ECE661 Computer Vision HW 6

Rih-Teng Wu

Email: wu952@purdue.edu

1. Introduction

In this homework, we apply Otsu's method to perform image segmentation. Otsu's method aims to find the optimal gray level that can best separate the foreground and the background of the image. The RGB segmentation is achieved by applying the Otsu's algorithm to the three RGB color channel separately, and then combine the segmentation results to get the final image segmentation. The texture-based segmentation is achieved by first computing the texture features with three different windows, apply the Otsu's algorithm to the three texture features separately, and then combine the segmentation results to get the final image segmentation. After the segmentation is done, we use contour extraction techniques to get the boundary of the foreground.

2. Otsu's Method

The procedure of the Otsu's method is described as follows. Given a gray-scale image, we can find the optimal gray level that best separate the foreground and the background by the following steps:

Step 1: Calculate the pixel intensity histogram of the image. For gray levels in [1, 256], compute the PDF of the gray levels:

$$p_i = \frac{h_i}{N}$$

Where h_i is the pixel intensity for gray level i, N is the total number of pixels, and p_i is the probability density for gray level i.

Step 2: Given a gray level threshold k, class 0 is defined as the pixels with gray level less or equal to k (background), and class 1 is defined as the pixels with gray level greater than k (foreground). Calculate the probability of class 0, ω_0, and the probability of class 1, ω_1:

$$\omega_0 = \sum_{i=1}^{k} h_i \text{, and } \omega_1 = 1 - \omega_0$$

Step 3: Calculate the mean for both the two classes:

$$\mu_0 = \frac{1}{\omega_0}\sum_{i=1}^{k} ip_i \text{, and } \mu_1 = \frac{1}{\omega_1}\sum_{i=k+1}^{256} ip_i$$

Step 4: Calculate the between-class variance, σ_b^2 :

$$\sigma_b^2 = \omega_0\omega_1(\mu_0 - \mu_1)^2$$

Step 5: Choose the threshold k that maximize the between-class variance σ_b^2.

Step 6: Construct a mask whose pixel value is 1 if the corresponding pixel in the gray-scale image is greater than the threshold k, and 0 otherwise.

Step 7: Repeat Step 1 to Step 6 for several iterations. To improve the segmentation result, the number of iteration is manually chosen depending on the original image.

3. RGB Image Segmentation

The image segmentation can be achieved by applying Otsu's method to the three RGB color channels separately, and then combine the segmentation results to get final segmentation of the image. The procedure is described as follows.

Step 1: Separate the three RGB color channels and convert them into three gray-scale images.

Step 2: Construct the mask for each channel using Otsu's method as described in Section 2.

Step 3: Combine the three masks by logical operator AND. To get a better segmentation result, the combination logic is chosen depending on the image. For the lake image, the blue mask is chosen as the foreground while the other two channel are chosen as the background, since the lake is mostly blue. Therefore, the overall mask for the lake image is:

$$mask = mask_b \ \& \ (\neg mask_r) \ \& \ (\neg mask_g)$$

Where $mask_b, mask_r$, and $mask_g$, are the masks obtained from the blue, red, and the green channel, respectively.

For the leopard image and the brain image, there is no dominant color. Therefore, the overall mask for the two images is:

$$mask = mask_b \ \& \ mask_r \ \& \ mask_g$$

4. Texture-based Image Segmentation

The texture-based segmentation method is similar to what we have done in Section 3. The only difference is that we use three texture feature channels as the input to Otsu's method instead of using the three RGB channels. The procedure is described as follows.

Step 1: Convert the image to gray-scale image.

Step 2: Generate a new gray-scale image whose pixel value is the variance of the gray-scale values in a $N \times N$ window around the corresponding pixels of the original gray-scale image.

Step 3: Do Step 1 to Step 2 for three different window sizes $N = 3, N = 5$, and $N = 7$. These three gray-scale images are considered to be the texture features of the original image.

Step 4: Treat the three texture features as the three channels of the original image, apply Otsu's method separately to get the three masks.

Step 5: Combine the three masks by logical operator AND. To get a better segmentation result, the combination logic is chosen depending on the image. It should be noted that the values in the texture feature represent the variance. Therefore, for the lake image, we select class 0 as the foreground. Also, we select the mask from $N = 7$ window as the overall mask for the lake image.

For the leopard image and the brain image, the overall mask is chosen as the combination of all the three masks:

$$mask = mask_3 \ \& \ mask_5 \ \& \ mask_7$$

Where $mask_3$, $mask_5$, and $mask_7$, are the masks obtained from the $N = 3$, $N = 5$, and the $N = 7$ window, respectively.

5. Contour Extraction

After the segmentation is done, the contour can be extract for better visualization. My contour extraction algorithm is implemented based on 8-neighbors. The foreground corresponds to the pixel values equal to 1 in the overall mask, while the background corresponds to the pixel values equal to 0 in the overall mask. For each pixel in the overall mask:

a. If the pixel value is 0, then it is not selected as the contour point.
b. If the pixel value is 1, and all its 8-neighbors are 1, then it is not selected as the contour point.
c. If the pixel value is 1, and not all of its 8-neighbors are 1, then it is selected as the contour point.

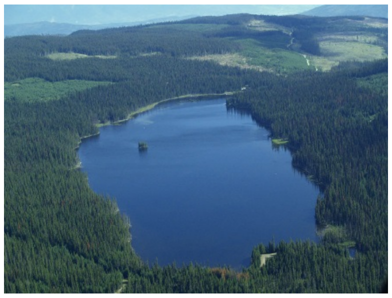
6. Observations

1. The performance of the segmentation results depends highly on the characteristic of the original image. We should select the appropriate segmentation method based on the input image.
2. According to the results, the texture-based method works better in the leopard image, while the RGB segmentation method works better in the other two images. It is reasonable since the texture-based method is suitable when our foreground image contains more textures than the background image. The leopard image is the case. In addition, the color characteristic in the lake image and the gray level difference in the brain image makes them more suitable for applying the RGB segmentation method.
3. Sometimes if we do some preprocessing on the images, the segmentation results can be enhanced. For instance, the RGB segmentation in the lake image, the RGB and texture segmentation in the brain image, are improved by first smoothing the original image with a Gaussian filter, then do the processing.
4. The contour extraction method could also influence the result. Right now my contour extraction algorithm would depict the boundary of those tiny foreground regions. Use other contour extraction algorithms may improve the results.

