Homework 6

ECE 661
Aziza Satkhozhina
asatkhoz@purdue.edu

Goals

The goal of this homework is to implement Otsu's binary segmentation and contour extraction algorithm.

Otsu's algorithm

Otsu's segmentation algorithm automatically clusters pixels into two groups: background and foreground. The main idea of Otsu's algorihtm is to find threshold that would maximize between-class variance and minimize within-class variance. Then, all pixels are classified into 2 classes using that threshold.

First step is to create a histogram of pixel values. Since we are taking 8-bit image, there are 256 possible values for pixels. We compute histogram h with 256 bins where the height of each bin corresponds to number of pixels that have that pixel value (from 0 to 255).

Given an image, we can estimate probability of pixel value i by simply dividing the height of bin i in the histogram by the total number of pixels N.

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

Given threshold k, probability of class 0 is sum of probabilities of pixel values smaller than k.

$$w_0 = \sum_{i=1}^{k} h_i \text{ and } w_1 = 1 - w_0$$

Since the goal is to maximize between class variance, we need to calculate the means and the variances of both classes.

$$\mu_1 = \frac{1}{w_0} \sum_{i=1}^{k} i p_i \text{ and } \mu_2 = \frac{1}{w_1} \sum_{i=k+1}^{256} i p_i$$

.

It is sufficient to increase between class variance $\sigma_b^2 = w_0 w_1 (\mu_0 - \mu_1)^2$. This will decrease within-class variance too. I calculated the between-class variance for each threshold k and picked the threshold that maximizes the variance. Depending on the image, sometimes Otsu's algorithm was run several times for better results.

Each of the RGB channels of color image was segmented separately. Then,

I combined results from RGB channels using AND operator, where I set the final segmented image pixel to 1 only if all three segmented channels have 1 at that pixel location. I also had to specify manually which class is foreground and which is background.

Texture segmentation

To perform texture segmentation, I first converted RGB image into grayscale using standard formula: $I = 0.2989R + 0.5870G + 0.1140B$. I then followed a simple approach by calculating the variance of gray scale image in NxN window. I performed texture segmentation for three values of $N = 3, 5, 7$ and combined all three results into one 3-channel image. Then, texture image is fed into Otsu's algorithm the same way as RGB image.

Connected Component Analysis

Connected Component Analysis is another standard approach used in Image Processing. After Otsu thresholding, we are left with a binary mask where pixel value 1 represents foreground and pixel 0 represents background. Next step is to extract the connected components. I used 8 pixel neighborhood to find connected components. Two-pass Connected Component algorithm first labels all connected components with a unique label, and then unites some connected components together. The alogorithm is the following:

Step 1) Set $n = 0$.

Step 2) For pixel $img(x, y)$, define its 4 neighbors as pixels located at the following locations $(x-1, y), (x-1, y-1), (x, y-1), (x+1, y-1)$. If all of its neighboring pixels have value of 0, then increment the value of n by 1 and set $img(x, y)$ to n. Otherwise, find the minimum non-zero value of its neighbors, and set $img(x, y)$ to that value. Record the values of the neighbors into equivalence list.

Step 3) Repeat step 2 with the next pixel.

Step 4) After all pixels were processed and united into connected components, it is necessary to unite connected components using the equivalence list. Do a second pass on the image and unite all connected components.

Step 5) Optional. Re-label all connected components from 1 to the number of the components.

Step 6) Optional. Delete connected components whose size is less than 100 pixels to get rid of noise.

Contour extraction

I implemented Square Tracing algorithm, one of the earliest contour detection algorithms. The pseudocode is below and it is applied to each connected component c_i separately.

Step 1) Set contour list B empty

Step 2) Scan the pixels and find first non-zero pixel of c_i at location px, py. Insert the locations of the pixel into list B.

Step 3) Move to the left adjacent pixel $px - 1, py$.

Step 4) At location x, y if the current pixel value is 1, then insert the location into set B and move to the left. If current pixel value is 0, move to the right.

Step 5) Repeat step 4 until $x, y = px, py$.

