Homework - 6

Sriram Karthik Badam

October 30, 2012

1. **Image Segmentation and Contouring**

   In this homework, Otsu's Method has been used to segment out a Region of Interest (foreground) of the image and then the Contour is extracted.
   Otsu's Algorithm: The goal of Otsu's Algorithm is to find a threshold on the pixel intensity that distinguishes background from foreground.

   (a) Split the color image into three gray-scale images.

   (b) On each gray-scale image do the following:

      i. Calculate the histogram of the image

      ii. Divide the histogram into two classes by finding the threshold bin 'k' that maximizes the Variance Bewteen classes. We do this sequentially by calculating the Between-class Variance for each value of 'k' in the input space (L bins of the histogram). The following set of formulae help you calculate the variance.

$$p_i = n_i/N$$

p_i stands for probability of level or bin 'i' in the histogram. The Weight of each class is given by

$$w_0 = Pr(C_0) = \sum_{i=1}^{k} p_i$$

$$w_1 = Pr(C_1) = \sum_{i=k+1}^{L} p_i$$

Note that $w_1 = 1 - w_0$. The above weight values can be updated cummulatively during each iteration for threshold 'k' using-

$$w_0^k = w_0^{k-1} + p_k$$

$$w_1^k = 1 - w_0^k$$

at k^{th} iteration and $w_0^0 = p_0$.
Mean of each class is given by

$$\mu_0 = \frac{\sum_{i=1}^{k} i p_i}{w_0}$$

$$\mu_1 = \frac{\mu_T - \mu_0}{w_1}$$

$$\mu_T = \sum_{i=1}^{L} ip_i$$

The updation of μ_0 and μ_1 is done by using

$$\mu_0^k = \frac{s^{k-1} + kp_k}{w_0^k}$$

where $s^{k-1} = \sum_1^{k-1} ip_i$

$$\mu_1^k = \frac{\mu_T - \mu_0^k}{1 - w_0^k}$$

Once you get the Mean values, the Between-class Variance is calculated using

$$\sigma_b^2 = w_0 w_1 (\mu_0 - \mu_1)^2$$

Using the above updation principles we iteratively calculate the value of σ_b^2 for each value of threshold 'k' to find the threshold that maximises the Between-class variance.
Note that maximizing σ_b^2 in-turn maximizes the descriminant criterion $\eta = \sigma_b^2/\sigma_T^2$, since σ_T^2 is independent of the threshold.

iii. Discard all pixels that fall outside this threshold 'k'.

iv. **Important:** Otsu's Algorithm may have to be applied multiple times to get the best foreground estimate. For the given example, I applied the Otsu's method more than once on the foreground for a few channels. The number of times to apply is found out manually offline by looking at the image channels after thresholding.

(c) We reconstruct the color image by merging the individual channel values.

$$im[i,j] = (channel1[i,j], channel2[i,j], channel3[i,j])$$

where im[i,j] stands for $(i,j)^{th}$ pixel in image and channel1, channel2, channel3 stand for the image channels after otsu's method.

(d) After this we can binarize the image by assigning pixel intensity to 255 if the original intensity is non-zero.
This gives a Binary foreground image consisting of the lake for the given example.

2. **Contour Extraction:** The Contours are extracted using the following algorithm:
For the non-zero intensity pixels in the foreground image, if any 4 - connected pixel neighbors has zero intensity then the pixel is on the boundary.

(a) Apply a binary mask on each pixel to compute the number of zero intensity neighboring pixels and if this count is greater than zero then the pixel of interest is on the boundary and its intensity value is retained. Note that im[i-1, j], im[i, j-1], im[i+1, j], im[i, j+1] are the pixels 4-connected to the pixel im[i, j] where im stands for image.

(b) If the pixel is not on the boundary i.e, it has only non-zero 4-connected neighboring pixels, then the pixel value is made zero.

This gives us a final image consising the outlines of the foreground components or contours in the image.

