ECE 661 Computer Vision: HW6

Qiuchen Zhai
qzhai@purdue.edu

October 12, 2020

1 Theory Question: The Strengths and the Weaknesses of Otsu Algorithm and the Watershed Algorithm

The Otsu algorithm is fast, easy for calculation and effective for automatic image thresholding. It attains good results even under noisy constraints. However, the Otsu algorithm evaluates the pixels by sides of the threshold. As the number of classes in an image increases, the time constraint and computation cost for the threshold selection also increase.

The watershed algorithm is sensitive to local minima, which can cause serious over-segmentation especially when the image is noisy. However, watershed supports segmentation with multiple classes while the Otsu only support two classes.

2 Description of Programming Tasks

2.1 Otsu Algorithm

Given input 8-bit image, the implementation of Otsu algorithm follows the steps:

1. Create histogram of image data.
   Given the image, we could construct a 256-level histogram that counts the number of pixels in each gray level $h[i] = n_i$, where i denotes the gray level taking value from 0 to 255 and n_i denotes the number of pixels at gray level i.

2. Calculate the PMF and the overall average gray level.
   After we finish the construction of histogram, we could compute the probability of a certain gray level i by dividing the corresponding number n_i of pixels by the total number of image pixels N.

$$p_i = \frac{n_i}{N} \tag{1}$$

   Then the overall average gray level is given by

$$\mu_T = \sum_{i=1}^{L} i p_i \tag{2}$$

3. Determine two classes by the gray level threshold k.
   Given the gray level threshold $l = k$, the image pixels are divided into two classes C_0 and C_1.

The probability ot the two classes are

$$\omega_0 = p(C_0) = \sum_{i=1}^{k} p_i \tag{3}$$

$$\omega_1 = p(C_1) = \sum_{i=k+1}^{L} p_i \tag{4}$$

And the two classes could be described by the mean and variance

$$\mu_0 = \sum_{i=1}^{k} i\frac{p_i}{\omega_0} \textit{ and } \delta_0^2 = \sum_{i=1}^{k} (i-\mu_0)^2 \frac{p_i}{\omega_0} \tag{5}$$

$$\mu_1 = \sum_{i=k+1}^{L} \textit{ and } \delta_1^2 = \sum_{i=k+1}^{L} (i-\mu_1)^2 \frac{p_i}{\omega_1} \tag{6}$$

4. Compute the between class variance.
   The between class variance is calculated by

$$\delta_B^2 = \omega_0\omega_1(\mu_1 - \mu_0)^2 \tag{7}$$

5. Determine the value of threshold.
   The value of threshold k is determined by the value k^* that maximizes the between class variance.
6. Repeat the above steps if necessary.
7. Generate the mask according the two classes.

2.2 Image Segmentation using RGB Values

The following steps describe the procedure of image segmentation using RGB values,

1. Using RGB values of image for segmentation, each color channel of the RGB image is treated as a single image. Thus, we first split the RGB image into three grayscale image.
2. For each color channel, we use the Otsu algorithm to generate an individual mask for overall segmentation.
3. The final mask for overall segmentation is obtained by taking the logical AND among the individual masks.
4. Obtain the foreground image using the final mask. If the mask is noisy, we might use dilation and erosion techniques to remove noise.

2.3 Texture-based Segmentation

The following steps describes the procedure of texture-based segmentation,

1. To extract some texture-based features, we first convert the RGB image into grayscale image.
2. Then a window of $N \times N$ is placed at each pixel. Within each window, the mean intensity is subtracted and the intensity variance is computed as a texture measure at the center pixel. The step is repeated for $N = 3, 5, 7$.

3. For each obtained image, we use the Otsu algorithm to obtain an individual mask. Then we get the final mask by taking the logical AND among the individual masks. If the merged mask is noisy, we could use dilation and erosion techniques to denoise the mask.

4. Finally, the foreground image is obtained by the merged mask.

2.4 Contour Extraction

Given the binary mask, the contour is extracted by scanning the mask pixels. A pixel is determined to be ”on the contour” if

- the pixel value is 1.
- In 3×3 neighborhood around the pixel, there exists at least one zero-valued pixel.

Otherwise, the pixel is not ”on the contour”.

3 Results

3.1 cat.jpg

3.1.1 Parameters

	color channel:[B, G, R]	window size:[3x3, 5x5, 7x7]
number of iterations	[1, 1, 1]	[1, 1, 1]
dilation	3x3, 1 iteration	7x7, 1 iteration
erosion	3x3, 1 iteration	9x9, 1 iteration

