ECE 661: Homework 4

Shengtong Zhang
Email: zhan3366@purdue.edu
(Fall 2020)

1 Theory Question

For Otsu Algorithm,

- Advantages: This algorithm is simple for calculation. And when the difference between the foreground object and the bachground is not too big, this algorithm can effectively segment the image.

- Disadvantage: Because the algorithm ignores the spatial information of the image, and uses the grayscale distribution of the image as the basis for segmenting the image which is sensitive to noise. When the area of the target and the background in the image is very different, it shows that the histogram does not have obvious double peaks, or the size of the two peaks is very different, the segmentation effect is not good, or the foreground object and the background cannot be accurately separated when there is a large overlap in the grayscale of the foreground object and the background.

For Watershed Algorithm,

- Advantages: This algorithm is Intuitive and fast. And it can be computed in parallel. And the result of this algorithm always producing complete boundaries.

- Disadvantage: Due to interference from noise points or other factors, this algorithm may get densely small areas, which is over-segmented. This is because there are many local minimal value points in the image, each point will be a small area of its own.

2 Ostu Algorithm

The Otsu algorithm can be described by the following steps:

1.First calculate the pixel grayscale level histogram. For the range 0 to 255 total 256 gray levels the probability distribution function is,

$$p_i = \frac{n_i}{N}$$

where n_i denotes the number of pixels at i^{th} grayscale level and N is the total number of pixels in the image.

2.Let k denotes the grayscale level that sepatates the foreground object and the background. Then we denote the pixels with grayscale level less than k as class C_0 and the pixels with grayscale level greater than k as class C_1. Then we can get the probability of C_0 and C_1, $\omega_0 = \sum_1^k p_i$ and $\omega_1 = \sum_{k+1}^{255} p_i$.

3.Then we can calculate the mean of each class and the between-class variance,

$$\mu_0 = \frac{1}{\omega_0}\sum_{1}^{k} ip_i, \quad \mu_1 = \frac{1}{\omega_1}\sum_{k+1}^{255} ip_i$$

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

4.Then we select the optimum threshold k^* that maximizes σ_b^2. And in the next step, we carry out masking of the image in such a way that the pixel value of the pixels with grayscale level lower than k^* are changed to 0 to mark the background and pixels with grayscale level greater than k^* are changed to 1 to mark the foreground.

5.To improve the segmentation results, repeat the above steps. The number of iterations is a hyperparameter.

3 Image Segmentation Using RGB Values

The image segmentation can be achieved by applying Otsu algorithm to the three RGB color channels separately, and then combine the segmentation results to get final segmentation of the image. The procedure is described as follows:

1.Separate the three RGB color channels and convert them into three grayscale images.

2.For each grayscale images, implement the Otsu algorithm.

3.Combine the results gotten from step 2 by logical operator AND to get a better segmentation result.

4 Texture-based Segmentation

We can exploit the spatial information of pixels using texture-based feature and represent the original image in texture space. More specifically, the texture-based feature for a pixel in this experiment is defined as the variance of the pixels values in a $N \times N$ window. We evaluate the original image with three different window sizes to obtain three channels for the texture space image representation, similar to the RGB channels in a normal color image.

5 Contour Extraction

After we have the binary image, we can extract the contour of the segmented region. When a pixel with value 1 has at least one neighbor pixel with value 0 for 8-neighbors, it's determined to be on the contour. Otherwise, the pixel is not on the contour.

6 Results

6.1 Cat

For the cat image,

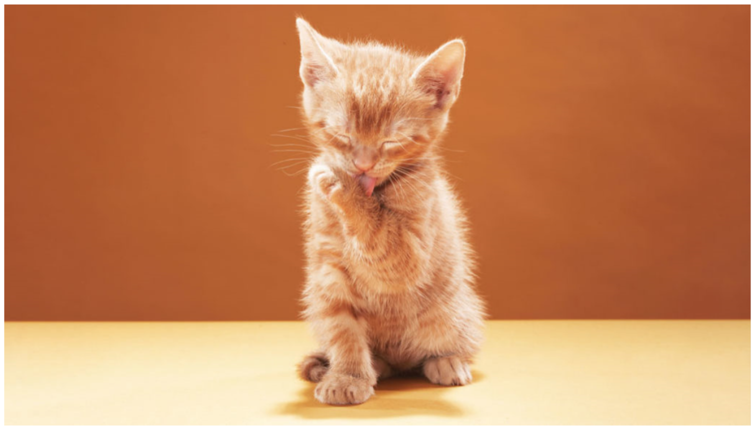

Figure 1: The original cat image

6.1.1 Image Segmentation Using RGB Values

The numbers of iterations for this method are [1,1,1].

