**ECE 661 Homework 6**

**Weichen Xu**

**xu1363@purdue.edu**

**1 Theory Questions (Difference between Otsu and Watershed Algorithm)**

Otsu algorithm use the whole image’s histogram of gray scale pixel values, then obtain the threshold which gives the maximal inter-class variance and minimal intra-class variance to create a binary image, assuming that one level is the foreground while the other level is the background. The strength of Otsu algorithm is that it has a simple implementation procedure while yields a good segmentation if the image has distinct foreground and background. The weakness of it is obvious when the image has multiple interest area of several pixel levels, it is hard for the algorithm to give a fining segmentation result.

Watershed algorithm use the concept of building watershed by continuously injecting water to the hole, until the whole plane is covered with water. The algorithm uses the gradients of gray image instead of image itself, it will look for regions of high-intensity gradients separating neighbored local minima. The outstanding strength of watershed algorithm is being able to give multiple divisions in separated regions, the weakness is it tends to have over-segmentation which it overlooks the small areas of interests.

**2 Otsu Algorithm**

The Otsu algorithm is a clustering-based method to perform image thresholding, it assumes that an image has two classes, one is the foreground pixels, the other is the background pixels. The algorithm calculates the threshold for two classes by minimizing the intra-class variance or maximizing the inter-class variance . We can express the weighted intra-class variance as following

While , are the probabilities of two classes, they satisfy that . , are variances within two classes. So the inter-class variance can be expressed as

While , , are mean pixel values for two classes and overall image. Since the image’s gray level has 256 bins, we can define them as following forms

is the current threshold for separating two classes, is the probability of pixel value in all pixel values.

We can find the threshold with the greatest value of , then this threshold is used to divide the whole gray-scale image into two classes, i.e. foreground and background.

**3 RGB Image segmentation**

Since a color image has three channels R, G, B, we will treat each channel of color image as a monochrome image. Then for three channels, we run Otsu algorithm on them respectively to obtain the threshold for separating foreground and background. Once we get the mask for three channels, we will combine three masks into an overall mask by using logical operator AND, thus obtaining the final segmentation of the image. Implementation is described below:

Step 1: Sperate a color image into three monochrome images.

Step 2: Run Otsu algorithm iteratively for three monochrome images to get masks.

Step 3: Combine three masks using logical operator AND.

**4 Texture-based Image Segmentation**

First, we convert a color image to a grayscale image, then we implement different sliding window (Here ) to obtain the variance in each window. Then we get three channels of features image, which is very similar to the original RGB channels of color image. We apply same operations of Otsu algorithm with the feature images.

Step 1: Convert the RGB image to grayscale image.

Step 2: Run three different sliding windows to get three features images of variances.

Step 3: Run Otsu algorithm iteratively for three features images to get masks.

Step 4: Combine three masks using logical operator AND.

**5 Contour Extraction**

In the previous image segmentation task, we already have a binary image mask, with the foreground pixel value is 1 and background pixel value is 0. To extract the contour of this binary mask, we need to implement contour extraction based on the 8-neighbors of a pixel. The criteria of

Step 1: Refine the binary mask using iteratively erosion and dilation operations with proper size.

Step 2: Decide a point is at the contour, if the pixel value is 1, and not all its 8-neighbors’ value are 1.

**6 Experiment Results**

**6.1 Task 1: RGB based Image Segmentation**

**A cat that is looking at the camera

Description automatically generated**

Figure 1: Cat image R channel Otsu result (1 iteration)

**A picture containing mammal, sitting, cat

Description automatically generated**

Figure 2: Cat image G channel Otsu result (1 iteration)

**A picture containing mammal, cat, sitting, looking

Description automatically generated**

Figure 3: Cat image B channel Otsu result (1 iteration)

**A picture containing mammal, sitting, cat, looking

Description automatically generated**

Figure 4: Cat image three channel combined segmentation mask

A close up of a logo

Description automatically generated

Figure 5: Cat image contour (erosion and dilation)