Table 1: Set of parameters

3.1.2 RGB-based Segmentation

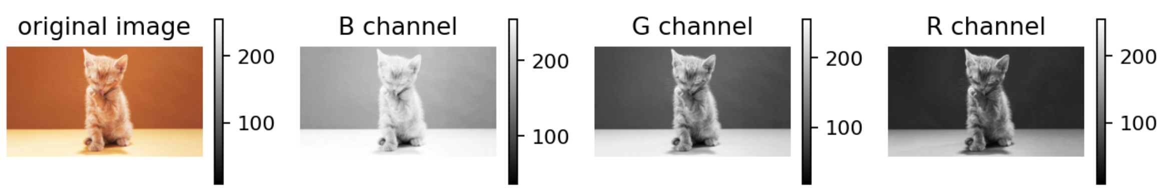


Figure 1: Original image and RGB channels

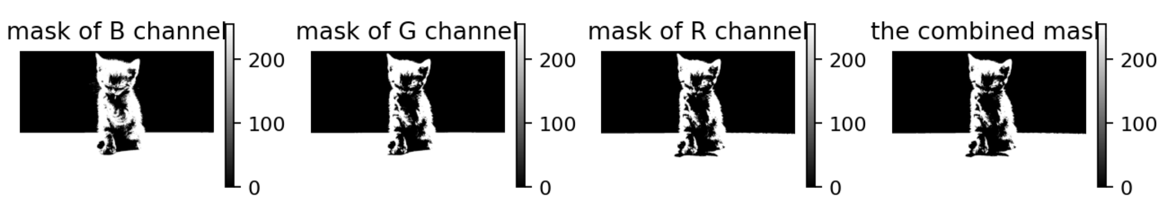


Figure 2: Generated masks of RGB channels

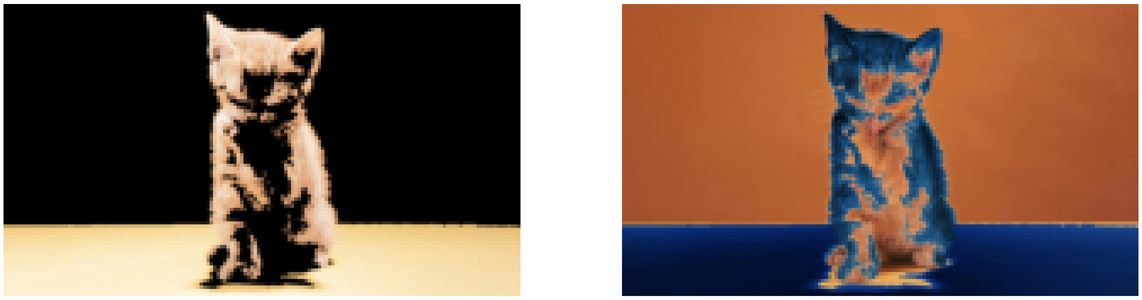


Figure 3: Foreground (left) and background (right) images

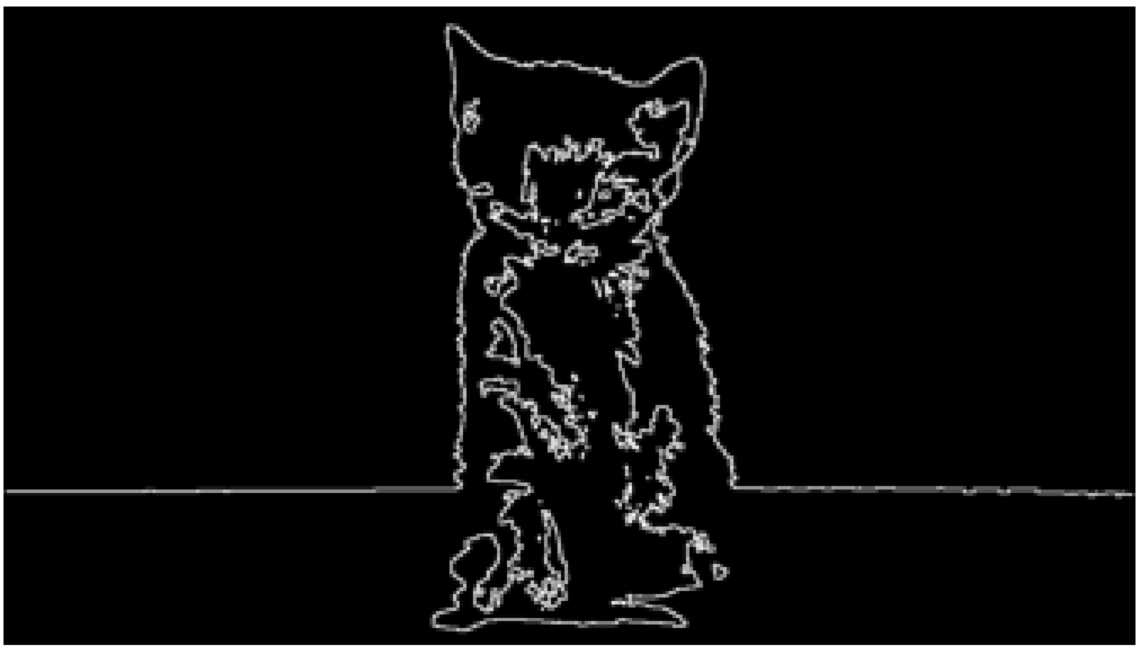


Figure 4: Extracted contour using RGB values

3.1.3 Texture-based Segmentation

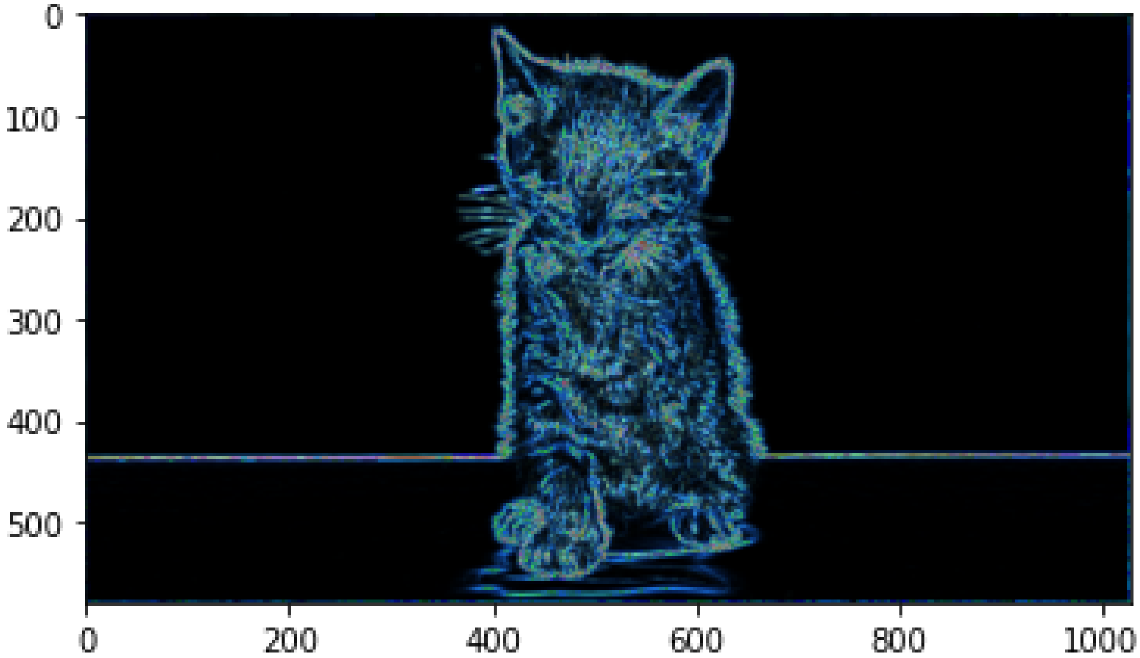


Figure 5: Texture image

