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 σ_w or maximizing the inter-class variance σ_b. We can express the weighted intra-class variance σ_w as following

$$\sigma_\omega{}^2 = \omega_0\sigma_0{}^2 + \omega_1\sigma_1{}^2$$

While ω_0, ω_1 are the probabilities of two classes, they satisfy that $\omega_0 + \omega_1 = 1$. $\sigma_0{}^2$, $\sigma_1{}^2$ are variances within two classes. So the inter-class variance σ_b can be expressed as

$$\sigma_\mathrm{b}{}^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0\omega_1(\mu_0 - \mu_1)^2$$

While μ_0, μ_1, μ_T 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

$$\mu_0 = \sum_{i=0}^{k-1} \frac{ip_i}{\omega_0}$$

$$\omega_0 = \sum_{i=0}^{k-1} p_i$$

$$\mu_1 = \sum_{i=k}^{255} \frac{ip_i}{\omega_1}$$

$$\omega_1 = \sum_{i=k}^{255} p_i$$

$$\mu_T = \sum_{i=0}^{255} i p_i$$

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

We can find the threshold with the greatest value of ${\sigma_\mathrm{b}}^2$, 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 $N \times N$ (Here $N = 3, 5, 7$) 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 $N \times N$ 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

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

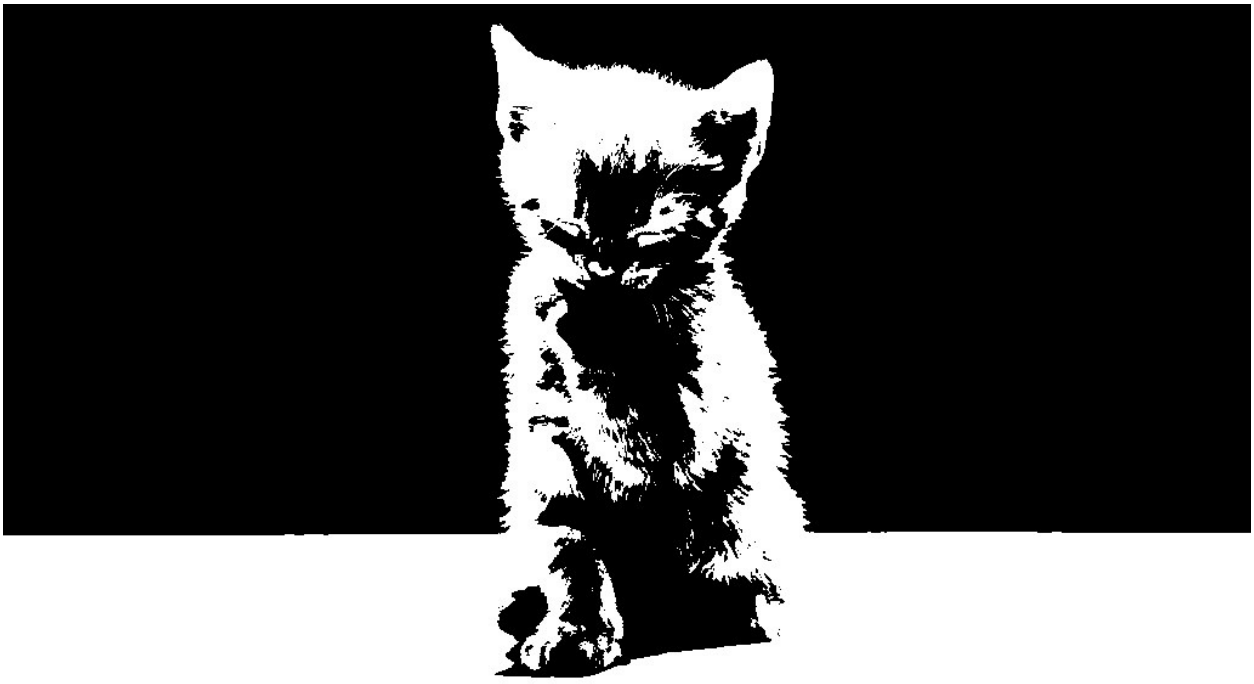

Figure 2: Cat image G channel Otsu result (1 iteration)

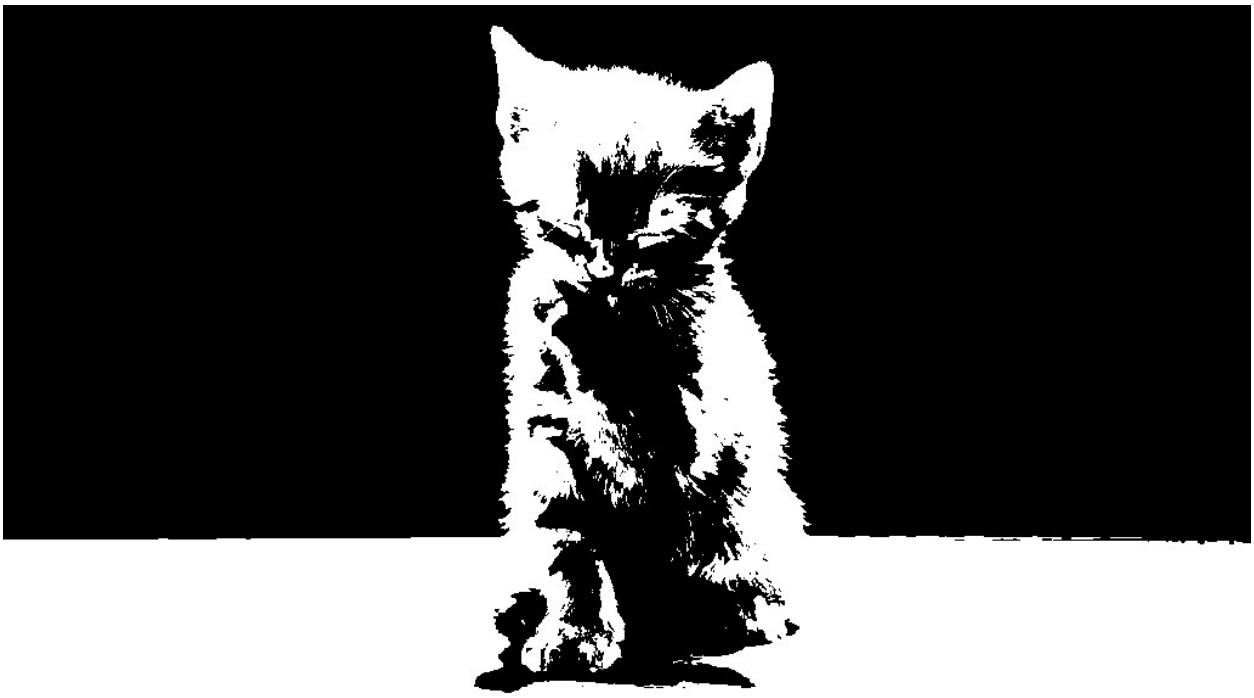

Figure 3: Cat image B channel Otsu result (1 iteration)

