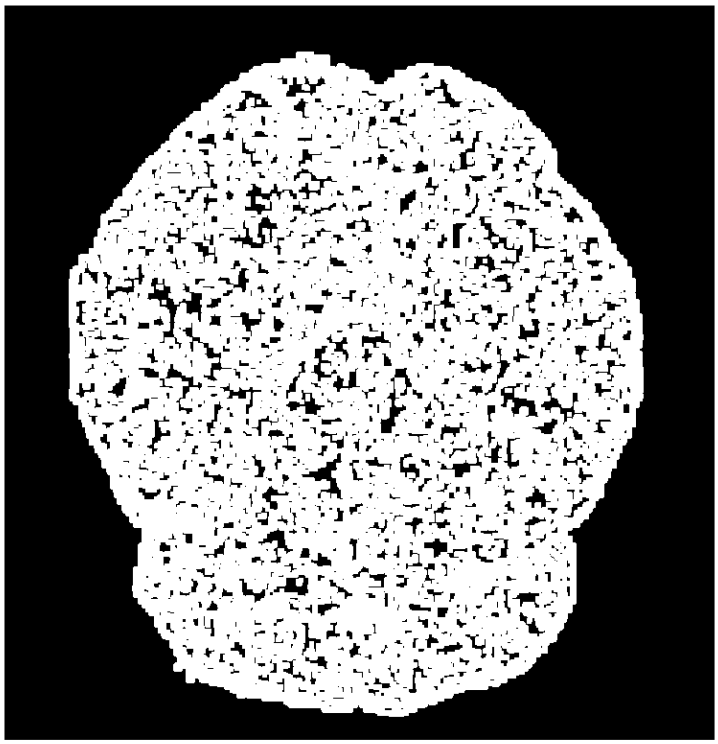

Figure 31: Segmentation mask using texture values post noise filtering

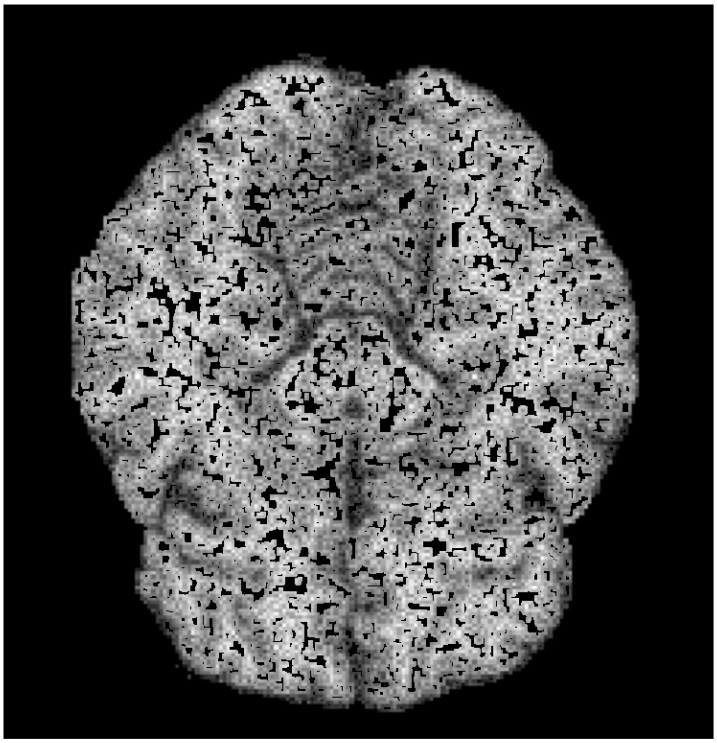

Figure 32: Segmentation using texture values post noise filtering

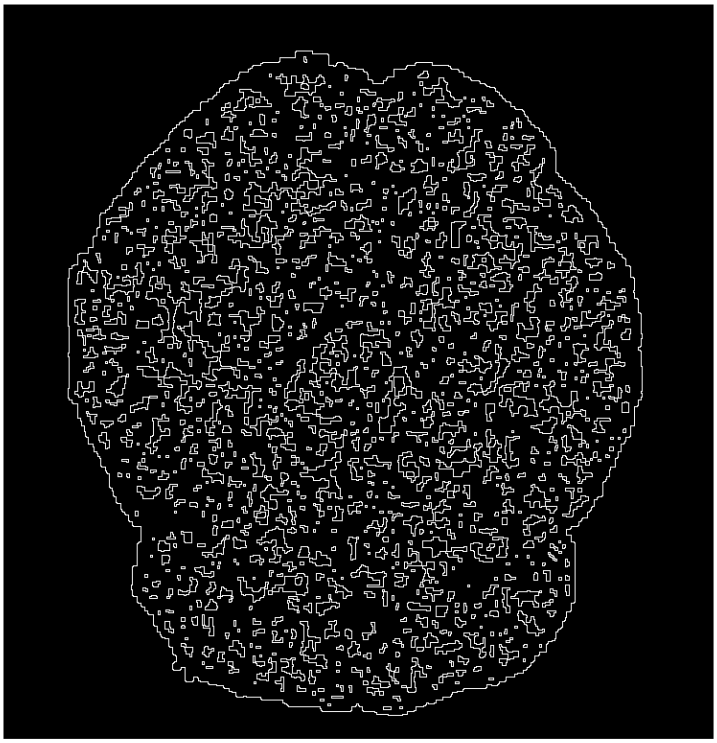

Figure 33: Contour based on texture Segmentation

# 8 Source Code

## 8.1 Main Function

```
clc
clear

cd HW6Pics
% I = imread('lake.jpg');
I = imread('leopard.jpg');
% I = imread('brain.jpg');
cd ..

I_gray = double(rgb2gray(I));
I = double(I);
I1 = I(:,:,1);
I2 = I(:,:,2);
```

```
I3 = I(:,:,3);

[a,b,~] = size(I);

% Masks for different color channels
maskR = zeros(a,b,3);
maskR(:,:,1) = ones(a,b);
maskG = zeros(a,b,3);
maskG(:,:,2) = ones(a,b);
maskB = zeros(a,b,3);
maskB(:,:,3) = ones(a,b);

%% Pixel Intensity based segmentation

% For lake image
% N_iterR = 4;     %Number of Iterations Otsu's algorithm needs to run
% N_iterG = 4;
% N_iterB = 1;

% For leopard image
N_iterR = 1;     %Number of Iterations Otsu's algorithm needs to run
N_iterG = 1;
N_iterB = 1;

% For brain image
% N_iterR = 3;     %Number of Iterations Otsu's algorithm needs to run
% N_iterG = 3;
% N_iterB = 3;

for i = 1:N_iterR
    Seg1 = Otsu_Seg(I1);
    I1 = I1.*Seg1;
end

for i = 1:N_iterG
    Seg2 = Otsu_Seg(I2);
    I2 = I2.*Seg2;
end

for i = 1:N_iterB