**A picture containing sitting, water, photo, table

Description automatically generated**

Figure 6: Pigeon image R channel Otsu result (1 iteration)

**A picture containing sitting, photo, water, table

Description automatically generated**

Figure 7: Pigeon image G channel Otsu result (1 iteration)

**A picture containing sitting, water, photo, dark

Description automatically generated**

Figure 8: Pigeon image B channel Otsu result (1 iteration)

**A picture containing sitting, water, photo, dark

Description automatically generated**

Figure 9: Pigeon image three channel combined segmentation mask

**A close up of a logo

Description automatically generated**

Figure 10: Pigeon image contour ( erosion twice and dilation)

**A picture containing field

Description automatically generated**

Figure 11: Fox image R channel Otsu result (1 iteration)

**A close up

Description automatically generated**

Figure 12: Fox image G channel Otsu result (1 iteration)

**An animal looking at the camera

Description automatically generated**

Figure 13: Fox image B channel Otsu result (1 iteration)

**A picture containing standing, snow, hill, tall

Description automatically generated**

Figure 14: Fox image three channel combined segmentation mask

**A close up of a logo

Description automatically generated**

Figure 15: Fox image contour ( erosion and dilation)

**6.2 Task 2: Texture based Image Segmentation**

**A picture containing food

Description automatically generated**

Figure 16: Cat variance texture image’s Otsu result (3 iterations)

**A close up of a logo

Description automatically generated**

Figure 17: Cat variance texture image’s Otsu result (3 iterations)

**Map

Description automatically generated**

Figure 18: Cat variance texture image’s Otsu result (3 iterations)

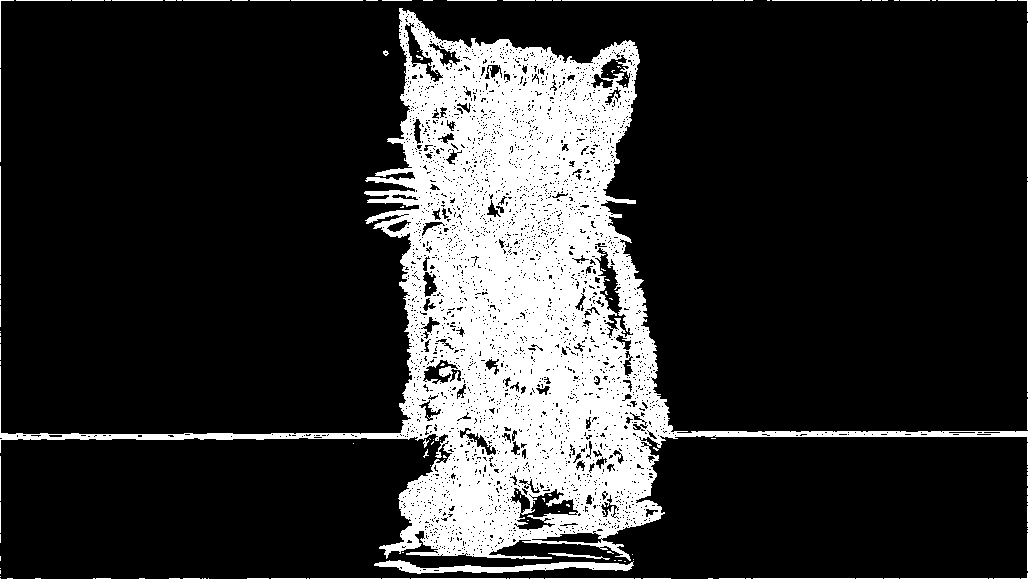
****

Figure 19: Cat texture image three channel combined segmentation mask

****

Figure 20: Cat texture image contour (erosion and dilation twice)

**A picture containing outdoor, water, standing, large

Description automatically generated**

Figure 21: Pigeon variance texture image’s Otsu result (2 iterations)