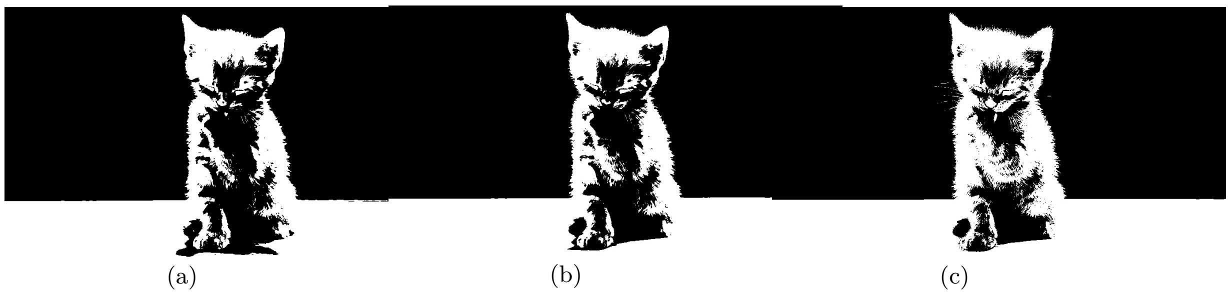

Figure 2: Masks get from different color channels

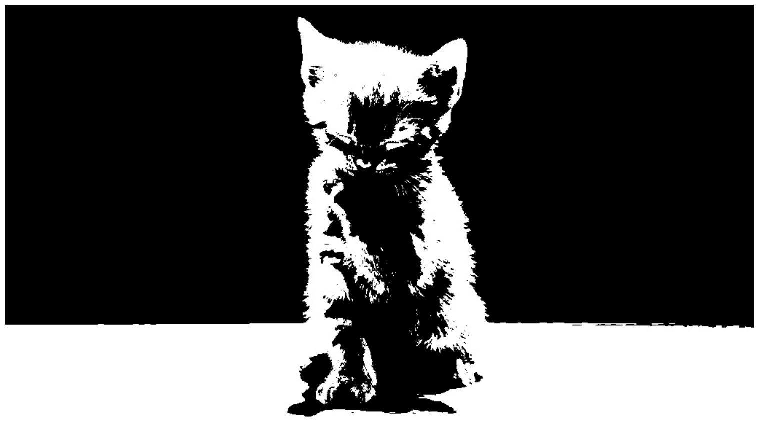

Figure 3: The overall foreground mask

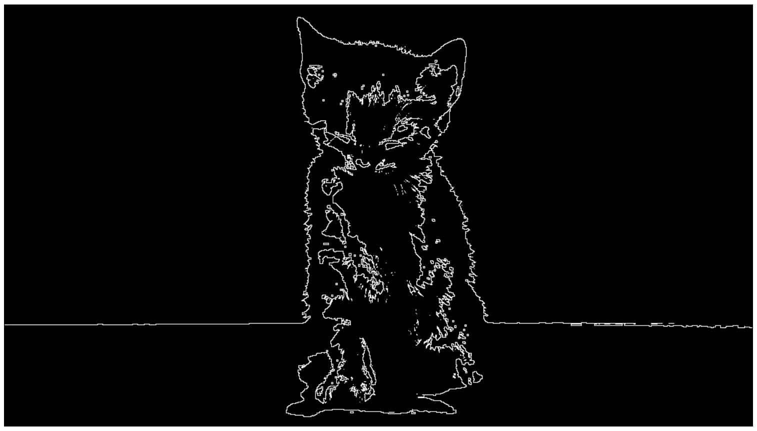

Figure 4: The final contour of the image

6.1.2 Texture-based Segmentation

The numbers of iterations for this method are [1,1,1]. The window sizes are chosen as [6×6,7×7,8×8]