    Seg3 = Otsu_Seg(I3);
    I3 = I3.*Seg3;
end

% Forming the complete Mask

% For lake image
% Seg = (1-Seg1).*(1-Seg2).*(Seg3);
% For leopard and brain image
Seg = (Seg1).*(Seg2).*(Seg3);

% Display Images
figure, imshow(I.*maskR/255);
figure, imshow(I.*maskG/255);
figure, imshow(I.*maskB/255);
figure, imshow(Seg);
figure, imshow(I.*repmat(Seg,[1 1 3])/255);
```

```
% Filtering the noise out of the segmentation
Seg = Filter_Seg(Seg);
figure, imshow(Seg);
figure, imshow(I.*repmat(Seg,[1 1 3])/255);

% Extracting the contour from the segmentation
C = Contour(Seg);
figure, imshow(C);

%% Generating texture images
I4 = Texture(I_gray,3);
I5 = Texture(I_gray,5);
I6 = Texture(I_gray,7);

% Texture Based Segmentation using Otsu
Seg4 = Otsu_Seg(I4);
Seg5 = Otsu_Seg(I5);
Seg6 = Otsu_Seg(I6);

% Forming the complete Mask
% For lake image
% Seg2 = (1-Seg4).*(1-Seg5).*(1-Seg6);
% For leopard and brain image
Seg2 = (Seg4).*(Seg5).*(Seg6);

% Display images
figure, imshow(I4/255);
figure, imshow(I5/255);
figure, imshow(I6/255);
figure, imshow(Seg2);
figure, imshow(I.*repmat(Seg2,[1 1 3])/255);

% Filtering the noise out of the segmentation
Seg2 = Filter_Seg(Seg2);
figure, imshow(Seg2);
figure, imshow(I.*repmat(Seg2,[1 1 3])/255);

% Extracting the contour from the segmentation
C2 = Contour(Seg2);
figure, imshow(C2);
```

