ECE661: Homework 6

Wan-Eih Huang

October 20, 2018

1 Logic and Method

In this assignment we are going to use two methods to segment the images. One is based on R, G, and B values. And the other is using local statistic value, variance, as texture measure. To separate the image into foreground and background, this assignment needs us use Otsu's algorithm. In addition, extracting contour after segmentation.

1.1 Otsu's Algorithm

Otsu's algorithm is a clustering-based thresholding method. It separates gray levels into two classes, foreground and background. (It also can be extended to multi-level, but here we only discuss bi-level case.)
There are three descriminant criteria: $\frac{\sigma_B^2}{\sigma_W^2}$, $\frac{\sigma_T^2}{\sigma_W^2}$, and $\frac{\sigma_B^2}{\sigma_T^2}$. σ_T^2 is the total variance of data set which is not a function of threshold value k. σ_W^2 is winthin-class variance given by

$$\sigma_W^2 = w_0\sigma_0^2 + w_1\sigma_1^2$$

w_0, w_1 are the probabilities of two classes and $w_0 + w_1 = 1$. σ_0^2 and σ_1^2 are the variances of two classes. σ_B^2 is between-class variance given by

$$\begin{aligned}\sigma_B^2 &= w_0(\mu_0 - \mu_T)^2 + w_1(\mu_1 - \mu_T)^2 \\ &= w_0 w_1(\mu_0 - \mu_1)^2\end{aligned}$$

μ_0, μ_1, and $\mu_T = w_0\mu_0 + w_1\mu_1$ are the mean values of class 0, class 1, and overall data. Because σ_W^2 is the second order, Otsu's algorithm adopts σ_B^2 which is the first order as its criterion to achieve best separation. That is, find the optimum value of k which maximize σ_B^2.
First, construct the 256-bin historgram of the image. Then we can compute w_0, w_1, μ_0, and μ_1 by

$$\begin{aligned}w_0 &= \textstyle\sum_{i=0}^{k-1} p_i = \sum_{i=0}^{k-1} \frac{n_i}{N} \\ w_1 &= 1 - w_0 \\ \mu_0 &= \textstyle\sum_{i=0}^{k-1} \frac{ip_i}{w_0} = \frac{\sum_{i=0}^{k-1} in_i}{\sum_{i=0}^{k-1} n_i} \\ \mu_1 &= \frac{\mu_T - w_0\mu_0}{w_1}\end{aligned}$$

where p_i is the probability of i gray level, n_i is the number of pixels with i gray level, and N is the total number of pixels.
Finally, the solution is obtained by iterating k from 0 to 255 and always keeping best-so-far threshold value.

1.2 RGB Based Segmentation

First, we separate a color image into three single-channel monochrome images. Then we can use Otsu's algorithm to segment the foreground and background. Last, create a mask for each channel and then merge them together.
However, this application is kind of image dependent. For flexibility, we set several parameters to adjust the program.
1. We may need to use Otsu's algorithm iteratively to obtain better result. Hence, we can set number of iterations of Otsu's algorithm.
2. For iterative Otsu's algorithm, we use the data of one class obtained from previous iteration. This class can be class 0 or class 1. In other words, the threshold value can be go up or down based on the mode we choose.
3. The foreground object may be with low gray levels. So, we have a flag to indicate the mask should be low gray level class or high gray level class.

1.3 Texture Based Segmentation

The process of texture based segmentation is very similar to the process of RGB based segmentation. We compute variance images with different window size. Using a sliding window to compute the variance within the window around the pixel and scaling the result to (0, 255). Then we use Otsu's algorithm again to create a mask for each window size. Last, merge the masks obtained from different window size. The parameters we can adjust is the same as last section.

1.4 Contour Extraction

Before contour extraction, we want to eliminate the noise on the segmentation image. There are some tiny holes in foreground or dots in background. We adopt dilation and erosion methods by square window. Unfortunately, the applications are also kind of dependent on images. Thus, we need to set the window size and number of iterations.
After we removed the noise, the contour extraction method is following the criterion; if the pixel is foreground and its 8 neighbors are not all in foreground, then it is the border pixel. All the border pixels form the contour.