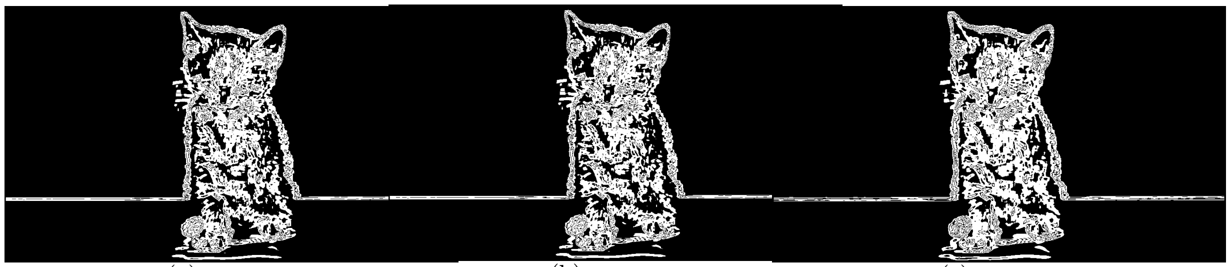

Figure 5: Masks get from different window sizes

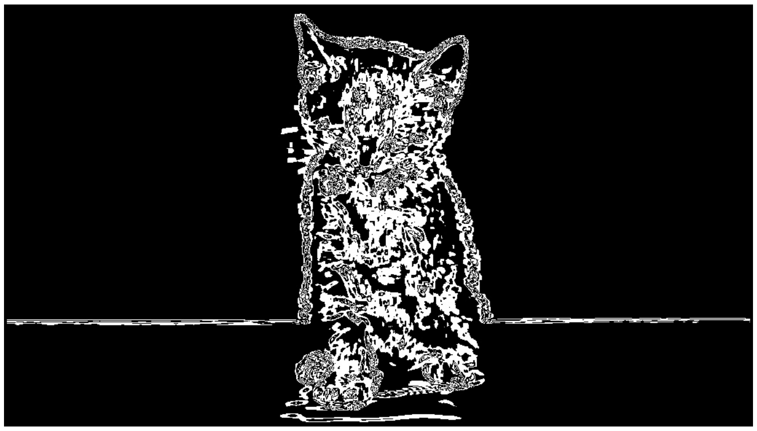

Figure 6: The overall foreground mask

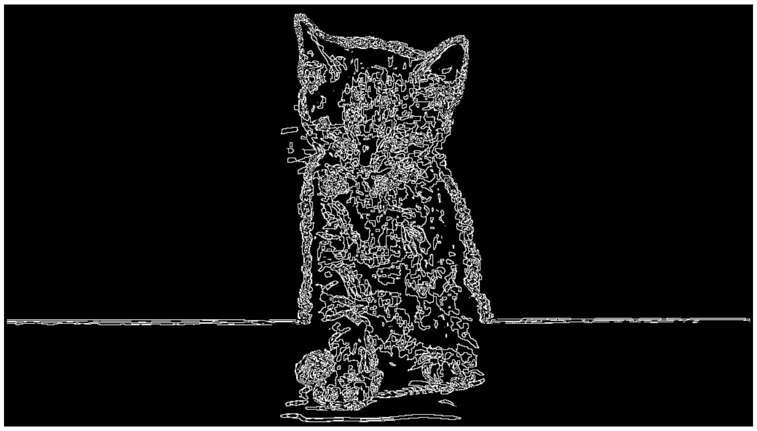

Figure 7: The final contour of the image

6.2 Pigeon

For the pigeon image,

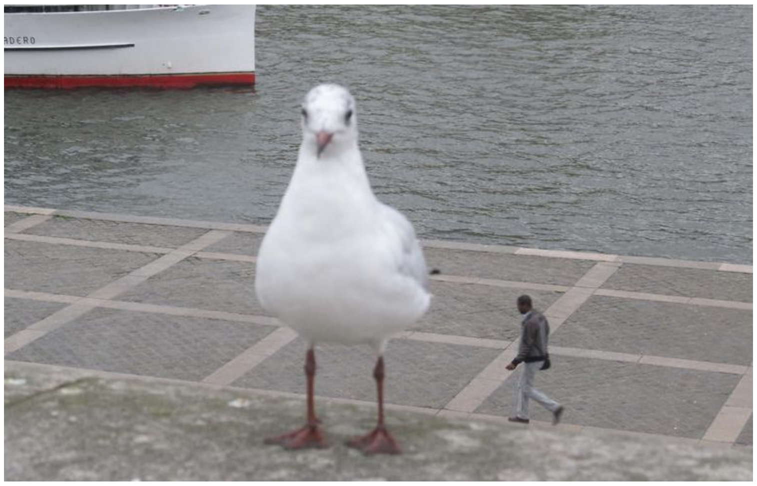

Figure 8: The original pigeon image

6.2.1 Image Segmentation Using RGB Values

The numbers of iterations for this method are [2,2,2].

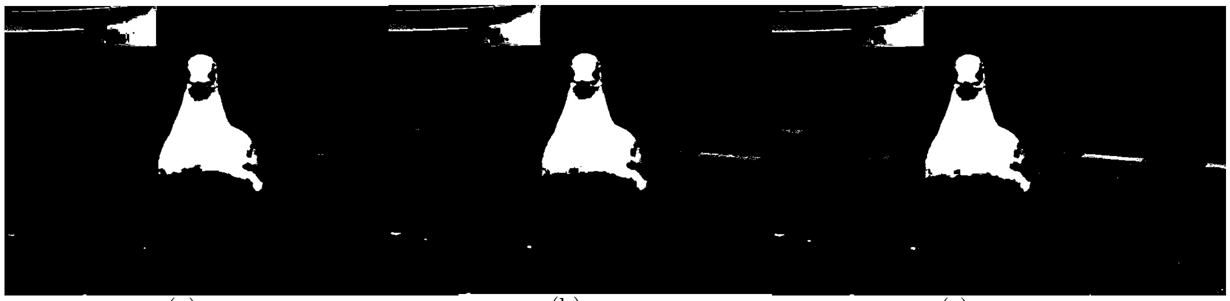
(a) (b) (c)