A picture containing outdoor, water, standing

Description automatically generated

Figure 22: Pigeon variance texture image’s Otsu result (2 iterations)



Figure 23: Pigeon variance texture image’s Otsu result (2 iterations)

A close up

Description automatically generated

Figure 24: Pigeon texture image three channel combined segmentation mask

A picture containing text

Description automatically generated

Figure 25: Pigeon texture image contour ( dilation twice)

A picture containing photo, front, grass, sitting

Description automatically generated

Figure 26: Fox variance texture image’s Otsu result (3 iterations)

A picture containing photo, sitting, front, standing

Description automatically generated

Figure 27: Fox variance texture image’s Otsu result (2 iterations)

A close up

Description automatically generated

Figure 28: Fox variance texture image’s Otsu result (3 iterations)

A picture containing photo, front, sitting, standing

Description automatically generated

Figure 29: fox texture image three channel combined segmentation mask

A picture containing sitting

Description automatically generated

Figure 30: Fox texture image contour (erosion twice and dilation twice)

**7 Observations**

For RGB-based image segmentation, the Otsu algorithm works well in the smooth regions, i.e. the regions with only few outstanding colors. In this case, the pigeon image where most of the body of pigeon is white, has a fair good segmentation result. However, for cat and fox images, where their body areas have significant color variance, the algorithm has a problem locating the image boundary.

For texture-based image segmentation, it works well in the case of image has abundant texture information, especially for the foreground has more texture information than the background, then the Otsu algorithm is prone to capture such foreground. The cat and fox image has a good performance on this texture-based segmentation method because of the animal foreground has large variance. The final contour image is vulnerable to the noise, so proper erosion and dilation refining methods are needed for binary mask to give a good contour image.

**8 Code**

*#!/usr/bin/env python  
# coding: utf-8*import numpy as np  
import cv2  
import matplotlib.pyplot as plt  
from scipy import signal  
  
  
def otsu(img, num\_iter, img\_dir, label):  
 Mask = np.zeros(img.shape, dtype = np.uint8)  
 channels = [**'B'**, **'G'**, **'R'**]  
 for i in range(3):  
 print(**'img'**, i+1)  
 gray = img[:, :, i]  
 temp = gray.flatten()  
 channel = channels[i]  
 for num in range(num\_iter[i]):  
 print(**'Iteration'**, num+1)  
 *# obtain the channel's histogram* hist, bin\_edges = np.histogram(temp, bins = 256, range = (0, 256))  
   
 *# obtain the overall weighted pixel value* sum\_all = np.sum(hist \* bin\_edges[0 : -1])  
 sum\_back = 0  
 sum\_fore = 0  
 num\_back = 0  
 num\_fore = 0  
 var\_best = 0  
 match = 0  
   
 *# iteratively calculate the inter-class variance over all available threshold,   
 # setting the threshold which give greatest inter-class variance* for j in range(256):  
 num\_back += hist[j]  
 num\_fore = np.sum(hist) - num\_back  
 sum\_back += hist[j] \* j  
 sum\_fore = sum\_all - sum\_back  
  
 if num\_back != 0 and num\_fore != 0:  
 avg\_back = sum\_back / num\_back  
 avg\_fore = sum\_fore / num\_fore  
 var = num\_back \* num\_fore \* (avg\_back - avg\_fore) \*\* 2  
  
 if var >= var\_best:  
 var\_best = var  
 match = j  
 print(match)  
 mask = np.zeros(gray.shape, dtype = np.uint8)  
 mask[gray > match] = 1  
   