1.5 Steps of Implementation

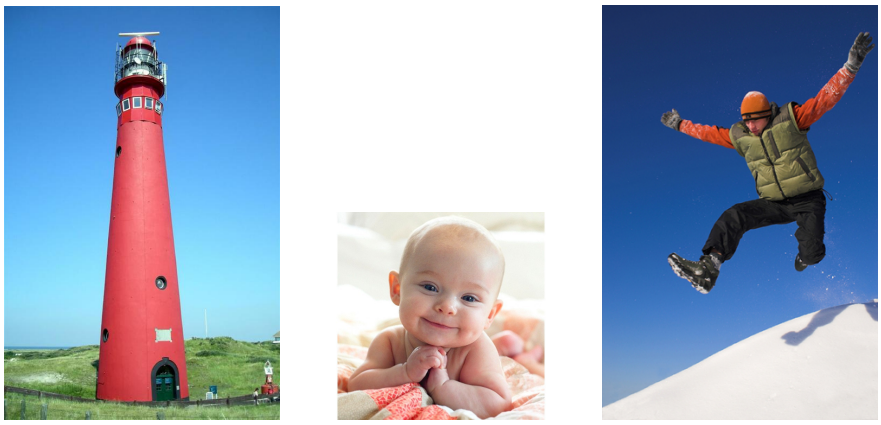
1.5.1 Task 1: RGB based segmentation

Step 1: Separate the input color image into three single channel images.
Step 2: Create a mask for each single channel image by iterative Otsu's algorithm.
Step 3: Merge the masks into one overall mask.
Step 4: Eliminate the noise by erosion and dilation.
Step 5: Extract the contour.

1.5.2 Task 2: Texture based segmentation

Step 1: Convert the input color image to grayscale image.
Step 2: Compute variance image for given window size. In this experiment, we use 3×3, 5×5, and 7×7. And scale each variance image to the range (0, 255).
Step 3: Create a mask for each variance image by iterative Otsu's algorithm.
Step 4: Merge the masks into one overall mask.
Step 5: Eliminate the noise by erosion and dilation.
Step 6: Extract the contour.

2 Result



(a) (b) (c)

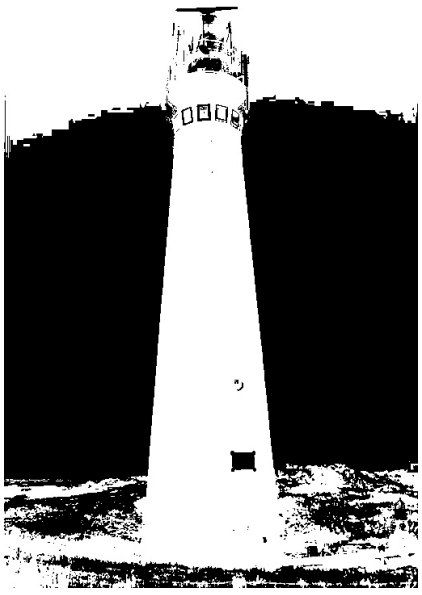
Figure 1: Original images. (a) Image 1 (b) Image 2 (c) Image 3

2.1 Task 1

Image 1:

(a) (b)

Figure 2: The foreground generated by R channel image. (a) The first iteration (k=139) (b) The second iteration (k=192)