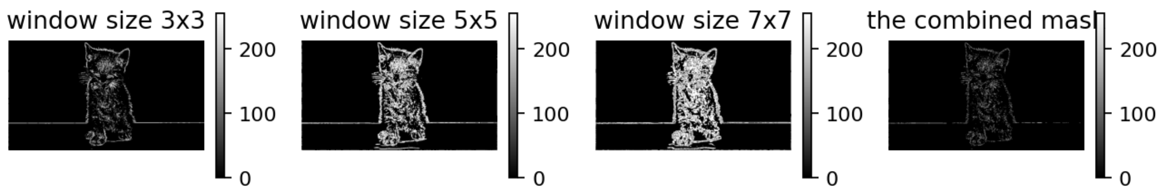


Figure 6: Generated masks of different window sizes

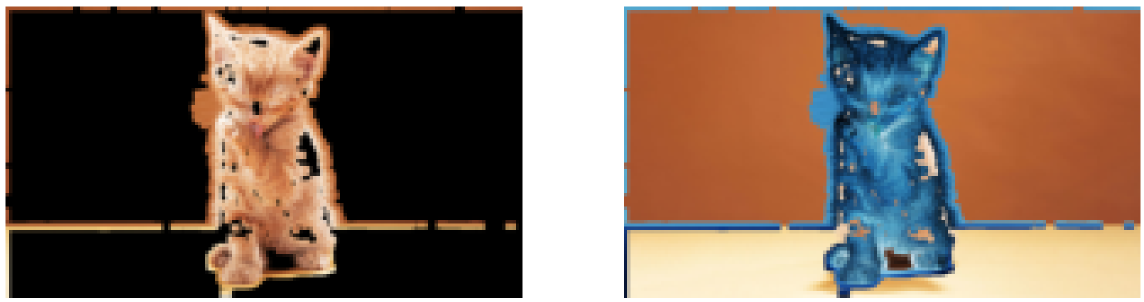


Figure 7: Foreground (left) and background (right) images

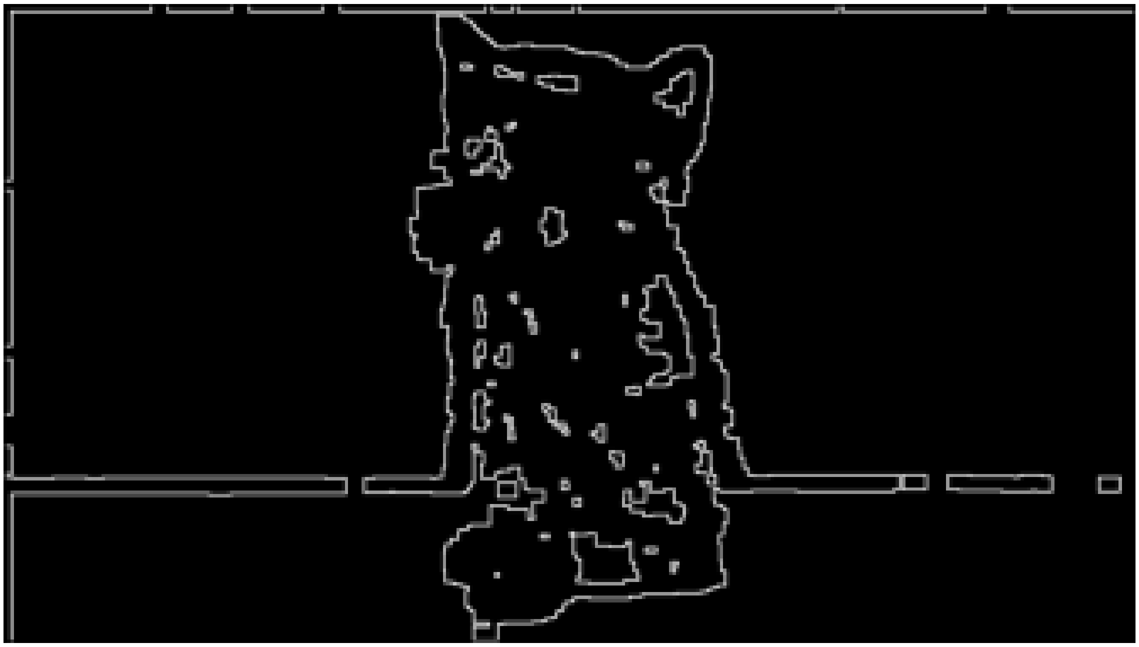


Figure 8: Extracted contour based on texture features

3.2 Red-Fox.jpg

3.2.1 Parameters

	color channel:[B, G, R]	window size:[3x3, 5x5, 7x7]
number of iterations	[2, 2, 1]	[1, 1, 1]
dilation	5x5, 1 iteration	7x7, 1 iteration
erosion	3x3, 1 iteration	3x3, 1 iteration