Figure 9: Masks get from different color channels

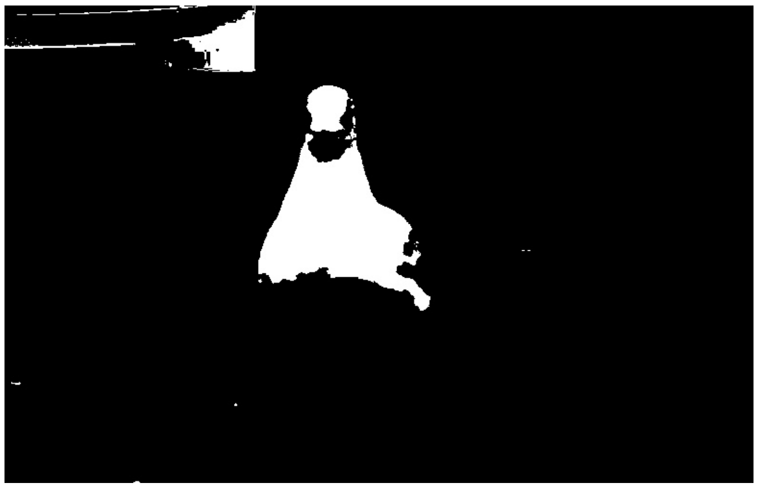

Figure 10: The overall foreground mask

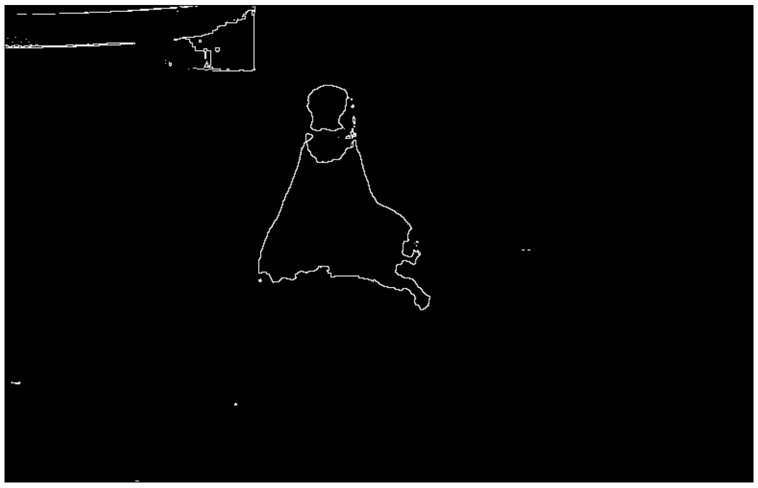

Figure 11: The final contour of the image

6.2.2 Texture-based Segmentation

The numbers of iterations for this method are [6,5,4]. The window sizes are chosen as [2×2,3×3,4×4]