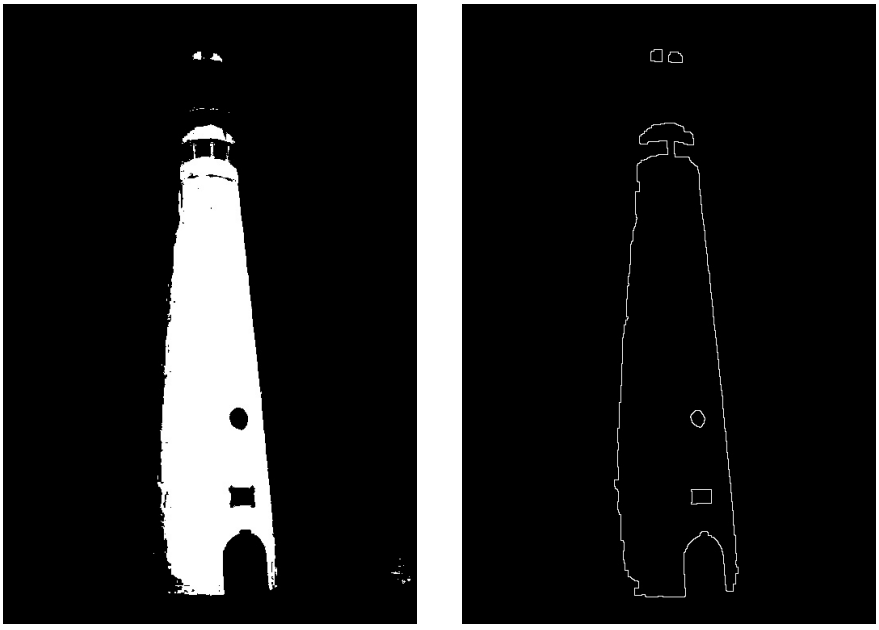


(a)

Figure 3: The foreground generated by G channel image. (k=153)

(a)

Figure 4: The foreground generated by B channel image. (k=163)



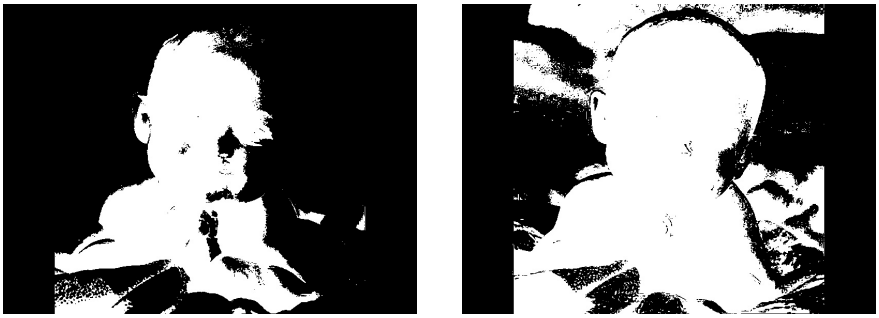
(a) (b)

Figure 5: (a) The overall foreground (b) The contour

Image 2:

	[R, G, B]
Number of iterations	[2,1,1]
Class 0 is the candidate set for next iteration ?	[No, NA, NA]
Foreground is low gray level class ?	[No, Yes, Yes]
Erosion	3×3, 1 iteration
Dilation	5×5, 2 iterations

Table 1: The parameters of image 1.



(a) (b)

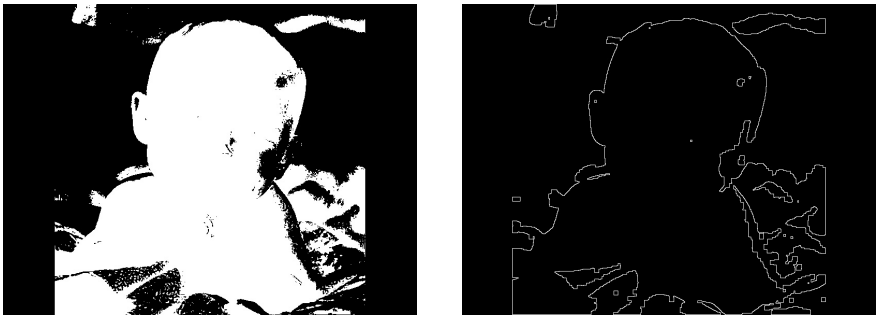
Figure 6: The foreground generated by R channel image. (a) The first iteration (k=207) (b) The second iteration (k=239)

(a) (b)

Figure 7: The foreground generated by G channel image. (a) The first iteration (k=186) (b) The second iteration (k=229)

(a) (b)

Figure 8: The foreground generated by B channel image. (a) The first iteration (k=180) (b) The second iteration (k=228)



(a) (b)

Figure 9: (a) The overall foreground (b) The contour

	[R, G, B]
Number of iterations	[2,2,2]
Class 0 is the candidate set for next iteration ?	[No, No, No]
Foreground is low gray level class ?	[Yes, Yes, Yes]
Erosion	3×3, 1 iteration
Dilation	5×5, 2 iterations

Table 2: The parameters of image 2.

