Prateek Gupta

gupta198@purdue.edu

ECE 661 Homework 6

**Theory Question**

Lecture 14 will present two very famous algorithms for image segmentation: The Otsu Algorithm and the Watershed Algorithm. These algorithms are as different as night and day. Present in your own words the strengths and the weaknesses of each. (Note that the Watershed algorithm uses the morphological operators that we will discuss in Lecture 13.)

The watershed algorithm is fairly simple to understand, and it is very intuitive. It is an algorithm that can easily be understood by people without a strong technical background. However, since the watershed algorithm starts by finding the smallest contour lines first, it can lead to over segmentation. Additionally, if there are very thin structures, the watershed algorithm is likely to not detect them. The Otsu algorithm is really good at picking up the slack of the watershed method, but it does have some downsides. Since it relies on having a distinct difference in intensity between background and foreground pixels, if the intensity does not have a bimodal distribution, Otsu will not work. Additionally, if the image is unequally lit, Otsu will have difficulty.

**Overview**

In this assignment, we will attempt to separate the foreground of an image from the background. This will be accomplished by separating each image into its red, green, and blue channels. Then, we will use the OTSU algorithm, and recombine the images on completion. There are two techniques for this. Each one will be discussed and implemented. Then, once the images have been segmented, we can extract the contours of the image and develop a contour map.

**Otsu’s Algorithm**

Otsu’s algorithm finds the threshold of gray values for distinguishing the foreground from the background. Of course, this assumes that there is indeed such a threshold that will allow us to make the discrimination. It does this by looking at the bimodal distribution of gray levels in a binary image and finds the threshold that maximizes the variance between the two classes. So, we need find the weighted sum of the variances within the two classes. This is done with the following equation

Here, the weights applied to each variance are computed by totaling the probabilities of a particular threshold t.

Since our goal is to maximize the variance between classes, we need to obtain an equation that represents the variance between classes. That is given below

Here, μ0, μ1, μT, are the mean values of class 0, class 1, and the overall data, respectively. The threshold is computed for each gray value, between 0 and 255, and the maximum threshold is kept.

**RGB Segmentation**

RGB segmentation utilizes the Otsu method. First, as the name implies, we separate an image into its red, green, and blue channels. Then we can utilize Otsu to find the threshold between the foreground and background and develop a mask for the channel. Then all three masks are combined using the logical AND operator. This may not yield the best result, so we can play with how many times we run Otsu for each channel.

**Texture Based Segmentation**

In texture-based segmentation, we calculate the variance of the image pixels using a sliding window of dimension N x N pixels. Here N is chosen to be 3, 5, or 7. We then apply the Otsu method to create a mask for each of the three window sizes, and then merge the three using the logical AND.

**Noise Removal**

Whether using RGB segmentation, or texture-based segmentation, there will likely be noise. This can be removed by applying a dilation and erosion several times. The amount of times varies, so we will do this as often as necessary, until there is minimal noise.

**Contour Extraction**

Once the images have been successfully segmented, whether through RGB texture based, and the noise removed, the contour can be extracted. We will perform a raster scan on the image, and if we find a foreground pixel whose 8-neighbors are not foreground pixels as well, we have found a border pixel, and this can be marked.