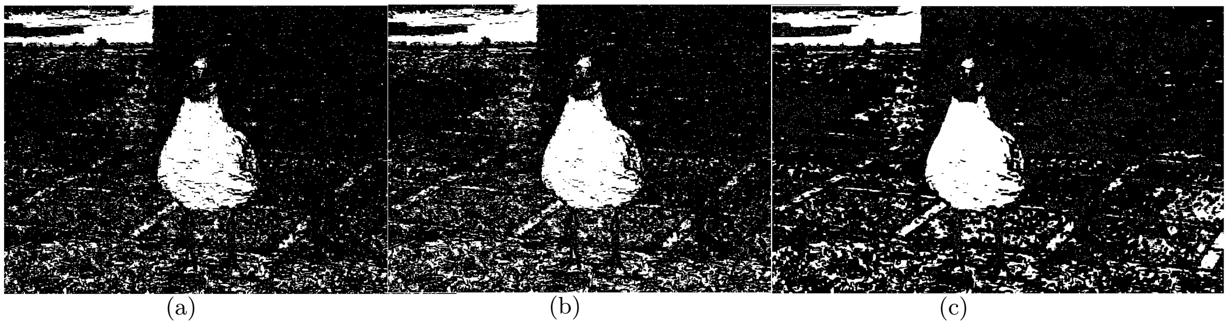

Figure 12: Masks get from different window sizes

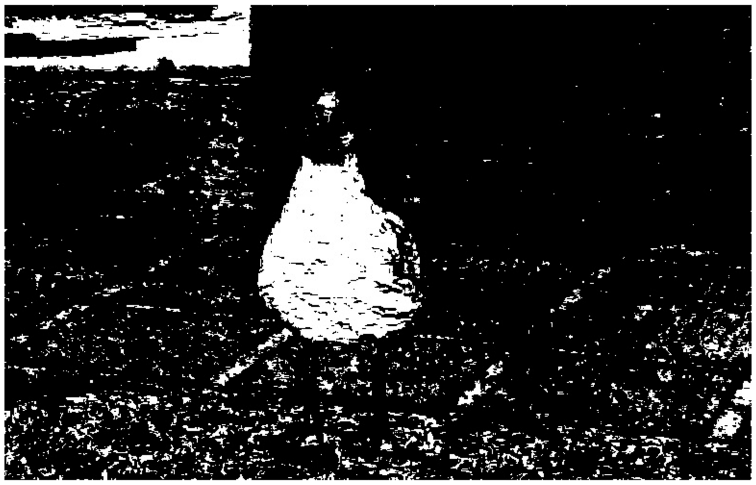

Figure 13: The overall foreground mask

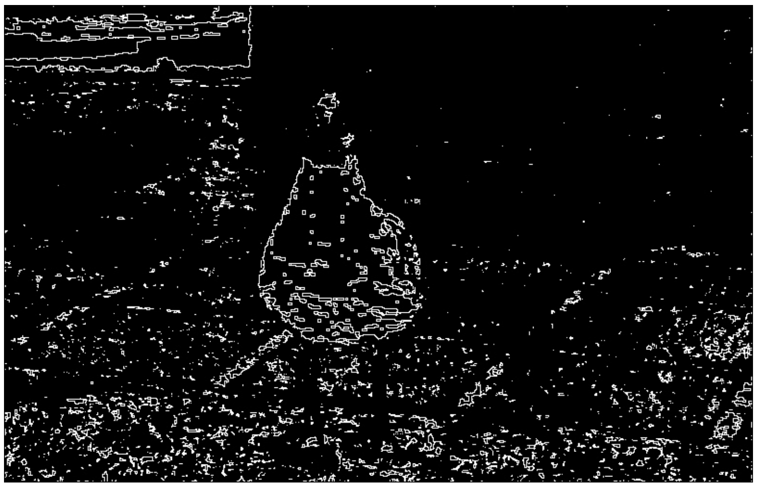

Figure 14: The final contour of the image

6.3 Red Fox

For the red fox image,

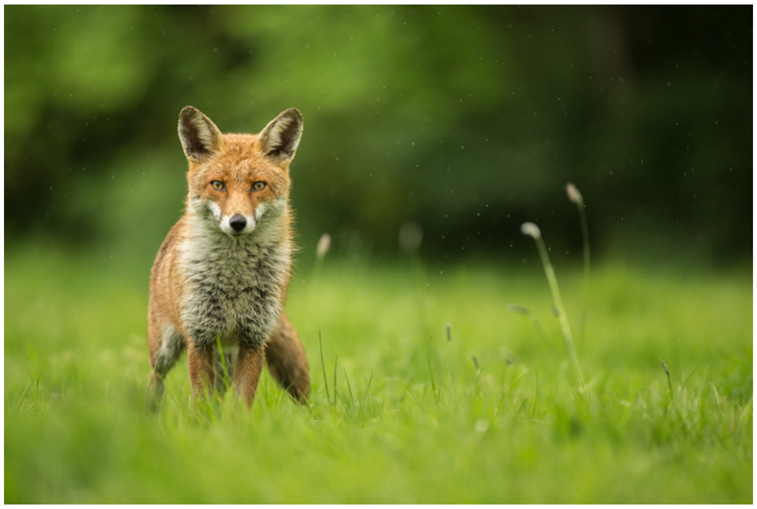

Figure 15: The original red fox image

6.3.1 Image Segmentation Using RGB Values

The numbers of iterations for this method are [2,2,2].