Table 2: Set of parameters

3.2.2 RGB-based Segmentation

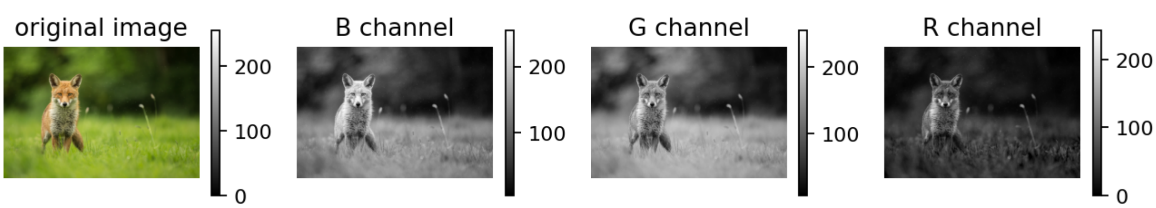


Figure 9: Original image and RGB channels

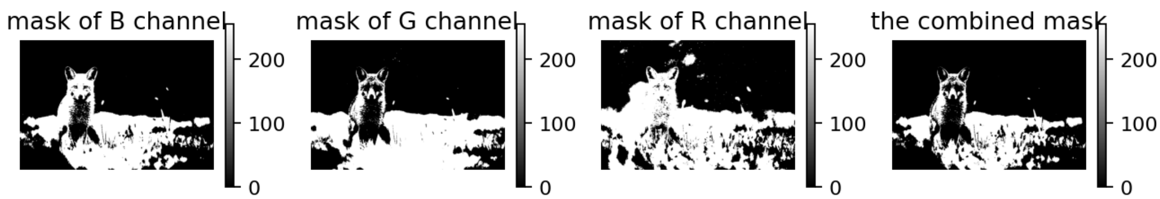


Figure 10: Generated masks of RGB channels

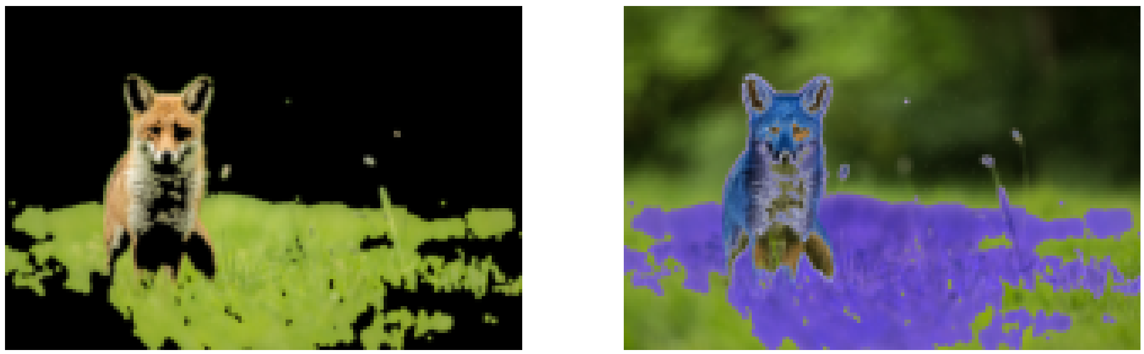


Figure 11: Foreground (left) and background (right) images

Figure 12: Extracted contour using RGB values

3.2.3 Texture-based Segmentation

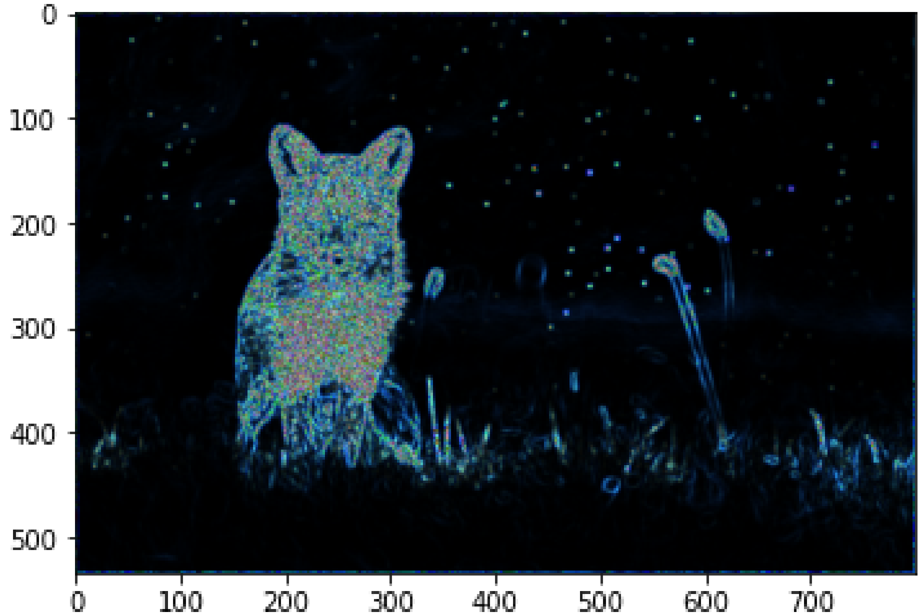


Figure 13: Texture image

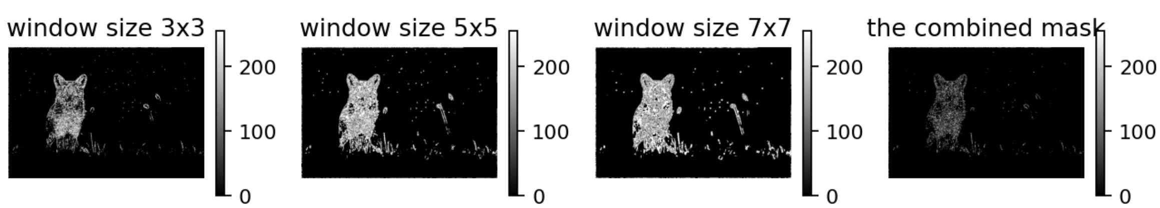


Figure 14: Generated masks of different window sizes

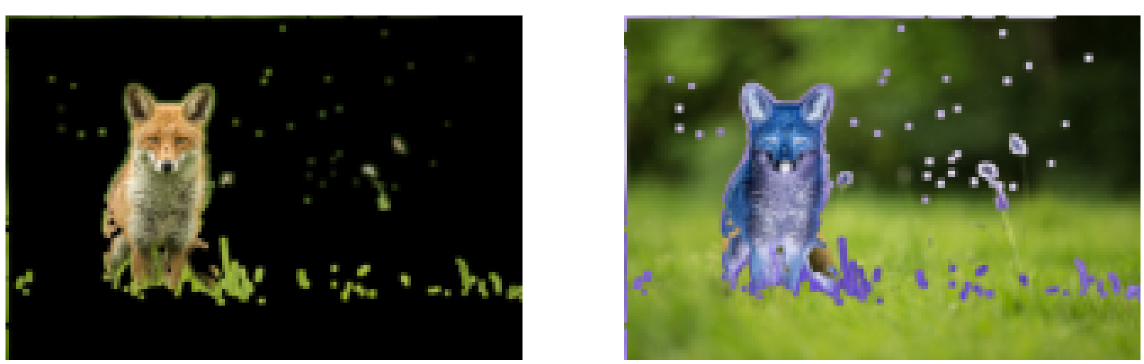


Figure 15: Foreground (left) and background (right) images

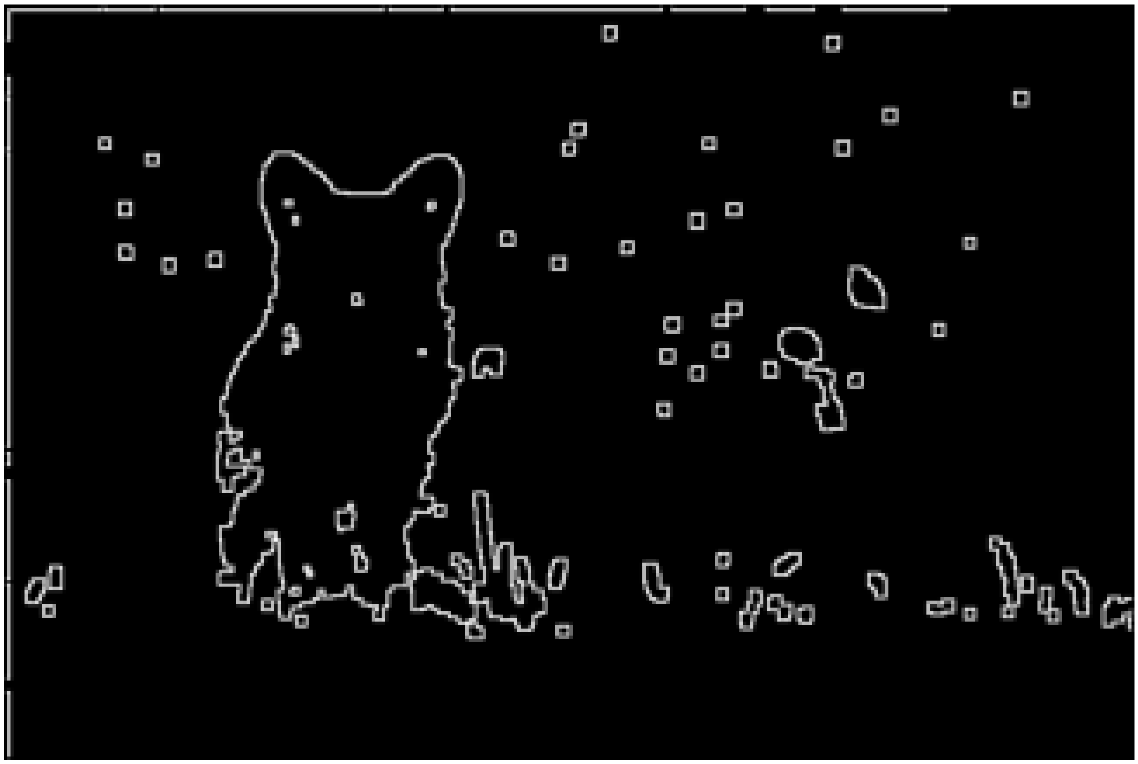


Figure 16: Extracted contour based on texture features

3.3 pigeon.jpg

3.3.1 Parameters

	color channel:[B, G, R]	window size:[3x3, 5x5, 7x7]
number of iterations	[2, 2, 2]	[1, 2, 2]
dilation	3x3, 2 iteration	9x9, 1 iteration
erosion	3x3, 1 iteration	3x3, 2 iteration