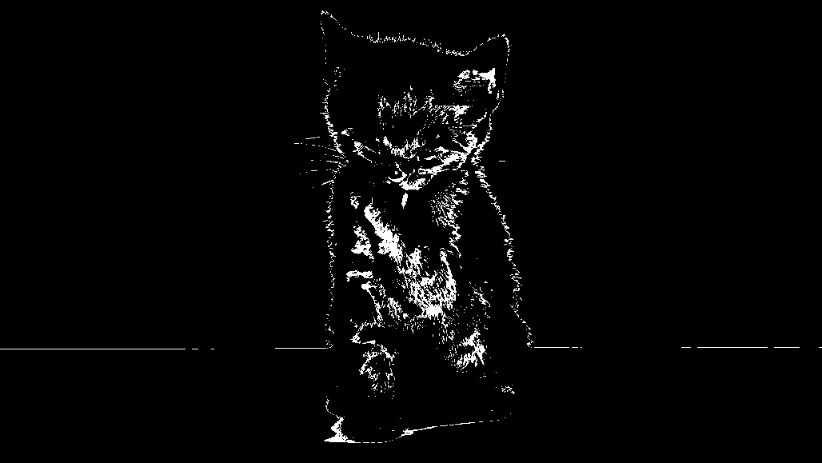
**Results**



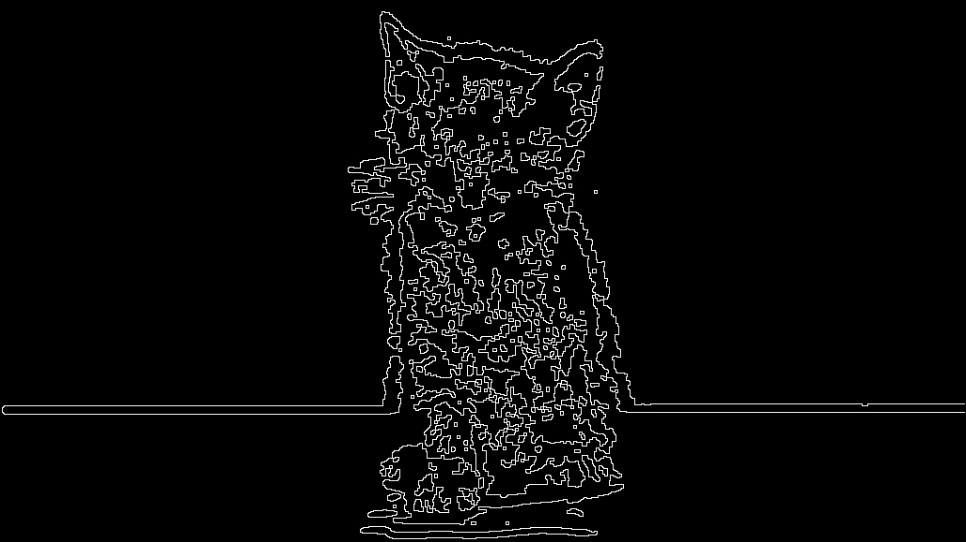
Original Image 1



RGB Segmentation



Texture Segmentation



Contour



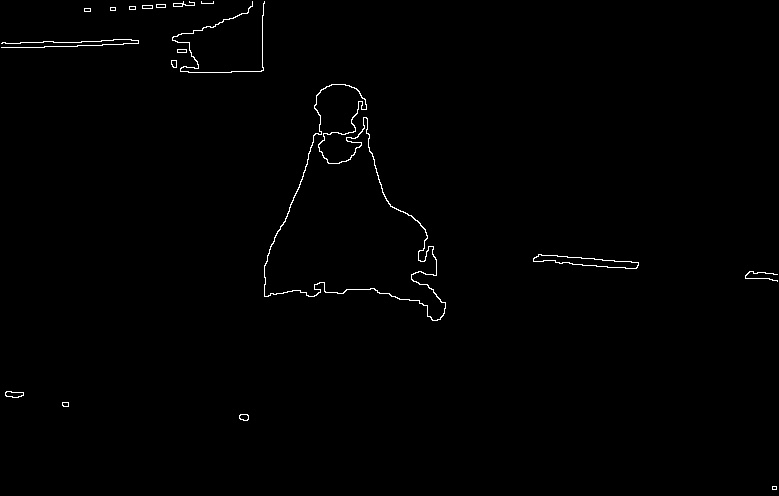
Original Image 2



RGB Segmentation



Texture Segmentation



Contour



Original Image 3



RGB Segmentation



Texture Segmentation



Contour

import numpy as np

import cv2

import matplotlib.pyplot as plt

def otsu(im):

hist, edges = np.histogram(im.flatten(), bins = 256, range=(0, 256)) #magic

numBack = 0; backSum = 0

N = im.shape[0] \* im.shape[1]

total = sum(hist \* edges[:-1])

maximum = 0

for i in range(len(hist)):