Image 3:

(a)

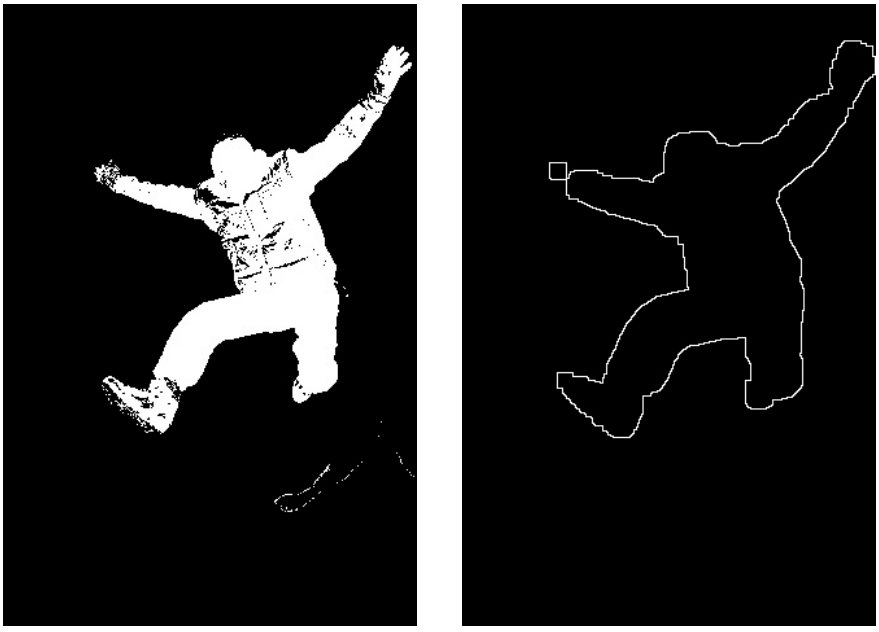
Figure 10: The foreground generated by R channel image. (k=136)

(a)

Figure 11: The foreground generated by G channel image. (k=150)

(a) (b)

Figure 12: The foreground generated by B channel image. (a) The first iteration (k=150) (b) The second iteration (k=81)



(a) (b)

Figure 13: (a) The overall foreground (b) The contour

	[R, G, B]
Number of iterations	[1,1,2]
Class 0 is the candidate set for next iteration ?	[NA, NA, Yes]
Foreground is low gray level class ?	[No, Yes, Yes]
Erosion	3×3, 1 iteration
Dilation	5×5, 3 iterations

Table 3: The parameters of image 3.

2.2 Task 2

Image 1:

(a) (b) (c)

Figure 14: The foreground generated by local variance with 3×3 window. (a) The first iteration (k=33) (b) The second iteration (k=8) (c) The third iteration (k=2)

(a) (b) (c)

Figure 15: The foreground generated by local variance with 5×5 window. (a) The first iteration (k=33) (b) The second iteration (k=9) (c) The third iteration (k=2)

(a) (b) (c)

Figure 16: The foreground generated by local variance with 7×7 window. (a) The first iteration (k=35) (b) The second iteration (k=9) (c) The third iteration (k=2)

(a) (b)

Figure 17: (a) The overall foreground (b) The contour

	$[3 \times 3, 5 \times 5, 7 \times 7]$
Number of iterations	[3,3,3]
Class 0 is the candidate set for next iteration ?	[Yes, Yes, Yes]
Foreground is low level class ?	[Yes, Yes, Yes]
Erosion	5×5, 2 iteration
Dilation	5×5, 1 iterations

Table 4: The parameters of image 1.

Image 2:

(a) (b) (c)

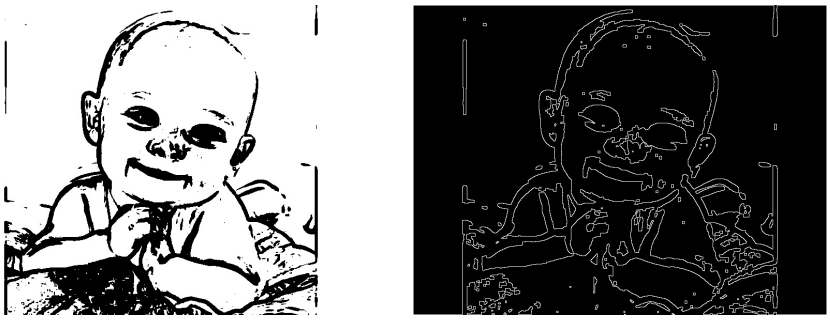
Figure 18: The foreground generated by local variance with 3×3 window. (a) The first iteration (k=16) (b) The second iteration (k=4) (c) The third iteration (k=1)