3. **Observations**

   (a) Otsu's Algorithm has to be applied multiple times on some channels to extract the lake perfectly. Since there is no way to programatically find if the foreground consists of only lake and no noise, this should be done manually by checking the output after otsu's method.

   (b) We may get some noise after the Otsu's algorithm . The noise consists of really small groups of pixels(one or two pixels) which are notified as foreground. These can be removed during the contour extraction.

   (c) For the second input man.jpg, the application of Otsu's algorithm has a limitation. It partially segments the rocks under the man. I guess these rocks also belong to foreground but they can't be extracted completely with a naive approach as Otsu's method.

   (d) I noticed that for complex images with foreground distributed among the three channels Otsu's method doesn't perform well. I guess thats another limitation of Otsu's method.

(a) **Results**

Figure 1: input image

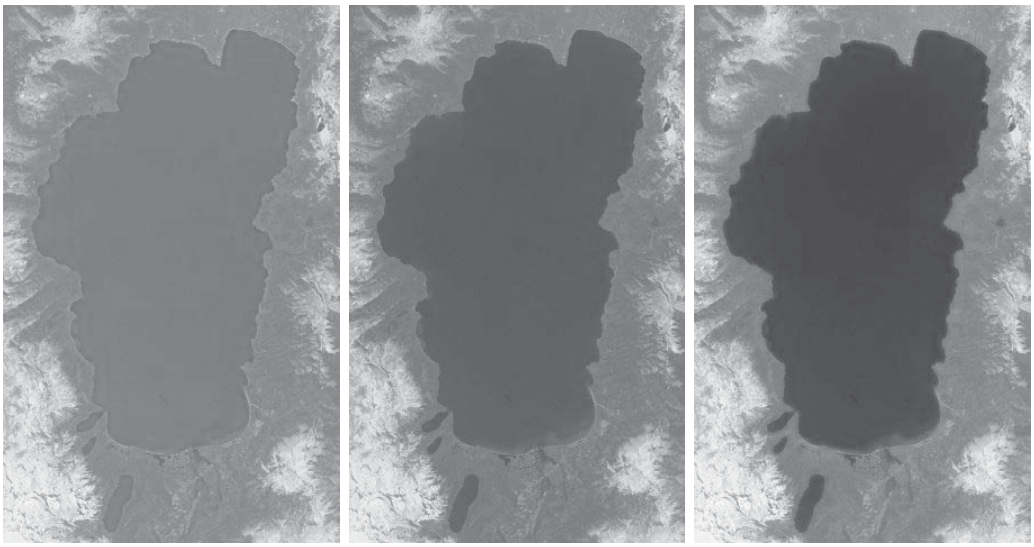

Figure 2: Split to three Channels

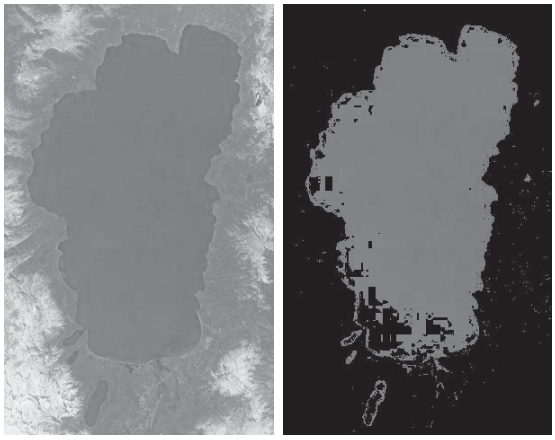

Figure 3: Otsu's Algorithm has been applied 3 times on the foreground to extract the lake in Channel 1

