ECE 661 Homework 6

Suyash Ail
sail@purdue.edu

October 12, 2020

The task of this homework is to implement an automatic thresholding method to separate the foreground from the background.We use the famous Otsu's algorithm to determine thresholds on the RGB channel and texture patterns.

1 Theory Questions

Question1

1. Otsu method is based on global thresholding of images and hence works only for a small domain of applications. Watershed algorithm is based on local thresholding hence more adaptive.

2. Otsu method considers the image to have a bimodal distribution. It is not suitable for segmentation in domains other than brightness. Watershed assumes image to be a 3d surface with pixel values as the z-axis. It performs well in case of poor contrast.

3. Otsu is widely used for its simple and fast implementation of binary foreground-background segmentaion. However it does not work well with multiple objects. Watershed is popular due to its fast implementation and does not have the limitation of multiple object segmentation.

4. Many of the watershed variants suffer from the problem of oversegmentation and are highly susceptible to noise. Otsu performs significantly well in images having noise.

2 Logic

2.1 Otsu' Algorithm

Ostu's algorithm is a thresholding technique based on clustering of image pixels. It separates the gray level into two classes-foreground and background.

There are three discriminant criterion in the algorithm: $\frac{\sigma_B^2}{\sigma_W^2}, \frac{\sigma_T^2}{\sigma_W^2}$ and $\frac{\sigma_B^2}{\sigma_T^2}$. σ_T^2 is independent of the threshold k. σ_W^2 is the within-class variance given by

$$\sigma_W^2 = \omega_0\sigma_0^2 + \omega_1\sigma_1^2$$

Here ω_0 and ω_1 are the class probabilities and $\omega_0 + \omega_1 = 1.\sigma_0^2$ and σ_1^2 are the class variance. σ_B^2 is the between class variance given by

$$\sigma_B^2 = \omega_0\omega_1(\mu_0 - \mu_1)^2$$

Here μ_0 and μ_1 are the class means.
Because σ_W^2 is of second order, the Otsu's algorithm uses σ_B^2 as its criterion for optimization. σ_B^2 is of order 1. The goal for Ostu's algorithm is to minimize the between class variance. That is find the optimum value k^* that will minimise the between class variance.

Steps:

1. Preprocess the image and extract the three channels -R,G and B. I have also used the inverse channels (255-R),(255-G),(255-B). For texture channels, I have obtained three channels N=3, N=5 and N=7.

2. For each channel setup a 256-bin histogram between the minimum and maximum value of pixel in the image.The 256 element long vector $C = (C_0, C_1...C_{255})$.

3. Iteratively set k_{index} from 0 to 255 such that all pixels smaller than the threshold represent the background class and all pixels greater than threshold represent the foreground class. the frequency of the background is given by

$$\omega_0 = \frac{\Sigma_{i=0}^{k_{index}} C_i}{\Sigma_{i=0}^{255} C_i}$$

   And the frequency of the foreground class is given by

$$\omega_1 = \frac{\Sigma_{k_{index+1}}^{255} C_i}{\Sigma_{i=0}^{255} C_i}$$

   The mean of the background class is given by

$$\mu_0 = \frac{\Sigma_{i=0}^{k_{index}} (C_i.i)}{\Sigma_{i=0}^{255} (C_i.i)}$$

   The mean of the foreground class is given by

$$\mu_1 = \frac{\Sigma_{k_{index+1}}^{255} (C_i.i)}{\Sigma_{i=0}^{255} (C_i.i)}$$

   The optimization problem is formulated as

$$k^*_{\text{index}} = \underset{k_{\text{index}}}{\text{argmax}}(\sigma_b^2) = \underset{k_{\text{index}}}{\text{argmax}}(\omega_0\omega_1 \left(\mu_0 - \mu_1\right)^2) \tag{1}$$

4. Use the optimal k to perform image segmentation. If the results are not satisfactory, perform the algorithm again using the thresholded image as input.

5. The final mask is obtained by taking element-wise AND of the three channels.

2.2 Texture-based Features

The purpose of texture based features is to provide better features than the R,G,B channels for image segmentation. The hypothesis is that the background and foreground have different neighbourhood variances. The neighbourhood is defined by using a 3x3, 5x5 and 7x7 window. For each sliding window across the gray scale image, we find the variance of pixels within the window and assign the center pixel the value. For the edge pixels, a zero padding of size (1,2,3) has been done on the grayscale image. After the window process, the additional padding pixels are removed.

2.3 Contour Extraction

For contour extraction, we look at the 8 conected neighbours of the pixel represented as foreground (255). If all the 8 neighbours are foreground, then the pixel is not considered in the contour image. If not all 8-neighbours are foreground, then the pixel is considered a contour pixel. Before performing contouring, we need to deal with the noise in the segmented image. We perform erosion followed by dilation with different smoothening window to get the best results.

2.4 Steps

2.4.1 RGB method

1. Separate image into R,G,B channels
2. Create a mask for each of the channels by using iterative Otsu's algorithm.
3. Merge the masks into a singe mask
4. Eliminate noise by erosion-dilation
5. Extract the contours

2.4.2 Threshold-based method

1. Convert the input image to grayscale
2. Compute the variance image using 3x3,5x5 and 7x7 window.
3. Create a mask for each of the variance images using iterative Otsu
4. Merge the masks into one single mask
5. Remove noise using erosion-dilation
6. Extract the contours

3 Observations

1. The RGB method worked well for detecting the foreground objects. But it was not able to completely fill all foreground pixels. Hence we only see partly filled images of fox, cat and pigeon.