(a) (b) (c)

Figure 19: The foreground generated by local variance with 5×5 window. (a) The first iteration (k=25) (b) The second iteration (k=6) (c) The third iteration (k=1)

(a) (b) (c)

Figure 20: The foreground generated by local variance with 7×7 window. (a) The first iteration (k=31) (b) The second iteration (k=8) (c) The third iteration (k=2)



(a) (b)

Figure 21: (a) The overall foreground (b) The contour

	$[3 \times 3, 5 \times 5, 7 \times 7]$
Number of iterations	[3,3,3]
Class 0 is the candidate set for next iteration ?	[Yes, Yes, Yes]
Foreground is low level class ?	[Yes, Yes, Yes]
Erosion	5×5, 1 iteration
Dilation	5×5, 1 iterations

Table 5: The parameters of image 2.

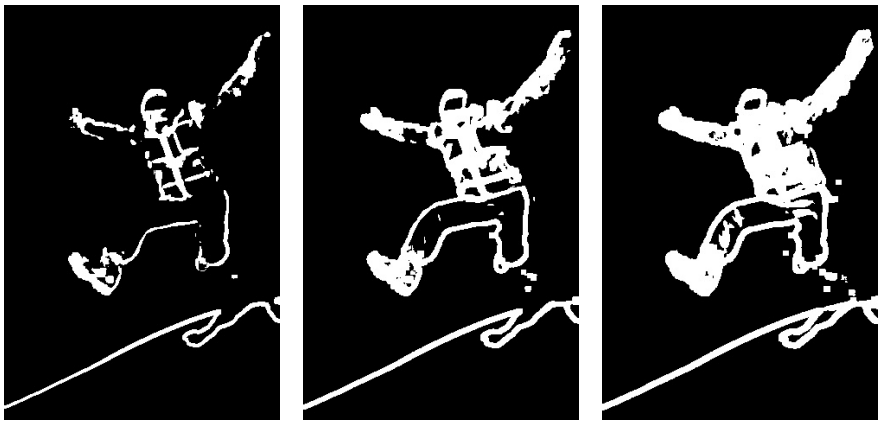
Image 3:

(a) (b) (c)

Figure 22: The foreground generated by local variance with 3×3 window. (a) The first iteration (k=28) (b) The second iteration (k=8) (c) The third iteration (k=2)

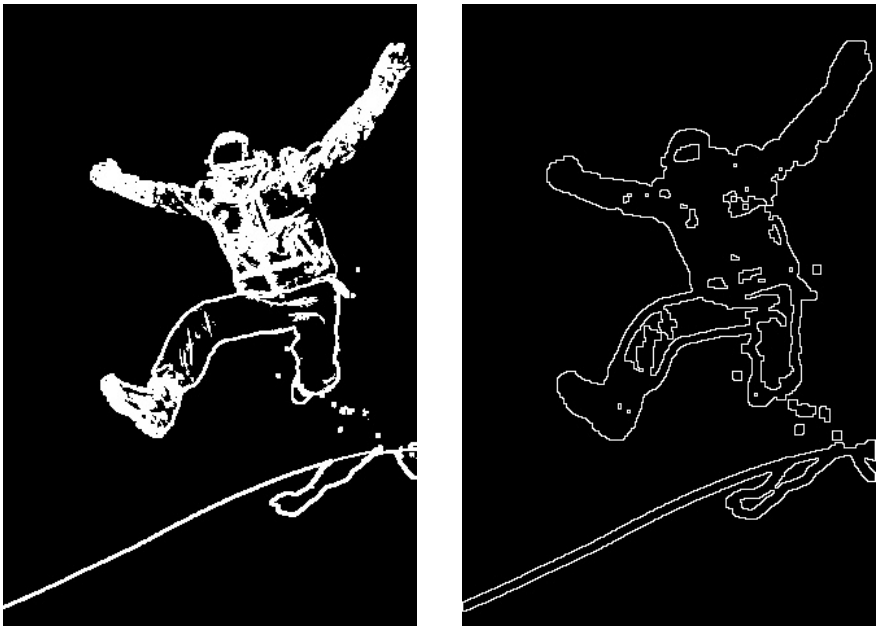
(a) (b) (c)

