ECE661 Computer Vision HW6

Chengjun Guo
guo456@purdue.edu

19 October 2022

1 Theory question

Lecture 15 presented 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 discussed in Lecture 14.)

1.1 Watershed Algorithm

In watershed algorithm, the image gradient is considered as a surface area with valleys. When we rain evenly, the first ridge line can be found when the water from one valley is meeting another valley. We put a dam at this dam and continue raining.
The advantage for this algorithm is that it will always produce closed contours at each ridge line. It is fast and intuitive.
The disadvantage for this algorithm is clear that the valley marking is critical. If we didn't mark valleys well, the result would be messed up. The common issue for this algorithm is over segmentation. The small ridges at each bottom of the valleys might be considered as a ridge lines.

1.2 Otsu algorithm

Otsu algorithm is implemented in this homework, the logic can be found in next section.
The advantage for this algorithm is that it works well with the bimodal. The calculation for this algorithm is simple.
The disadvantage of this algorithm is that it is not robust to the noise in a picture. Also, when the object has small area compared to a very large background area, the answer would not return a good result. Also, the illumination for the image should be uniform. It can be easily seen in the later section of the otsu output of box, the shadow of the box is considered as part of the box.

2 Logic

The code includes three defined functions, and the most of coding is the implementation for each images using the defined function.
The first function is otsu_algo, it is used to deal with one channel image or gray-scale image. It implement the otsu algorithm to find the threshold of the image and use the threshold to return a segmented image.
The second function is get_texture_image, it is returning the variance image that with input of window boundary length. It is also used for gray-scale image.
The last function is get_contour, it is used to mark the object border of the image.

2.1 Algorithms and implementation

2.1.1 otsu algorithm

For otsu algorithm, the pixel gray scale level histogram is from 0 to 255.

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

denotes the probability of pixel with gray level i, where n_i is the number of gray level i pixels and N is the total pixel numbers.
The pixels are dichotomized into two classes C_0 and C_1 by a threshold k. Then the probabilities for each class can be calculated by:

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

$$\omega_1 = \sum_{i=k+1}^{255} p_i$$

The mean of each class would be:

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

$$\mu_1 = \sum_{i=k+1}^{255} \frac{ip_i}{\omega_1}$$

. The variance between two classes can be found by:

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

where μ_T is the mean of total pixels.
We loop through all level of threshold k and find the k that maximize the variance which separate the foreground and background best. We will use that k as the threshold to divide the image.

2.1.2 segmentation using rgb values

For this algorithm, we perform the otsu algorithm in the previous section on each channel of the image. For output from each channel, we combine them with bitwise AND operator.

2.1.3 segmentation using texture

For this algorithm, we convert the image to grey scale image and then calculate the texture image. We use a window to swipe through the image pixel by pixel. The variance of the window would be the new value at the center point of the window. The new texture image would be used as the input for the otsu algorithm.

2.1.4 contour extraction

For contour extraction, we loop through the image with window size 3. In each window, if the center point have value and the pixels around it has empty value, we will consider this point as a edge point. After applying this algorithm to the image, the contour of the image would be depicted.

3 Data and images:

3.1 cat

The coefficient for cat:

	value		
window size for grey scale	channel1	channel2	channel3
	3	5	7
iteration for bgr	blue	green	red
	1	1	1

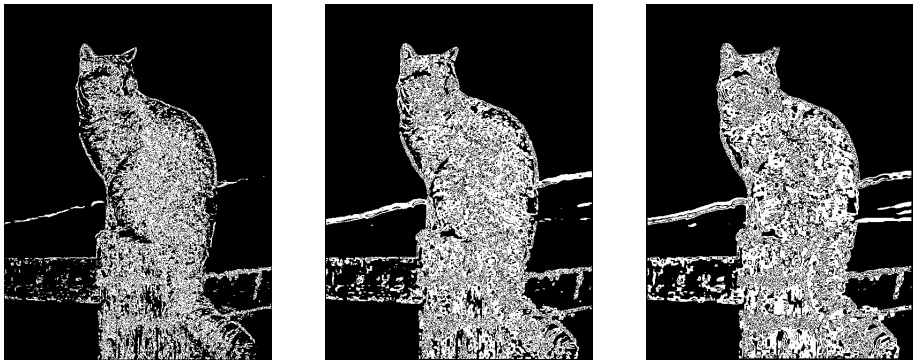
The cat rgb method output:

(a) blue (b) green (c) red

Figure 1: cat b g r channel

Figure 2: cat rgb output

The cat texture based method output:



(a) 3 (b) 5 (c) 7

Figure 3: cat grey scale

Figure 4: cat texture output

Contour extracted:

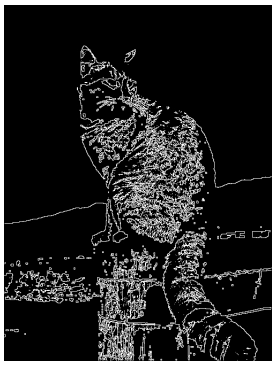


Figure 5: cat contour

3.2 car

The coefficient for car:

	value		
window size for grey scale	channel1	channel2	channel3
	3	5	7
iteration for bgr	blue	green	red
	1	1	1

The car rgb method output:

(a) blue (b) green (c) red

Figure 6: car b g r channel

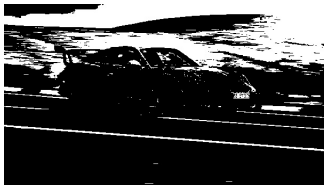


Figure 7: car rgb output

The car texture based method output:

(a) 3 (b) 5 (c) 7

Figure 8: car grey scale

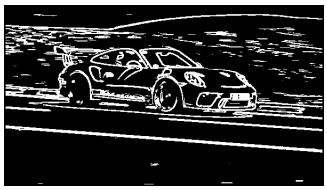


Figure 9: car texture output

Contour extracted:

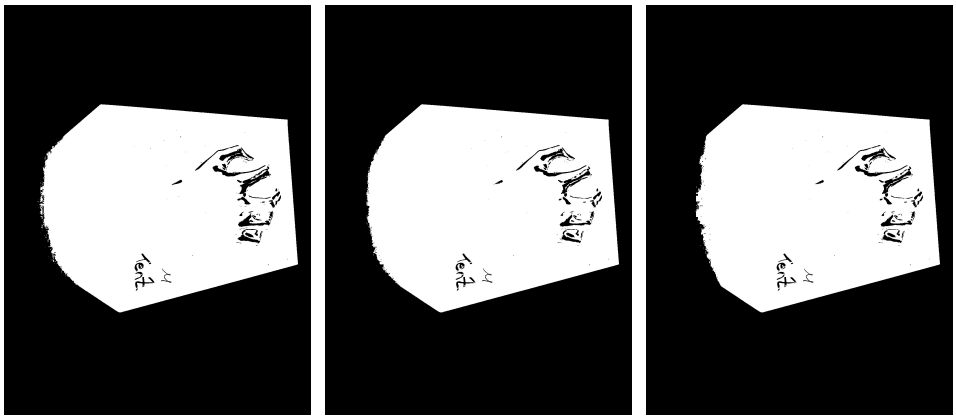
Figure 10: car contour

3.3 box

The coefficient for box:

	value		
window size for grey scale	channel1	channel2	channel3
	3	5	7
iteration for bgr	blue	green	red
	1	1	1

The box rgb method output:



(a) blue (b) green (c) red

Figure 11: box b g r channel

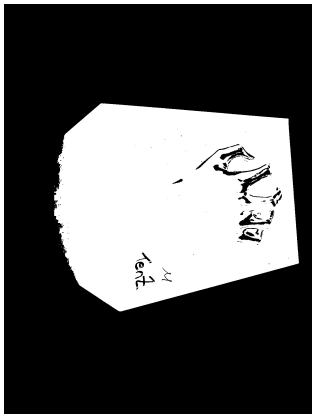
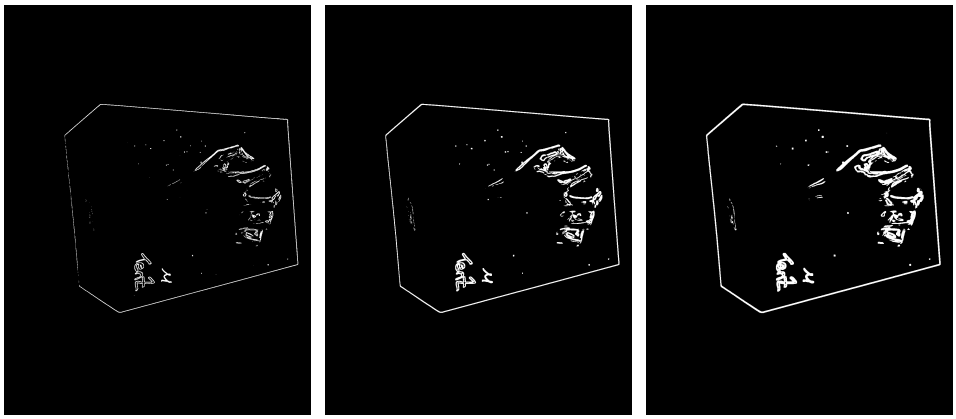


Figure 12: box rgb output

The box texture based method output:



(a) 3 (b) 5 (c) 7

Figure 13: box grey scale

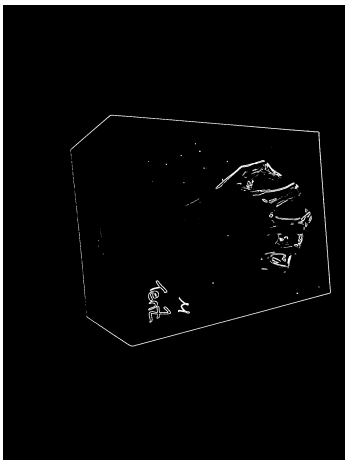


Figure 14: box texture output

Contour extracted:

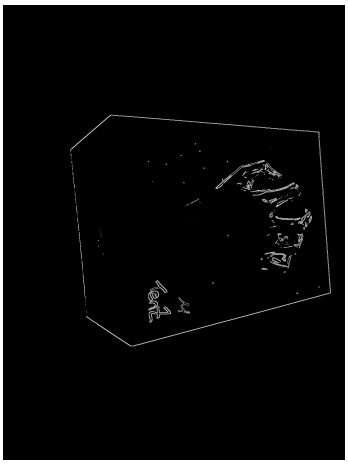


Figure 15: box contour

3.4 gum

The coefficient for gum:

	value		
window size for grey scale	channel1	channel2	channel3
	3	5	7
iteration for bgr	blue	green	red
	1	1	1

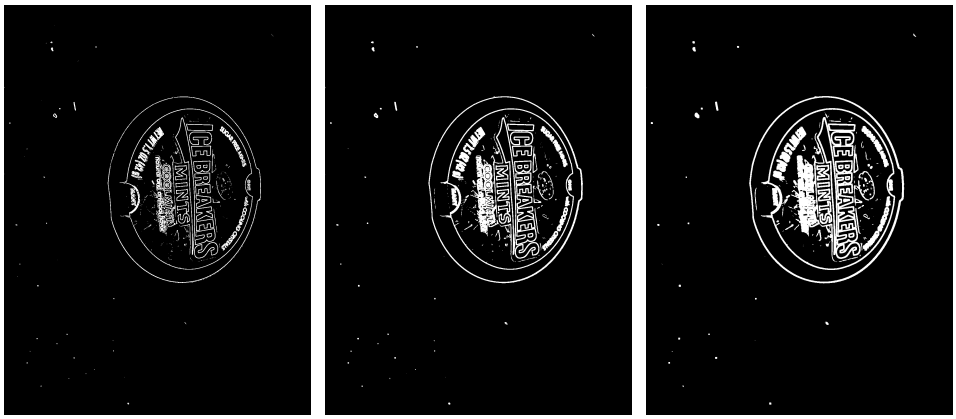
The gum rgb method output:

(a) blue (b) green (c) red

Figure 16: gum b g r channel

Figure 17: gum rgb output

The gum texture based method output:



(a) 3 (b) 5 (c) 7

Figure 18: gum grey scale

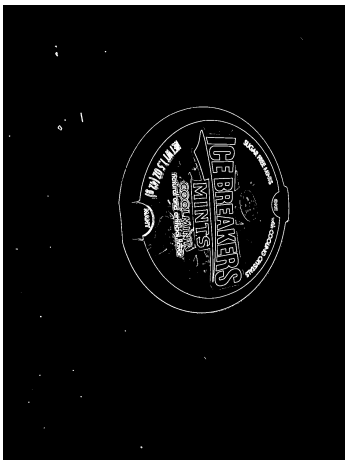


Figure 19: gum texture output

Contour extracted:

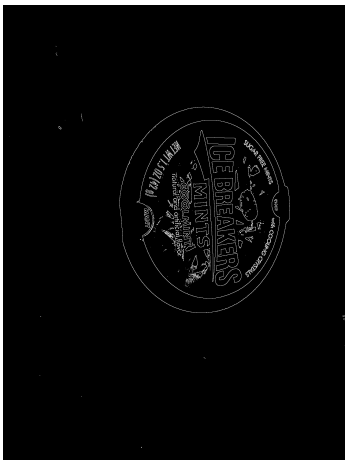


Figure 20: gum contour

4 Observation

4.1 RGB based segmentation

For the RGB otsu algorithm, the output are not good at separating the background and the object. Take the box image as example. The box is black and the shadow of the box is relatively dark compared to the background. It is hard for this algorithm to separate the shadow and the object with merely a threshold. Since the image is mainly black and white, the rgb channels have similar output. For a colorful input such as the car image, the green channel can separate the car best among the three channels since the car is green. Also, object filled with a different color than the background generally have better result for RGB segmentation.

4.2 Texture-based segmentation

For the texture based segmentation, it is sensitive to the edge. Take the box image as example, the edges is clearly detected. The shadow didn't affect the result. However, the edges in the background is also detected. Take the car as example, the edges in the background is hard to eliminate from the object. For different window size, take the gum as example. If the window size is too large, for a point in the middle of some part, the window will unexpectedly cover the part outside and result in a high variance. In this case, the image will look noisy for texture 3 in gum.