2. The contour detection gave good results for RGB method. 8-neighbourhood was used which gave good contours.

3. Too much tuning was required and heavy use of erosion and dilation to reduce noise in the images. Even then noise is still present.

4. RGB method performed very well on pigeon image on detecting pigeon as the foreground.

5. The texture based method performed vastly superior to the RGB method. The textures detected in the fox and cat image largely represented the foreground object and hence Otsu on the texture image gave very good segmentation results.

6. The texture based method did not do a good job on the pigeon image as the textures detected a lot of noise in the background. On reducing noise using erosion-dilation, some part of the pigeon also disappeared. Here RGB method performed better.

4 Results

4.1 Fox

	Rounds	Invert	Erosion	Dilation
R Channel	4	No	3x3,1	5x5,1
G Channel	2	Yes	3x3,1	5x5,1
B Channel	6	No	3x3,1	5x5,1

4.1.1 RGB

Figure 1: R channel

Figure 2: G channel

Figure 3: B channel

Figure 4: R mask (k=130)

Since the image contains lot of green pixels of grass, we want to neglect it in foreground. Hence we will use inverse channel output (255-G)

Figure 5: G mask (k=105)

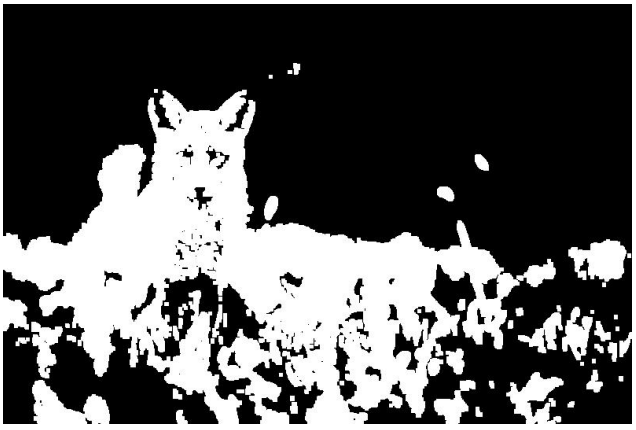

Figure 6: B mask (k=51)

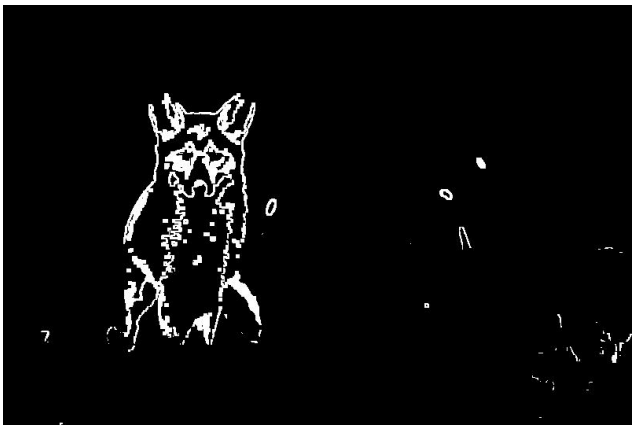

Figure 7: Segmented Image

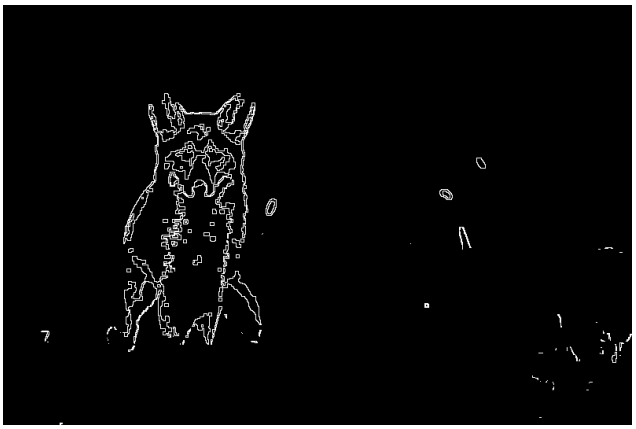

Figure 8: Contour Image-8 neighbourhood

4.1.2 Texture based

	Rounds	Invert	Erosion	Dilation
3x3	1	No	3x3,1	5x5,2
5x5	2	No	3x3,1	5x5,2
7x7	2	No	3x3,1	5x5,2