Figure 4: Cat image three channel combined segmentation mask

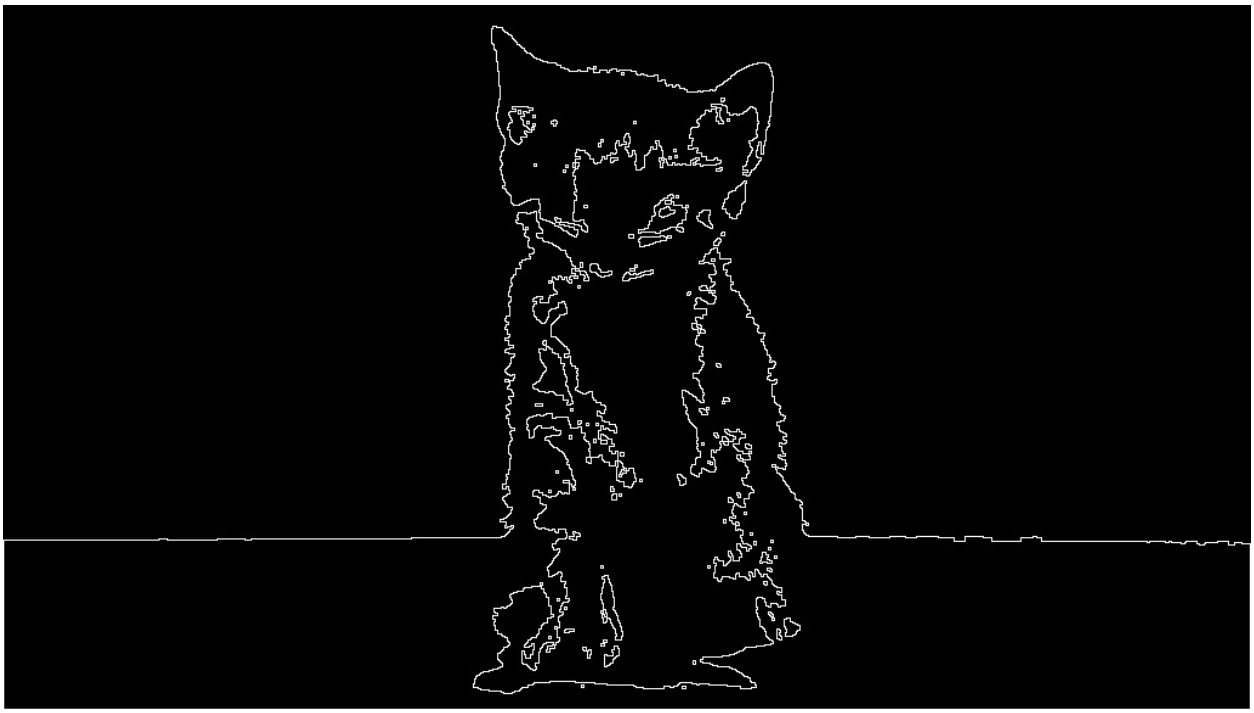

Figure 5: Cat image contour (3×3 erosion and 3×3 dilation)

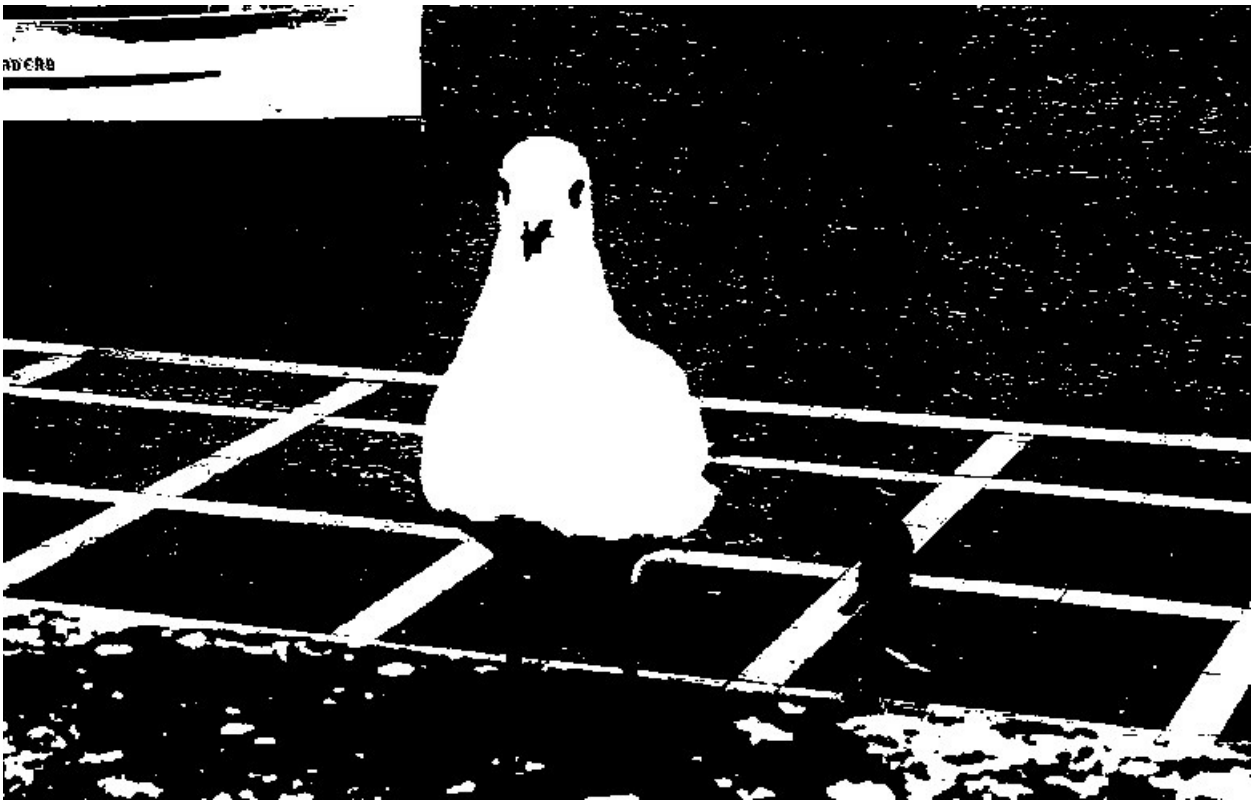

Figure 6: Pigeon image R channel Otsu result (1 iteration)

