# Mean Shift Based Image Segementation

Yuxin Fan
57881161

December 9, 2018

## Introduction

This project is to do image segmentation by mean shift.
Mean shift is a non-parametric feature-space analysis technique for locating the maxima of a density function, a so-called mode-seeking algorithm.
The value of mean shift can be calculated by

$$m(x) = \frac{\sum_{i=1}^{n} x_i K(\frac{(x-x_i)^2}{h})}{\sum_{i=1}^{n} K(\frac{(x-x_i)^2}{h})} - x$$

which can be simply regard as weighed mean of x where kernel function K is the weight function.
One simple choice is to choose K function as

$$K(x) = \begin{cases} 1 & \text{x} \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Or choose it as Guassian kernel function

$$K(\frac{x - x_i}{h}) = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-x_i)^2}{2h^2}}$$

For my experience in this project, Guassian is much better. I finally choose Guassian as K.

## Implementation Details

Firstly, Read a image and resize its size to reduce scale of data if needed. Convert uint type image data to double type data.

**Compute Mean Shift**
Set space kernel size hs and color kernel size hr. Let GRB color value and position value as features of a pixel. For each pixel, compute the mean shift of it and store the pixel position where mean shift is converged.

calculation code for each pixel,

```
%reshape y to [position x, position y, d1, d2, d3 ]
pixel = double( [x y reshape(img(y,x,:),1,d)] );

x_left = max(x-hs,1);
x_right = min(x+hs,n);
length = x_right-x_left+1;

y_up = max(y-hs,1);
y_down = min(y+hs,m);
hight = y_down-y_up+1;

squares = length*hight;
%iw is a 1 X nw matrix, value from 0 to nw-1.
serial = (0:(squares-1))';
position_xs = floor(serial/hight)+x_left;
position_ys = mod(serial,hight)+y_up;
%change interset region [height * length * 3] to [(height
   *length) * 3]
reshape_= reshape(img(y_up:y_down,x_left:x_right,:),[],d)
   ;
%region_sr is (height*length)*5 which contains [position
   of each pixel of the region, d1, d2, d3]
region_sr = [position_xs position_ys reshape_];

%compute mean shift, current pixel conv to whick pixel
while true
%compute (|x_i[s,r]-x[s,r]|/h[s,r])**2 lable as M
  g_para = (region_sr-repmat(pixel,squares,1)) ./ repmat(
     h,squares,1);
  g_para = sum(g_para.*g_para, 2);
  %compute K(M)
  record = pixel;
  K = (2*pi)^(-0.5)*exp(-g_para./2);
  sumK=sum(K);
  BigK = repmat(K,1,(d+2));
  pixel = sum(region_sr(:,:).*BigK)/sumK;
  step = norm(pixel-record);
  if step < thr * max(hs,hr)
    break;
  end
end
a_conv_b(y,x,:) = pixel;
```

**Clustering/Merge**
In this part, I implement a easy used supergrid to merge points to regions. Firstly, make

a big grid which contains 0s edges and 1s entrys. Then use the 0(two nearby pixel is not merged) and 1(two nearby pixle) to fill the entries between the original 1s and then label same regions to values.

```
m = size_i(1);
n = size_i(2);

chess = ones( 2*m+1, 2*n+1);
chess(1:2:(2*m+1),:) = zeros( m+1, 2*n+1);
chess(:,1:2:(2*n+1)) = zeros( 2*m+1, n+1);
%cluster row
grad = abs(modes_map(:,2:end,1:2)-modes_map(:,1:(end-1)
   ,1:2));
space_cluster =  grad < (hs);

grad = abs(modes_map(:,2:end,3:end)-modes_map(:,1:(end-1)
   ,3:end));
color_cluster = grad < (hr);
is_nearby = cat(3,space_cluster,color_cluster);
chess(2:2:2*m,3:2:(2*n-1)) = all(is_nearby, 3);

%cluster col
space_cluster = abs(modes_map(2:end,:,1:2)-modes_map(1:(
   end-1),:,1:2)) < (hs);
color_cluster = abs(modes_map(2:end,:,3:end)-modes_map
   (1:(end-1),:,3:end)) < (hr);
is_nearby = cat(3, space_cluster, color_cluster);
chess(3:2:(2*m-1),2:2:2*n) = all(is_nearby, 3);

segm = bwlabel( chess, 4 );
segm = segm( 2:2:2*m, 2:2:2*n );
```

## Experiments & Results

Parameters are set as hs = 32, hr = 8, thr = 0.01.