Figure 9: Texture Image for 3x3 window

Figure 10: Texture Image for 5x5 window

Figure 11: Texture Image for 7x7 window

Figure 12: Mask for 3x3 texture image

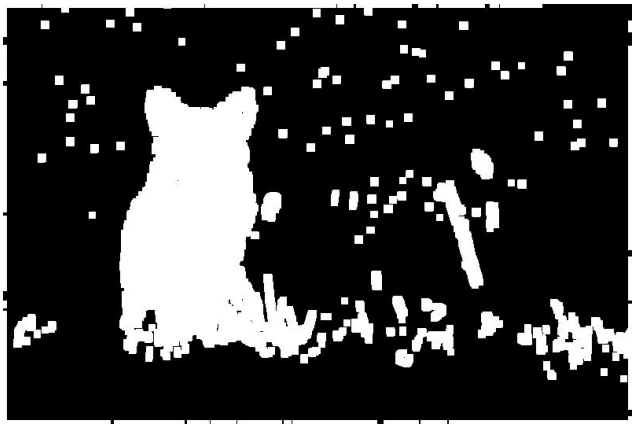

Figure 13: Mask for 5x5 texture image

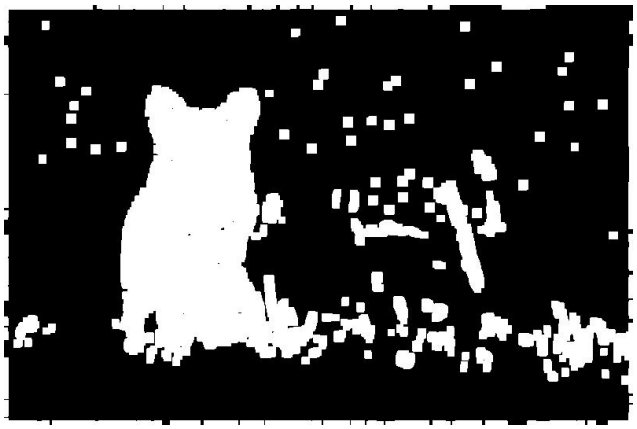

Figure 14: Mask for 7x7 texture image

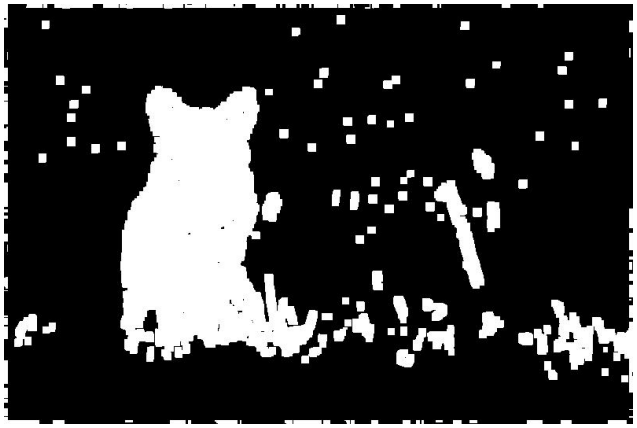

Figure 15: Segmented image

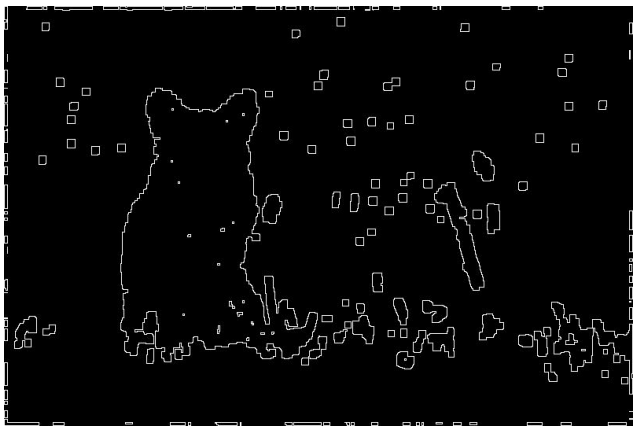

Figure 16: Contour Image

4.2 Cat

4.2.1 RGB

	Rounds	Invert	Erosion	Dilation
R Channel	2	No	5x5,1	5x5,2
G Channel	1	Yes	5x5,1	5x5,2
B Channel	1	Yes	5x5,1	5x5,2

Figure 17: R channel

Figure 18: G channel

Figure 19: B channel

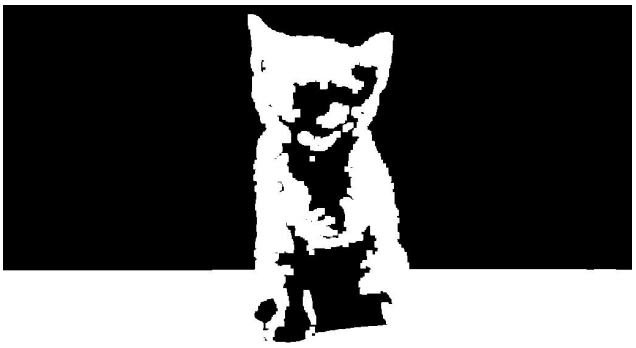

Figure 20: R mask (k=211)

Since the image predominantly contains only red color, I have inverted the blue and green channels to keep red cat as background.

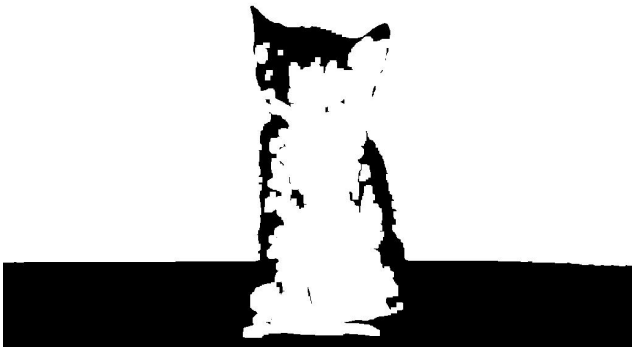

Figure 21: G mask (k=68)

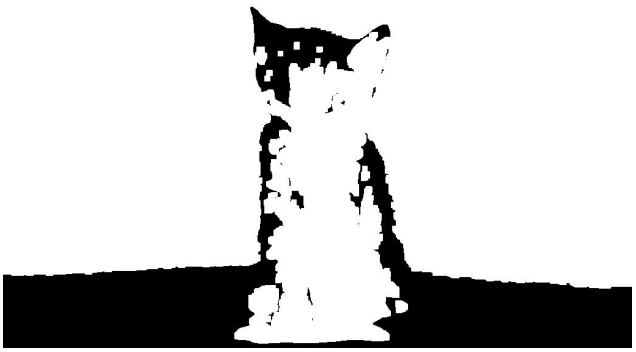

Figure 22: B mask (k=108)