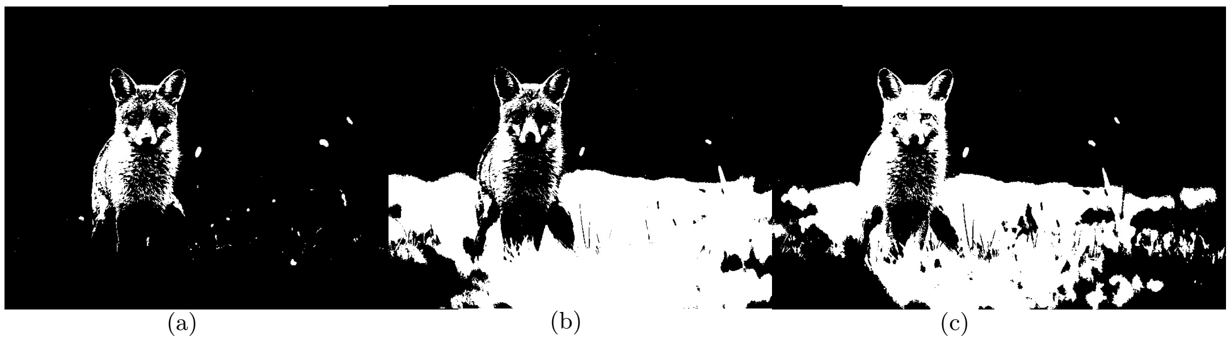

Figure 16: Masks get from different color channels

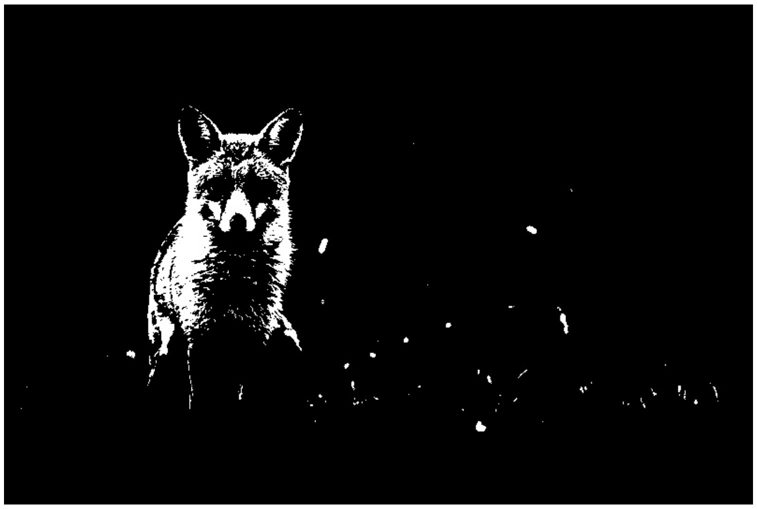

Figure 17: The overall foreground mask

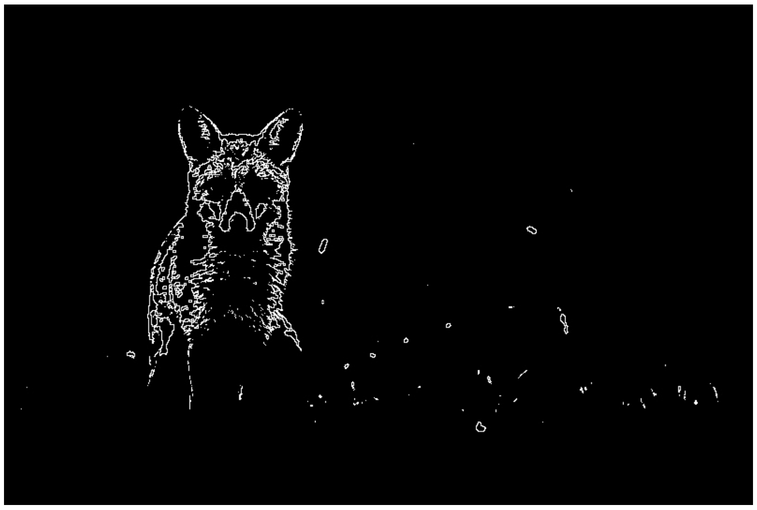

Figure 18: The final contour of the image

6.3.2 Texture-based Segmentation

The numbers of iterations for this method are [1,1,1]. The window sizes are chosen as [5×5,6×6,7×7]

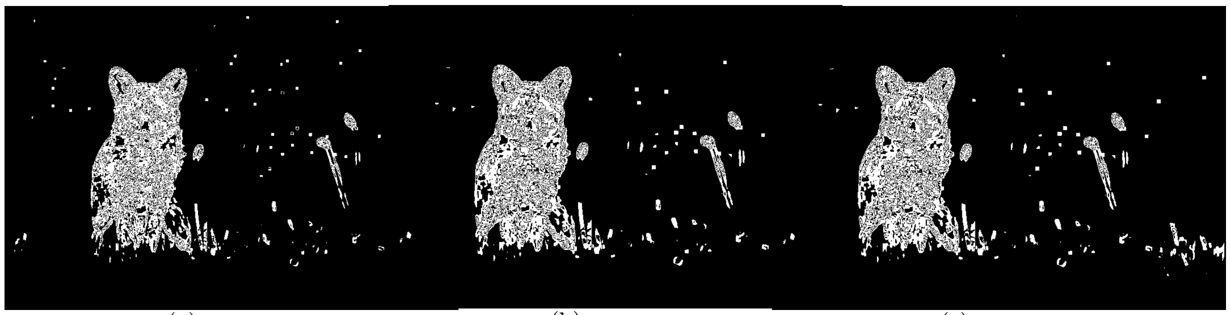
(a) (b) (c)