Left and right are defined as follows. Initially, if you are at pixel location x, y, "North" is defined as moving to the pixel at location $x, y - 1$, "South" to location $x, y + 1$, "West" to location $x - 1, y$, and "East" to location $x + 1, y$. However, once we start moving, the direction is defined by the previous move. Every move changes the coordinate system, therefore "right", "left" is relative to the previos move. It is summarized below:

If $(prevDirection == \text{“}North''$)
  $left => West$
  $right => East$
elseif$(prevDirection == \text{“}South'')$
  $left => East$
  $right => West$
elseif$(prevDirection == \text{“}West'')$
  $left => South$
  $right => North$
elseif$(prevDirection == \text{“}East'')$
  $left => North$
  $right => South$
end

Results

Otsu's algoruthm applied on the texture features performs much better than RGB channels. In Pic 1. for example, the lake and the trees have approximately same amount of red and green, however lake has much more blue color in it. Therefore Blue color would be suitable for segmentation, while red and green channels make the result more noisy. Texture feature looks at the smoothness

of the region, therefore it is a better feature for extracting objects from an image (especially for the lake). Pic 2 was a difficult image to segment as the tiger has multiple colors, and the colors match his background. Also the texture of the tiger is not uniform, so it is difficult to segment it using texture segmentation as well.

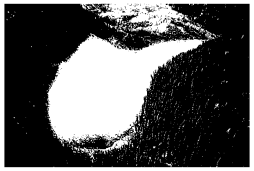
Output images

Figure 1: Original pic. 1

Figure 2: Otsu segmentation of Pic. 1 (RGB channels)