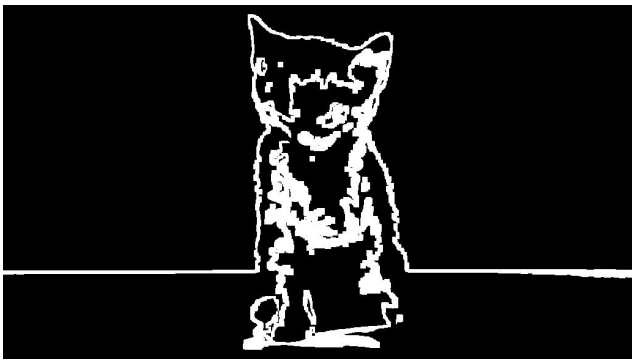

Figure 23: Segmented Image

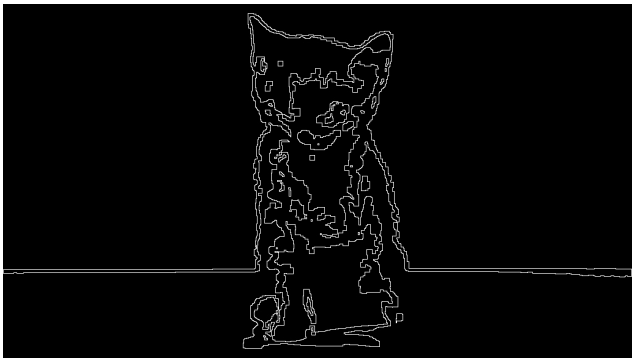

Figure 24: Contour Image-8 neighbourhood

4.2.2 Texture based

	Rounds	Invert	Erosion	Dilation
3x3	1	No	3x3,1	5x5,2
5x5	3	No	3x3,1	5x5,2
7x7	3	No	3x3,1	5x5,2

Figure 25: Texture Image for 3x3 window

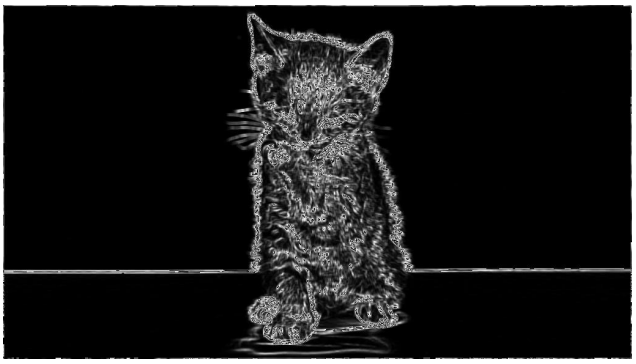

Figure 26: Texture Image for 5x5 window

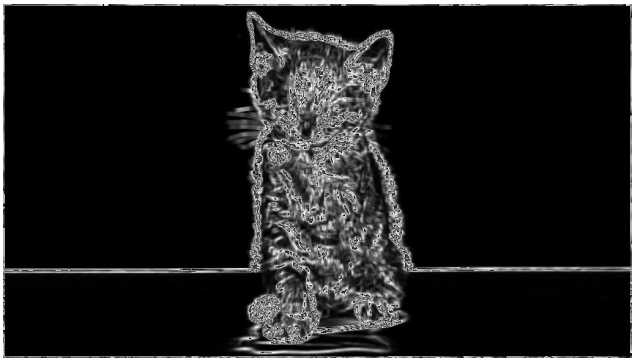

Figure 27: Texture Image for 7x7 window

Figure 28: Mask for 3x3 texture image

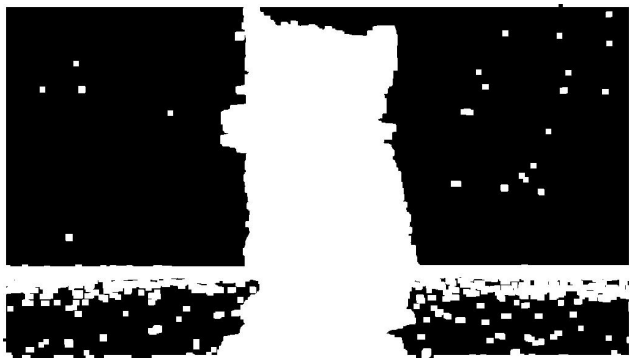

Figure 29: Mask for 5x5 texture image

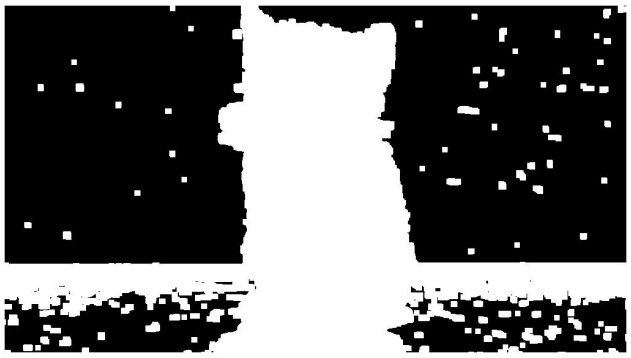

Figure 30: Mask for 7x7 texture image

Figure 31: Segmented image

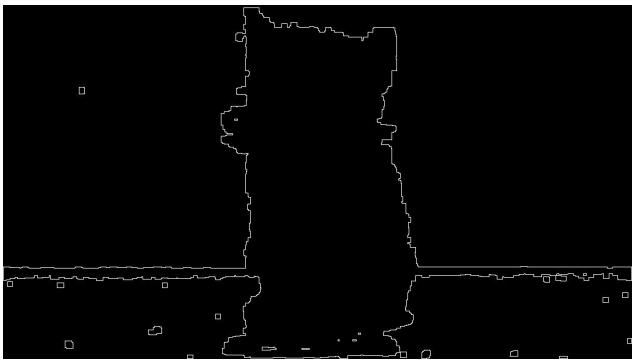

Figure 32: Contour Image

4.3 Pigeon

4.3.1 RGB

	Rounds	Invert	Erosion	Dilation
R Channel	2	No	3x3,1	3x3,1
G Channel	2	No	3x3,1	3x3,1
B Channel	1	No	3x3,1	3x3,1

