ECE 661: Homework 6

Wei Xu
Email: xu1639@purdue.edu
Due date: 11:59 pm, Oct. 19, 2022
(Fall 2022)

1 Theory Question

(1) Otsu Algorithm

Advantage. The algorithm is simple. It can effectively segment the image when the area difference between the foreground object and the background is not large.

Disadvantage. When the areas of the foreground object and the background in the image are very different, there will not be the obvious two peaks in the histogram, or the sizes of the two peaks are very different. Then the segmentation will not be good, and the foreground object and the background cannot be accurately separated, especially when they have a large overlap of grayscale values. This is due to the fact that the method ignores the spatial information of the image, while using the grayscale distribution of the image as the basis for the image segmentation, and is also quite sensitive to noise.

(2) Watershed Algorithm

Advantage. The algorithm detects contours with closure and it is fast in detection.

Disadvantage. It does not learn from the region that is obviously foreground, for example to build a mixed cloth Gaussian model. And there will be mis-segmentation or over-segmentation in complex scenes.

2 Implementation Description

2.1 Otsu algorithm

The algorithm exhaustively searches for the threshold that maximizes the between-class variance. It can be implemented through the following steps.

(1) Compute the histogram and probabilities of each intensity level (from 0 to $L-1$, L bins in total).

(2) For each possible threshold $1 \leq k \leq L-2$, compute class probabilities w and class means μ. The class probabilities are

$$w_0(k) = P_0(C_0) = \sum_{i=0}^{k-1} p_i \tag{1}$$

$$w_1(k) = P_1(C_1) = \sum_{i=k}^{L-1} p_i \tag{2}$$

The class means are

$$\mu_0(k) = \sum_{i=0}^{k-1} iPr(i|C_0) = \sum_{i=0}^{k-1} \frac{ip_i}{w_0(k)} \tag{3}$$

$$\mu_1(k) = \sum_{i=k}^{L-1} iPr(i|C_1) = \sum_{i=k}^{L-1} \frac{ip_i}{w_1(k)} \tag{4}$$

Then compute the between-class scatter

$$\sigma_b^2(k) = w_0(k)[\mu_T - \mu_0(k)]^2 + w_1(k)[\mu_T - \mu_1(k)]^2 = w_0(k)w_1(k)[\mu_1(k) - \mu_0(k)]^2 \tag{5}$$

where $\mu_T = \sum_{i=0}^{L-1} ip_i$. Record $\sigma_b^2(k)$.

(3) The desired threshold k^* corresponds to the maximum $\sigma_b^2(k)$. The mask is generated according to k^*.

(4) Repeat steps (2) and (3) until the mask is satisfying.

2.2 Otsu algorithm using RGB channels

The RGB channels are split. And each channel yields a mask through the Otsu algorithm. Then combine the masks. It can be implemented through the following steps.

(1) Split the RGB channels into three layers. Treat each layer as a single image.

(2) Run Otsu algorithm with each layer to get three masks.

(3) Combine the three masks with 'AND' operator to get better segmentation.

2.3 Otsu algorithm using texture features

In this method, the texture features are considered. Then run Otsu algorithm with the features. It can be implemented through the following steps.

(1) Convert the RGB image into grayscale.

(2) Use a $N \times N$ window sliding on the image. And compute the variance within the window for each pixel. Then get a variance layer having the same size as the original image.

(3) Repeat the last step with different kernel sizes N to get different layers.

(4) Treat these layers as channels. Run the algorithm with the similar logic as Otsu algorithm using RGB channels. Then get the final segmentation.

2.4 Contour extraction

To get better contour extraction, firstly do erosion and dilation to eliminate noise. Then place a window sliding on the mask to recognize the contour. It can be implemented through the following steps.

(1) Do erosion with the kernel size of 3.

(2) Do dilation withe the kernel size of 3.

(3) Use a 3×3 window sliding on the mask. If a pixel is in the foreground and at least one of the surrounding 8 pixels is in the background, consider this pixel being on contour.

3 Task 1

3.1 Optimal set of parameters

The optimal set of parameters for Task 1 is shown in Tab. 1.

Table 1: Optimal set of parameters in task 1.

image	method	parameter	value
Cat	RGB channels	number of iteration	$[1, 1, 1]$
	texture features	window size, N	$[7, 9, 11]$
		number of iteration	$[5, 5, 5]$
Car	RGB channels	number of iteration	$[1, 1, 1]$
	texture features	window size, N	$[3, 5, 7]$
		number of iteration	$[1, 1, 1]$

3.2 Image segmentation and contour extraction

3.2.1 RGB channels method

The results for *Cat* image are shown in Fig. 1-3. And the results for *Car* image are shown in Fig. 4-6.