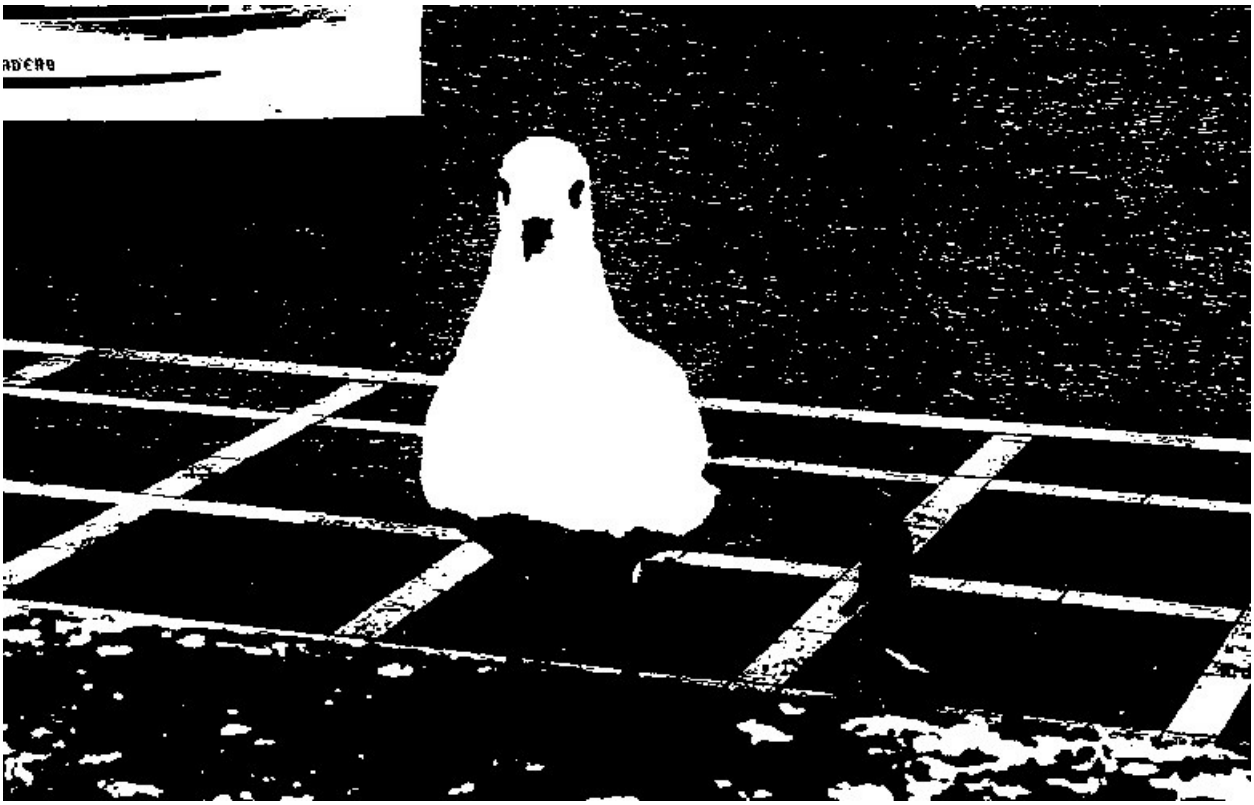

Figure 7: Pigeon image G channel Otsu result (1 iteration)

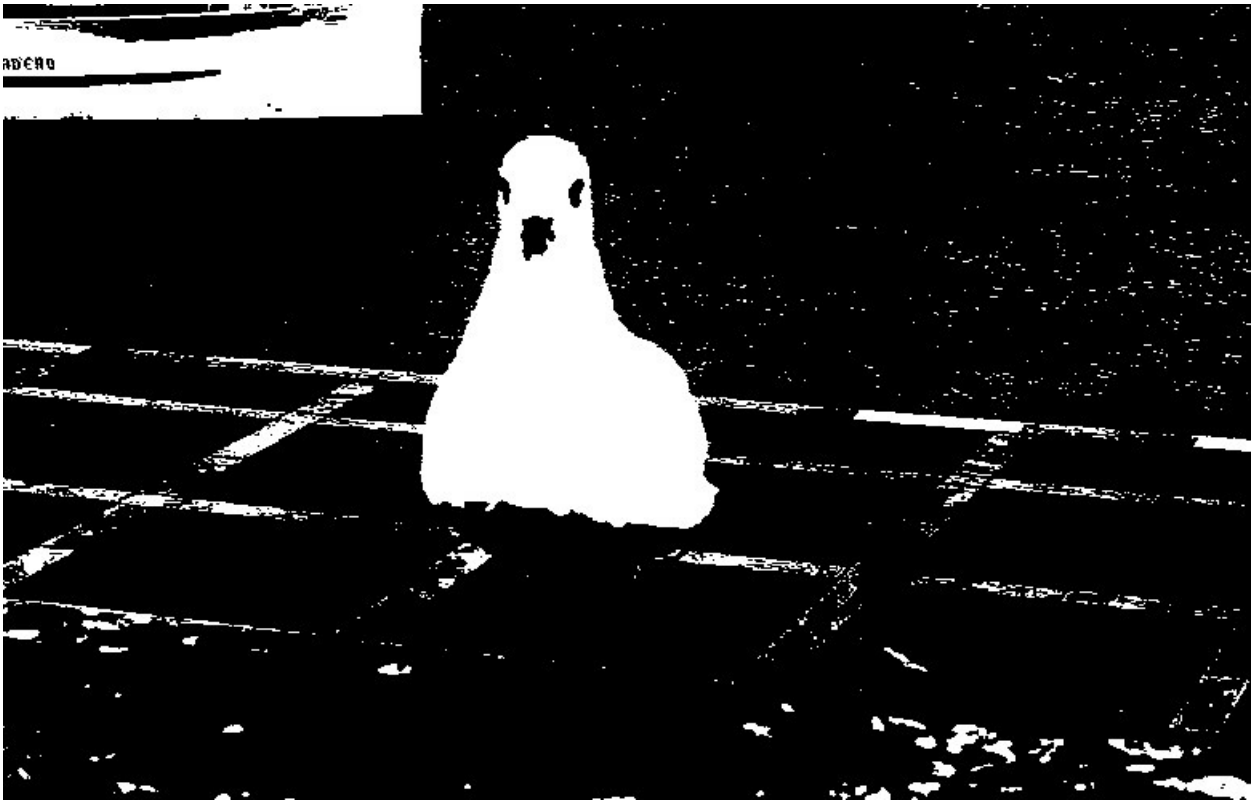

Figure 8: Pigeon image B channel Otsu result (1 iteration)