Figure 23: The foreground generated by local variance with 5×5 window. (a) The first iteration (k=29) (b) The second iteration (k=8) (c) The third iteration (k=2)



(a) (b) (c)

Figure 24: The foreground generated by local variance with 7×7 window. (a) The first iteration (k=32) (b) The second iteration (k=10) (c) The third iteration (k=3)



(a) (b)

Figure 25: (a) The overall foreground (b) The contour

	$[3 \times 3, 5 \times 5, 7 \times 7]$
Number of iterations	[3,3,3]
Class 0 is the candidate set for next iteration ?	[Yes, Yes, Yes]
Foreground is low level class ?	[No, No, No]
Erosion	No
Dilation	5×5, 1 iterations

Table 6: The parameters of image 3.

3 Observation

RGB based segmentation
In image 1, there are three clear color regions, red lighthouse, blue sky, and green grass. Because we define foreground is red lighthouse, we can expect the pixel values in foreground should be high in red, low in blue and green. So we have the clue to set foreground flag which indicates high or low level class is foreground. And this segmentation method did a great job in image 1.
In image 2, there is no dominant color. Most pixels in background are close to white. Hence, we expect the pixel values in foreground are relative low in 3 channels. However, it is hard to separate the foreground baby, blanket, and some shadows in image, if we only threshold the single-channel image independently and ignore the relationship among three channels.
From the observation on image 3, we can know the pixel values on the jumping man are low in green and blue channel. In addition, the pixels on the arms of the jumping man are high in red. However, the pixels on the pants are low in red. Thus, it is difficult to segment the person in red channel. In this image, I did a little change in merging the masks. That is, if two of three channels support this area is foreground, then it is foreground.
Texture based segmentation
In all three images, when we use larger window size, the segmentations are more coarse.
Moreover, in image 1 both lighthouse and blue sky are flat. In means both they have low variance statistic characteristic. So, when we use texture to segment the image 1, we obtain these two regions as foreground. The mask acts like edge detection.
Similarly, in image 2, the baby's skin and near-white background do not have obvious texture. However, the baby's eyes, nose, and mouth have larger variations. Thus our result shows the flat area and masked parts are like edges. Because they have relatively higher variance.
In image 3, the pixel values of the jumping man are with higher variance except the pants area. The sky and snow are with relative low variance. Therefore, the texture based segmentation can partially segment the person except the pants.

4 Source Code

4.1 Task 1

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 18 19:23:33 2018
RGB based segmentation
@author: wehuang
"""

import cv2
import matplotlib.pyplot as plt
import numpy as np
from util_basic import *

# Parameters
label = ['b', 'g', 'r']
num_iter = [1, 1, 2]
flag = [1, 1, 0]
go_lower = [0, 0, 0]
erode_size = 3
erode_iter = 1
dilate_size = 5
dilate_iter = 2

# Main program starts from here!
img = cv2.imread('HW6Pics/lighthouse.jpg')
mask = np.zeros(img.shape, dtype=np.uint8)
for i in range(3):
    img_mono = img[:,:,i]
    mask[:,:,i] = run_otsu(img_mono, num_iter[i], label[i],
        flag[i], go_lower[i])

# Merge masks
mask_overall = mask[:,:,0]*mask[:,:,1]*mask[:,:,2]
#mask_overall = np.floor((mask[:,:,0]+mask[:,:,1]+mask[:,:,2])
    /2) # for image 3
img_out = np.uint8(mask_overall*255)
cv2.imwrite('otsu.jpg', img_out)