(a) R (b) G (c) B

Figure 1: Masks from different color channels of *Cat* image.

Figure 2: Combined mask from RGB-based masks of *Cat* image.

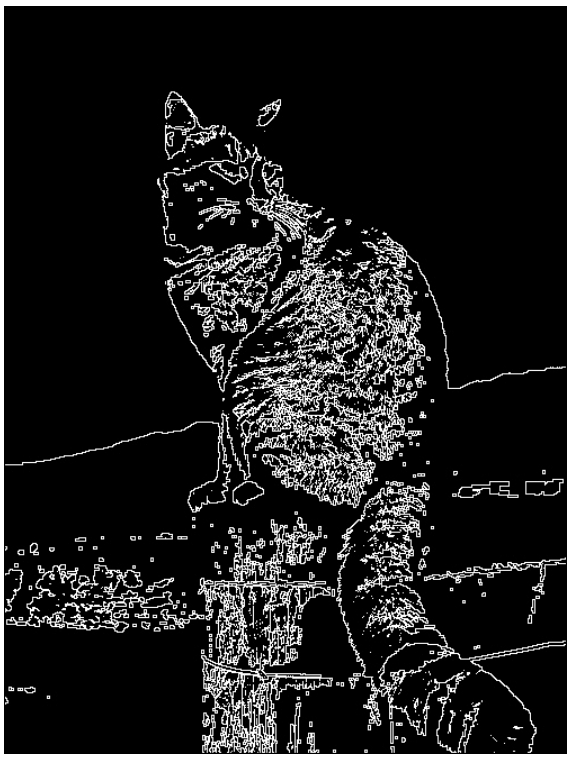
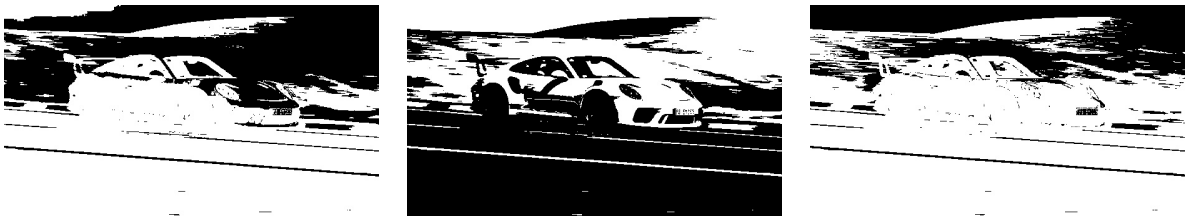


Figure 3: Contour from RGB-based mask of *Cat* image.



(a) R (b) G (c) B

Figure 4: Masks from different color channels of *Car* image.

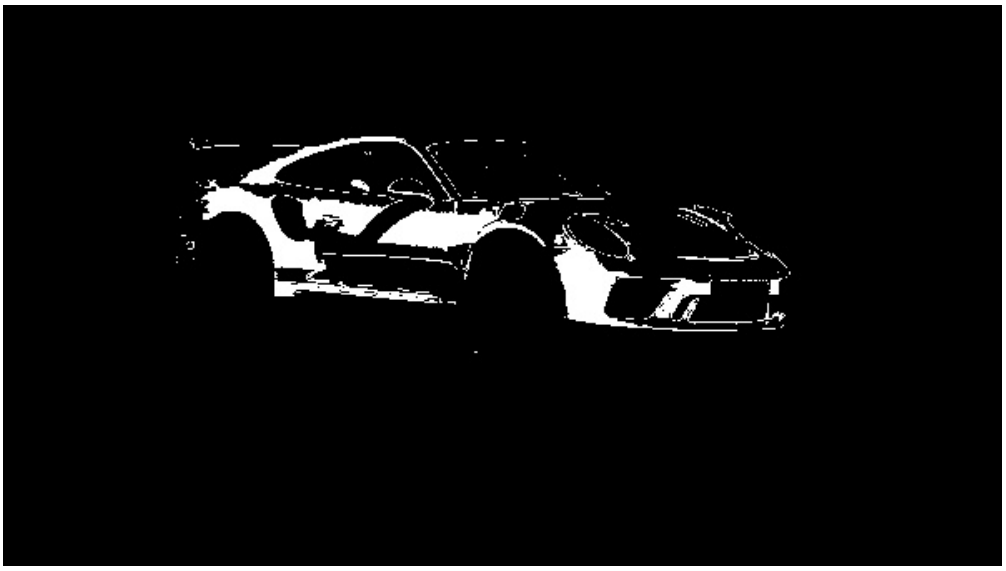


Figure 5: Combined mask from RGB-based masks of *Car* image.