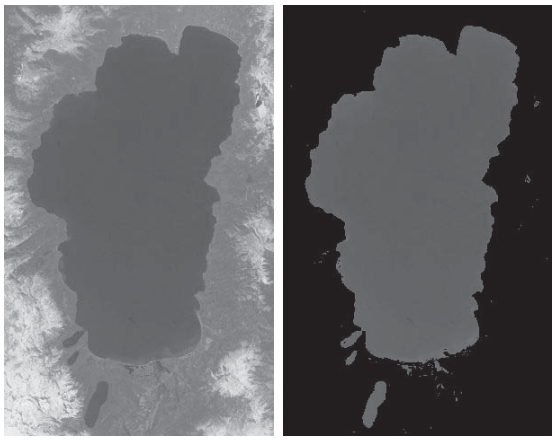

Figure 4: Otsu's Algorithm has been applied 2 times on the foreground to extract the lake in Channel 2

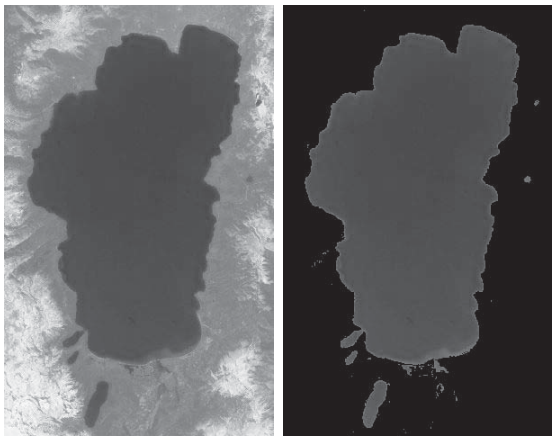

Figure 5: Otsu's Algorithm has been applied 1 time to extract the lake in Channel 3

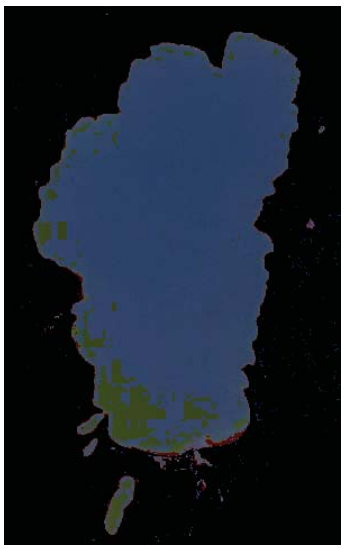

Figure 6: Merged image

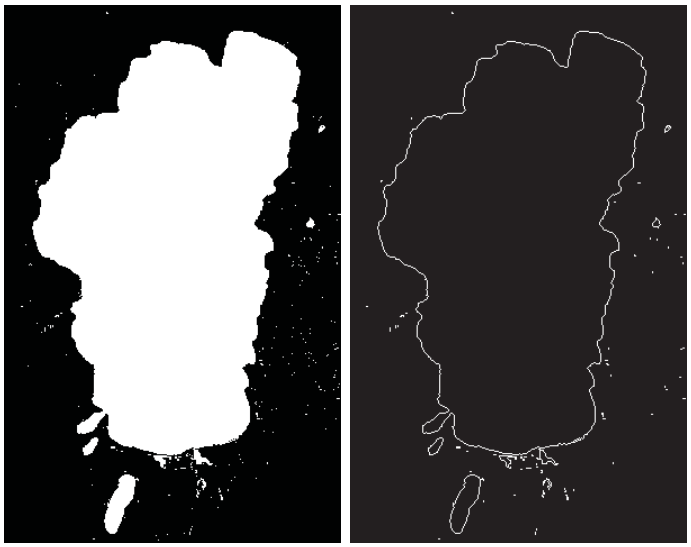

Figure 7: Contour extraction of the binary image of lake

Figure 8: input image

Figure 9: Split to three Channels

Figure 10: Otsu's Algorithm has been applied 2 times on the foreground to extract the man in Channel 1

Figure 11: Otsu's Algorithm has been applied 3 times on the foreground to extract the man in Channel 2

Figure 12: Otsu's Algorithm has been applied 3 times to extract the man in Channel 3