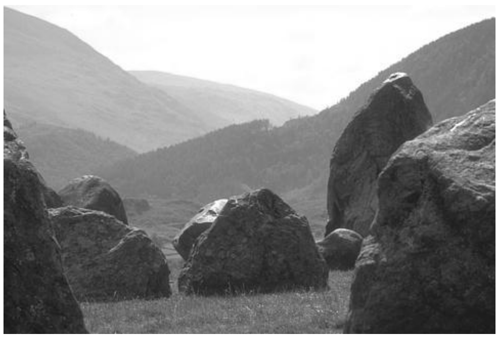

Figure 1: Input Image.

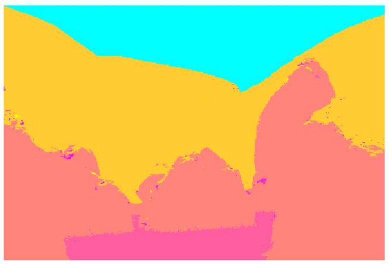

Figure 2: Segmented Image.

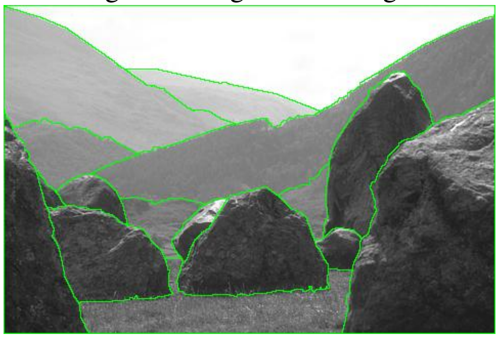

Figure 3: Human Segmented.

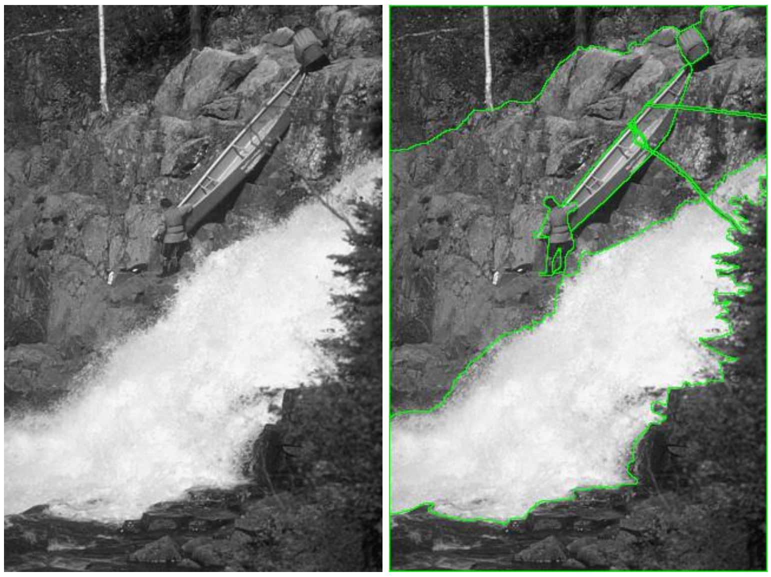
Figure 4: left: Input Image. Right: Human Segmented.

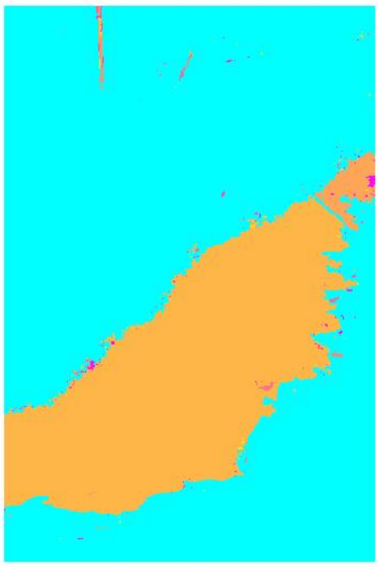
Figure 5: Segmented Image.