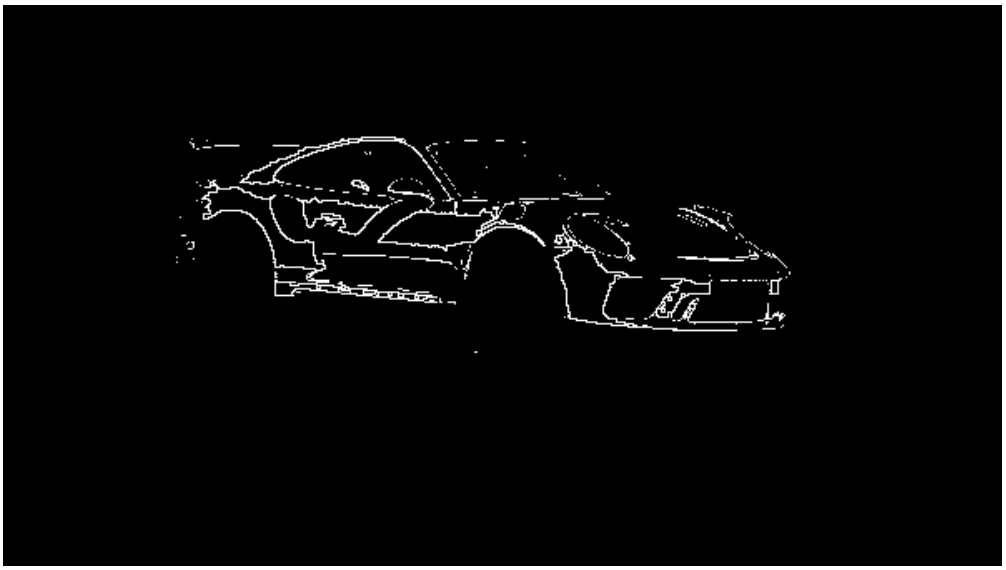


Figure 6: Contour from RGB-based mask of *Car* image.

3.2.2 Texture features method

The results for *Cat* image are shown in Fig. 7-9. And the results for *Car* image are shown in Fig. 10-12.

(a) $N = 7$ (b) $N = 9$ (c) $N = 11$

Figure 7: Masks from different window size of *Cat* image.

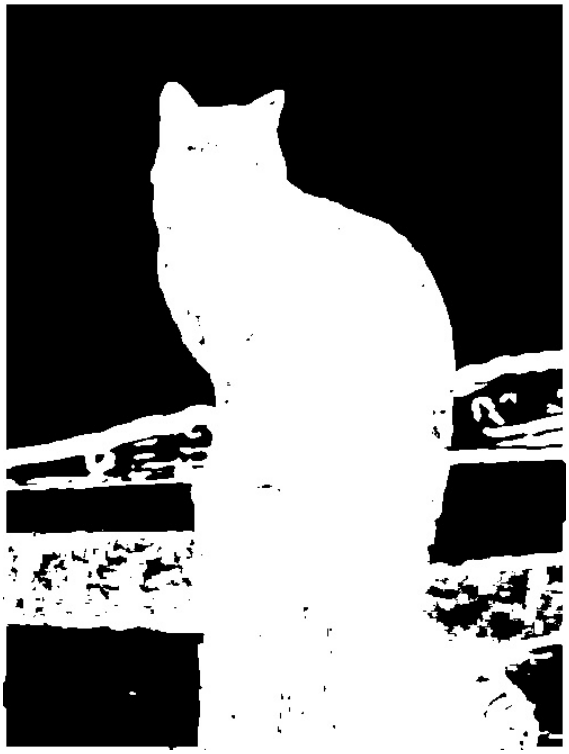


Figure 8: Combined mask from texture-based masks of *Cat* image.