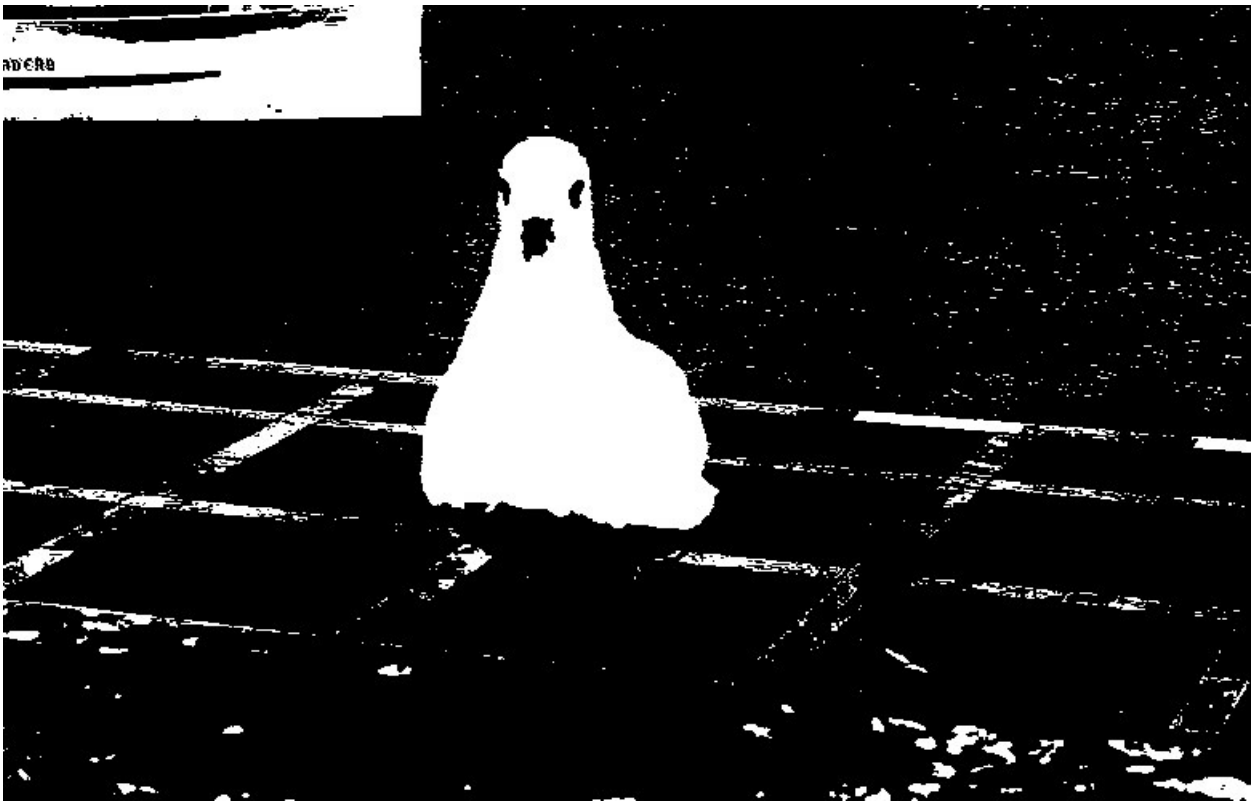

Figure 9: Pigeon image three channel combined segmentation mask

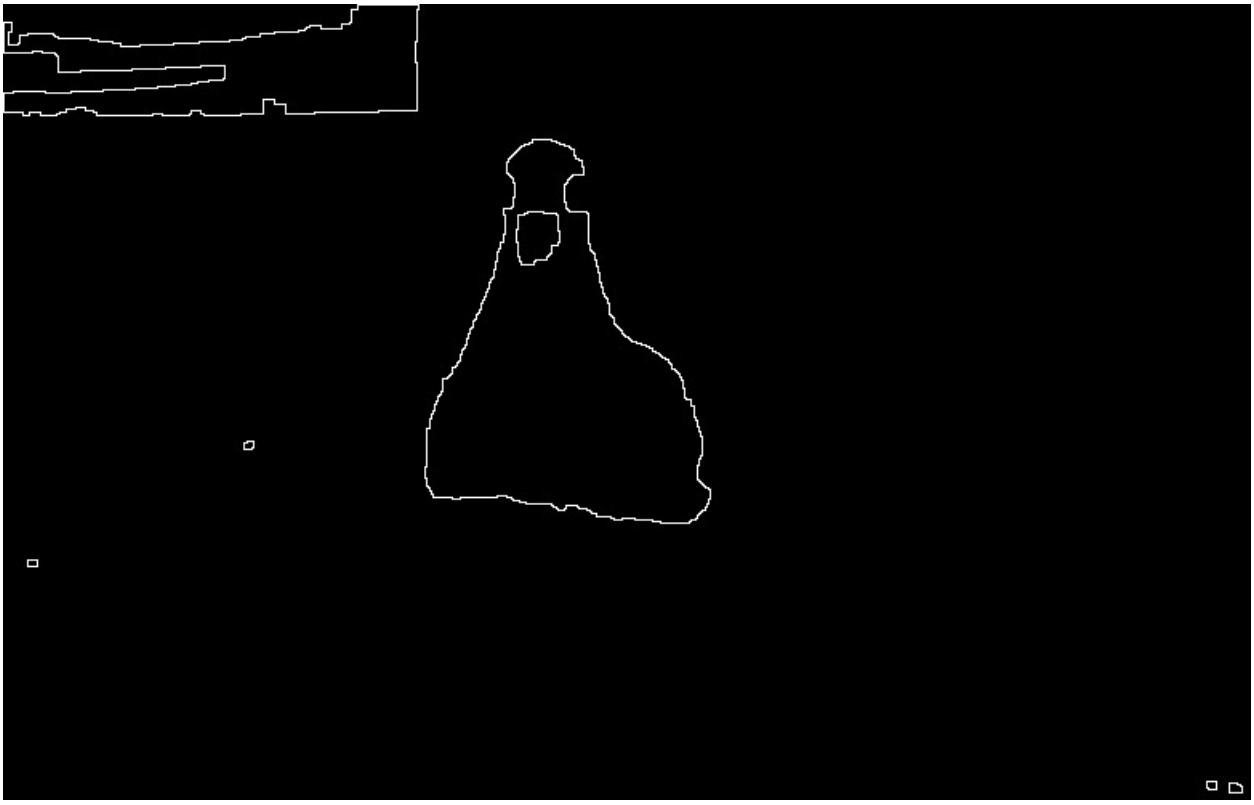

Figure 10: Pigeon image contour (5×5 erosion twice and 5×5 dilation)

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

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

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

Figure 14: Fox image three channel combined segmentation mask

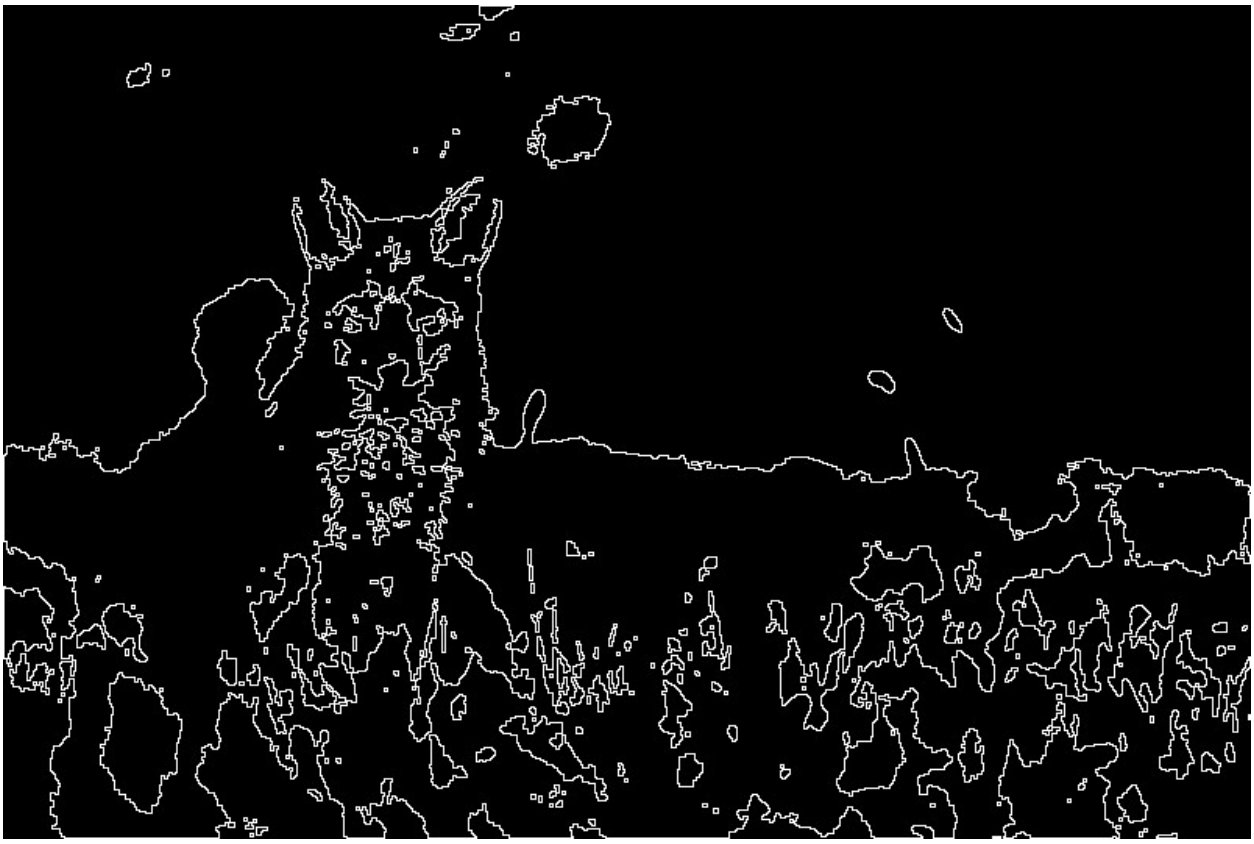

Figure 15: Fox image contour (3×3 erosion and 3×3 dilation)