7. Results

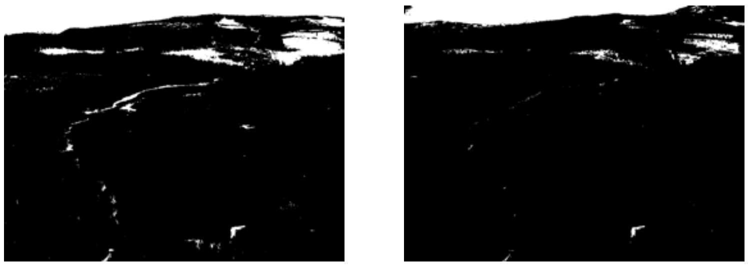
7.1 Lake Image

Original image:

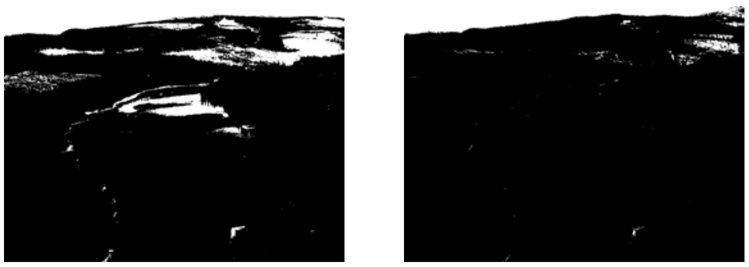


RGB segmentation:

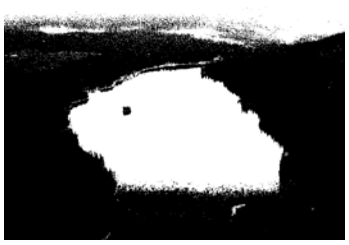
R-channel with 2 iteration of the Otsu’s method:



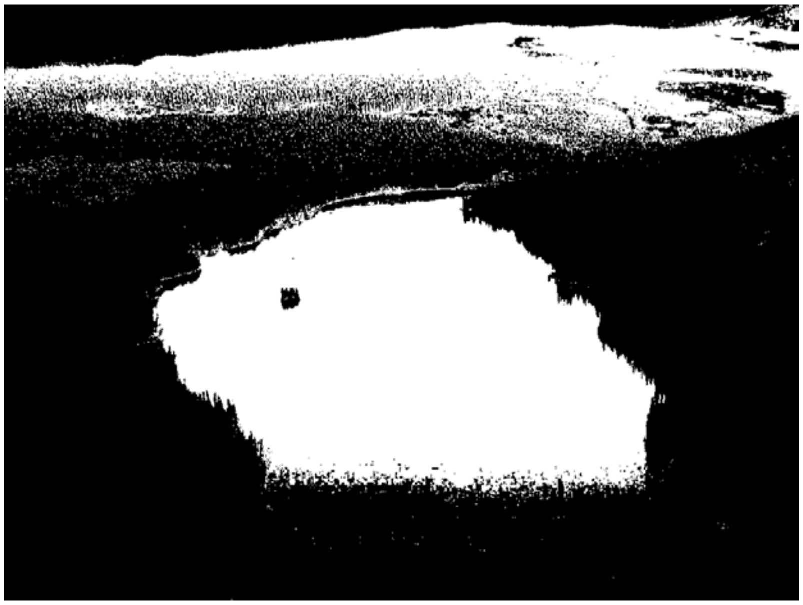
G-channel with 2 iteration of the Otsu’s method:



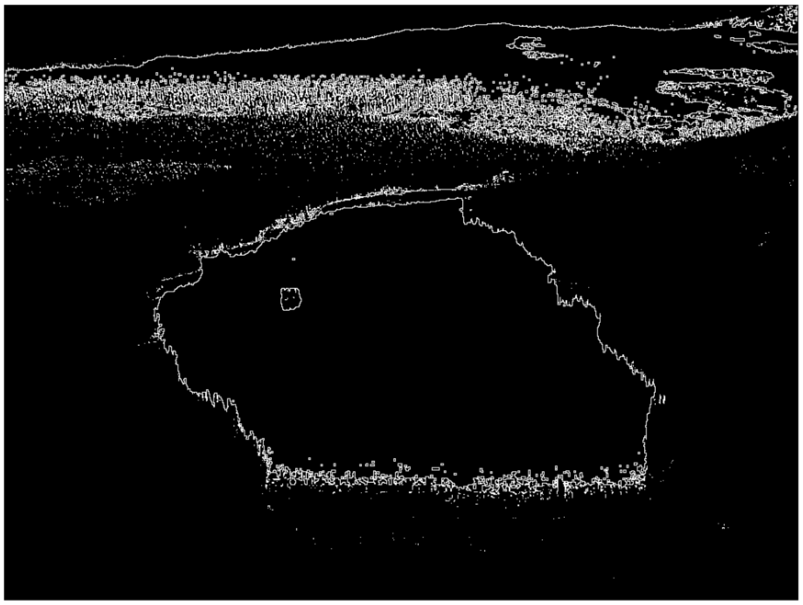
B-channel with 1 iteration of the Otsu’s method:



Overall mask:

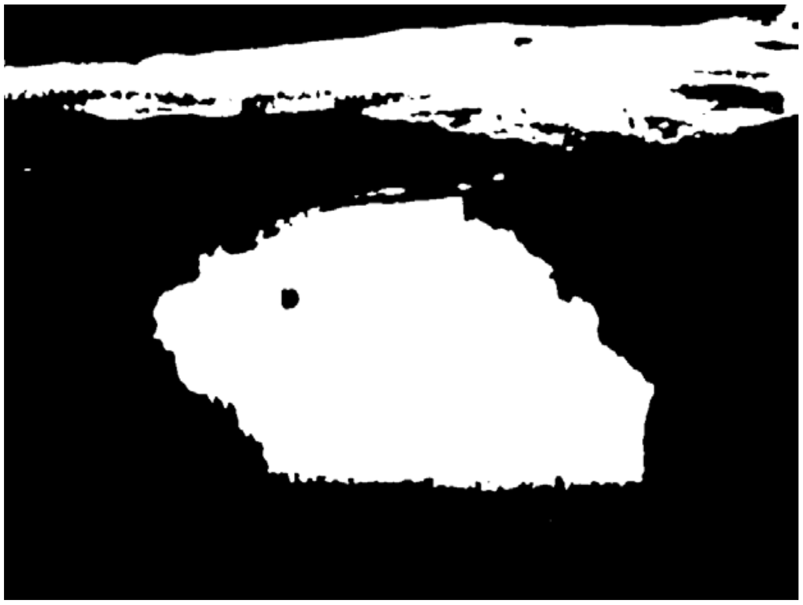


Contour:

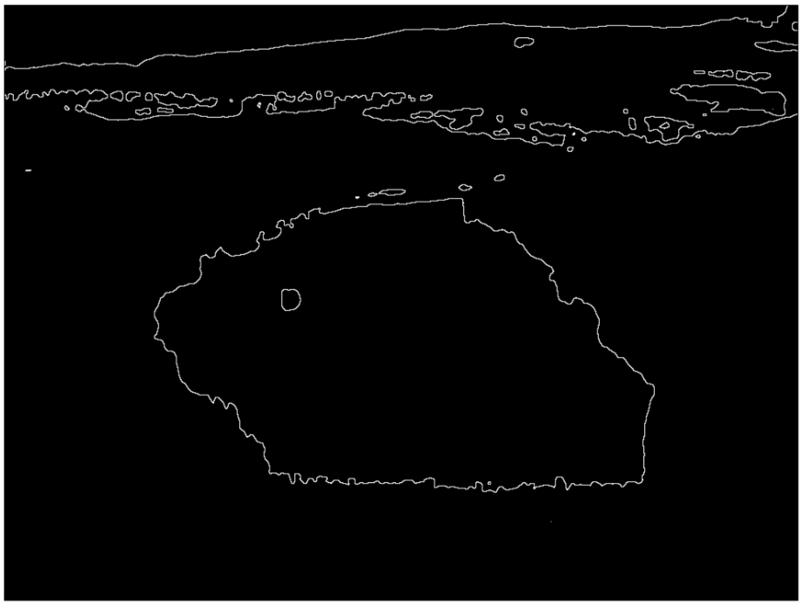


In this case, we can improve our result by smoothing the image with Gaussian low-pass filter of $\sigma = 2.5$, then apply the Otsu’s method.

Overall mask with Gaussian smoothing:



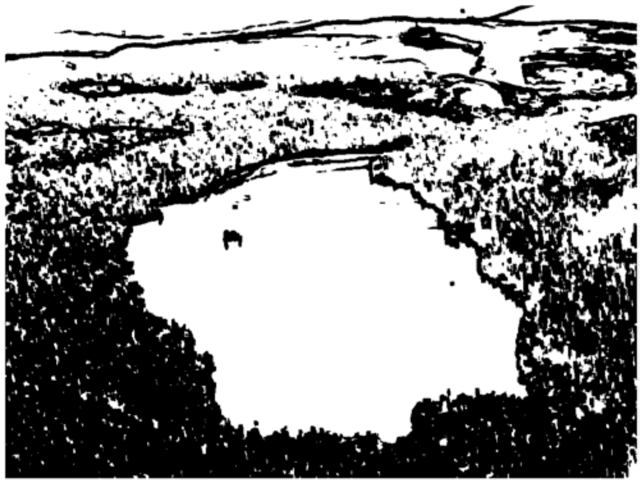
Contour:



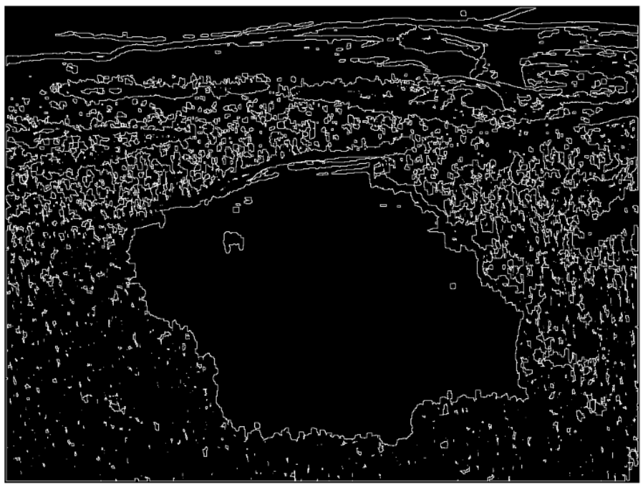
Texture-based segmentation:

N=7 channel with 4 iteration of the Otsu’s method:

Overall mask:



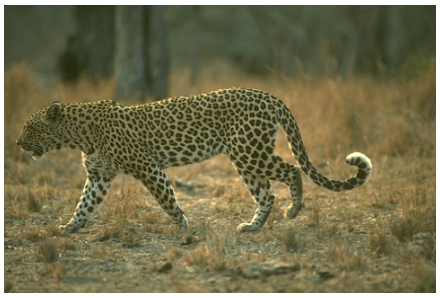
Contour:



In this case, applying Gaussian filter do not improve the result.

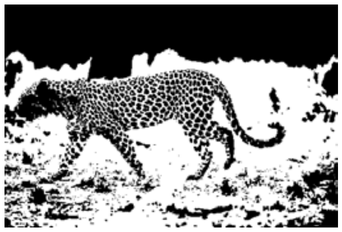
7.2 Leopard Image

Original image:

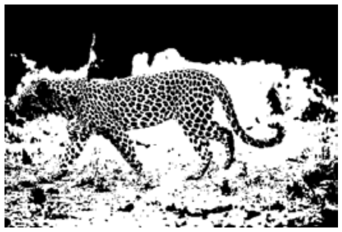


RGB segmentation:

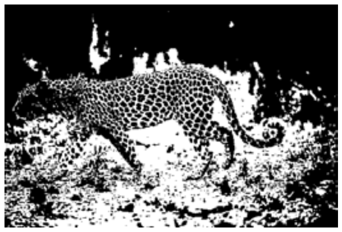
R-channel with 1 iteration of the Otsu's method:

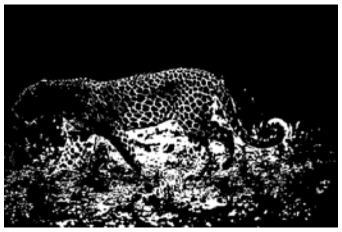


G-channel with 1 iteration of the Otsu's method:

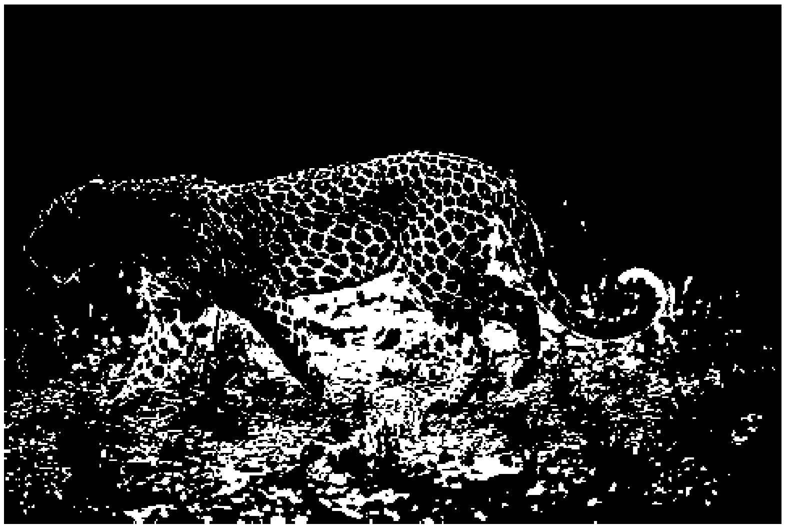


B-channel with 2 iteration of the Otsu's method:

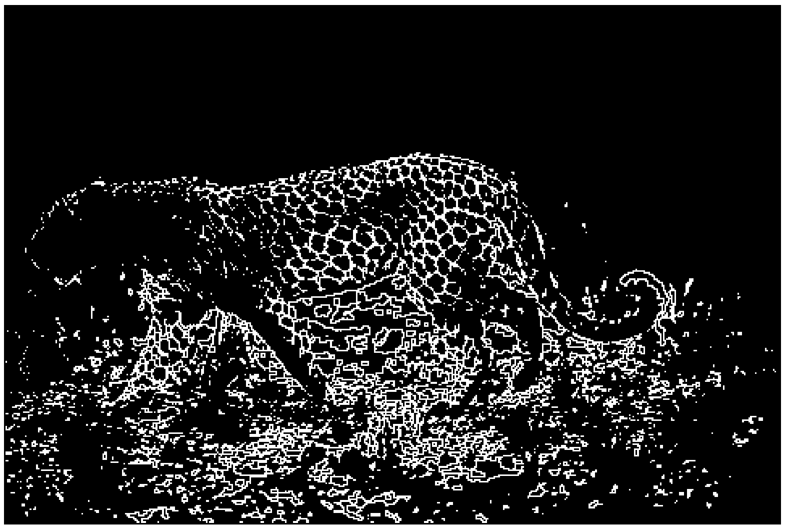




Overall mask:

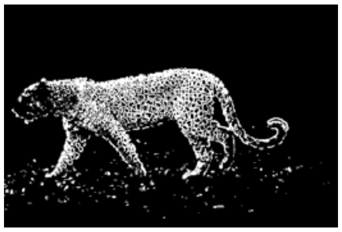


Contour:

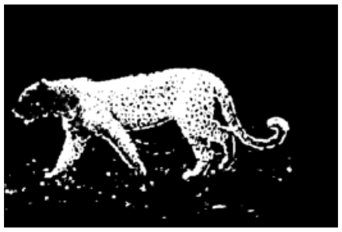


Texture-based segmentation:

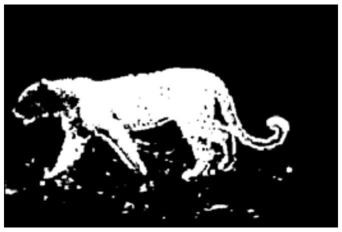
N=3 channel with 1 iteration of the Otsu’s method:



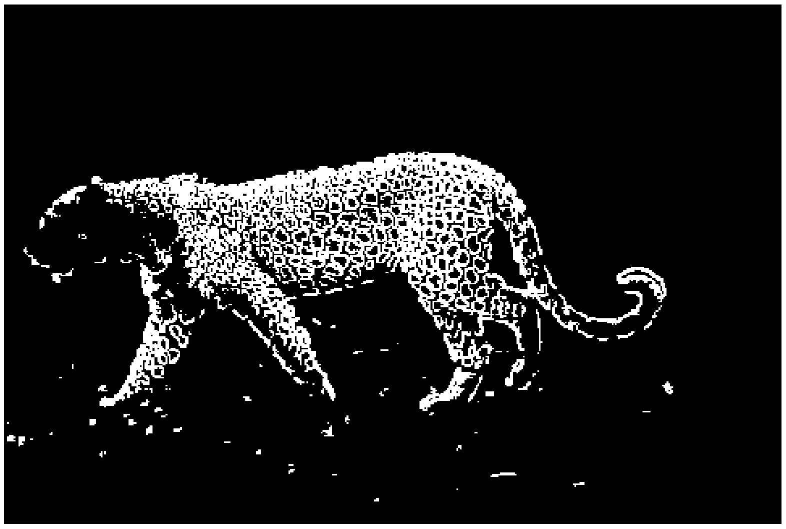
N=5 channel with 1 iteration of the Otsu’s method:



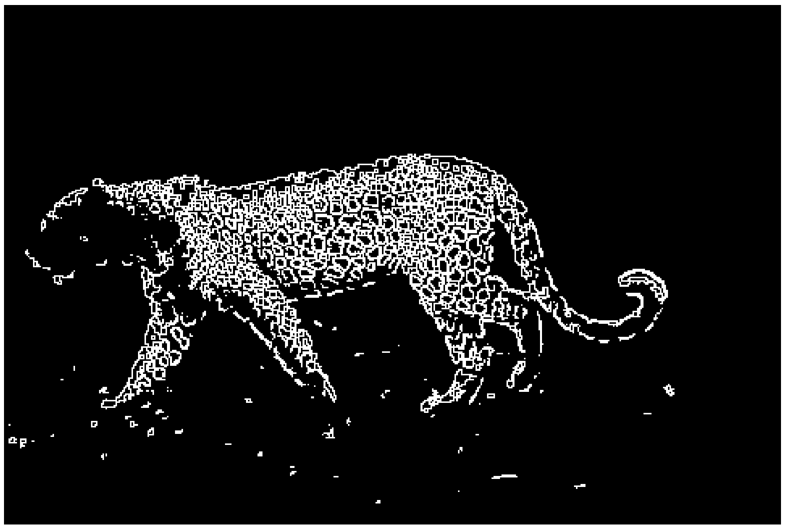
N=7 channel with 1 iteration of the Otsu’s method:



Overall mask:

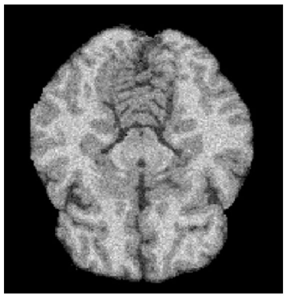


Contour:



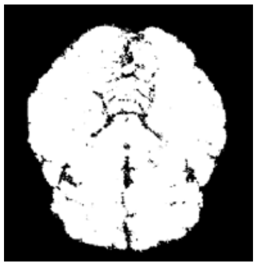
7.3 Brain Image

Original image:

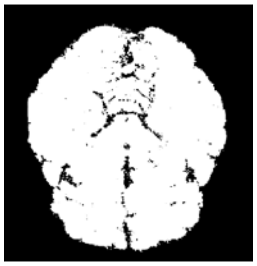


RGB segmentation:

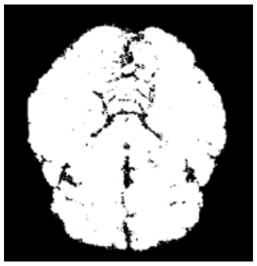
R-channel with 1 iteration of the Otsu’s method:



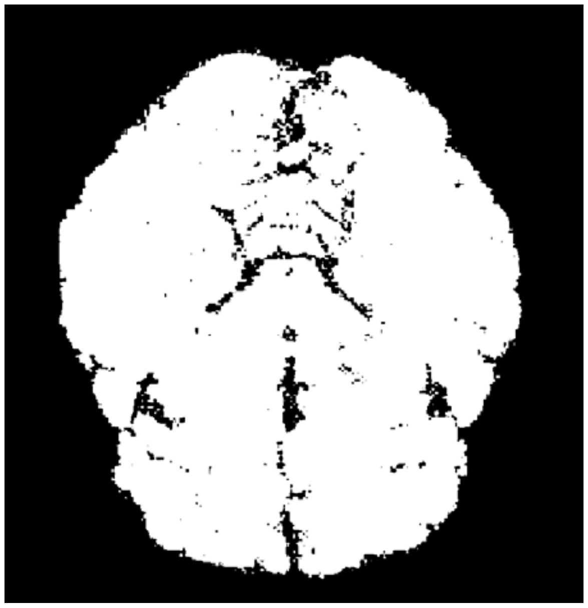
G-channel with 1 iteration of the Otsu’s method:



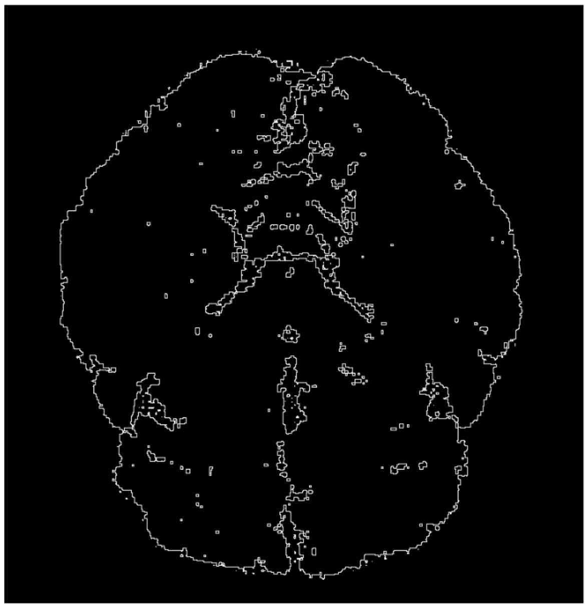
B-channel with 1 iteration of the Otsu’s method:



Overall mask:

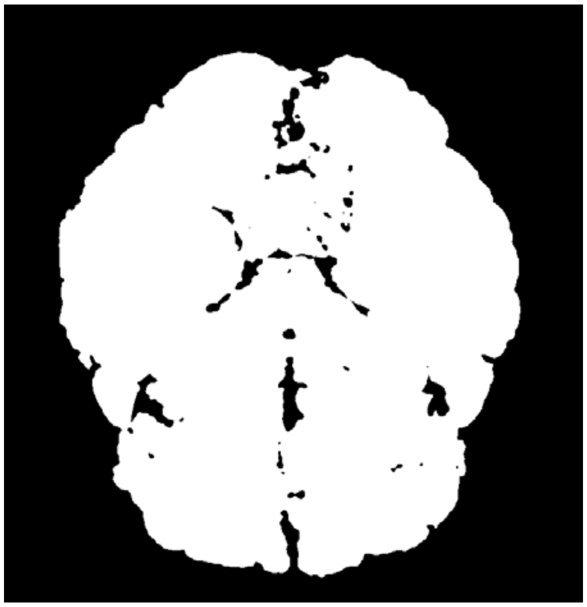


Contour:

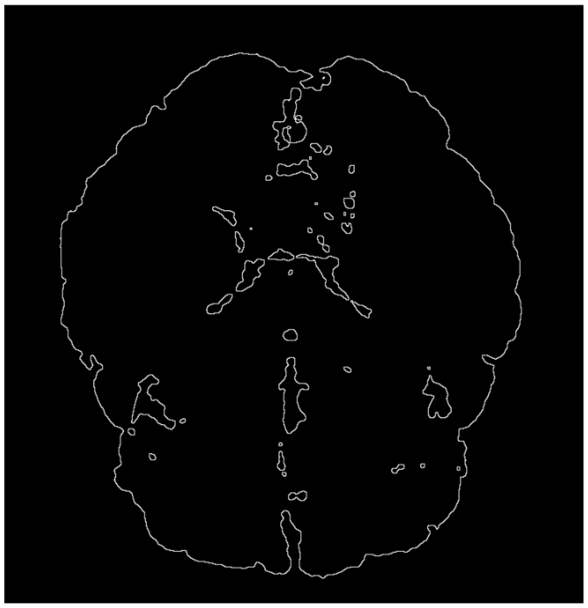


In this case, we can improve our result by smoothing the image with Gaussian low-pass filter of $\sigma = 2.5$, then apply the Otsu's method.