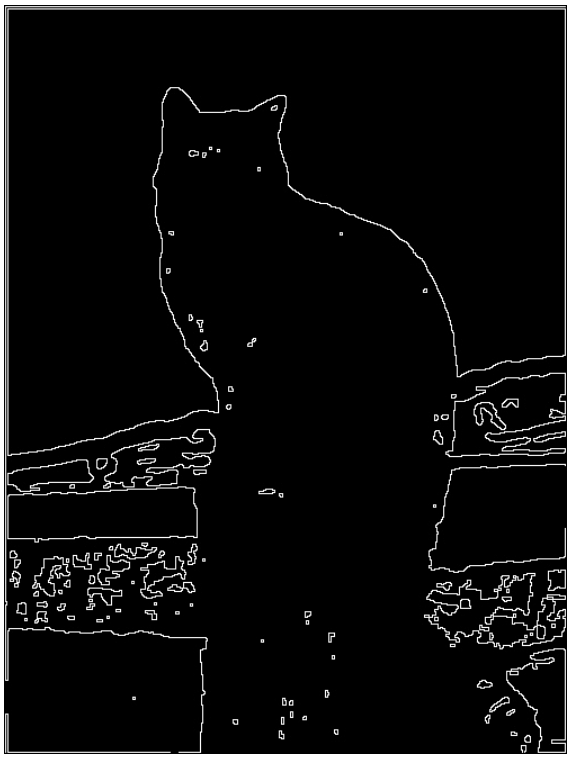
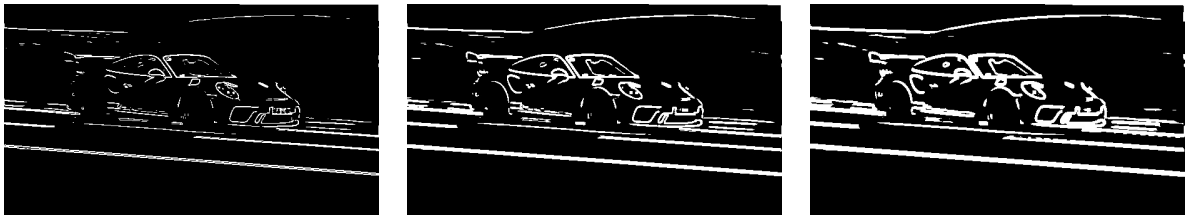


Figure 9: Contour from texture-based mask of *Cat* image.



(a) $N = 3$ (b) $N = 5$ (c) $N = 7$

Figure 10: Masks from different window size of *Car* image.

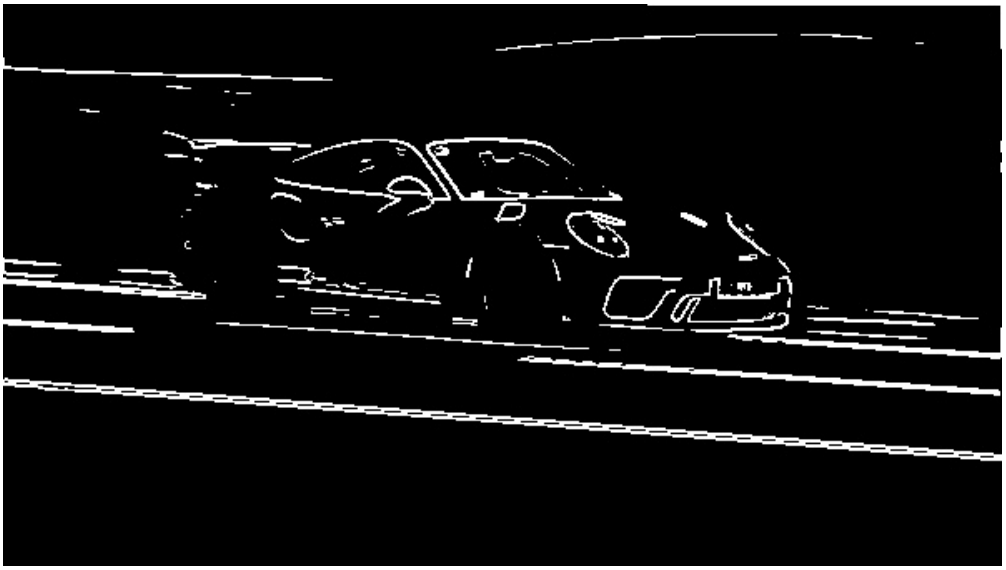


Figure 11: Combined mask from texture-based masks of *Car* image.