6.2 Task 2: Texture based Image Segmentation

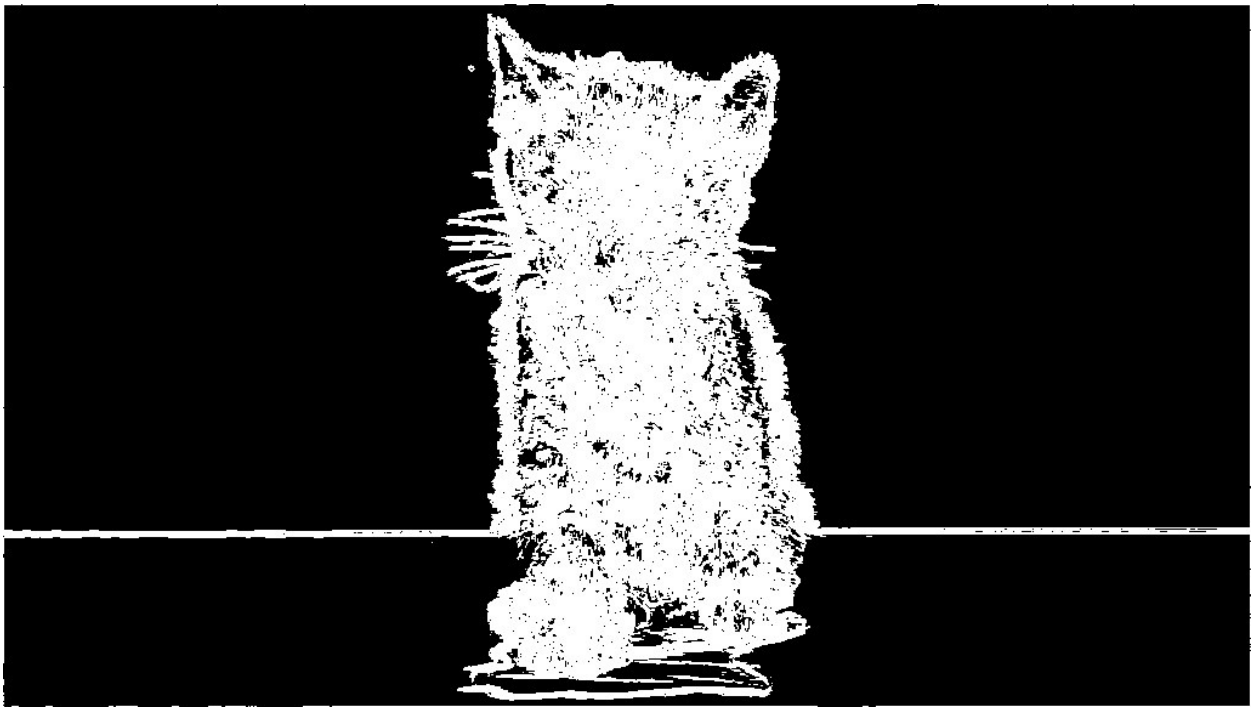

Figure 16: Cat 3×3 variance texture image's Otsu result (3 iterations)

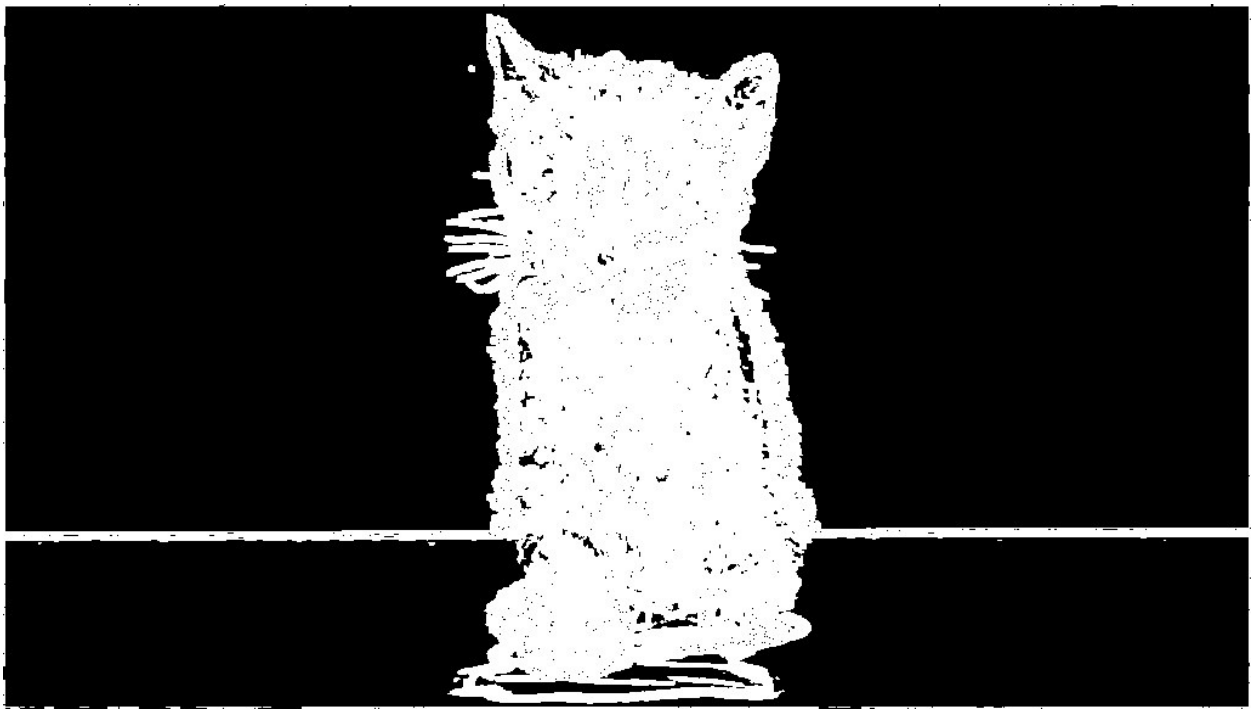

Figure 17: Cat 5×5 variance texture image's Otsu result (3 iterations)