Table 3: Set of parameters

3.3.2 RGB-based Segmentation

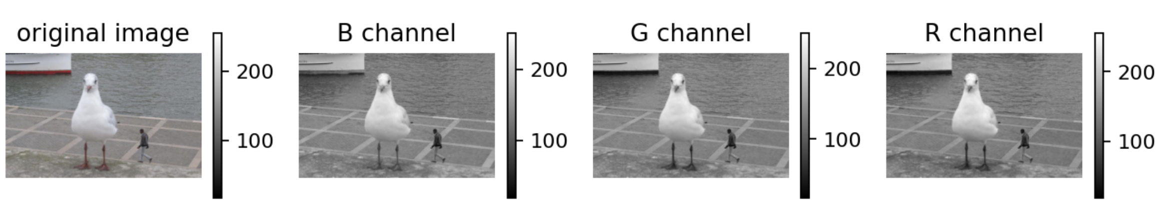


Figure 17: Original image and RGB channels

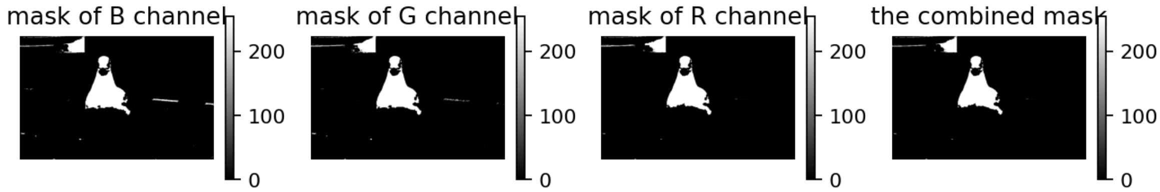
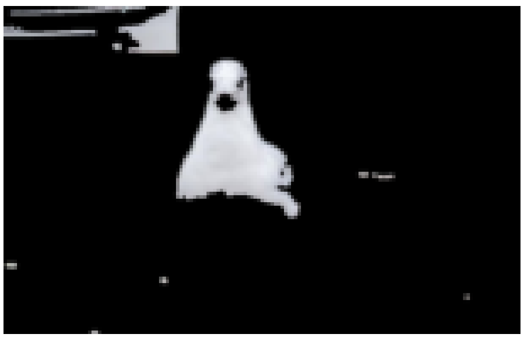