Figure 33: R channel

Figure 34: G channel

Figure 35: B channel

Figure 36: R mask (k=169)

Figure 37: G mask (k=182)

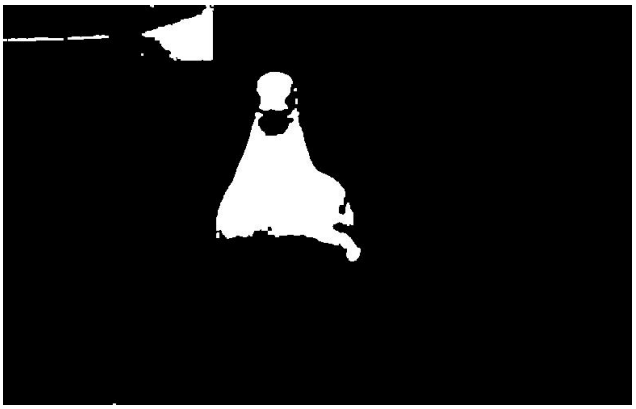

Figure 38: B mask (k=196)

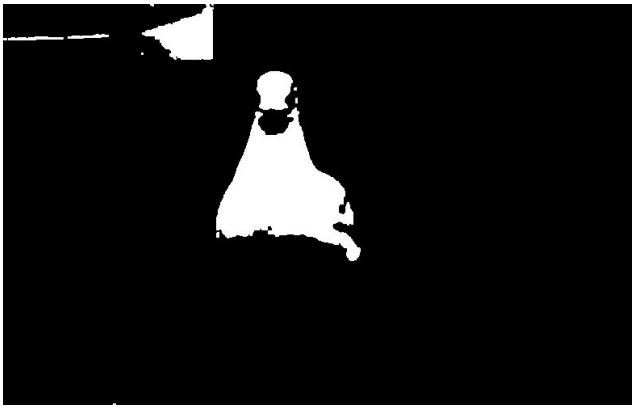

Figure 39: Segmented Image

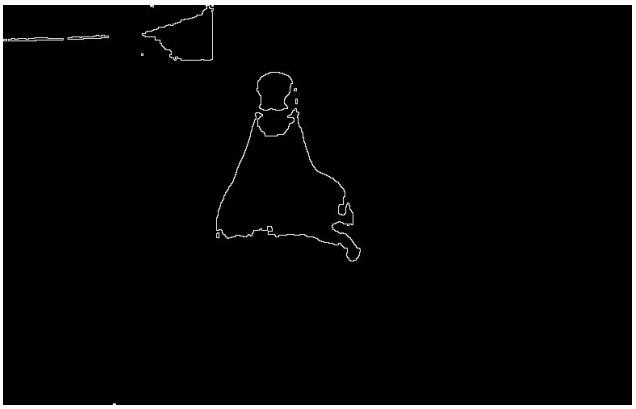

Figure 40: Contour Image-8 neighbourhood

4.3.2 Texture based

	Rounds	Invert	Erosion	Dilation
3x3	2	Yes	3x3,2	5x5,1
5x5	2	Yes	3x3,2	5x5,1
7x7	1	Yes	3x3,2	5x5,1

Figure 41: Texture Image for 3x3 window

Figure 42: Texture Image for 5x5 window

Figure 43: Texture Image for 7x7 window

Since the texture images recognized majoring the background pixels, we will invert all the three channels and then perform thresholding.