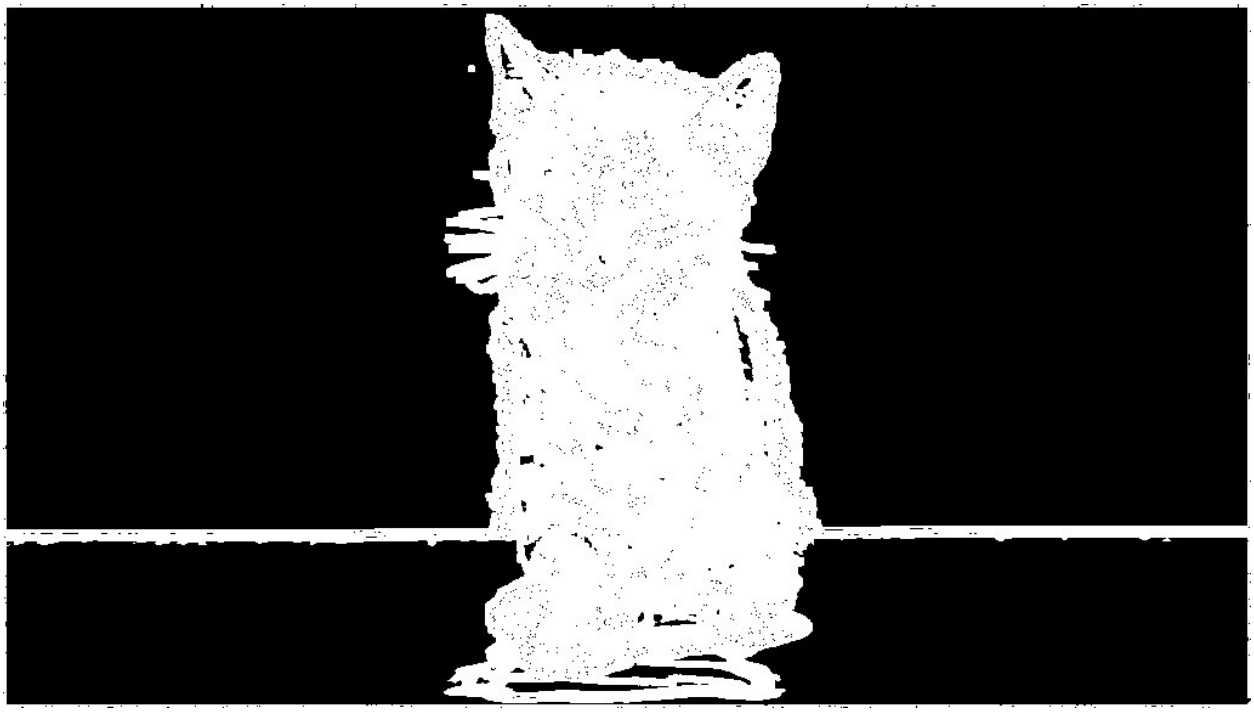

Figure 18: Cat 7×7 variance texture image's Otsu result (3 iterations)

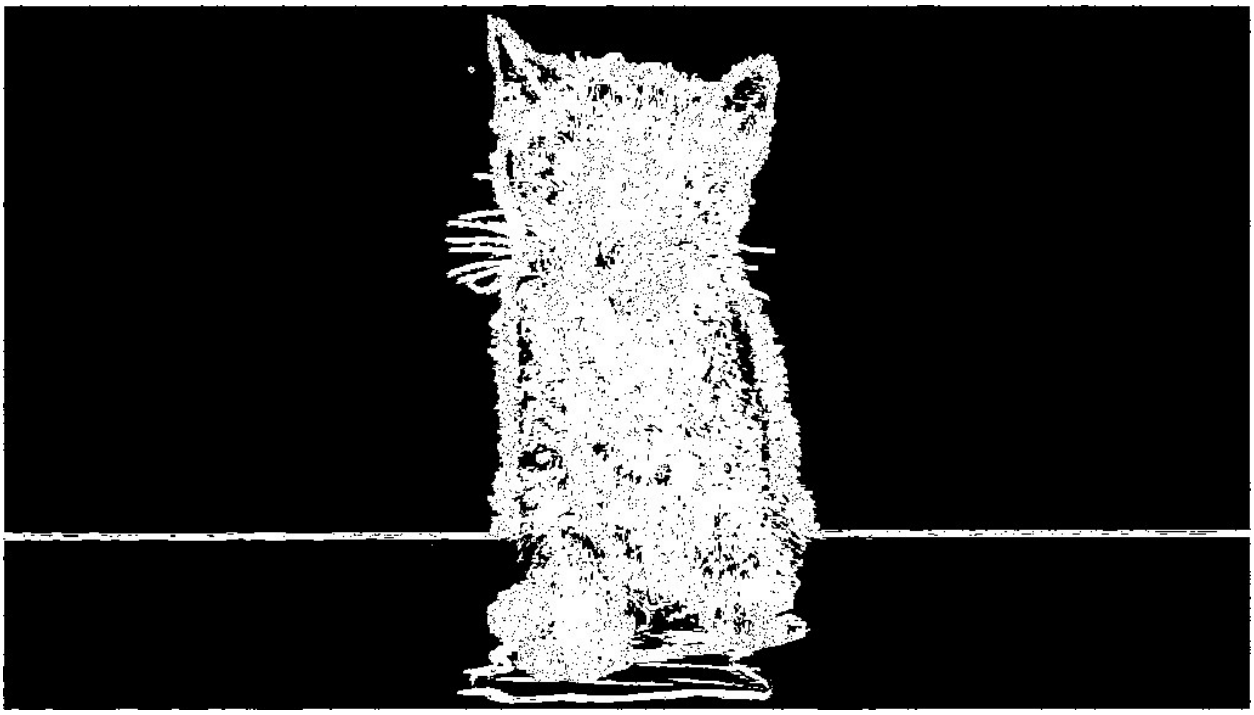

Figure 19: Cat texture image three channel combined segmentation mask

Figure 20: Cat texture image contour (3×3 erosion and 5×5 dilation twice)

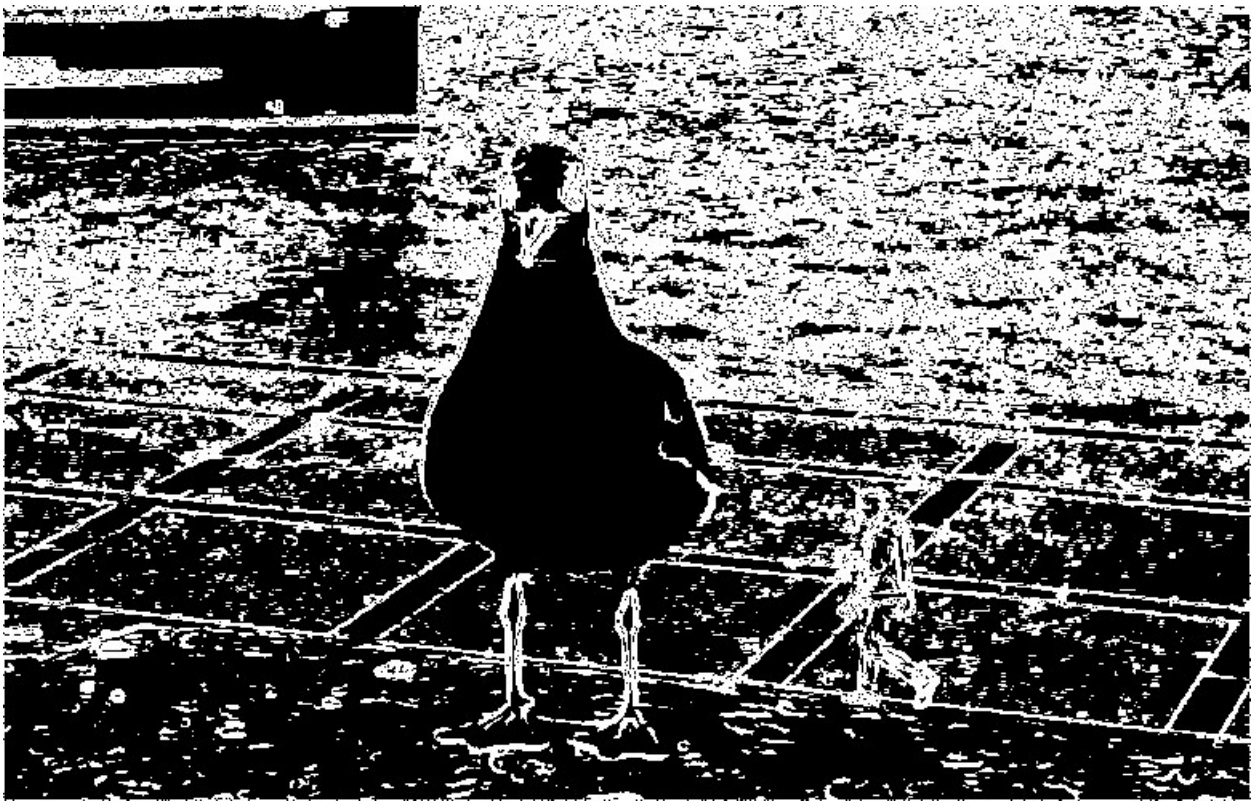

Figure 21: Pigeon 3×3 variance texture image's Otsu result (2 iterations)