Figure 18: Generated masks of RGB channels



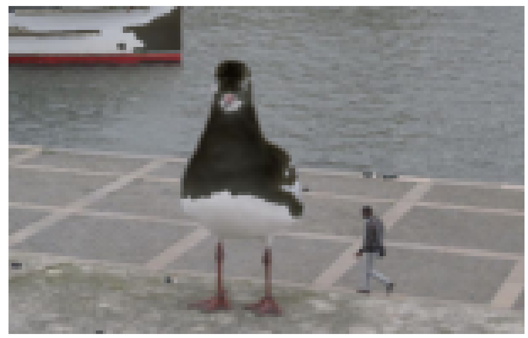


Figure 19: Foreground (left) and background (right) images

Figure 20: Extracted contour using RGB values

3.3.3 Texture-based Segmentation

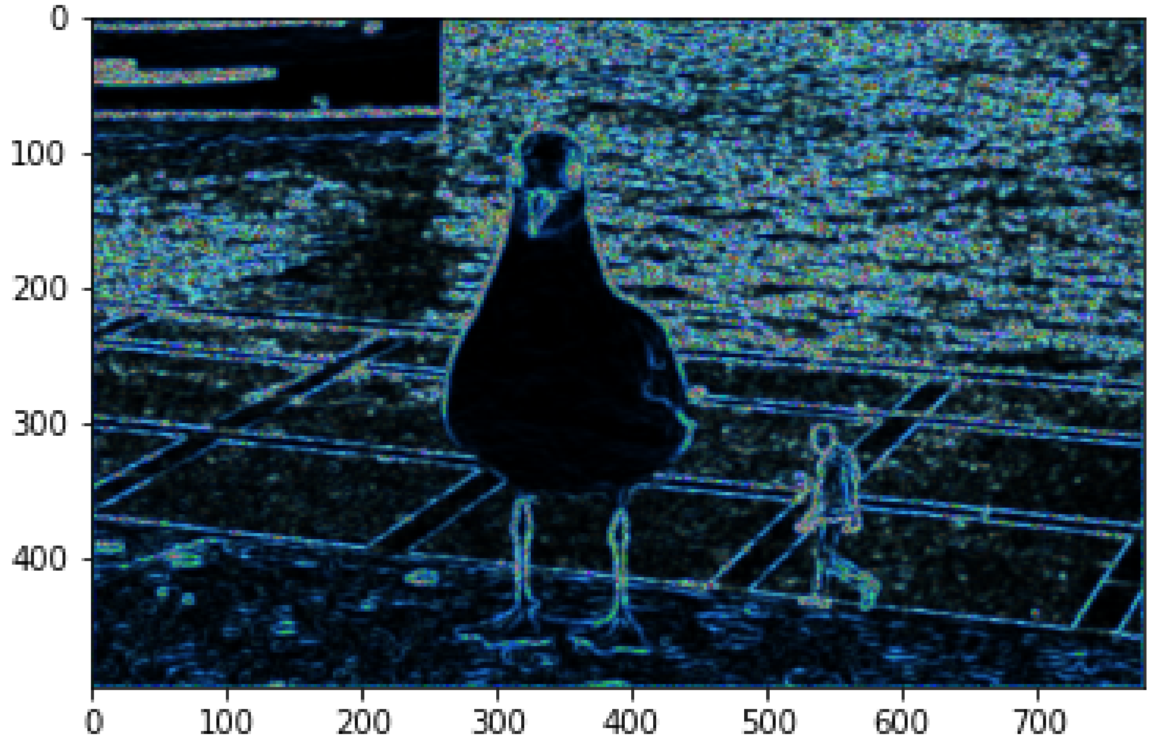


Figure 21: Texture image

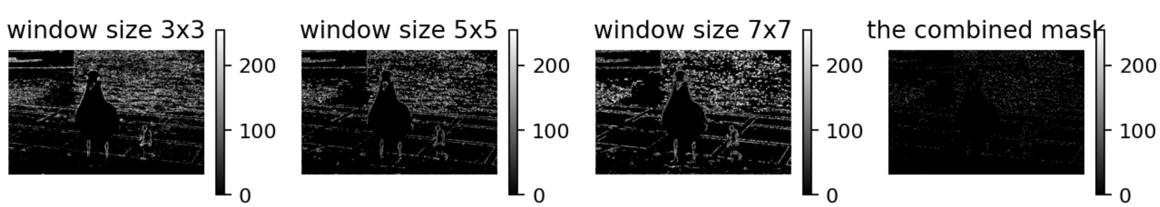


Figure 22: Generated masks of different window sizes

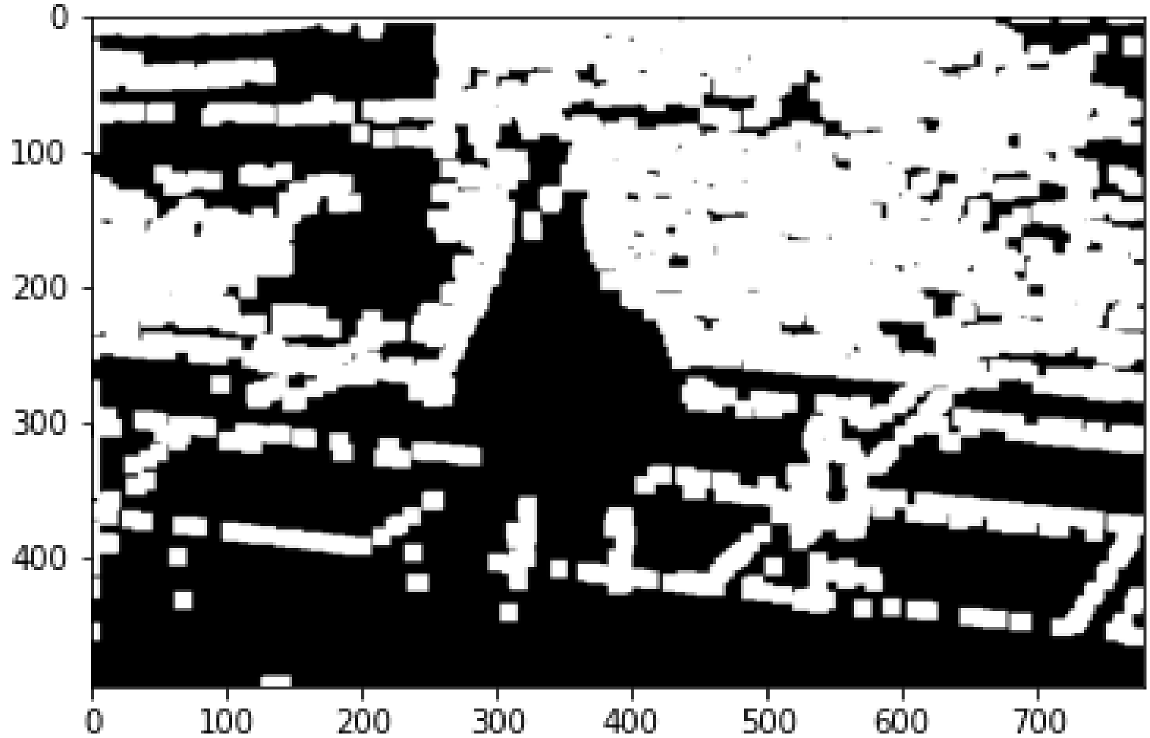


Figure 23: Denoised mask

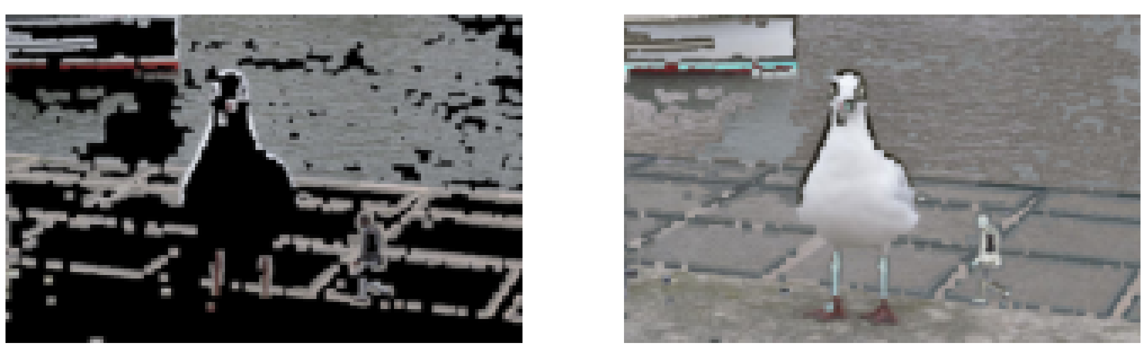


Figure 24: Foreground (left) and background (right) images

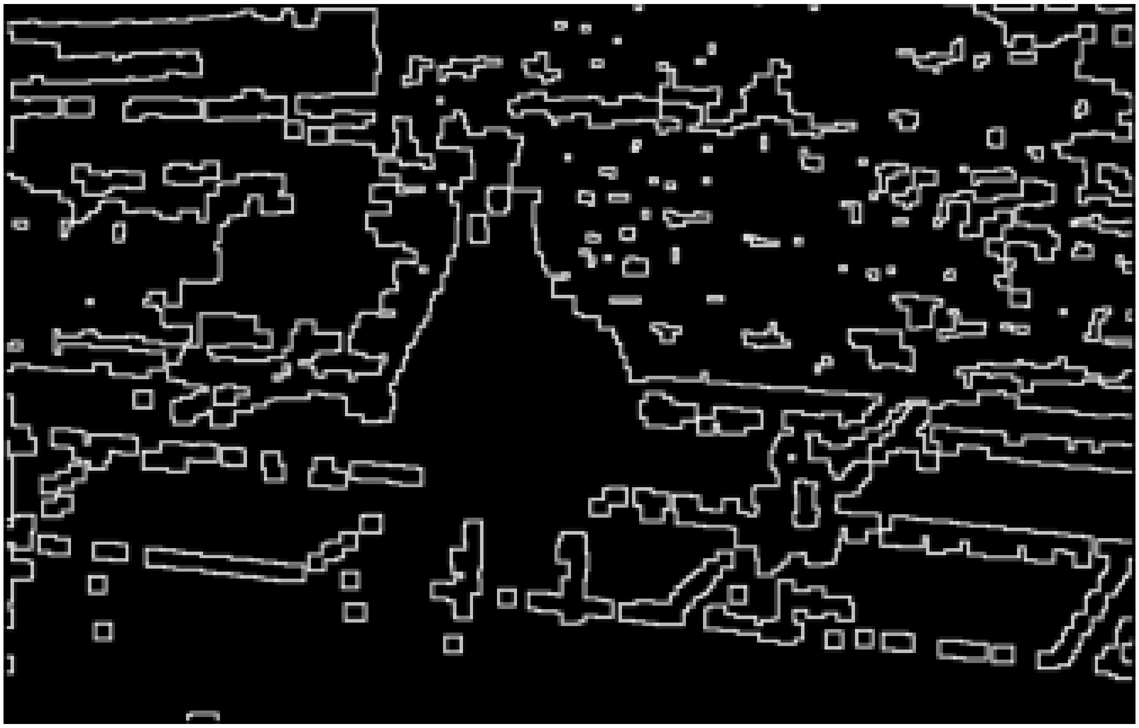


Figure 25: Extracted contour based on texture features

4 Observations

- The RGB-based method works better in extracting smooth contours, while the texture-based method contains more features in the foreground image.
- The performance of two methods are related to the characteristics of original image. While choosing the approach for image segmentation, we should choose the algorithm based on the features of original image.
- Both methods are sensitive to the number of iterations of Otsu algorithm.
- Dilation adds pixels to the object boundaries in an image, while erosion removes pixels on object boundaries. It's important to choose appropriate kernel sizes and iteration numbers to avoid over dilation or over erosion.

5 Source Code

5.1 utils.py

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from skimage import io


def compute_thres(img_mono):
    """
    Compute the threshold that maximizes the between class variance
    """
    img = img_mono.flatten()
    hist, bin_edges = np.histogram(img, bins=256, range=(0, 256), density=True)
    plt.hist(hist, bins=256)
    plt.show()
    plt.clf()
    pr_hist = hist * np.arange(256)
    mean_total = np.sum(pr_hist)
    optimum = -1e3
    for i in range(256):
        omega = np.sum(hist[:i+1])
        mu = np.sum(pr_hist[:i+1])
        if ((mean_total*omega - mu) **2 / (omega * (1-omega)) > optimum) and (omega
    !=0) and (omega!=1):
            optimum = (mean_total*omega - mu) **2 / (omega * (1-omega))
            thres = i

    return thres


def otsu_mask(img_mono, num_iter):
    """
    Otsu Algorithm
    """
    mask = np.ones(img_mono.shape, dtype=np.uint8)
    for i in range(num_iter):
        thres = compute_thres(img_mono[np.nonzero(mask)])
        # print(thres)
        ret, mask = cv2.threshold(img_mono, thres, 255, cv2.THRESH_BINARY)

    return mask


def otsu_rgb(img, num_iter):
    """
    Get mask using Otsu algorithm using RGB values
    """
    b, g, r = cv2.split(img)
    b_iter, g_iter, r_iter = num_iter[0], num_iter[1], num_iter[2]
    mask_b = otsu_mask(b, b_iter)
    mask_g = otsu_mask(g, g_iter)
    mask_r = otsu_mask(r, r_iter)
    mask = cv2.bitwise_and(cv2.bitwise_and(mask_b, mask_g), mask_r)
    # Plot
    figure(num=None, figsize=(10, 1.5), dpi=160, facecolor='w', edgecolor='k')
    plt.subplot(141)
    plt.imshow(mask_b, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title('mask of B channel')
```

```python
    plt.subplot(142)
    plt.imshow(mask_g, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title('mask of G channel')
    plt.subplot(143)
    plt.imshow(mask_r, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title(r'mask of R channel')
    plt.subplot(144)
    plt.imshow(mask, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title(r'the combined mask')