 *# create the input for next iteration by removing pixel larger than threshold* temp1 = [n for n in temp if n <= match]   
 temp = temp1  
 plt.imshow(mask, cmap = **'gray'**)  
 *#plt.show()* cv2.imwrite(img\_dir + label + **'\_'** + channel + **'\_iteration'** + str(num+1) +**'.jpg'**, mask\*255)  
 Mask[:, :, i] = mask  
   
 return Mask  
  
def dilation(img, size, num):  
 kernel = np.ones((size, size))  
 mask = cv2.dilate(img, kernel, iterations = num)  
 return mask  
  
def erosion(img, size, num):  
 kernel = np.ones((size, size))  
 mask = cv2.erode(img, kernel, iterations = num)  
 return mask  
  
def get\_contour(mask\_all):  
 *# get the 8-neighbors to decide if a point is at contour* contour = np.zeros(mask\_all.shape, dtype = np.uint8)  
 for i in range(mask\_all.shape[0]):  
 for j in range(mask\_all.shape[1]):  
 if mask\_all[i, j] > 0:  
 neighbor = mask\_all[i-1 : i+2, j-1 : j+2]  
 if np.sum(neighbor.flatten()) < 9: *# not all 8-neighbors are 1 is valid contour point* contour[i, j] = 1  
   
 return contour  
  
  
  
  
def get\_texture(img):  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 mask = np.zeros(gray.shape, dtype = np.uint8)  
 mask\_all = np.zeros(img.shape, dtype = np.uint8)  
 layers = [3, 5, 7] *# three window size 3, 5, 7* for num in range(3):  
 N = layers[num]  
 edge = np.int((N - 1) / 2)  
   
 *# padding the original mask with edge of 0s* temp = np.zeros((mask.shape[0] + 2\*edge, mask.shape[1] + 2\*edge), dtype = np.uint8)  
 temp[edge:temp.shape[0]-edge, edge:temp.shape[1]-edge] = gray  
   
 for i in range(mask.shape[0]):  
 for j in range(mask.shape[1]):  
 x = i + edge  
 y = j + edge  
 window = temp[x - edge : x + edge + 1, y - edge : y + edge + 1]  
 mask[i, j] = np.var(window) *# assign the variance in the window to be pixel value* mask\_all[:, :, num] = mask  
   
 *# normalize the variance value to [0, 255]* mask\_min = np.min(mask\_all.flatten())  
 mask\_max = np.max(mask\_all.flatten())  
 mask\_all = np.uint8(np.around((mask\_all - mask\_min) / (mask\_max - mask\_min) \* 255 ))  
 return mask\_all  
  
  
  
directory = **"/home/xu1363/Documents/ECE 661/hw6/hw6\_images/"**file1 = **"cat.jpg"**file2 = **"pigeon.jpeg"**file3 = **"Red-Fox\_.jpg"**img1 = cv2.imread(directory+file1,cv2.IMREAD\_COLOR)  
img2 = cv2.imread(directory+file2,cv2.IMREAD\_COLOR)  
img3 = cv2.imread(directory+file3,cv2.IMREAD\_COLOR)  
  
  
  
img\_dir = **"/home/xu1363/Documents/ECE 661/hw6/output/img1/"**label = **'img1'**num\_iter = [1, 1, 1]  
mask = otsu(img1, num\_iter, img\_dir, label)  
mask\_all = mask[:, :, 0] \* mask[:, :, 1] \* mask[:, :, 2]  
  
print(**'Combine three channels'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_combined.jpg'**, mask\_all \* 255)  
  
mask\_all = erosion(mask\_all, 3, 1)  
print(**'erosion'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
mask\_all = dilation(mask\_all, 3, 1)  
print(**'dilation'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
print(**'Refined'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
contour = get\_contour(mask\_all)  
print(**'Contour'**)  
plt.imshow(contour \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_contour.jpg'**, contour \* 255)  
  
  
  
  
img\_dir = **"/home/xu1363/Documents/ECE 661/hw6/output/img2/"**label = **'img2'**num\_iter = [1, 1, 1]  
mask = otsu(img2, num\_iter, img\_dir, label)  
mask\_all = mask[:, :, 0] \* mask[:, :, 1] \* mask[:, :, 2]  
  
print(**'Combine three channels'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_combined.jpg'**, mask\_all \* 255)  
  
mask\_all = erosion(mask\_all, 5, 2)  
print(**'erosion'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
mask\_all = dilation(mask\_all, 5, 1)  
print(**'dilation'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
print(**'Refined'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
contour = get\_contour(mask\_all)  
print(**'Contour'**)  
plt.imshow(contour \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_contour.jpg'**, contour \* 255)  
  