Figure 22: Pigeon 5×5 variance texture image's Otsu result (2 iterations)

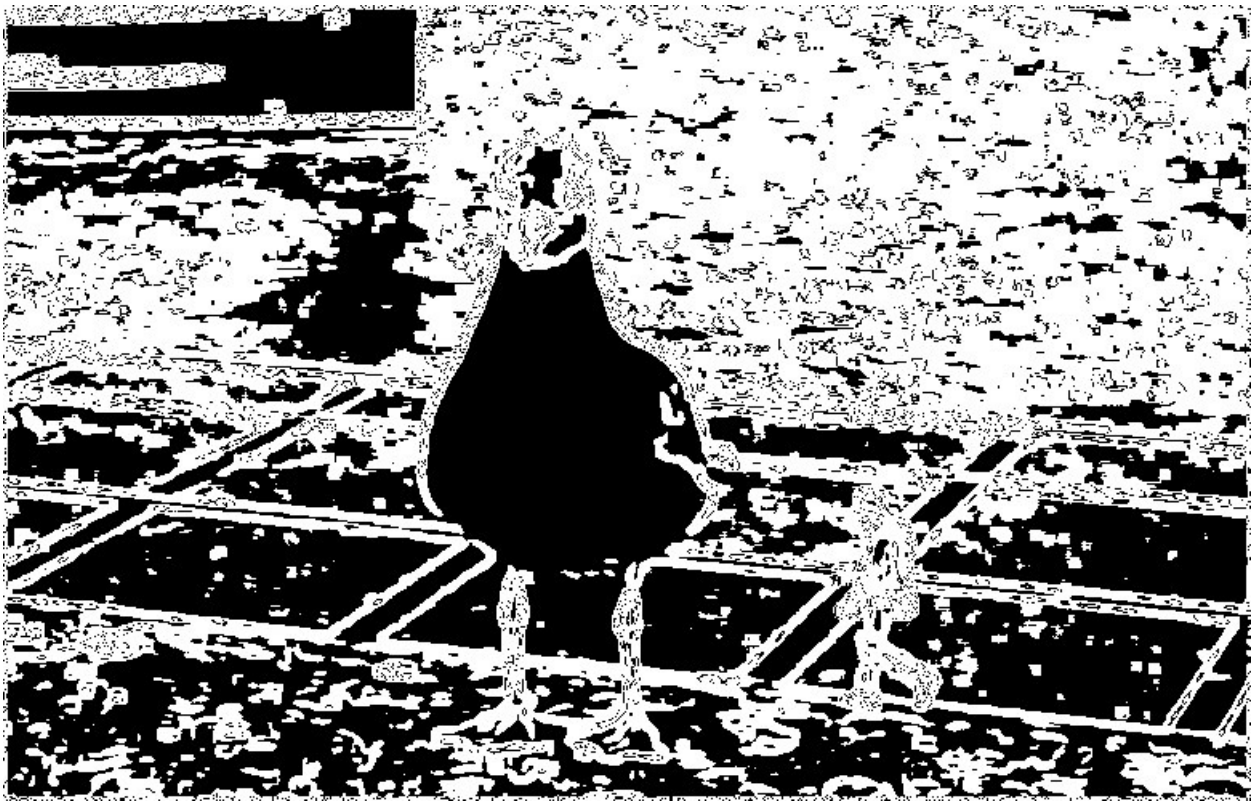

Figure 23: Pigeon 7×7 variance texture image's Otsu result (2 iterations)

Figure 24: Pigeon texture image three channel combined segmentation mask

Figure 25: Pigeon texture image contour (5×5 dilation twice)

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

Figure 27: Fox 5×5 variance texture image's Otsu result (2 iterations)

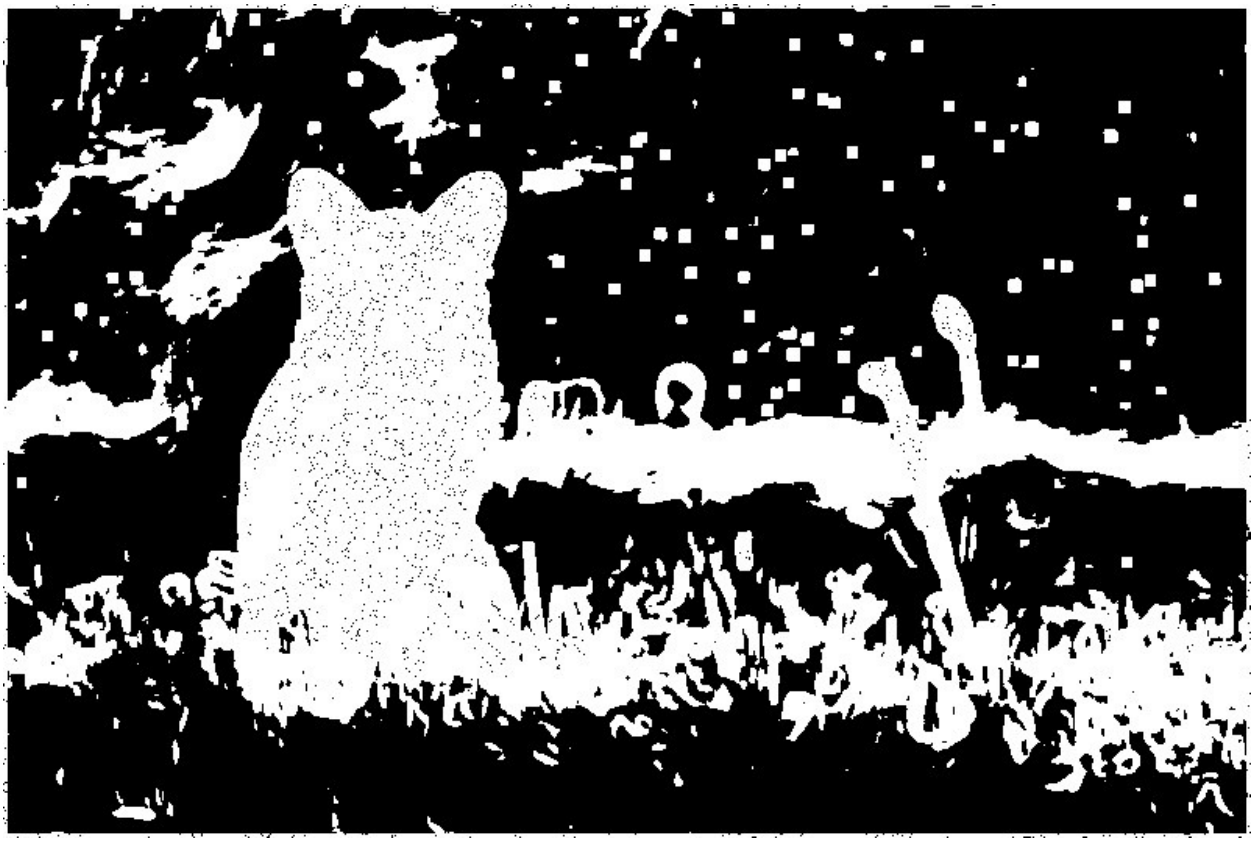

Figure 28: Fox 7×7 variance texture image's Otsu result (3 iterations)