    plt.show()
    plt.clf()
    return mask_b, mask_g, mask_r, mask


def plot_rgb(img):
    """
    Plot the original image and the RGB channels
    """
    b, g, r = cv2.split(img)
    figure(num=None, figsize=(10, 1.5), dpi=160, facecolor='w', edgecolor='k')
    plt.subplot(141)
    plt.imshow(img, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title('original image')
    plt.subplot(142)
    plt.imshow(b, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title('B channel')
    plt.subplot(143)
    plt.imshow(g, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title(r'G channel')
    plt.subplot(144)
    plt.imshow(r, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title(r'R channel')
    plt.show()
    plt.clf()


def refine_mask(mask, kernel_size, num_iter, method):
    """
    Denoise mask
    :return:
    """
    kernel = np.ones((kernel_size, kernel_size), dtype=np.uint8)
    if method == 'dilation' or 'dilate':
        output = cv2.dilate(mask, kernel, iterations=num_iter)
    elif method == 'erosion' or 'erose':
        output = cv2.erode(mask, kernel, iterations=num_iter)
    # plt.subplot(121)
    # plt.imshow(mask, cmap='gray')
    # plt.axis('off')
    # plt.title('before')
```

```
    # plt.subplot(122)
    # plt.imshow(output, cmap='gray')
    # plt.axis('off')
    # plt.title('after')
    # plt.show()
    # plt.clf()
    return output


def get_contour(mask, window_size):
    """
    Obtain the contour
    """
    output = np.zeros(mask.shape, dtype=np.uint8)
    for i in range(window_size, mask.shape[0]-window_size):
        for j in range(window_size, mask.shape[1]-window_size):
            if np.min(mask[i - window_size:i+window_size+1, j-window_size:j+
    window_size+1])==0:
                output[i, j] = 255

    output[mask == 0] = 0
    plt.imshow(output, cmap='gray')
    plt.axis('off')
    plt.show()
    plt.clf()
    return output


def get_texture(img, win_size = [3, 5, 7], Niter):
    """
    Get mask from Otsu algorithm based on texture features
    """
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    mask = np.zeros(gray.shape, dtype=np.uint8)
    texture_img = np.zeros(img.shape, dtype=np.uint8)
    masks = np.zeros(img.shape, dtype=np.uint8)
    for n in range(len(win_size)):
        N = win_size[n]
        temp = np.zeros((mask.shape[0] + 2 * int((N - 1) / 2), mask.shape[1] + 2 * int
    ((N - 1) / 2)), dtype=np.uint8)
        temp[int((N - 1) / 2):temp.shape[0] - int((N - 1) / 2), int((N - 1) / 2):temp.
    shape[1] - int((N - 1) / 2)] = gray
        for i in range(mask.shape[0]):
            for j in range(mask.shape[1]):
                x = i + int((N - 1) / 2)
                y = j + int((N - 1) / 2)
                window = temp[x - int((N - 1) / 2): x + int((N - 1) / 2), y - int((N -
     1) / 2): y + int((N - 1) / 2)]
                mask[i, j] = np.var(window)
        texture_img[:, :, n] = mask
        masks[:, :, n] = otsu_mask(mask, Niter[n])
    mask = cv2.bitwise_and(cv2.bitwise_and(masks[:, :, 0], masks[:, :, 1]), masks[:,
    :, 2])
    figure(num=None, figsize=(10, 1.5), dpi=160, facecolor='w', edgecolor='k')
    plt.subplot(141)
    plt.imshow(masks[:, :, 0], cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title('window size 3x3')
    plt.subplot(142)
    plt.imshow(masks[:, :, 1], cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title('window size 5x5')
```