# Eliminate noise
mask_overall = erosion(mask_overall, erode_size, erode_iter)
mask_overall = dilation(mask_overall, dilate_size, dilate_iter
    )
# Contour
find_contour(mask_overall)
```

4.2 Task 2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
```

```
Created on Thu Oct 18 23:48:47 2018
Texture based segmentation using local variance
@author: wehuang
"""

import cv2
import matplotlib.pyplot as plt
import numpy as np
from util_basic import *

def texture_stat(img_gray, N):
    variance = np.zeros(img_gray.shape)
    for j in range(img_gray.shape[0]):
        for i in range(img_gray.shape[1]):
            half = np.int((N-1)/2)
            patch = img_gray[max(0,j-half):min(img_gray.shape
                [0],j+half+1), max(0,i-half):min(img_gray.
                shape[1],i+half+1)]
            variance[j][i] = np.var(patch)

    # Scale to [0, 255]
    variance = np.uint8(np.round(255*variance/(np.max(variance
        )-np.min(variance))))
    return variance

# Parameter
window_size = [3, 5, 7]
num_iter = [3, 3, 3]
label = ['3', '5', '7']
flag = [0, 0, 0]
go_lower = [1, 1, 1]
erode_size = 5
erode_iter = 1
dilate_size = 5
dilate_iter = 1

# Main program starts from here!
img = cv2.imread('HW6Pics/ski.jpg')
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
mask = np.zeros(img.shape, dtype=np.uint8)
for i, N in enumerate(window_size):
    text_img = texture_stat(img_gray, N) # compute local
        variance
    mask[:,:,i] = run_otsu(text_img, num_iter[i], label[i],
        flag[i], go_lower[i])

# Merge masks
mask_overall = mask[:,:,0]*mask[:,:,1]*mask[:,:,2]
img_out = np.uint8(mask_overall*255)
cv2.imwrite('text.jpg', img_out)

# Eliminate noise
#mask_overall = erosion(mask_overall, erode_size, erode_iter)
mask_overall = dilation(mask_overall, dilate_size, dilate_iter
```

```
    )
# Contour
find_contour(mask_overall)
```

4.3 Basic functions

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 19 11:16:34 2018
Basic functions for image segmentation
@author: wehuang
"""

import cv2
import matplotlib.pyplot as plt
import numpy as np

def otsu(data):
    hist, bin_edges = np.histogram(data, bins=256, range
        =(0,256))

    N = len(data)
    sum_total = sum(hist*bin_edges[:-1])
    n_back = 0 # number of pixels in background
    sum_back = 0 # sum of background
    best_score = -1
    for t in range(256):
        n_back += hist[t]
        n_fore = N - n_back # number of pixels in foreground
        if n_back == 0 or n_fore == 0:
            continue
        sum_back += t*hist[t]
        sum_fore = np.int(sum_total - sum_back) # sum of
            foreground
        score = n_back*n_fore*(sum_back/n_back-sum_fore/n_fore
            )**2
        if score >= best_score:
            threshold = t
            best_score = score

    return threshold

def run_otsu(img_mono, num_iter, label, flag, go_lower):
    a = img_mono.flatten()
    for n in range(num_iter):
        threshold = otsu(a)
        # Generate mask
        print(threshold)
        mask = np.zeros(img_mono.shape, dtype=np.uint8)
        if flag: # low level is foreground
            mask[img_mono <= threshold] = 1
        else: # high level is foreground
            mask[img_mono > threshold] = 1
```

```
        img_out = mask*255
        #plt.imshow(img_out, cmap="gray")
        #plt.show()
        cv2.imwrite('otsu_'+label+str(n+1)+'.jpg', img_out)
        if go_lower: # low level class for next iteration
            a_tmp = [i for i in a if i <= threshold]
        else: # high level class for iteration
            a_tmp = [i for i in a if i > threshold]
        a = np.asarray(a_tmp)

    return mask

def erosion(mask, erode_size, erode_iter):
    kernel = np.ones((erode_size,erode_size), np.uint8)
    mask = cv2.erode(mask, kernel, iterations=erode_iter)
    img_out = np.uint8(mask*255)
    cv2.imwrite('otsu_erode.jpg', img_out)

    return mask

def dilation(mask, dilate_size, dilate_iter):
    kernel = np.ones((dilate_size,dilate_size), np.uint8)
    mask = cv2.dilate(mask, kernel, iterations=dilate_iter)
    img_out = np.uint8(mask*255)
    cv2.imwrite('otsu_dilate.jpg', img_out)

    return mask

def find_contour(mask):
    contour = np.zeros(mask.shape, dtype=np.uint8)
    for j in range(mask.shape[0]):
        for i in range(mask.shape[1]):
            if mask[j][i] == 0:
                continue
            patch = mask[j-1:j+2,i-1:i+2]
            if sum(patch.flatten()) < 9:
                contour[j][i] = 1

    img_out = contour*255
    cv2.imwrite('contour.jpg', img_out)
```

5 Reference

1. https://en.wikipedia.org/wiki/Otsu%27s_method