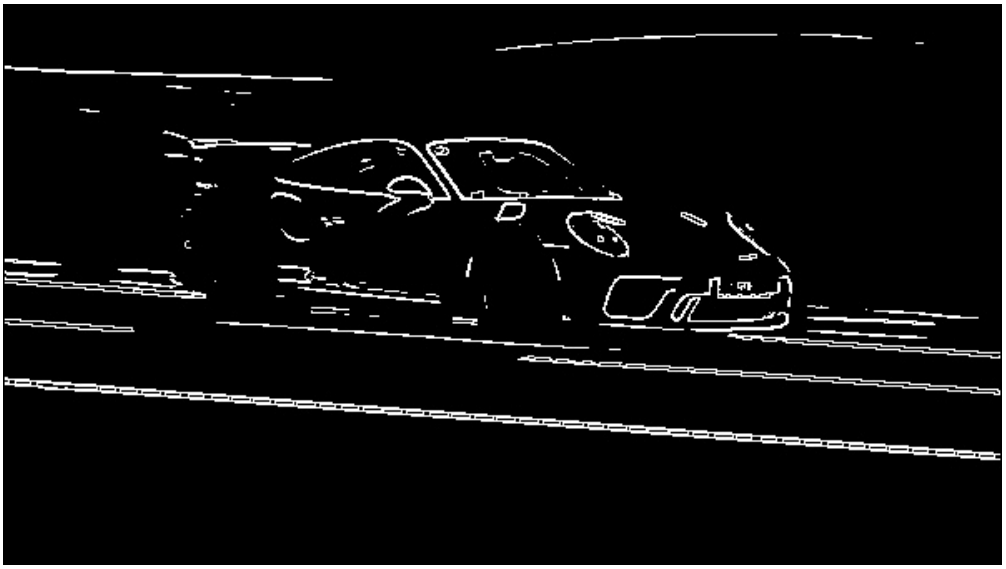


Figure 12: Contour from texture-based mask of *Car* image.

3.3 Discussion

Both methods do better on the *Cat* image, but they do not work well on the *Car* image.

For the RGB-based method, the performance is more stable. The cat and lawn can be separated from the background well on the blue channel. Because it is easier to recognize the sky with the blue channel. The car can be separated from the background well on the green channel due to its green color.

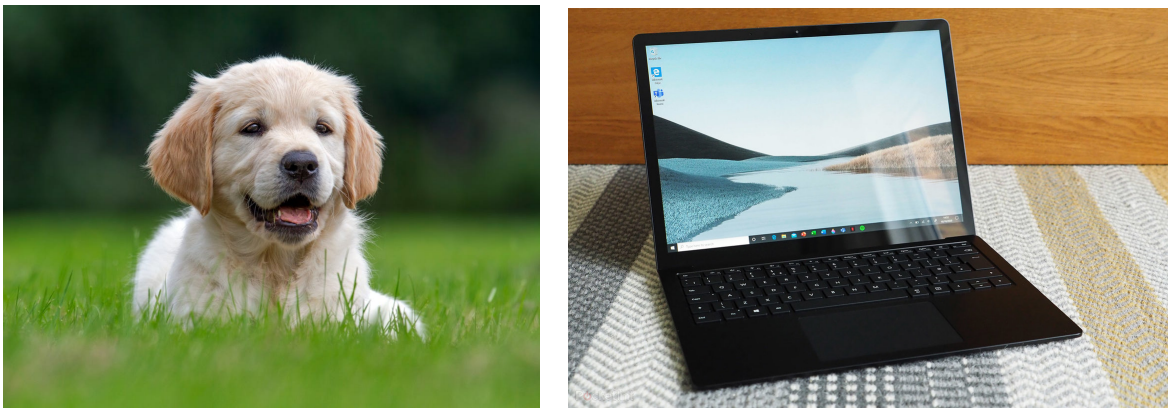
For the texture-based method, the performance is highly related to the image. It does better on the *Cat* image than the RGB-based method. Because the fur of the car has a complex texture. So the texture-based method can recognize it better. But for the *Car* image, there is no obvious

texture on the car, so the recognition is not satisfying. And the texture from the background also affects the results a lot.

4 Task 2

4.1 Inputs

The input images of Task 2 are shown in Fig. 13. The expected results should show the separation of a foreground object (dog and laptop) from the background.



(a) *Dog* (b) *Laptop*

Figure 13: Input images of Task 2.

4.2 Optimal set of parameters

The optimal set of parameters for Task 2 is shown in Tab. 2.

Table 2: Optimal set of parameters in task 2.

image	method	parameter	value
Dog	RGB channels	number of iteration	$[1, 1, 2]$
	texture features	window size, N	$[5, 9, 13]$
		number of iteration	$[10, 10, 10]$
Laptop	RGB channels	number of iteration	$[2, 2, 1]$
	texture features	window size, N	$[5, 9, 13]$
		number of iteration	$[2, 2, 2]$

4.3 Image segmentation and contour extraction

4.3.1 RGB channels method

The results for *Dog* image are shown in Fig. 14-16. And the results for *Laptop* image are shown in Fig. 17-19.