```
    plt.subplot(143)
    plt.imshow(masks[:, :, 2], cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title(r'window size 7x7')
    plt.subplot(144)
    plt.imshow(mask, cmap='gray')
    plt.colorbar()
    plt.axis('off')
    plt.title(r'the combined mask')
    plt.show()
    plt.clf()

    return mask, texture_img
```

utils.py

5.2 main.py

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
from skimage import io
from utils import *

# Read images
img1 = io.imread('/hw6_images/cat.jpg')
img2 = io.imread('/hw6_images/Red-Fox_.jpg')
img3 = io.imread('/hw6_images/pigeon.jpeg')
# ================================================================================

# img1
img = img1
plot_rgb(img)
iters = [1, 1, 1]
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# b, g, r = cv2.split(img)
# img = cv2.merge((b,g,r))
# RGB_based
mask_b, mask_g, mask_r, final_mask = otsu_rgb(img, num_iter=iters)
final_mask = refine_mask(final_mask, kernel_size=3, num_iter=1, method='dilation')
final_mask = refine_mask(final_mask, kernel_size=3, num_iter=1, method='erosion')

foreground_img = cv2.bitwise_and(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
background_img = cv2.bitwise_xor(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
# plt.subplot(121)
# plt.imshow(foreground_img, cmap='gray')
# plt.axis('off')
# plt.subplot(122)
# plt.imshow(background_img, cmap='gray')
# plt.axis('off')
# plt.show()
# plt.clf()

contour_img = get_contour(final_mask, window_size=1)
plt.imshow(contour_img, cmap='gray')
plt.axis('off')
plt.show()
plt.clf()
# Texture-based
Niter = [1, 1, 1]
```

```
mask, texture_img = get_texture(img, win_size=[3, 5, 7], Niter=Niter)
plt.imshow(texture_img, cmap='gray')
plt.show()
plt.clf()

mask = refine_mask(mask, kernel_size=7, num_iter=1, method='dilation')
mask = refine_mask(mask, kernel_size=9, num_iter=1, method='erosion')

foreground_img = cv2.bitwise_and(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
background_img = cv2.bitwise_xor(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
# plt.subplot(121)
# plt.imshow(foreground_img, cmap='gray')
# plt.axis('off')
# plt.subplot(122)
# plt.imshow(background_img, cmap='gray')
# plt.axis('off')
# plt.show()
# plt.clf()

contour_img = get_contour(mask, window_size=1)
plt.imshow(contour_img, cmap='gray')
plt.axis('off')
plt.show()
plt.clf()


#
    ==========================================================================================

# img2
img = img2
plot_rgb(img)
iters = [2, 2, 1]
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# b, g, r = cv2.split(img)
# img = cv2.merge((b,g,r))
# RGB_based
mask_b, mask_g, mask_r, final_mask = otsu_rgb(img, num_iter=iters)
final_mask = refine_mask(final_mask, kernel_size=5, num_iter=1, method='dilation')
final_mask = refine_mask(final_mask, kernel_size=3, num_iter=1, method='erosion')

foreground_img = cv2.bitwise_and(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
background_img = cv2.bitwise_xor(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
# plt.subplot(121)
# plt.imshow(foreground_img, cmap='gray')
# plt.axis('off')
# plt.subplot(122)
# plt.imshow(background_img, cmap='gray')
# plt.axis('off')
# plt.show()
# plt.clf()

contour_img = get_contour(final_mask, window_size=1)
plt.imshow(contour_img, cmap='gray')
plt.axis('off')
plt.show()
plt.clf()
# Texture-based
Niter = [1, 1, 1]
mask, texture_img = get_texture(img, win_size=[3, 5, 7], Niter=Niter)
plt.imshow(texture_img, cmap='gray')
plt.show()
plt.clf()
```

```
mask = refine_mask(mask, kernel_size=7, num_iter=1, method='dilation')
mask = refine_mask(mask, kernel_size=3, num_iter=1, method='erosion')

foreground_img = cv2.bitwise_and(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
background_img = cv2.bitwise_xor(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
# plt.subplot(121)
# plt.imshow(foreground_img, cmap='gray')
# plt.axis('off')
# plt.subplot(122)
# plt.imshow(background_img, cmap='gray')
# plt.axis('off')
# plt.show()
# plt.clf()

contour_img = get_contour(mask, window_size=1)
plt.imshow(contour_img, cmap='gray')
plt.axis('off')
plt.show()
plt.clf()


#
    ==========================================================================================

# img3
img = img3
plot_rgb(img)
iters = [2, 2, 2]
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# b, g, r = cv2.split(img)
# img = cv2.merge((b,g,r))
# RGB_based
mask_b, mask_g, mask_r, final_mask = otsu_rgb(img, num_iter=iters)
final_mask = refine_mask(final_mask, kernel_size=3, num_iter=2, method='dilation')
final_mask = refine_mask(final_mask, kernel_size=3, num_iter=1, method='erosion')

foreground_img = cv2.bitwise_and(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
background_img = cv2.bitwise_xor(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
# plt.subplot(121)
# plt.imshow(foreground_img, cmap='gray')
# plt.axis('off')
# plt.subplot(122)
# plt.imshow(background_img, cmap='gray')
# plt.axis('off')
# plt.show()
# plt.clf()

contour_img = get_contour(final_mask, window_size=1)
plt.imshow(contour_img, cmap='gray')
plt.axis('off')
plt.show()
plt.clf()
# Texture-based
Niter = [1, 2, 2]
mask, texture_img = get_texture(img, win_size=[3, 5, 7], Niter=Niter)
plt.imshow(texture_img, cmap='gray')
plt.show()
plt.clf()

mask = refine_mask(mask, kernel_size=9, num_iter=1, method='dilation')
mask = refine_mask(mask, kernel_size=3, num_iter=2, method='erosion')

foreground_img = cv2.bitwise_and(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
background_img = cv2.bitwise_xor(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
```

```
# plt.subplot(121)
# plt.imshow(foreground_img, cmap='gray')
# plt.axis('off')
# plt.subplot(122)
# plt.imshow(background_img, cmap='gray')
# plt.axis('off')
# plt.show()
# plt.clf()

contour_img = get_contour(mask, window_size=1)
plt.imshow(contour_img, cmap='gray')
plt.axis('off')
plt.show()
plt.clf()
```

main.py