(a) R channel

(b) G channel



(c) B channel

Figure 3: Pic.1 Texture Image

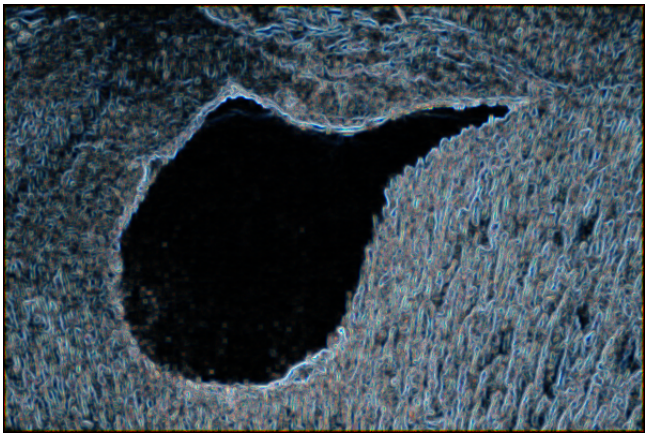


Figure 4: Otsu segmentation of Pic. 1

(a) RGB image

(b) Texture image

Figure 5: Pic.1 Connected Component

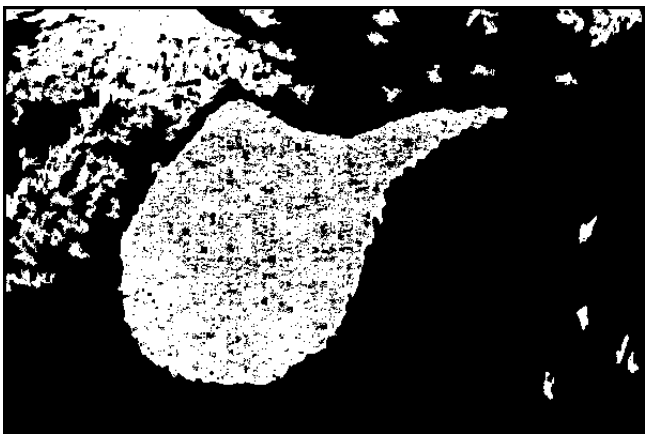


Figure 6: Pic.1 Contour image

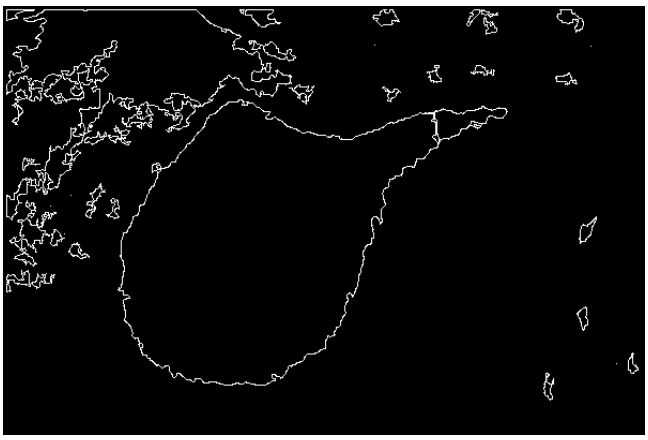


Figure 7: Original pic. 2

Figure 8: Pic.2 Texture Image

Figure 9: Otsu segmentation of Pic. 2

(a) RGB image

(b) Texture image

Figure 10: Pic. 2 Connected Component

Figure 11: Pic. 2 Contour image

Matlab code

```
%*****************************************************************************
%*****************************************************************************

img = imread('pic2.jpg');
str = 'pic2';
s = size(img);

%Otsu applied on RGB channels
otsuRGB = zeros(s);
for ch = 1:3
    otsuRGB(:,:,ch) = otsu(img(:,:,ch),1);
end

%combine three channels to get one BW image
bwotsu = zeros(s(1),s(2));
for i = 1:s(1)
    for j = 1:s(2)
        if(otsuRGB(i,j,1) > 0 && otsuRGB(i,j,2)>0 && otsuRGB(i,j,3)>0)
            bwotsu(i,j) = 1;
        end
    end
end
imwrite(bwotsu,strcat(str,'_otsuRGB.png'));

%convert RGB into grayscale
img_g = rgb2gray(img);
%Texture segmentation applied on gray scale image for three Ns
tim = zeros(s);
for ch = 1:3
    I = text_segm(img_g,1+ch*2);
    tim(:,:,ch) = I/max(max(I))*255;
end
tim = uint8(tim);
imwrite(tim,strcat(str,'_texture.png'));

%apply otsu on the texture image
im = zeros(s);
for ch = 1:3
    im(:,:,ch) = otsu(tim(:,:,ch),1);
end

%combine three texture images into one bw
bw = zeros(s(1),s(2));
for i = 1:s(1)
    for j = 1:s(2)
        if(im(i,j,1) > 0 && im(i,j,2)>0 && im(i,j,3)>0)
            bw(i,j) = 1;
        end
    end
end
imwrite(bw,strcat(str,'_OtsuTexture.png'));
```

```
%apply connect component analysis
[img_con, N] = myconn(bw);
img_con = delete_small_cc(img_con,100);
imwrite(img_con,strcat(str,'conn.png'));

%apply contour extraction algorithm
cont = find_contour(img_con);
imwrite(cont,strcat(str,'_contour.png'))

%*************************************************************************
%*************************************************************************

function img = otsu(img,nruns)
s = size(img);

N = 256; %number of graylevels
bw = ones(s(1),s(2));
for r = 1:nruns
    %compute the histogram of colors
    h = zeros(1,N);
    for i = 1:s(1)
        for j = 1:s(2)
            if(bw(i,j)==1)
                h(img(i,j)+1) = h(img(i,j)+1)+1;
            end
        end
    end

    %compute probabilities of each gray level
    p = h/(s(1)*s(2));


    %omoute between class variance for different thresholds
    varB = zeros(1,N);
    for k = 1:N
        w0 = sum(p(1:k));
        t1 = sum([1:k].*p(1:k));
        t2 = sum([k+1:N].*p(k+1:N));
        w1 = 1 - w0;
        mu0 = t1/w0;
        mu1 = t2/w1;
        varB(k) = w0*w1*(mu1-mu0)^2;
    end

    %find the largest between class variance
    ind = find(varB == max(varB));

    %compute new segmented image (foreground)
%     bw = zeros(s(1),s(2));
    new_img = zeros(size(img));
    for i = 1:s(1)
        for j = 1:s(2)
            if(bw(i,j)==1 && img(i,j) > ind(1)) %less or equal depending on
what is foreground
                new_img(i,j) = img(i,j);
```

```
                bw(i,j) =1;
            else
                bw(i,j)=0;
            end
        end
    end
    img = new_img;
end


%*************************************************************************
%*************************************************************************

function tim = text_segm(img,N)
s = size(img);
tim = zeros(size(img));
w = (N-1)/2;

for i = 1+w:s(1)-w
    for j = 1+w:s(2)-w
        patch = img(i-w:i+w, j-w:j+w);
        patch = patch(:);
        m = mean(patch);
        tim(i,j) = 1/length(patch)*sum((patch-m).^2);
    end
end

%*************************************************************************
%*************************************************************************

function [img, n ] = myconn(img)

s = size(img);
n = 1;
cnt = 0;

%first pass (label all non zero pixels)
equiv = [];
for i = 1:s(1)
    for j = 1:s(2)
        if(img(i,j)==1)
            neighbors = [];