(a) R (b) G (c) B

Figure 14: Masks from different color channels of *Dog* image.

Figure 15: Combined mask from RGB-based masks of *Dog* image.

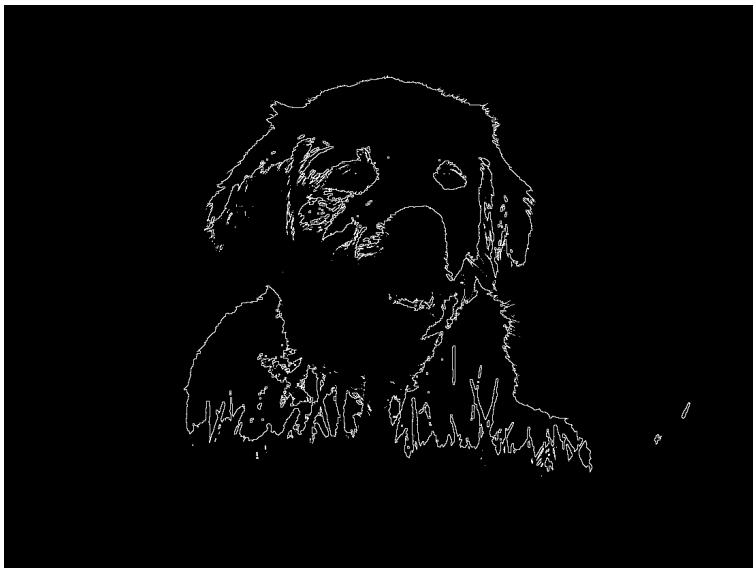


Figure 16: Contour from RGB-based mask of *Dog* image.

(a) R (b) G (c) B

Figure 17: Masks from different color channels of *Laptop* image.

Figure 18: Combined mask from RGB-based masks of *Laptop* image.

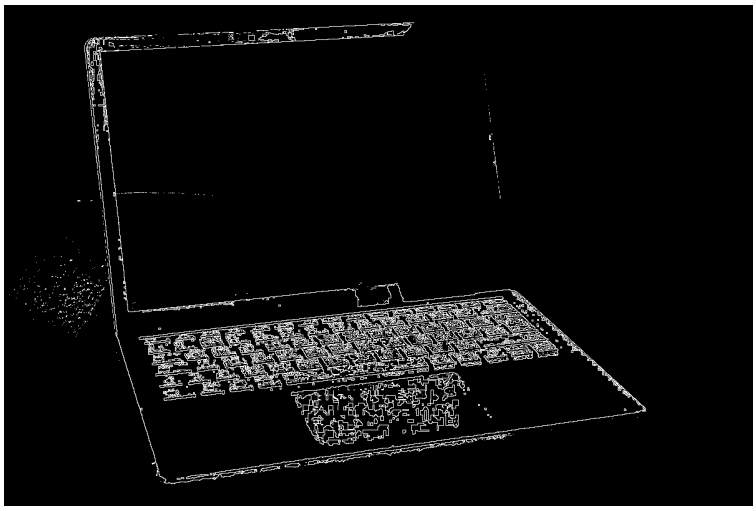


Figure 19: Contour from RGB-based mask of *Laptop* image.

4.3.2 Texture features method

The results for *Dog* image are shown in Fig. 20-22. And the results for *Laptop* image are shown in Fig. 23-25.

(a) $N = 5$ (b) $N = 9$ (c) $N = 13$

Figure 20: Masks from different window size of *Dog* image.

Figure 21: Combined mask from texture-based masks of *Dog* image.

Figure 22: Contour from texture-based mask of *Dog* image.

(a) $N = 5$ (b) $N = 9$ (c) $N = 13$

Figure 23: Masks from different window size of *Laptop* image.

Figure 24: Combined mask from texture-based masks of *Laptop* image.

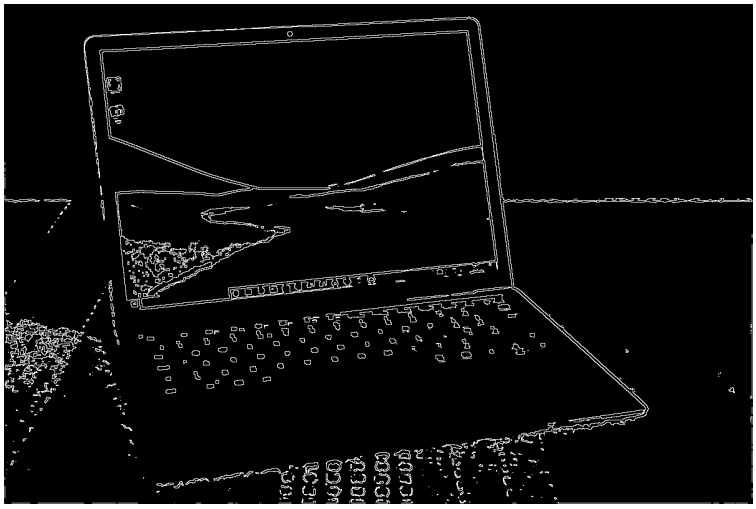


Figure 25: Contour from texture-based mask of *Laptop* image.