Figure 13: Merged image

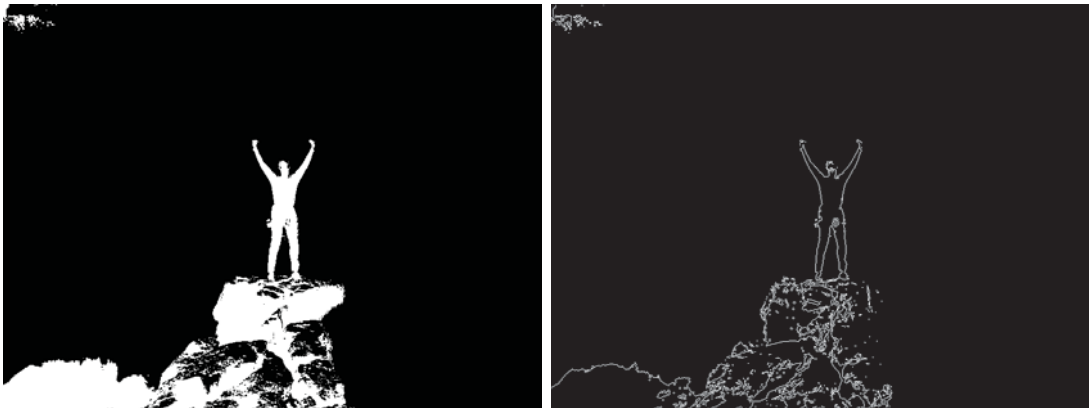

Figure 14: Contour extraction of the binary image of man

4. Code: otsu.py

```
#!/usr/bin/python
#
# Author: Sriram Karthik Badam
# Date: Oct 27, 2012
#

import sys, os
import cv
import numpy as np

NUMBER_OF_BINS = 256

"""
Comment:
Computes the threshold for a given histogram, minValue and maxValue
hist-> histogram of the image
minValue-> minimum bin index histogram
maxValue-> maximum bin index in histogram
Example usage:
otsu(hist, 0, 255)
otsu(hist, 18, 100)-> Computes the threshold for the sub-histogram between 18,
100 bins of the histogram hist
"""
def otsu(hist, minValue, maxValue):
  total_pixels = 0

  #probability of each bin
  p = np.zeros(maxValue - minValue + 1, float)
  sum = 0
  for i  in range(minValue, maxValue+1):
    total_pixels += cv.QueryHistValue_1D(hist, i)

  for i  in range(minValue, maxValue+1):
    p[i] = cv.QueryHistValue_1D(hist, i)/total_pixels
    sum += i*p[i]
  #print p

  #Weight of classes
  w = np.zeros(2, float)
  #cummulative sum -> sum(i*p[i])
  c_sum = np.zeros(2, float)
  #mean
  mean = np.zeros(2, float)

  w[0] = 0
  max_variance = 0
  threshold = 0

  #threshold k
  for k in range(minValue, maxValue+1):
    #weight of foreground and background classes
    w[0] += p[k]
    w[1] = 1 - w[0]

    if w[0] == 1:
      break
```

```
    if w[0] != 0:
      c_sum[0] += k*p[k]
      c_sum[1] = sum  - c_sum[0]

      #Mean of background and foreground classes
      mean[0] = c_sum[0]/w[0]
      mean[1] = c_sum[1]/w[1]

      variance_between = w[0]*w[1]*np.power((mean[0] - mean[1]), 2)

      #Save the threshold of maximum Between class variance
      if variance_between > max_variance:
        max_variance = variance_between
        threshold = k

  return threshold


def main():
  #input image
  input_image = cv.LoadImage("lake.jpg", cv.CV_LOAD_IMAGE_UNCHANGED)

  #splits input image to 3 channels
  image_channel1 = cv.CreateImage((input_image.width, input_image.height), input_image
      .depth, 1)
  image_channel2 = cv.CreateImage((input_image.width, input_image.height), input_image
      .depth, 1)
  image_channel3 = cv.CreateImage((input_image.width, input_image.height), input_image
      .depth, 1)
  cv.Split(input_image, image_channel1, image_channel2, image_channel3, None)
  cv.SaveImage("InputChannel1.png", image_channel1)
  cv.SaveImage("InputChannel2.png", image_channel2)
  cv.SaveImage("InputChannel3.png", image_channel3)

  global NUMBER_OF_BINS
  bin = NUMBER_OF_BINS
  intensity_range = [0, 255]
  hist1 = cv.CreateHist([bin], cv.CV_HIST_ARRAY, [intensity_range], 1)
  hist2 = cv.CreateHist([bin], cv.CV_HIST_ARRAY, [intensity_range], 1)
  hist3 = cv.CreateHist([bin], cv.CV_HIST_ARRAY, [intensity_range], 1)

  #Image Channel 1
  cv.CalcHist([image_channel1], hist1, 0, None)