Overall mask with Gaussian smoothing:

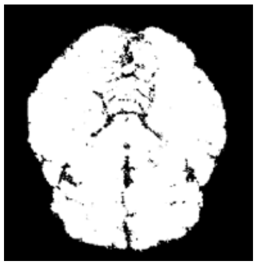
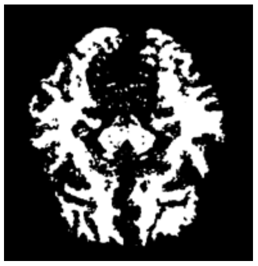


Contour:

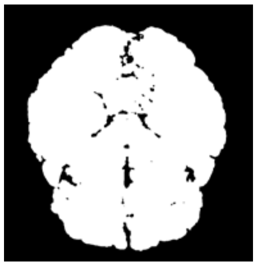


According to the original brain image, if we want to differentiate between the relatively white region and the gray region, we can use different iteration numbers for Otsu's method. To improve the result, we also smooth the image with Gaussian low-pass filter of $\sigma = 2.5$.

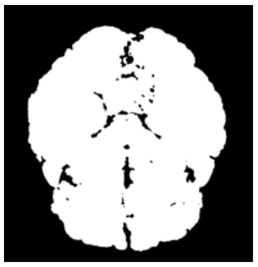
R-channel with 2 iteration of the Otsu's method:

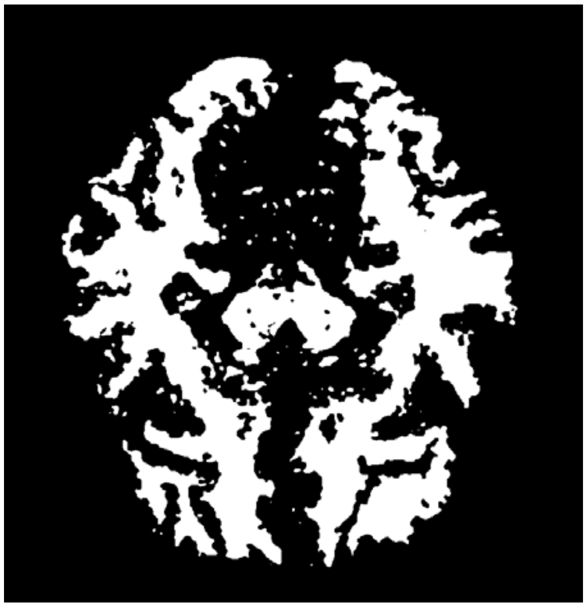
G-channel with 1 iteration of the Otsu's method:



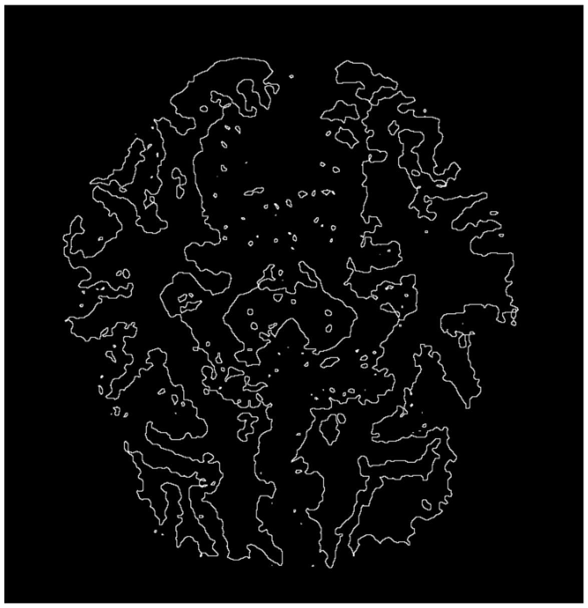
B-channel with 1 iteration of the Otsu's method:



Overall mask with Gaussian smoothing:

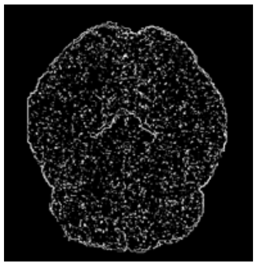


Contour:

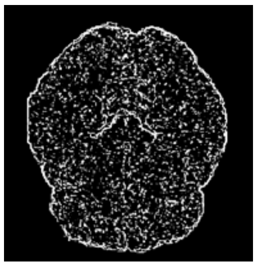


Texture-based segmentation:

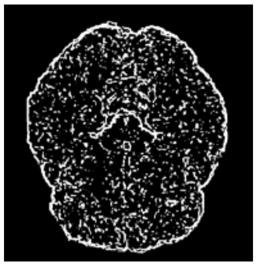
N=3 channel with 1 iteration of the Otsu’s method:



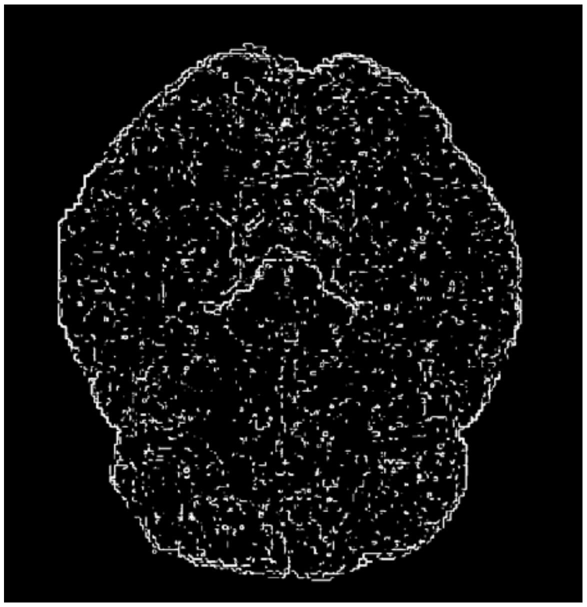
N=5 channel with 1 iteration of the Otsu’s method:



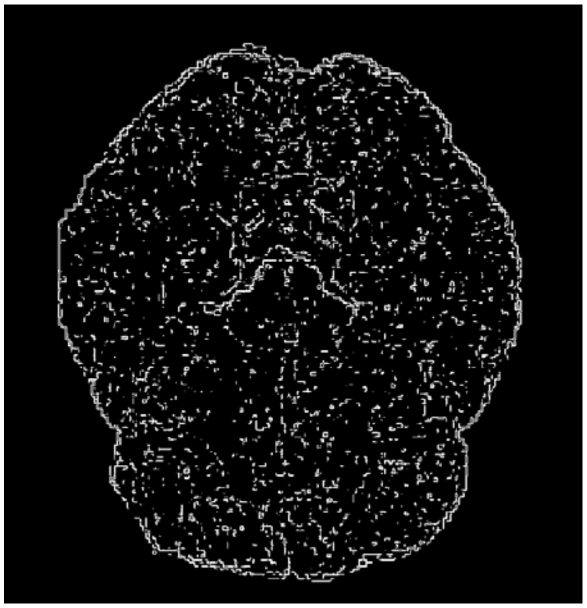
N=7 channel with 1 iteration of the Otsu’s method:



Overall mask:

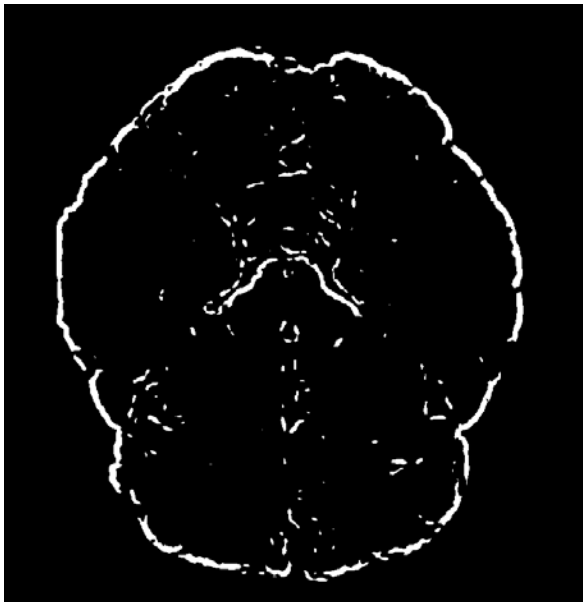


Contour:

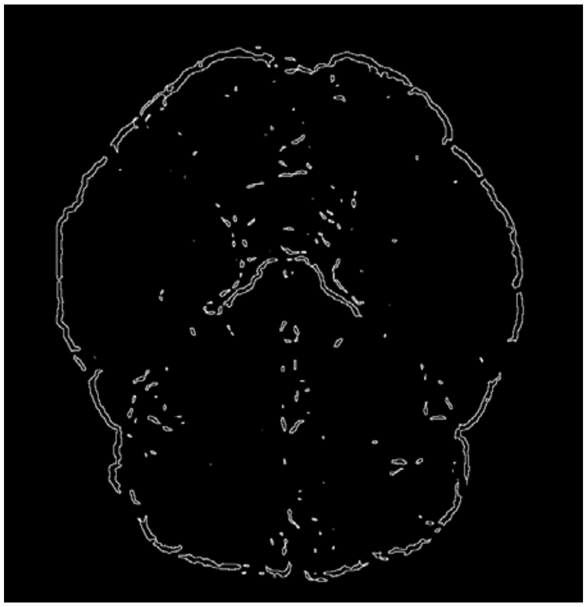


In this case, we can improve our result by smoothing the image with Gaussian low-pass filter of $\sigma = 2.5$, then apply the Otsu's method.

Overall mask with Gaussian smoothing:



Contour:



Code: hw6_RihTengWu.m

```
% ECE 661 HW 6 - Otsu's method, Texture based segmentation, Contour
extraction
% Student: Rih-Teng Wu

clc
clear all
close all

% ==== load images for different cases
Case = 3;  % 1: lake; 2: leopard, 3: brain
switch Case
    case 1
        image = imread('lake.jpg');
        % ==== try gaussian lowpass filter,
        % ==== for texture-based, do not filter the image
        image = imgaussfilt(image,2.5);
    case 2
        image = imread('leopard.jpg');
    case 3
        image = imread('brain.jpg');
        image = imgaussfilt(image,2.5); % try gaussian lowpass filter
end
figure, imshow(image);

% ==== Show RGB images
image_black = zeros(size(image,1),size(image,2));
image_r = cat(3,image(:,:,1),image_black,image_black); % red channel
figure, imshow(image_r);
image_g = cat(3,image_black,image(:,:,2),image_black); % green channel
figure, imshow(image_g);
image_b = cat(3,image_black,image_black,image(:,:,3)); % blue channel
figure, imshow(image_b);

% ==== RGB images segmentation using Otsu
switch Case
    case 1
        N_iter_r = 2; N_iter_g = 2; N_iter_b = 1; % different iterations for
RGB channels, 2, 2, 1
    case 2
        N_iter_r = 1; N_iter_g = 1; N_iter_b = 2; % 1,1,2
    case 3
        N_iter_r = 2; N_iter_g = 1; N_iter_b = 1; % 2,1,1; 1,1,1
end
mask_r = Otsu_RGB(image_r,N_iter_r); % Perform Otsu separately for each
channel
mask_g = Otsu_RGB(image_g,N_iter_g);
mask_b = Otsu_RGB(image_b,N_iter_b);

figure, imshow(mask_r);
figure, imshow(mask_g);
figure, imshow(mask_b);

% ==== apply AND operator to get overall mask
```

```
switch Case
    case 1
        mask_overall = mask_b & ~mask_r & ~mask_g; % for the lake image
    case 2
        mask_overall = mask_b & mask_r & mask_g; % for the leopard image
    case 3
        mask_overall = mask_b & mask_r & mask_g; % for the brain image
end
figure, imshow(mask_overall);

% ==== perform texture segmentation
texture_3 = text_seg(image,3);
texture_5 = text_seg(image,5);
texture_7 = text_seg(image,7);

image_3 = cat(3,texture_3,image_black,image_black);
image_5 = cat(3,image_black,texture_5,image_black);
image_7 = cat(3,image_black,image_black,texture_7);

switch Case
    case 1
        % ==== for lake, consider small variance as the foreground image
        image_3 = uint8(255*ones(size(image,1),size(image,2),3)) - image_3;
        image_5 = uint8(255*ones(size(image,1),size(image,2),3)) - image_5;
        image_7 = uint8(255*ones(size(image,1),size(image,2),3)) - image_7;
        N_iter_3 = 1; N_iter_5 = 2; N_iter_7 = 4; % 1,2,4
    case 2
        N_iter_3 = 1; N_iter_5 = 1; N_iter_7 = 1;
    case 3
        N_iter_3 = 1; N_iter_5 = 1; N_iter_7 = 1;
end

mask_3 = Otsu_RGB(image_3,N_iter_3);
mask_5 = Otsu_RGB(image_5,N_iter_5);
mask_7 = Otsu_RGB(image_7,N_iter_7);

figure, imshow(mask_3);
figure, imshow(mask_5);
figure, imshow(mask_7);

% ==== apply AND operator to get overall mask
switch Case
    case 1
%           mask_overall_text = mask_3 & mask_5 & mask_7; % for the lake
image
          mask_overall_text = mask_7; % for the lake image
    case 2
        mask_overall_text = mask_3 & mask_5 & mask_7; % for the leopard image
    case 3
        mask_overall_text = mask_3 & mask_5 & mask_7; % for the brain image
end
figure, imshow(mask_overall_text);

% ==== Contour extraction
contour_RGB = ContourExtraction(mask_overall);
```

```
contour_text = ContourExtraction(mask_overall_text);

figure, imshow(contour_RGB);
figure, imshow(contour_text);
```