  
  
  
img\_dir = **"/home/xu1363/Documents/ECE 661/hw6/output/img3/"**label = **'img3'**num\_iter = [1, 1, 1]  
mask = otsu(img3, num\_iter, img\_dir, label)  
mask\_all = mask[:, :, 0] \* mask[:, :, 1] \* mask[:, :, 2]  
  
print(**'Combine three channels'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_combined.jpg'**, mask\_all \* 255)  
  
mask\_all = erosion(mask\_all, 3, 1)  
print(**'erosion'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
mask\_all = dilation(mask\_all, 3, 1)  
print(**'dilation'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
print(**'Refined'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
contour = get\_contour(mask\_all)  
print(**'Contour'**)  
plt.imshow(contour \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_contour.jpg'**, contour \* 255)  
  
  
  
  
*# get the texture information for N = 3, 5, 7*img1\_texture = get\_texture(img1)  
img2\_texture = get\_texture(img2)  
img3\_texture = get\_texture(img3)  
  
  
  
  
img\_dir = **"/home/xu1363/Documents/ECE 661/hw6/output/img1/"**label = **'img1texture'**num\_iter = [3, 3, 3]  
mask = otsu(img1\_texture, num\_iter, img\_dir, label)  
mask\_all = mask[:, :, 0] \* mask[:, :, 1] \* mask[:, :, 2]  
  
print(**'Combine three channels'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_combined.jpg'**, mask\_all \* 255)  
  
mask\_all = erosion(mask\_all, 3, 1)  
print(**'erosion'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
mask\_all = dilation(mask\_all, 5, 2)  
print(**'dilation'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
print(**'Refined'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
contour = get\_contour(mask\_all)  
print(**'Contour'**)  
plt.imshow(contour \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_contour.jpg'**, contour \* 255)  
  
  
  
  
img\_dir = **"/home/xu1363/Documents/ECE 661/hw6/output/img2/"**label = **'img2texture'**num\_iter = [2, 2, 2]  
mask = otsu(img2\_texture, num\_iter, img\_dir, label)  
mask\_all = mask[:, :, 0] \* mask[:, :, 1] \* mask[:, :, 2]  
  
print(**'Combine three channels'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_combined.jpg'**, mask\_all \* 255)  
  
*#mask\_all = erosion(mask\_all, 5, 1)*print(**'erosion'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
mask\_all = dilation(mask\_all, 5, 2)  
print(**'dilation'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
  
print(**'Refined'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
contour = get\_contour(mask\_all)  
print(**'Contour'**)  
plt.imshow(contour \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_contour.jpg'**, contour \* 255)  
  
  
  
  
img\_dir = **"/home/xu1363/Documents/ECE 661/hw6/output/img3/"**label = **'img3texture'**num\_iter = [3, 2, 3]  
mask = otsu(img3\_texture, num\_iter, img\_dir, label)  
mask\_all = mask[:, :, 0] \* mask[:, :, 1] \* mask[:, :, 2]  
  
print(**'Combine three channels'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_combined.jpg'**, mask\_all \* 255)  
  
  
  
mask\_all = dilation(mask\_all, 3, 2)  
print(**'dilation'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
mask\_all = erosion(mask\_all, 5, 2)  
print(**'erosion'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
print(**'Refined'**)  
plt.imshow(mask\_all \* 255, cmap = **'gray'**)  
plt.show()  
  
contour = get\_contour(mask\_all)  
print(**'Contour'**)  
plt.imshow(contour \* 255, cmap = **'gray'**)  
plt.show()  
cv2.imwrite(img\_dir + label + **'\_contour.jpg'**, contour \* 255)