            if(i > 1 && img(i-1,j) > 0)
                neighbors = [neighbors img(i-1,j)];
            end
            if(j > 1 && img(i,j-1)>0 )
                neighbors = [neighbors img(i,j-1)];
            end
            if(i> 1 && j >1 && img(i-1,j-1) > 0)
                 neighbors = [neighbors img(i-1,j-1)];
            end
            if(i> 1 && j < s(1) &&  img(i-1,j+1) > 0)
                 neighbors = [neighbors img(i-1,j+1)];
            end
```

```
            if(isempty(neighbors))
                n = n + 1;
                img(i,j) = n;
            else
                label = min(neighbors);
                img(i,j) = label;
                k = unique(neighbors);
                if(length(k) > 1 || (length(k)==1 && k~=label) )
                    for m = 1:length(k)
                        if(label ~= k(m))
                            equiv{k(m)} = label;
                        end
                    end
                end
            end
        end
    end
end

%pass #2  : unite connected components in equivalence class
for i = 1:s(1)
    for j = 1:s(2)
        if(img(i,j) > 0)
            if(img(i,j) < length(equiv))
                parent = equiv{img(i,j)};
                if(~isempty(parent))
                    while(~isempty(equiv{parent}))
                        parent = equiv{parent};
                    end
                    img(i,j) = parent;
                end
            end
        end
    end
end

%make the labels to be from 1 to N
k = unique(unique(img));
k(find(k==0))=[];

for i = 1:s(1)
    for j  = 1:s(2)
        if(img(i,j)>1)
            ind = find(k==img(i,j));
            img(i,j) = ind;
        end
    end
end


%*************************************************************************
%*************************************************************************

function img = delete_small_cc(img,N)

k = unique(unique(img));
```

```
k(find(k==0))=[];

for i = 1:length(k)
    [row col] = find(img == k(i));
    if(length(row)< N)
        for j = 1:length(row)
            img(row(j),col(j))=0;
        end
    end
end

%*************************************************************************
%*************************************************************************

function cont = find_contour(img_con)

%find unique labels
k = unique(unique(img_con));
k(find(k==0))=[];
cont = zeros(size(img_con));
s = size(img_con);

for i = 1:length(k)
    [row col] = find(img_con == k(i));
    I = zeros(size(img_con));

    %create a binary image
    for i = 1:length(row)
        I(row(i),col(i))=1;
    end

    %find a non zero pixel
    ind = find(row == max(row));
    y = row(ind(1));
    x = col(ind(1));

    %initialize the contour list
    B = [y x];
    py = y; px = x;
    d = 3;

    x = x - 1;

    %1 is up, 2 bottom, 3 left, 4 right
    while(x~=px || y~=py)
        %if its a white pixel move to the left
       if(I(y,x)==1)
            B = [B; y x];
            if(d == 1)%up
                d = 3;
                if(x-1>0)
                    x = x-1;
                end
            elseif(d==2)
                d = 4;
```

```
                if(x+1<=s(2))
                    x = x + 1;
                end
            elseif(d == 3) %previos step was to the left
                d = 2; %down
                if(y+1 <= s(1))
                    y = y + 1;
                end
            else
                d = 1;
                if(y-1 > 0)
                    y = y - 1;
                end
            end
        else %if pixel is black move to the right
            if(d == 1)%up
                d = 4;
                if(x+1 <= s(2))
                    x = x+1;
                end
            elseif(d==2)
                d = 3;
                if(x-1 > 0)
                    x = x - 1;
                end
            elseif(d == 3) %previos step was to the left
                d = 1; %down
                if(y-1 >0)
                y = y - 1;
                end
            else
                d = 2;
                if(y+1 <= s(1))
                y = y + 1;
                end
            end
        end
    end

    for i = 1:size(B,1)
        cont(B(i,1),B(i,2))=1;
    end
end
```