Figure 19: Masks get from different window sizes

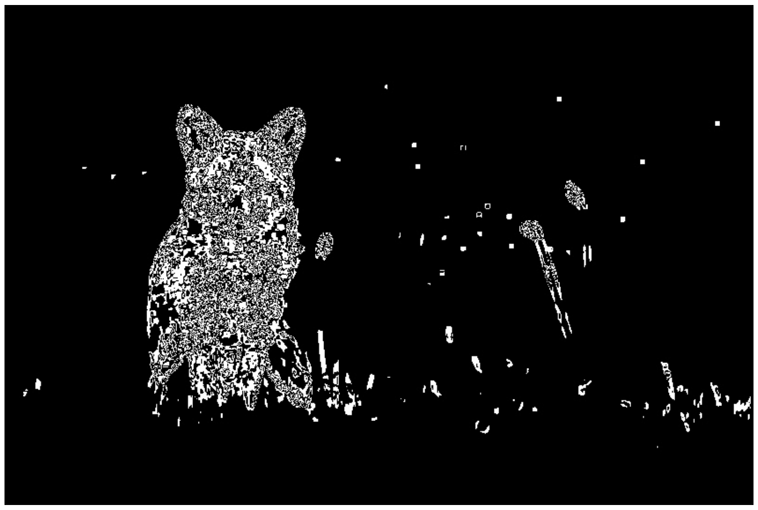

Figure 20: The overall foreground mask

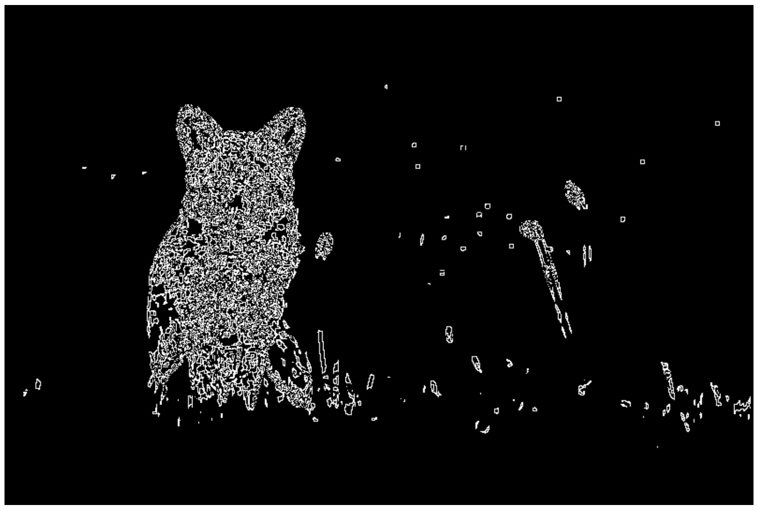

Figure 21: The final contour of the image

7 Discussion

• The quality of the segmentation results highly depends on the original images. We should select an appropriate method for different kinds of images.

• The segmentation method based on the RGB channels seems can produce a more smooth result than the method based on texture features. But there are also some problems with this method. For example, for the cat image. This method can't separate the table with the cat because the cat is lighter than the wall but darker than the table.

• The method based on texture features seems more sensitive to the noise. There are many small areas in the results of this method. This is obvious in the result of pigeon. And if we use a larger window size, there are more pixels we want (which should be considered as foreground) are included in the final result. However, the segmentation result will also be more coarse meanwhile.