Figure 44: Mask for 3x3 texture image

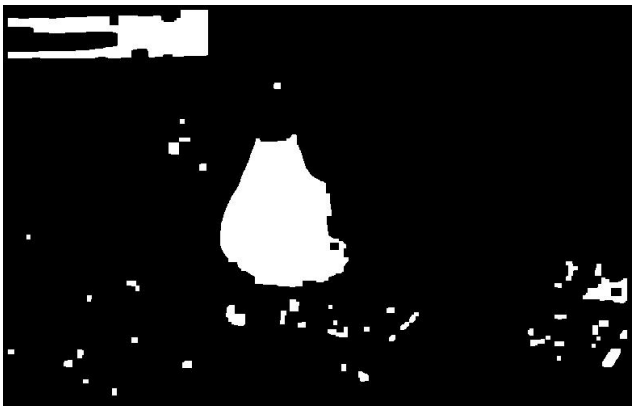

Figure 45: Mask for 5x5 texture image

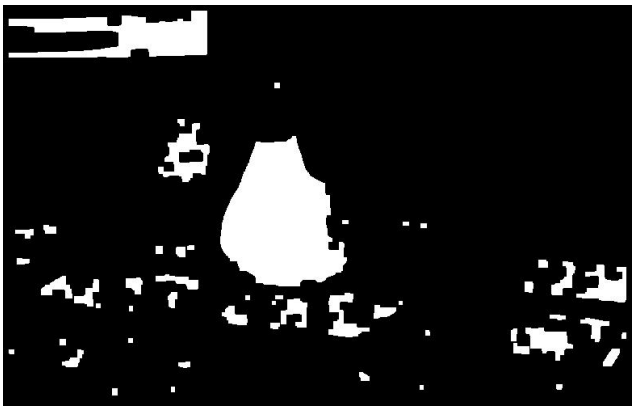

Figure 46: Mask for 7x7 texture image

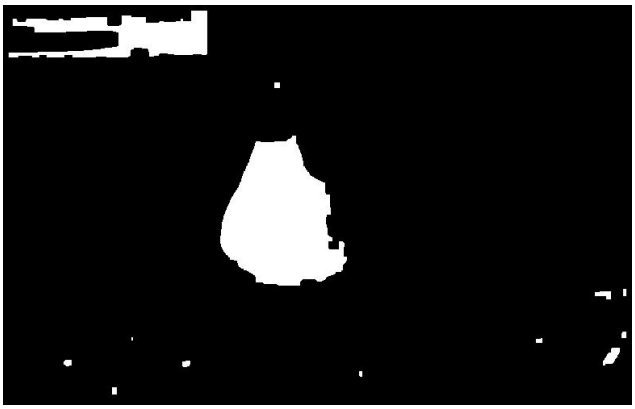

Figure 47: Segmented image

Figure 48: Contour Image

5 Code Listings

```
# -*- coding: utf-8 -*-
"""hw6_Suyash_Ail.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1QiHfeA59Hqlf4nSCdlwsuYW6alHmCMje
"""

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

def otsu(image_mono, round=False,rounds=1):
  min_threshold=0
  k_rounds=[]
  mask_img_rounds=[]
  for rnd in range(rounds):
    #count,_= np.histogram(image_mono,bins=256-min_threshold, range=(0,255))
    count,_= np.histogram(image_mono.flatten(),bins=np.arange(min_threshold
    -0.5,256))
    freq, sigma_bsquare = np.zeros((256,)),np.zeros((256,))
```

```
    freq[min_threshold:]=count/np.max(count)
    for i in range(min_threshold,256):
      omega0 = np.sum(freq[min_threshold:i+1])
      omega1 = np.sum(freq[i+1:])
      mu0 = np.sum(freq[min_threshold:i+1]*np.arange(min_threshold,i+1))
      mu1 = np.sum(freq[i+1:]*np.arange(i+1,256))
      sigma_bsquare[i] = omega0 * omega1 * (mu0-mu1) * (mu0-mu1)
    k_opt = np.argmax(sigma_bsquare)
    #k_rounds.append(k_opt)
    mask = (image_mono>k_opt)
    #mask_img_rounds.append(mask)
    if round:
      min_threshold=k_opt
    else:
      break

  #return mask_img_rounds,k_rounds
  return np.uint8(mask),k_opt

############## trial ################
def get_otsu_image(image,round=False,rounds=[1,1,1],invert=[0,0,0],erosion
    =[3,1],dialation=[3,1]):
  R,G,B = image[:,:,0],image[:,:,1],image[:,:,2]
  Ri=255-R;Gi=255-G;Bi=255-B
  if invert[0]==1:
    R_masks,R_k=otsu(Ri,round,rounds[0])
  else:
    R_masks,R_k=otsu(R,round,rounds[0])
  if invert[1]==1:
    G_masks,G_k=otsu(Gi,round,rounds[1])
  else:
    G_masks,G_k=otsu(G,round,rounds[1])
  if invert[2]==1:
    B_masks,B_k=otsu(Bi,round,rounds[2])
  else:
    B_masks,B_k=otsu(B,round,rounds[2])

  ############ erosion on mask images ####################
  erosion_win=erosion[0]
  erosion_iterations=erosion[1]
  erosion_kernel = np.ones((erosion_win,erosion_win),np.uint8)
  R_masks = cv2.erode(R_masks,erosion_kernel,iterations = erosion_iterations)
  G_masks = cv2.erode(G_masks,erosion_kernel,iterations = erosion_iterations)
  B_masks = cv2.erode(B_masks,erosion_kernel,iterations = erosion_iterations)

  ############## dialation on eroded images ###########
  dialation_win=dialation[0]
  dialation_iterations=dialation[1]
  dialation_kernel = np.ones((dialation_win,dialation_win),np.uint8)
  R_masks = cv2.dilate(R_masks,dialation_kernel,iterations =
    dialation_iterations)
  G_masks = cv2.dilate(G_masks,dialation_kernel,iterations =
    dialation_iterations)
  B_masks = cv2.dilate(B_masks,dialation_kernel,iterations =
    dialation_iterations)