Code: Otsu_RGB.m

```
function [ mask ] = Otsu_RGB( image, N )
% This function separate the foreground and the background of the
% input image based on Otsu's method
% Author: Rih-Teng Wu
% mask: 1 for foreground; 0 for background
% image: the red, green, or blue image of the original image (uint8 format)
% N: number of iteration for Otsu's method

% ==== convert to double and grayscale
image = rgb2gray(image);
image = double(image);
image_shift = image + 1; % shift [0 255] to [1 256]

% ==== calculate image intensity histogram
L = 256; % Gray level from 0 to L-1 (0~255)
[h, w] = size(image); % the height and width of the image
p_total = h*w;

% ==== calculate pdf of pixel intensity
pixel_count = zeros(L,1);
pixel_pdf = zeros(L,1); % initial the pdf of pixel intensity

for i = 1:L
    pixel_count(i) = length(find(image_shift(:) == i));
    pixel_pdf(i) = pixel_count(i)/p_total;
end

% ==== Otsu's method
thresh = 1; % initial optimum gray level threshold in [1 256]

for i = 1:N
    sigma_b.thresh = []; % initial sigma_b (between-class variance)
    sigma_b.value = [];
    count = 1;

    for j = thresh:L
        % ==== apply Otsu only for gray level in [thresh,L-1]
        % ==== for the next interation, apply Otsu in the foreground
        % ==== threshold will be updated

        % ==== calculate W0 and W1, mu_0, mu_1, and between-class variance
        W0 = sum(pixel_pdf(thresh:j,1));
        W1 = sum(pixel_pdf(j+1:L,1));

        mu_0 = sum([thresh:j]'.*pixel_pdf(thresh:j,1))/W0;
        mu_1 = sum([j+1:L]'.*pixel_pdf(j+1:L,1))/W1;

        sigma_b.thresh(count) = j;
        sigma_b.value(count) = W0*W1*(mu_0-mu_1)^2;
        count = count + 1;
    end
    % ==== update threshold
    sigma_max = max([sigma_b.value]);
    max_idx = find(sigma_b.value == sigma_max);
    thresh = sigma_b.thresh(max_idx);

    % ==== update histogram
    [row col] = find(image_shift >= thresh);
    p_total = length(row);
    pixel_count = zeros(L,1);
    pixel_pdf = zeros(L,1); % initial the pdf of pixel intensity

    for k = thresh:L
        pixel_count(k) = length(find(image_shift(:) == k));
        pixel_pdf(k) = pixel_count(k)/p_total;
    end
end

% ==== get mask
thresh = thresh - 1; % convert back to [0 255];
mask = zeros(h,w);   % initial of mask

for i = 1:h
    for j = 1:w
        if image(i,j)> thresh
            mask(i,j) = 1;
        end
    end
end

end
```

Code: text_seg.m

```
function [ texture ] = text_seg( image , N )
% This function output the texture feature by using a N by N window
% The feature corresponds to the variance of the pixels in the window
% Author: Rih-Teng Wu
% texture: the output texture feature
% image: original image with uint8 format
% N: defines the window size, need to be an odd number

% ==== convert to grayscale, then convert to double
image_g = rgb2gray(image);
image_g = double(image_g);

[h w] = size(image_g);
w_half = (N-1)/2; % half of window size,
np = N*N;         % number of pixel in the window

texture = zeros(h,w); % initial output

for i = 1+w_half:h-w_half
    for j = 1+w_half:w-w_half
        img_window = image_g(i-w_half:i+w_half,j-w_half:j+w_half);
        value = img_window(:);
        m_window = mean(value);
        % ==== calculate variance
        texture(i,j) = 1/np*sum((value-m_window).^2);
    end
end

% ==== scale to [0 255], then convert to uint8 format
texture = texture./max(texture(:))*255;
texture = uint8(texture);
end
```

Code: ContourExtraction.m

```
function [ contour ] = ContourExtraction( mask )
% This function output the contour from the mask
% Author: Rih-Teng Wu
% contour: the contour that separate the foreground and the
% background
% mask: the mask obtained from image segmentation

[h w] = size(mask);
contour = zeros(h,w); % initial contour
N = 3;                % 3x3 window, with 8-neighbors
w_half = (N-1)/2;

% ==== only if the center pixel value is 1, and not all its surrounding
pixels
% equals to 1, is considered as a valid contour point

for i = 1+w_half:h-w_half
    for j = 1+w_half:w-w_half
        img_window = mask(i-w_half:i+w_half,j-w_half:j+w_half);
        if img_window(2,2)== 1
            values = img_window(:);
            if sum(values)< N*N
                contour(i,j) = 1;
            end
        end
    end
end

end
```