8 Code

```python
import numpy as np
import cv2
from matplotlib import pyplot as plt

def get_texture_image(bgr, Ns):
        gray = cv2.cvtColor(bgr, cv2.COLOR_BGR2GRAY)
        texture = np.zeros( (bgr.shape[0], bgr.shape[1], len(Ns)) )
        for i in range(len(Ns)):
                w_h = int(Ns[i]/2)
                for r in range(w_h, bgr.shape[0]-w_h):
                        for c in range(w_h, bgr.shape[1]-w_h):
                                var = np.var( gray[r-w_h:r+w_h+1, c-w_h
                                    :c+w_h+1] )
                                texture[r,c,i] = var
        return texture.astype(np.uint8)

def get_contour(mask):
        contour = np.zeros(mask.shape).astype(np.uint8)
        w_h = 1
        for r in range(w_h, mask.shape[0]-w_h):
                for c in range(w_h, mask.shape[1]-w_h):
                        if mask[r,c] == 0:
                                continue
                        if np.min( mask[r-w_h:r+w_h+1, c-w_h:c+w_h+1] )
                            == 0:
                                contour[r,c] = 255
        return contour

def Otsu_for_gray_img(image, num_iters,reverse_mask):
        mask = np.ones(image.shape).astype(np.uint8) * 255
```

```
    for i in range(num_iters):
        hist ,_ = np.histogram(image[np.nonzero(mask)], bins=np.arange(257), density=True)
        ith_hist = np.multiply(hist, np.arange(256))
        mu_T = np.sum(ith_hist)
        b_vars = np.zeros(256)
        for k in range(256):
            omega_0 = np.sum(hist[:k])
            omega_1 = np.sum(hist[k+1:])
            omega_mu_0 = np.sum(ith_hist[:k])
            omega_mu_1 = np.sum(ith_hist[k+1:])
            if omega_mu_0 == 0.0 or omega_mu_1 == 0.0:
                continue
            b_vars[k] = omega_0*omega_1*(omega_mu_0/omega_0-omega_mu_1/omega_1)**2
        k_star = np.argmax(b_vars)
        _,mask = cv2.threshold(image, k_star, 255, cv2.THRESH_BINARY)
        if reverse_mask:
            mask = cv2.bitwise_not(mask)
    return mask

def Otsu_for_rgb_img(image, num_iterss, reverse_masks):
    # BGR channels
    channels = cv2.split(image)
    masks = []
    mask_together = np.ones(channels[0].shape).astype(np.uint8) * 255
    for i in range(len(channels)):
        mask = Otsu_for_gray_img(channels[i], num_iterss[i], reverse_masks[i])
        mask_together = cv2.bitwise_and(mask_together, mask)
        masks.append(mask)
    return masks, mask_together

def result():
    # image_cat = 'inputs/cat.jpg'
    # num_iterss = [1,1,1]
    # reverse_masks = [0,0,0]
    # color = cv2.imread(image_cat)
    # image = color
    # channels = cv2.split(image)
    # masks, mask_together = Otsu_for_rgb_img(image, num_iterss, reverse_masks)
    # for i in range(len(channels)):
    #         cv2.imwrite('outputs/cat_mask_' + str(i) + '.jpg', masks[i])
    # cv2.imwrite('outputs/cat_finalmask.jpg', mask_together)
```

```
        # foreground = cv2.bitwise_and(color, cv2.cvtColor(
            mask_together, cv2.COLOR_GRAY2BGR))
        # contour = get_contour(mask_together)
        # cv2.imwrite('outputs/cat_foreground.jpg', foreground)
        # cv2.imwrite('outputs/cat_contour.jpg', contour)


        image_cat = 'inputs/cat.jpg'
        num_iterss = [1,1,1]
        reverse_masks = [0,0,0]
        Ns = [6,7,8]
        color = cv2.imread(image_cat)
        texture = get_texture_image(color,Ns)
        cv2.imwrite('outputs/cat_texture.jpg', texture)
        image = texture
        channels = cv2.split(image)
        masks, mask_together = Otsu_for_rgb_img(image, num_iterss,
            reverse_masks)
        for i in range(len(channels)):
                cv2.imwrite('outputs/cat_texture_mask_' + str(i) + '.
                    jpg', masks[i])
        cv2.imwrite('outputs/cat_texture_finalmask.jpg', mask_together)
        foreground = cv2.bitwise_and(color, cv2.cvtColor(mask_together,
             cv2.COLOR_GRAY2BGR))
        contour = get_contour(mask_together)
        cv2.imwrite('outputs/cat_texture_foreground.jpg', foreground)
        cv2.imwrite('outputs/cat_texture_contour.jpg', contour)


        # image_pigeon = 'inputs/pigeon.jpeg'
        # num_iterss = [2,2,2]
        # reverse_masks = [0,0,0]
        # color = cv2.imread(image_pigeon)
        # image = color
        # channels = cv2.split(image)
        # masks, mask_together = Otsu_for_rgb_img(image, num_iterss,
            reverse_masks)
        # for i in range(len(channels)):
        #       cv2.imwrite('outputs/pigeon_mask_' + str(i) + '.jpg',
            masks[i])
        # cv2.imwrite('outputs/pigeon_finalmask.jpg', mask_together)
        # foreground = cv2.bitwise_and(color, cv2.cvtColor(
            mask_together, cv2.COLOR_GRAY2BGR))
        # contour = get_contour(mask_together)
        # cv2.imwrite('outputs/pigeon_foreground.jpg', foreground)
        # cv2.imwrite('outputs/pigeon_contour.jpg', contour)


        # image_pigeon = 'inputs/pigeon.jpeg'
        # num_iterss = [6,5,4]
        # reverse_masks = [1,1,1]
        # Ns = [2,3,4]
        # color = cv2.imread(image_pigeon)
        # texture = get_texture_image(color,Ns)
        # cv2.imwrite('outputs/pigeon_texture.jpg', texture)
        # image = texture
        # channels = cv2.split(image)
        # masks, mask_together = Otsu_for_rgb_img(image, num_iterss,
            reverse_masks)
        # for i in range(len(channels)):
        #       cv2.imwrite('outputs/pigeon_texture_mask_' + str(i) +
            '.jpg', masks[i])
        # cv2.imwrite('outputs/pigeon_texture_finalmask.jpg',
            mask_together)
        # foreground = cv2.bitwise_and(color, cv2.cvtColor(
            mask_together, cv2.COLOR_GRAY2BGR))
        # contour = get_contour(mask_together)
        # cv2.imwrite('outputs/pigeon_texture_foreground.jpg',
            foreground)
        # cv2.imwrite('outputs/pigeon_texture_contour.jpg', contour)

        # image_Red_Fox = 'inputs/Red-Fox.jpg'
        # num_iterss = [2,2,2]
        # reverse_masks = [0,0,0]
        # color = cv2.imread(image_Red_Fox)
        # image = color
        # channels = cv2.split(image)
        # masks, mask_together = Otsu_for_rgb_img(image, num_iterss,
            reverse_masks)
        # for i in range(len(channels)):
        #       cv2.imwrite('outputs/Red_Fox_mask_' + str(i) + '.jpg',
            masks[i])
        # cv2.imwrite('outputs/Red_Fox_finalmask.jpg', mask_together)

        # foreground = cv2.bitwise_and(color, cv2.cvtColor(
            mask_together, cv2.COLOR_GRAY2BGR))
        # contour = get_contour(mask_together)
        # cv2.imwrite('outputs/Red_Fox_foreground.jpg', foreground)
        # cv2.imwrite('outputs/Red_Fox_contour.jpg', contour)


        # image_Red_Fox = 'inputs/Red-Fox.jpg'
        # num_iterss = [1,1,1]
        # reverse_masks = [0,0,0]
        # Ns = [5,6,7]
        # color = cv2.imread(image_Red_Fox)
        # texture = get_texture_image(color,Ns)
        # cv2.imwrite('outputs/Red_Fox_texture.jpg', texture)
        # image = texture
        # channels = cv2.split(image)
        # masks, mask_together = Otsu_for_rgb_img(image, num_iterss, reverse_masks)
        # for i in range(len(channels)):
        #       cv2.imwrite('outputs/Red_Fox_texture_mask_' + str(i) + '.jpg', masks[i])
        # cv2.imwrite('outputs/Red_Fox_texture_finalmask.jpg', mask_together)
        # foreground = cv2.bitwise_and(color, cv2.cvtColor(mask_together, cv2.COLOR_GRAY2BGR))
        # contour = get_contour(mask_together)
        # cv2.imwrite('outputs/Red_Fox_texture_foreground.jpg', foreground)
        # cv2.imwrite('outputs/Red_Fox_texture_contour.jpg', contour)


        print("finish")


result()
```