```

```
76
77    plt.subplot(1,3,1)
78    plt.imshow(R_masks,cmap='gray')
79    seg_img = cv2.bitwise_and(G_masks,B_masks)
80    #seg_img = np.logical_and(G,B)
81    plt.subplot(1,3,2)
82    plt.imshow(G_masks,cmap='gray')
83    seg_img = cv2.bitwise_and(seg_img,R_masks)
84    #seg_img = np.logical_and(seg_img,R)
85    plt.subplot(1,3,3)
86    plt.imshow(B_masks,cmap='gray')
87    plt.imsave("R_mask.jpg",R_masks,cmap='gray')
88    plt.imsave("G_mask.jpg",G_masks,cmap='gray')
89    plt.imsave("B_mask.jpg",B_masks,cmap='gray')
90    return seg_img,[R_k,G_k,B_k]
91
92  def get_contours(image):
93    contour_img=np.zeros_like(image)
94    for i in range(contour_img.shape[0]):
95      for j in range(contour_img.shape[1]):
96        if image[i,j]==1:
97          window_img=image[i-1:i+2,j-1:j+2]
98          if np.sum(window_img)!=9:
99            contour_img[i,j]=255
100   return contour_img
101
102 im1 = cv2.imread("/content/drive/My Drive/hw6_images/Red-Fox_.jpg")
103 im1=cv2.cvtColor(im1, cv2.COLOR_BGR2RGB)
104 plt.imshow(im1)
105 im1_gray = cv2.cvtColor(im1,cv2.COLOR_RGB2GRAY)
106
107 (R, G, B) = (im1[:,:,0], im1[:,:,1], im1[:,:,2])
108 (iB, iG, iR) = (255-B, 255-G, 255-R)
109 plt.figure(figsize=(10,10))
110 plt.subplot(1,3,1)
111 plt.imshow(R,cmap='gray')
112 plt.axis('off')
113 plt.title("red")
114 plt.subplot(1,3,2)
115 plt.imshow(G,cmap='gray')
116 plt.axis('off')
117 plt.title("green")
118 plt.subplot(1,3,3)
119 plt.imshow(B,cmap='gray')
120 plt.axis('off')
121 plt.title("blue")
122 cv2.imwrite("fox_red.jpg",R)
123 cv2.imwrite("fox_green.jpg",G)
124 cv2.imwrite("fox_blue.jpg",B)
125
126 plt.figure(figsize=(20,40))
127 seg_image,[R_k,G_k,B_k]=get_otsu_image(im1,round=True,rounds=[5,2,6],invert
      =[0,1,0],erosion=[3,1],dialation=[5,1])
128 plt.figure()
129 plt.imshow(seg_image.astype(int),cmap='gist_gray')
130 print(R_k,G_k,B_k)
131 plt.imsave("fox_seg.jpg",seg_image,cmap='gray')
132
```

```
contour_image = get_contours(seg_image)
plt.imshow(contour_image,cmap='gray')
cv2.imwrite("fox_contour.jpg",contour_image)

im2 = cv2.imread("/content/drive/My Drive/hw6_images/cat.jpg")
im2=cv2.cvtColor(im2, cv2.COLOR_BGR2RGB)
plt.imshow(im2)
im2_gray = cv2.cvtColor(im2,cv2.COLOR_RGB2GRAY)

(R, G, B) = (im2[:,:,0], im2[:,:,1], im2[:,:,2])
(iB, iG, iR) = (255-B, 255-G, 255-R)
plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(R,cmap='gray')
plt.axis('off')
plt.title("red")
plt.subplot(1,3,2)
plt.imshow(G,cmap='gray')
plt.axis('off')
plt.title("green")
plt.subplot(1,3,3)
plt.imshow(B,cmap='gray')
plt.axis('off')
plt.title("blue")
cv2.imwrite("cat_red.jpg",R)
cv2.imwrite("cat_green.jpg",G)
cv2.imwrite("cat_blue.jpg",B)

plt.figure(figsize=(20,40))
seg_image2,[R_k,G_k,B_k]=get_otsu_image(im2,round=True,rounds=[7,1,1],invert
    =[0,1,1],erosion=[5,1],dialation=[5,2])
plt.figure()
plt.imshow(seg_image2.astype(int),cmap='gist_gray')
print(R_k,G_k,B_k)
plt.imsave("cat_seg.jpg",seg_image2,cmap='gray')

contour_image2 = get_contours(seg_image2)
plt.imshow(contour_image2,cmap='gray')
cv2.imwrite("cat_contour.jpg",contour_image2)

im3 = cv2.imread("/content/drive/My Drive/hw6_images/pigeon.jpeg")
im3=cv2.cvtColor(im3, cv2.COLOR_BGR2RGB)
plt.imshow(im3)
im3_gray = cv2.cvtColor(im3,cv2.COLOR_RGB2GRAY)

(R, G, B) = (im3[:,:,0], im3[:,:,1], im3[:,:,2])
(iB, iG, iR) = (255-B, 255-G, 255-R)
plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(R,cmap='gray')
plt.axis('off')
plt.title("red")
plt.subplot(1,3,2)
plt.imshow(G,cmap='gray')
plt.axis('off')
plt.title("green")
plt.subplot(1,3,3)
plt.imshow(B,cmap='gray')
```

```
plt.axis('off')
plt.title("blue")
cv2.imwrite("pig_red.jpg",R)
cv2.imwrite("pig_green.jpg",G)
cv2.imwrite("pig_blue.jpg",B)

plt.figure(figsize=(20,40))
seg_image3,[R_k,G_k,B_k]=get_otsu_image(im3,round=True,rounds=[15,19,20],
    invert=[0,0,0])
plt.figure()
plt.imshow(seg_image3.astype(int),cmap='gist_gray')
print(R_k,G_k,B_k)
plt.imsave("pigeon_seg.jpg",seg_image3,cmap='gray')

contour_image3 = get_contours(seg_image3)
plt.imshow(contour_image3,cmap='gray')
plt.imsave("pig_contour.jpg",contour_image3,cmap='gray')









#### get texture from image #######
image_mono=im1_gray

def get_var_image(image,window_size,pad_style='zeros'):
  #use the function to get texture based features from the grayscale images.
    The features are generated by calculating