  #Apply otsu's algorithm multiple times since single time doesn't give a good
      threshold for channel1
  threshold1 = otsu(hist1, 0, 255)
  threshold1 = otsu(hist1, 0, threshold1)
  threshold1 = otsu(hist1, 0, threshold1)
  #computes Channel1 foreground Image
  for i in range(input_image.width):
    for j in range(input_image.height):
      if image_channel1[j, i] > threshold1:
        image_channel1[j,i] = 0
  cv.SaveImage("Channel1.png", image_channel1)

  #Image Channel 2
```

```
cv.CalcHist([image_channel2], hist2, 0, None)
#Applys otsu's algorithm twice to get the lake
threshold2 = otsu(hist2, 0, 255)
threshold2 = otsu(hist2, 0, threshold2)
#computes Channel2 foreground Image
for i in range(input_image.width):
  for j in range(input_image.height):
    if image_channel2[j, i] > threshold2:
      image_channel2[j,i] = 0
cv.SaveImage("Channel2.png", image_channel2)

#Image Channel 3
cv.CalcHist([image_channel3], hist3, 0, None)
threshold3 = otsu(hist3, 0, 255)
#computes Channel3 foreground Image
for i in range(input_image.width):
  for j in range(input_image.height):
    if image_channel3[j, i] > threshold3:
      image_channel3[j,i] = 0
cv.SaveImage("Channel3.png", image_channel3)

print "threshold are: ",threshold1, threshold2, threshold3

#merges the three channels
for i in range(input_image.width):
  for j in range(input_image.height):
      input_image[j, i] = [image_channel1[j,i], image_channel2[j, i], image_channel3
          [j,i]]

cv.SaveImage("input_after_threshold.png", input_image)

#Creates Binary image of the foreground
for i in range(input_image.width):
  for j in range(input_image.height):
      if input_image[j, i][0] != 0 or input_image[j,i][1] != 0 or input_image[j, i
          ][2] != 0:
        input_image[j, i] = [255, 255, 255]

cv.SaveImage("Binary_foreground.png", input_image)


#Contour Extraction
out_image = cv.CreateImage((input_image.width, input_image.height), input_image.
    depth, 1)
cv.Zero(out_image)
count = 0
for i in range(1, input_image.width-1):
  for j in range(1, input_image.height-1):
    if input_image[j, i][0] != 0 or input_image[j, i][1] != 0 or input_image[j, i
        ][2]!=0:
      #checks 4 connected components!
      count = 0
      for index in [-1, 1]:
        neighbor_value = input_image[j+index, i]
  if neighbor_value[0] == 0 and neighbor_value[1] == 0 and neighbor_value[2] == 0:
          count += 1
      for index in [-1, 1]:
```

```
            neighbor_value = input_image[j, i+index]
        if neighbor_value[0] == 0 and neighbor_value[1] == 0 and neighbor_value[2] == 0:
                count += 1
            if count > 0 and count < 4:#if atleast one of the neighbors and atmost 3
                neighbors are of zero intensity
            out_image[j, i] = 255

    cv.SaveImage("Result.png", out_image)

if __name__=="__main__":
    main()
```