5 Code

```
import numpy as np
import cv2


def otsu_algo(image,iteration,reverse):
    #input gray image or one channel
    mask = np.ones(image.shape)
    for i in range(iteration):
        hist,bin_edges = np.histogram(image[np.nonzero(
            mask)], bins=np.arange(257), density=True)
        mu_T = np.mean(image[np.nonzero(mask)])
        i_hist = np.multiply(hist,np.arange(256))
        max_var = 0
        threshold = 0
        for k in range(256):
            omega_0 = np.sum(hist[:k])
            omega_1 = np.sum(hist[k:])
            mu_0 = np.sum(i_hist[:k])
            mu_1 = np.sum(i_hist[k:])
            if omega_0 == 0 or omega_1 == 0:
                continue
            var = omega_0 * omega_1 * (mu_0/omega_0 -
                mu_1/omega_1) ** 2
            if var > max_var:
                max_var = var
                threshold = k
        if reverse == 0:
            _, mask = cv2.threshold(image, threshold,
                255,cv2.THRESH_BINARY)
        else:
            _, mask = cv2.threshold(image, threshold,
                255,cv2.THRESH_BINARY_INV)
    return mask



def get_texture_image(image,N):
    #input gray image or one channel
    texture = np.zeros(image.shape)
    w = int((N-1)/2)
    for i in range(w,image.shape[0]-w):
```

```
        for j in range(w,image.shape[1]-w):
            window = image[i-w:i+w+1,j-w:j+w+1]
            texture[i,j] = np.var(window)
    return texture



def get_contour(mask):
    w = 1
    contour = np.zeros(mask.shape)
    for i in range(1,mask.shape[0]-1):
        for j in range(1,mask.shape[1]-1):
            if mask[i,j] == 0:
                continue
            if np.min(mask[i-w:i+w+1,j-w:j+w+1]) == 0:
                contour[i,j] = 255
    return contour



if __name__ == '__main__':

    # image1 cat
    name1 = 'cat.jpg'
    image1 = cv2.imread(name1)
    channel1, channel2, channel3 = [image1[:,:,0],image1
        [:,:,1],image1[:,:,2]]
    c1_mask = otsu_algo(channel1,1,1)
    c2_mask = otsu_algo(channel2,1,1)
    c3_mask = otsu_algo(channel3,1,1)
    cv2.imwrite('cat_mask_b.jpg', c1_mask)
    cv2.imwrite('cat_mask_g.jpg', c2_mask)
    cv2.imwrite('cat_mask_r.jpg', c3_mask)
    mask = np.ones(c1_mask.shape).astype(np.uint8) * 255
    for i in [c1_mask,c2_mask,c3_mask]:
        mask = cv2.bitwise_and(i,mask)
    cv2.imwrite('cat_mask.jpg',mask.astype(np.uint8))
    contour = get_contour(mask)
    cv2.imwrite('cat_rgb_contour.jpg',contour)

    image1 = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
    texture1 = get_texture_image(image1, 3).astype(np.
        uint8)
    texture2 = get_texture_image(image1, 5).astype(np.
        uint8)
```

```
texture3 = get_texture_image(image1, 7).astype(np.
    uint8)
c1_mask = otsu_algo(texture1,1,0)
c2_mask = otsu_algo(texture2,1,0)
c3_mask = otsu_algo(texture3,1,0)
cv2.imwrite('cat_texture_1.jpg', c1_mask)
cv2.imwrite('cat_texture_2.jpg', c2_mask)
cv2.imwrite('cat_texture_3.jpg', c3_mask)
mask = np.ones(c1_mask.shape).astype(np.uint8) * 255
for i in [c1_mask,c2_mask,c3_mask]:
    mask = cv2.bitwise_and(i,mask)
cv2.imwrite('cat_texture.jpg', mask.astype(np.uint8))
contour = get_contour(mask)
cv2.imwrite('cat_texture_contour.jpg',contour)

#image2 car

# name1 = 'car.jpg'
# image1 = cv2.imread(name1)
# channel1, channel2, channel3 = [image1[:,:,0],
    image1[:,:,1],image1[:,:,2]]
# c1_mask = otsu_algo(channel1,1,0)
# c2_mask = otsu_algo(channel2,1,0)
# c3_mask = otsu_algo(channel3,1,0)
# cv2.imwrite('car_mask_b.jpg', c1_mask)
# cv2.imwrite('car_mask_g.jpg', c2_mask)
# cv2.imwrite('car_mask_r.jpg', c3_mask)
# mask = np.ones(c1_mask.shape).astype(np.uint8) *
    255