## 8.2 Otsu Segmentation Function

```
function [ Seg ] = Otsu_Seg( I )
%Function that uses Otsu's algorithm to segment the given grayscale image I
[a,b] = size(I);

%Otsu Algorithm
% Parameter and Variable Initializations
N = a*b;
K = 256;
level = 0:K-1;
p = zeros(1,K);
n = zeros(1,K);
mask0 = zeros(1,K);
```

```
mask1 = ones(1,K);
varB = zeros(1,K);

% Calculate the pi and ni for every intensity level i
for i = 1:K
    Pixels = (I==i);
    n(i) = sum(Pixels(:));
    p(i) = n(i)/N;
end

% Calculate the between class variances for each choice of threshold K
for k = 1:K
    mask0(k) = 1;
    mask1(k) = 0;
    w0 = sum(mask0.*p);
    w1 = sum(mask1.*p);
    mu0 = sum(mask0.*level.*p)/w0;
    mu1 = sum(mask1.*level.*p)/w1;
    varB(k) = w0*w1*((mu0 - mu1)^2);
end

% Find best threshold that maximizes the between class variance
k_opt = find(varB == max(varB),1);

% Noisy Segmentation
Seg = (I>(k_opt-1));

end
```

## 8.3 Segmentation Filtering Function

```
function [ Seg ] = Filter_Seg( Seg )
%Function to filter the noise from the generated segmentation

% Filtering the noise in the segmentation
% Background noise reduction (only for lake image)
% SE = strel('square',30);
% Seg = imerode(Seg,SE);
% Seg = imdilate(Seg,SE);

% Foreground noise reduction
SE = strel('ball',10,10);
Seg = imdilate(Seg,SE);
Seg = imerode(Seg,SE);

% Making Seg a logical image again
Seg = (Seg>0);

end
```

## 8.4 Contour Extraction Function

```
function [ C ] = Contour( I )
%Function to detect contours in the segmented image I
```

```
[m,n] = size(I);
C = zeros(m,n);

N = 3;
fact = floor((N-1)/2);
I = padarray(I,[(N-1)/2 (N-1)/2]);

for i = 1:m
    for j = 1:n
        if(I(fact+i,fact+j) == 0)
            C(i,j) = 0;
        else
            patch = I(fact+(i-fact):fact+(i+fact),...
                fact+(j-fact):fact+(j+fact));
            if(sum(patch(:)) ~= N*N)
                C(i,j) = 1;
            end
        end
    end
end

end
```

## 8.5 Texture Image Formation Function

```
function [ T ] = Texture( I,N )
%Function to generate the texture image from the gray scale image I and for
%a window size N
[m,n] = size(I);
T = zeros(m,n);

fact = (N-1)/2;
I = padarray(I,[(N-1)/2 (N-1)/2]);

for i = 1:m
    for j = 1:n
        patch = I(fact+(i-fact):fact+(i+fact),fact+(j-fact):fact+(j+fact));
        T(i,j) = var(patch(:));
    end
end

end
```