numBack += hist[i]

numFore = N - numBack

if numBack > 0 and numFore > 0:

backSum += i\*hist[i]

foreSum = total - backSum

t = numBack \* numFore \* (backSum/numBack - foreSum/numFore)\*\*2

if t >= maximum:

threshold = i

maximum = t

return threshold

def rgbSeg(im):

threshold = otsu(im)

mask1 = np.zeros(im.shape)

mask2 = np.zeros(im.shape)

for i in range(im.shape[0]):

for j in range(im.shape[1]):

if im[i][j] > threshold:

mask1[i][j] = 255

else:

mask2[i][j] = 255

return [mask1, mask2]

def texture(im, N):

var = np.zeros(im.shape)

for i in range(im.shape[0]):

for j in range(im.shape[1]):

window = im[max(0, i - N/2) : min(im.shape[0], i + N/2 + 1)][max(0, j - N/2): min(im.shape[1], j + N/2 + 1)]

var[i, j] = np.var(window)

return np.uint8(var)

def contour(im):

mask = np.zeros(im.shape)

for i in range(im.shape[0]):

for j in range(im.shape[1]):

if im[i,j] > 0:

mask[i,j] = 255

return mask

im = cv2.imread("Images/cat.jpg")

imBlue = im[:,:,0]

imGreen = im[:,:,1]

imRed = im[:,:,2]

otsuRed1, otsuRed2 = rgbSeg(imRed)

otsuGreen1, otsuGreen2 = rgbSeg(imGreen)

otsuBlue1, otsuBlue2 = rgbSeg(imBlue)

rgb = otsuRed1 \* otsuGreen2 \* otsuBlue2

cv2.imwrite("Results/catRGB.jpg", rgb)

im = cv2.imread("Images/pigeon.jpeg")

imBlue = im[:,:,0]

imGreen = im[:,:,1]

imRed = im[:,:,2]

otsuRed1, otsuRed2 = rgbSeg(imRed)

otsuGreen1, otsuGreen2 = rgbSeg(imGreen)

otsuBlue1, otsuBlue2 = rgbSeg(imBlue)

rgb = otsuRed1 \* otsuGreen1 \* otsuBlue1

cv2.imwrite("Results/pigeonRGB.jpg", rgb)

im = cv2.imread("Images/Red-Fox\_.jpg")

imBlue = im[:,:,0]

imGreen = im[:,:,1]

imRed = im[:,:,2]

otsuRed1, otsuRed2 = rgbSeg(imRed)

otsuGreen1, otsuGreen2 = rgbSeg(imGreen)

otsuBlue1, otsuBlue2 = rgbSeg(imBlue)

rgb = otsuRed1 \* otsuGreen2 \* otsuBlue1

cv2.imwrite("Results/foxRGB.jpg", rgb)

im = cv2.imread("Images/cat.jpg")

imGray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

var = texture(imGray, 3)

mask13, mask23 = rgbSeg(var)

var = texture(imGray, 5)

mask15, mask25 = rgbSeg(var)

var = texture(imGray, 7)

mask17, mask27 = rgbSeg(var)

tot = mask13 \* mask15 \* mask17

cv2.imwrite("Results/catText.jpg", tot)

im = cv2.imread("Images/pigeon.jpeg")

imGray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

var = texture(imGray, 3)

mask13, mask23 = rgbSeg(var)

var = texture(imGray, 5)

mask15, mask25 = rgbSeg(var)

var = texture(imGray, 7)

mask17, mask27 = rgbSeg(var)

tot = mask13 \* mask15 \* mask17

cv2.imwrite("Results/pigeonText.jpeg", tot)

im = cv2.imread("Images/Red-Fox\_.jpg")

imGray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

var = texture(imGray, 3)

mask13, mask23 = rgbSeg(var)

var = texture(imGray, 5)

mask15, mask25 = rgbSeg(var)

var = texture(imGray, 7)

mask17, mask27 = rgbSeg(var)

tot = mask13 \* mask15 \* mask17

cv2.imwrite("Results/foxText.jpg", tot)