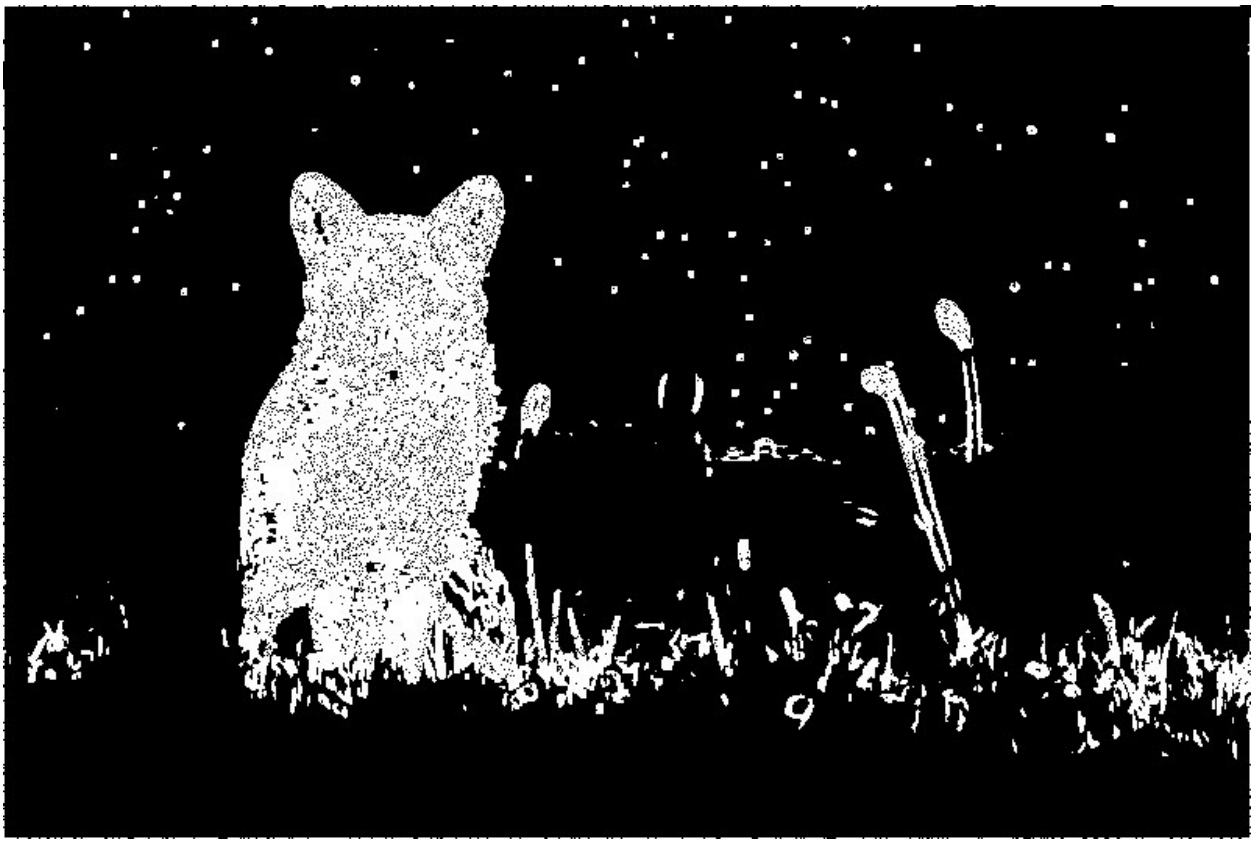

Figure 29: fox texture image three channel combined segmentation mask

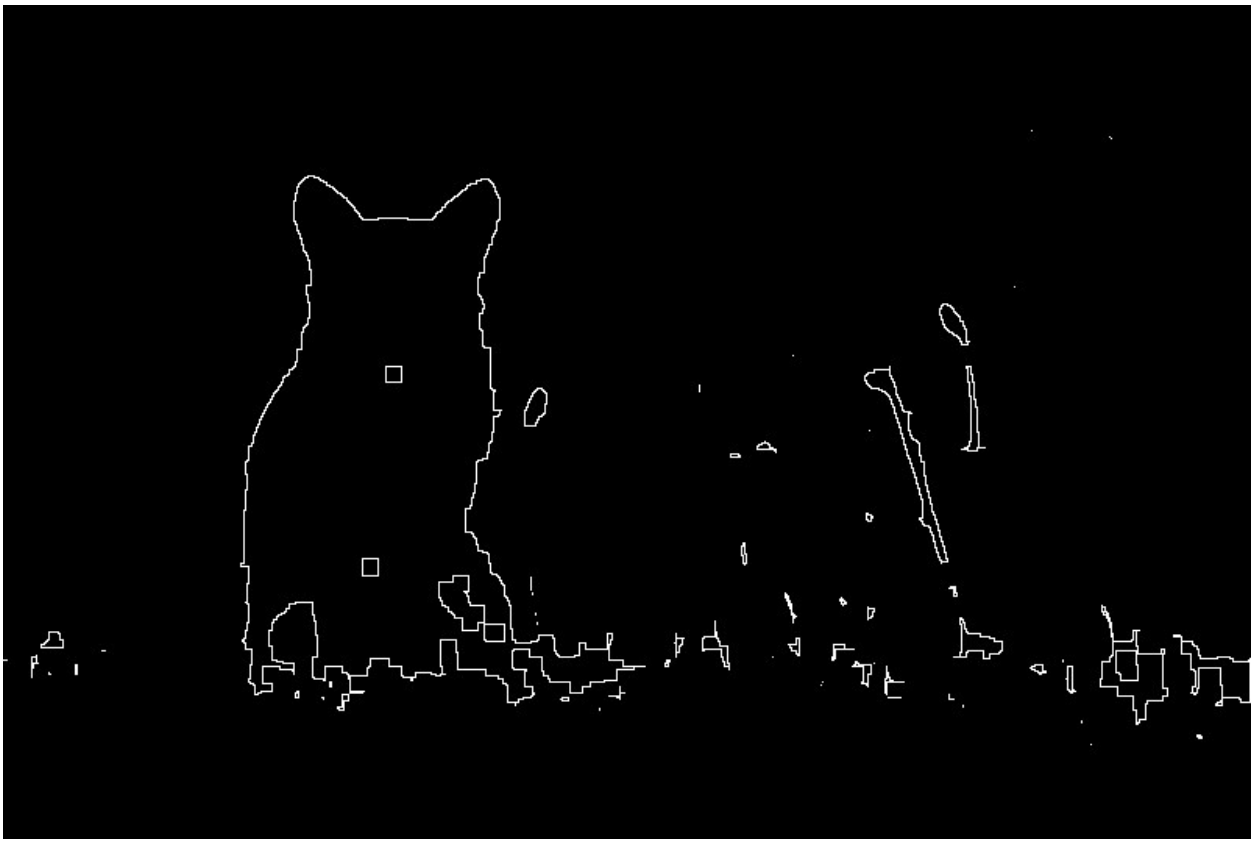

Figure 30: Fox texture image contour (3×3 erosion twice and 5×5 dilation twice)

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)
```