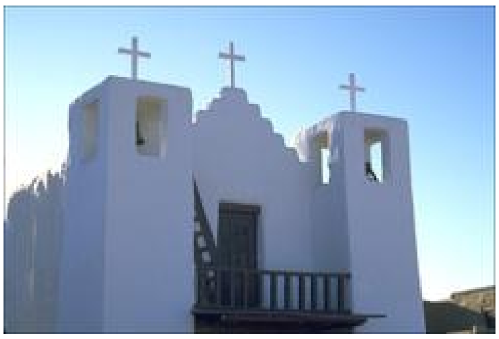

Figure 6: Input Image.

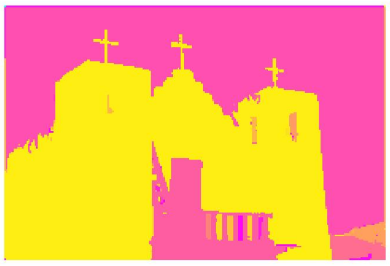

Figure 7: Segmented Image.

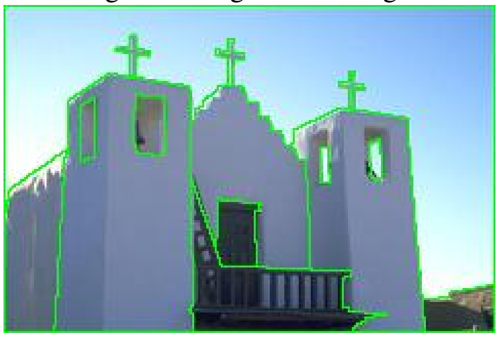

Figure 8: Human Segmented.

**An interesting Anime image**

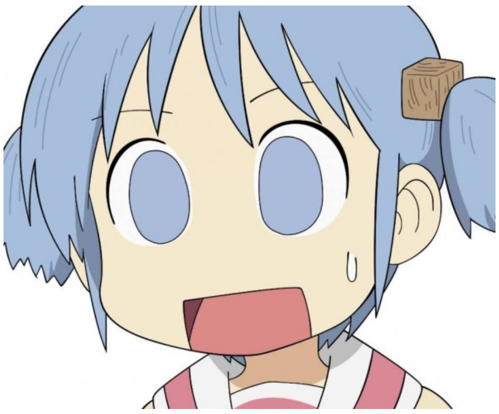

Figure 9: Input Image.

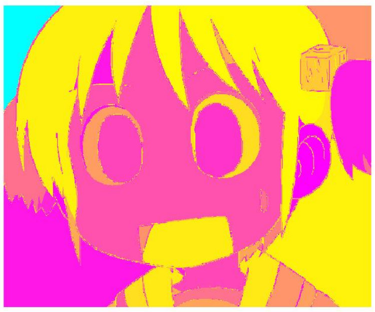

Figure 10: Segmented Image.

## Discussions

1. Firstly, I choose a simple kernel function g=0 or 1 to continue my work, however, it did not perform well. Lots of noise or very small pixel regions appear in the result. Nomatterhow I change the hs and hr parameters, the noise stays there anyway. Then, I choose Gaussian kernel function to do again, the result is much better. Segementation is reasonable and noise are small.

2. I implement a simle image segementation method. But did not do lots of optimization on it. Someting I want to do but did not implement contains that merge small areas to nearby large areas, smooth or sharpen image before mean shift.

3. Result 1 (Figure 3) looks very well. Stones and mountains and the sky are segemented clearly. But a part of the rightmost stone is be regared as mountain. The reason may be that the color of reflect light on the stone is more similar to mountain. This problem possibly can be solved by set a lower color kernel size but may cause over-segmentation.

4. Result 2 (Figure 5) looks well. Upper stones and lower stones are segmented perfectly as one group even they are divided by waters. And a branch are also be segmented out.

5. Result 3 (Figure 8) is fine. Sky, white wall and stairs and door are segmented clearly.

6. Result 4 (Figure 10) is anime image. This algorithm seems that not fit anime style image. Maybe the color of anime image are much sharp so that this function can easily segment all parts of a character which is extremely over-segmented.