4.4 Discussion

In this task, both methods do well as expected.

The RGB-based method does better than the texture-based method in general. The RGB-based method provides more smooth results. The dog is recognized well. Notice that even in the green channel, the head of the dog is separated. And the laptop is separated from the background well, even though there is some noise.

For the texture-based method, it is easy to be affected by the noise. The dog is not recognized well compared with the cat in *Cat* image. Although they both have fur, *Dog* image also contains some detailed grass texture, so it is harder to separate the dag. And the result of the *Laptop* image has more noise from the table.

5 Source Code

```
import argparse
import cv2
import copy
import numpy as np

def otsu_grayscale(img, num_iter, inverse):
    # Otsu algorithm for a single channel
    mask = np.ones(img.shape, dtype=bool)
    for i in range(num_iter):
        hist, bins = np.histogram(img[mask],\
            np.arange(0,257), density=True)
        x_value = -np.inf
        threshold = -np.inf
        for b in bins[1:-1]:
            w_0 = np.sum(hist[:b])
            w_1 = np.sum(hist[b:])
            if w_0 == 0 or w_1 == 0:
                continue
            mu_0 = np.sum(np.multiply(bins[:b],hist[:b])) / w_0
            mu_1 = np.sum(np.multiply(bins[b:-1],hist[b:])) / w_1
            x = w_0 * w_1 * (mu_1 - mu_0)**2
            if x >= x_value:
                x_value = x
                threshold = b
        if inverse:
            mask[img > threshold] = 0
        else:
            mask[img < threshold] = 0
    return mask

def otsu_rgb_channel(img, num_iter=[1,1,1],\
    inverse=[0,0,0], negtive=False):
    # Otsu algorithm using RGB channels
    num_channel = img.shape[2]
    mask = np.zeros((img.shape), dtype=int)
    for i in range(num_channel):
        print('Processing %dth channel out of %d...'\
            %(i+1, num_channel))
```

```
        img_channel = img[:,:,i]
        mask[:,:,i] = otsu_grayscale(img_channel,\
            num_iter[i], inverse[i])
    if negtive:
        mask = 1-mask
    mask_segmentation = np.zeros((img.shape[0:2]), dtype=int)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if np.sum(mask[i,j,:]) == num_channel:
                mask_segmentation[i,j] = 1
    return mask, mask_segmentation

def pad_with(vector, pad_width, iaxis, kwargs):
    # image padding
    pad_value = kwargs.get('padder', 10)
    vector[:pad_width[0]] = pad_value
    vector[-pad_width[1]:] = pad_value

def otsu_texture_feature(img, N_list, num_iter,\
    inverse, negtive=False):
    # Otsu algorithm using texture features
    channels = np.zeros((img.shape[0],img.shape[1],len(N_list)),\
        dtype=float)
    for k in range(len(N_list)):
        pad_size = N_list[k] // 2
        img_pad = np.pad(img, pad_size, pad_with, padder=np.mean(img))
        for i in range(img.shape[0]):
            for j in range(img.shape[1]):
                window = img_pad[i:i+N_list[k], j:j+N_list[k]]
                channels[i,j,k] = np.var(window)
        channels[:,:,k] = (channels[:,:,k]-np.amin(channels[:,:,k]))\
            / (np.amax(channels[:,:,k])-np.amin(channels[:,:,k])) * 255
    mask, mask_segmentation = otsu_rgb_channel(\
        channels.astype(np.uint8), num_iter, inverse, negtive)
    return mask, mask_segmentation

def get_contour(mask, ed=False):
    # contour recognition
    img_contour = np.zeros(mask.shape, dtype=bool)
    mask_denoise = copy.deepcopy(mask)
    if ed:
        mask_padded_e = np.pad(mask_denoise, 1, pad_with, padder=0)
        for i in range(mask.shape[0]):
            for j in range(mask.shape[1]):
                if np.sum(mask_padded_e[i:i+3, j:j+3]) == 9:
                    mask_denoise[i,j] = 1
                else:
                    mask_denoise[i,j] = 0
        mask_padded_d = np.pad(mask_denoise, 1, pad_with, padder=0)
        for i in range(mask.shape[0]):
            for j in range(mask.shape[1]):
                if mask_padded_d[i+1, j+1] == 1:
                    mask_denoise[i:i+3, j:j+3] = 1
    for i in range(1,mask.shape[0]-1):
```

```
        for j in range(1,mask.shape[1]-1):
            if mask_denoise[i,j] == 0:
                continue
            elif np.amin(mask_denoise[i-1:i+2, j-1:j+2]) == 0:
                img_contour[i,j] = 1
    return img_contour

def plot_mask(mask_seg, title, mask=None):
    # masks plot
    cv2.imwrite('%s.jpg'%title, mask_seg*255)
    if mask is not None:
        for i in range(mask.shape[2]):
            cv2.imwrite('%s_%d.jpg'%(title,i), mask[:,:,i]*255)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    # '1.1' -- Task 1, RGB-based method
    # '1.2' -- Task 1, texture-based method
    # '2.1' -- Task 2, RGB-based method
    # '2.2' -- Task 2, texture-based method
    parser.add_argument('-t', '--task', type=str, default='2.2',\
        help='choose a task', choices=['1.1','1.2','2.1','2.2'])
    args = parser.parse_args()

    if args.task == '1.1':
        car = cv2.imread('./HW6-Images/car.jpg')
        cat = cv2.imread('./HW6-Images/cat.jpg')
        mask, mask_seg = otsu_rgb_channel(car, [1,1,1], [1,0,1])
        plot_mask(mask_seg, 'car_rgb', mask)
        car_contour = get_contour(mask_seg)
        plot_mask(car_contour, 'car_rgb_contour')
        mask, mask_seg = otsu_rgb_channel(cat, [1,1,1], [1,1,1])
        plot_mask(mask_seg, 'cat_rgb', mask)
        cat_contour = get_contour(mask_seg, True)
        plot_mask(cat_contour, 'cat_rgb_contour')
    if args.task == '1.2':
        car = cv2.imread('./HW6-Images/car.jpg', cv2.IMREAD_GRAYSCALE)
        cat = cv2.imread('./HW6-Images/cat.jpg', cv2.IMREAD_GRAYSCALE)
        mask, mask_seg = otsu_texture_feature(car, [3,5,7],\
            [1,1,1], [0,0,0])
        plot_mask(mask_seg, 'car_texture', mask)
        car_contour = get_contour(mask_seg)
        plot_mask(car_contour, 'car_texture_contour')
        mask, mask_seg = otsu_texture_feature(cat, [7,9,11],\
            [5,5,5], [1,1,1], True)
        plot_mask(mask_seg, 'cat_texture', mask)
        cat_contour = get_contour(mask_seg, True)
        plot_mask(cat_contour, 'cat_texture_contour')
    if args.task == '2.1':
        dog = cv2.imread('./HW6-Images2/dog.jpg')
        cup = cv2.imread('./HW6-Images2/cup.jpg')
        mask, mask_seg = otsu_rgb_channel(dog, [1,1,2], [0,0,0])
        plot_mask(mask_seg, 'dog_rgb', mask)
        dog_contour = get_contour(mask_seg)
```

```
147         plot_mask(dog_contour, 'dog_rgb_contour')
148         mask, mask_seg = otsu_rgb_channel(cup, [2,2,1], [1,1,1])
149         plot_mask(mask_seg, 'cup_rgb', mask)
150         cup_contour = get_contour(mask_seg)
151         plot_mask(cup_contour, 'cup_rgb_contour')
152     if args.task == '2.2':
153         dog = cv2.imread('./HW6-Images2/dog.jpg', cv2.IMREAD_GRAYSCALE)
154         cup = cv2.imread('./HW6-Images2/cup.jpg', cv2.IMREAD_GRAYSCALE)
155         mask, mask_seg = otsu_texture_feature(dog, [5,9,13],\
156             [10,10,10], [1,1,1], True)
157         plot_mask(mask_seg, 'dog_texture', mask)
158         dog_contour = get_contour(mask_seg)
159         plot_mask(dog_contour, 'dog_texture_contour')
160         mask, mask_seg = otsu_texture_feature(cup, [5,9,13],\
161             [2,2,2], [1,1,1], True)
162         plot_mask(mask_seg, 'cup_texture', mask)
163         cup_contour = get_contour(mask_seg)
164         plot_mask(cup_contour, 'cup_texture_contour')
```