# for i in [c1_mask,c2_mask,c3_mask]:
#     mask = cv2.bitwise_and(i,mask)
# cv2.imwrite('car_mask.jpg',mask.astype(np.uint8))
# contour = get_contour(mask)
# cv2.imwrite('car_rgb_contour.jpg',contour)
#
# image1 = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
# texture1 = get_texture_image(image1, 3)
# texture2 = get_texture_image(image1, 5)
# texture3 = get_texture_image(image1, 7)
# c1_mask = otsu_algo(texture1,3,0)
# c2_mask = otsu_algo(texture2,3,0)
# c3_mask = otsu_algo(texture3,3,0)
# cv2.imwrite('car_texture_1.jpg', c1_mask)
# cv2.imwrite('car_texture_2.jpg', c2_mask)
# cv2.imwrite('car_texture_3.jpg', c3_mask)
```

```
# mask = np.ones(c1_mask.shape).astype(np.uint8) *
    255
# for i in [c1_mask,c2_mask,c3_mask]:
#     mask = cv2.bitwise_and(i.astype(np.uint8),mask)
# cv2.imwrite('car_texture.jpg', mask.astype(np.uint8
    ))
# contour = get_contour(mask)
# cv2.imwrite('car_texture_contour.jpg',contour)


# image3 box

# name1 = 'box.jpg'
# image1 = cv2.imread(name1)
# channel1, channel2, channel3 = [image1[:,:,0],
    image1[:,:,1],image1[:,:,2]]
# c1_mask = otsu_algo(channel1,1,1)
# c2_mask = otsu_algo(channel2,1,1)
# c3_mask = otsu_algo(channel3,1,1)
# cv2.imwrite('box_mask_b.jpg', c1_mask)
# cv2.imwrite('box_mask_g.jpg', c2_mask)
# cv2.imwrite('box_mask_r.jpg', c3_mask)
# mask = np.ones(c1_mask.shape).astype(np.uint8) *
    255
# for i in [c1_mask,c2_mask,c3_mask]:
#     mask = cv2.bitwise_and(i,mask)
# cv2.imwrite('box_mask.jpg',mask.astype(np.uint8))
# contour = get_contour(mask)
# cv2.imwrite('box_rgb_contour.jpg',contour)
#
# image1 = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
# texture1 = get_texture_image(image1, 3)
# texture2 = get_texture_image(image1, 5)
# texture3 = get_texture_image(image1, 7)
# c1_mask = otsu_algo(texture1,3,0)
# c2_mask = otsu_algo(texture2,3,0)
# c3_mask = otsu_algo(texture3,3,0)
# cv2.imwrite('box_texture_1.jpg', c1_mask)
# cv2.imwrite('box_texture_2.jpg', c2_mask)
# cv2.imwrite('box_texture_3.jpg', c3_mask)
# mask = np.ones(c1_mask.shape).astype(np.uint8) *
    255
# for i in [c1_mask,c2_mask,c3_mask]:
#     mask = cv2.bitwise_and(i.astype(np.uint8),mask)
# cv2.imwrite('box_texture.jpg', mask.astype(np.uint8
    ))
```

```
# contour = get_contour(mask)
# cv2.imwrite('box_texture_contour.jpg',contour)

#image4 gum

# name1 = 'gum.jpg'
# image1 = cv2.imread(name1)
# channel1, channel2, channel3 = [image1[:,:,0],
    image1[:,:,1],image1[:,:,2]]
# c1_mask = otsu_algo(channel1,1,0)
# c2_mask = otsu_algo(channel2,1,0)
# c3_mask = otsu_algo(channel3,1,0)
# cv2.imwrite('gum_mask_b.jpg', c1_mask)
# cv2.imwrite('gum_mask_g.jpg', c2_mask)
# cv2.imwrite('gum_mask_r.jpg', c3_mask)
# mask = np.ones(c1_mask.shape).astype(np.uint8) *
    255
# for i in [c1_mask,c2_mask,c3_mask]:
#     mask = cv2.bitwise_and(i,mask)
# cv2.imwrite('gum_mask.jpg',mask.astype(np.uint8))
# contour = get_contour(mask)
# cv2.imwrite('gum_rgb_contour.jpg',contour)
#
# image1 = cv2.cvtColor(image1,cv2.COLOR_BGR2GRAY)
# texture1 = get_texture_image(image1, 3)
# texture2 = get_texture_image(image1, 5)
# texture3 = get_texture_image(image1, 7)
# c1_mask = otsu_algo(texture1,3,0)
# c2_mask = otsu_algo(texture2,3,0)
# c3_mask = otsu_algo(texture3,3,0)
# cv2.imwrite('gum_texture_1.jpg', c1_mask)
# cv2.imwrite('gum_texture_2.jpg', c2_mask)
# cv2.imwrite('gum_texture_3.jpg', c3_mask)
# mask = np.ones(c1_mask.shape).astype(np.uint8) *
    255
# for i in [c1_mask,c2_mask,c3_mask]:
#     mask = cv2.bitwise_and(i.astype(np.uint8),mask)
# cv2.imwrite('gum_texture.jpg', mask.astype(np.uint8
    ))
# contour = get_contour(mask)
# cv2.imwrite('gum_texture_contour.jpg',contour)
```