  #the variance of the pixels inside a nxn window.

 #get the window
 win_size = window_size
 win_half=int((win_size-1)/2)
 win_half

 #pad the image with zeros to compute variance of edge pixels
 if pad_style=='zeros':
   pad_img = np.zeros((image.shape[0]+win_size-1,image.shape[1]+win_size-1))
   pad_img[win_half:pad_img.shape[0]-win_half,win_half:pad_img.shape[1]-
    win_half]=image

 if pad_style=='same':
   pass

 if pad_style=='reflective':
   pass

 #compute the variacnce
 var_img=np.zeros_like(pad_img)
 for i in range(win_half,pad_img.shape[0]-win_half):
   for j in range(win_half,pad_img.shape[1]-win_half):
     var_img[i,j]=np.var(pad_img[i-win_half:i+win_half+1,j-win_half:j+win_half
    +1])
 var_img = var_img[win_half:var_img.shape[0]-win_half,win_half:var_img.shape
```

```
    [1]-win_half]
 return var_img

def get_texture_image(image,window_size=[3,5,7]):
  texture_image=np.zeros((image.shape[0],image.shape[1],3),dtype=np.uint8)
  texture_image[:,:,0]=get_var_image(image,window_size[0]).astype(np.uint8)
  texture_image[:,:,1]=get_var_image(image,window_size[1]).astype(np.uint8)
  texture_image[:,:,2]=get_var_image(image,window_size[2]).astype(np.uint8)
  return texture_image

texture_image=get_texture_image(im1_gray,window_size=[3,5,7])

#var_img2=get_var_image(im1_gray,5)
plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(np.uint8(texture_image[:,:,0]),cmap='gray')
plt.subplot(1,3,2)
plt.imshow(np.uint8(texture_image[:,:,1]),cmap='gray')
plt.subplot(1,3,3)
plt.imshow(np.uint8(texture_image[:,:,2]),cmap='gray')
plt.imsave("fox_text3.jpg",texture_image[:,:,0],cmap='gray')
plt.imsave("fox_text5.jpg",texture_image[:,:,1],cmap='gray')
plt.imsave("fox_text7.jpg",texture_image[:,:,2],cmap='gray')

texture_image.dtype

seg_image_text,[R_k,G_k,B_k]=get_otsu_image(texture_image,round=True,rounds
    =[1,5,5],invert=[0,0,0],erosion=[3,1],dialation=[5,2])
plt.figure()
plt.imshow(seg_image_text,cmap='gray')
plt.imsave("fox_text_seg.jpg",seg_image_text,cmap='gray')

contour_image_text1 = get_contours(seg_image_text)
plt.imshow(contour_image_text1,cmap='gray')
plt.imsave("fox_text_cont.jpg",contour_image_text1,cmap='gray')

texture_image=get_texture_image(im2_gray,window_size=[3,5,7])

#var_img2=get_var_image(im1_gray,5)
plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(np.uint8(texture_image[:,:,0]),cmap='gray')
plt.subplot(1,3,2)
plt.imshow(np.uint8(texture_image[:,:,1]),cmap='gray')
plt.subplot(1,3,3)
plt.imshow(np.uint8(texture_image[:,:,2]),cmap='gray')
plt.imsave("cat_text3.jpg",texture_image[:,:,0],cmap='gray')
plt.imsave("cat_text5.jpg",texture_image[:,:,1],cmap='gray')
plt.imsave("cat_text7.jpg",texture_image[:,:,2],cmap='gray')

seg_image_text,[R_k,G_k,B_k]=get_otsu_image(texture_image,round=True,rounds
    =[1,5,5],invert=[0,0,0],erosion=[3,1],dialation=[5,2])
plt.figure()
plt.imshow(seg_image_text,cmap='gray')
plt.imsave("cat_text_seg.jpg",seg_image_text,cmap='gray')

contour_image_text1 = get_contours(seg_image_text)
plt.imshow(contour_image_text1,cmap='gray')
```

```
plt.imsave("cat_text_cont.jpg",contour_image_text1,cmap='gray')

texture_image=get_texture_image(im3_gray,window_size=[3,5,7])

#var_img2=get_var_image(im1_gray,5)
plt.figure(figsize=(10,10))
plt.subplot(1,3,1)
plt.imshow(np.uint8(texture_image[:,:,0]),cmap='gray')
plt.subplot(1,3,2)
plt.imshow(np.uint8(texture_image[:,:,1]),cmap='gray')
plt.subplot(1,3,3)
plt.imshow(np.uint8(texture_image[:,:,2]),cmap='gray')
plt.imsave("pig_text3.jpg",texture_image[:,:,0],cmap='gray')
plt.imsave("pig_text5.jpg",texture_image[:,:,1],cmap='gray')
plt.imsave("pig_text7.jpg",texture_image[:,:,2],cmap='gray')

seg_image_text,[R_k,G_k,B_k]=get_otsu_image(texture_image,round=True,rounds
    =[5,5,5],invert=[1,1,1],erosion=[3,3],dialation=[5,1])
plt.figure()
plt.imshow(seg_image_text,cmap='gray')
plt.imsave("pig_text_seg.jpg",seg_image_text,cmap='gray')

contour_image_text1 = get_contours(seg_image_text)
plt.imshow(contour_image_text1,cmap='gray')
plt.imsave("pig_text_cont.jpg",contour_image_text1,cmap='gray')

"""Check why wrong!!!"""

'''
kernel = np.ones((3,3))/9

mean_img=cv2.filter2D(image_mono,-1,kernel)
print(image_mono.shape)
print(mean_img.shape)
#mean_sub_image=image_mono-mean_img
win_sqr_mean = cv2.filter2D(image_mono**2,-1,kernel)
var_img = win_sqr_mean-mean_img**2
plt.imshow(var_img,cmap='gray')
'''
```