

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 .

$$p_i = \frac{n_i}{N} \quad (1)$$

Then the overall average gray level is given by

$$\mu_T = \sum_{i=1}^L i p_i \quad (2)$$

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 of the two classes are

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

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

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

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

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

4. Compute the between class variance.

The between class variance is calculated by

$$\delta_B^2 = \omega_0 \omega_1 (\mu_1 - \mu_0)^2 \quad (7)$$

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 to 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

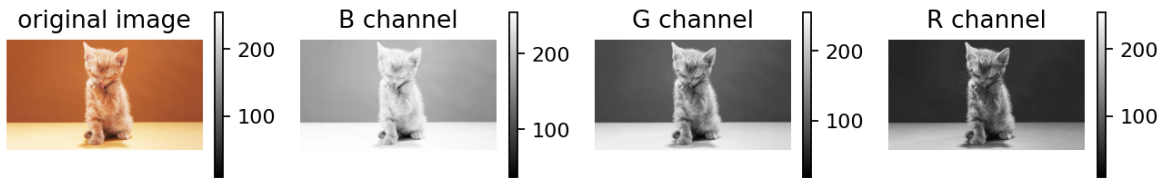


Figure 1: Original image and RGB channels

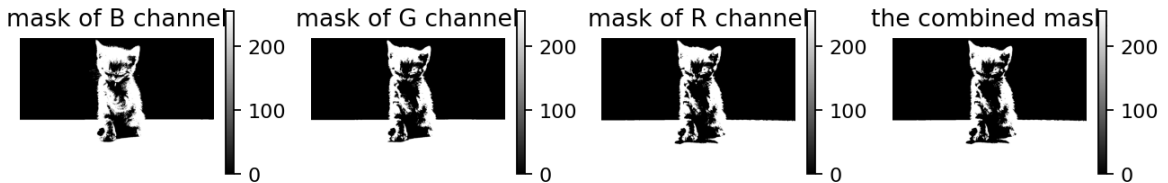


Figure 2: Generated masks of RGB channels

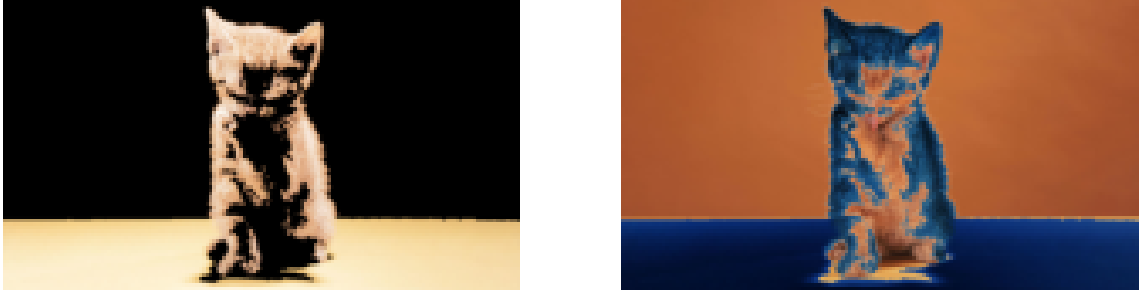


Figure 3: Foreground (left) and background (right) images

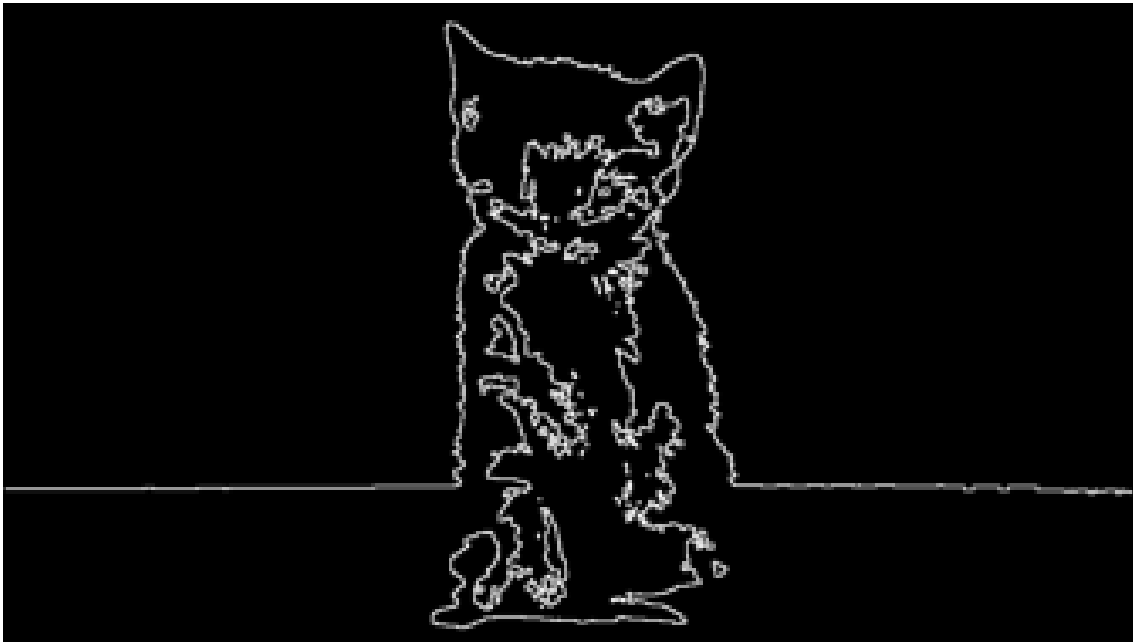


Figure 4: Extracted contour using RGB values

3.1.3 Texture-based Segmentation

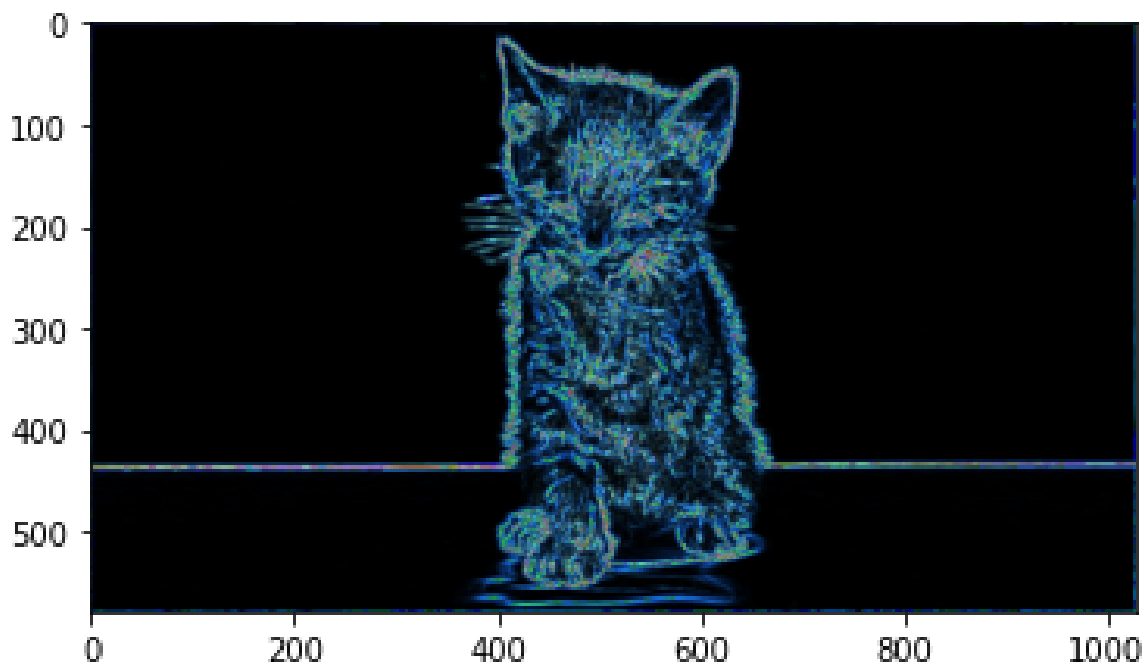


Figure 5: Texture image

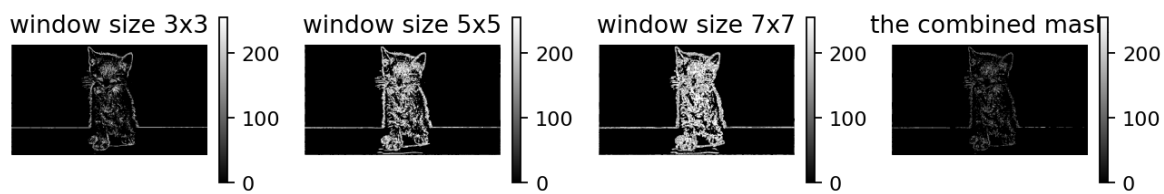


Figure 6: Generated masks of different window sizes

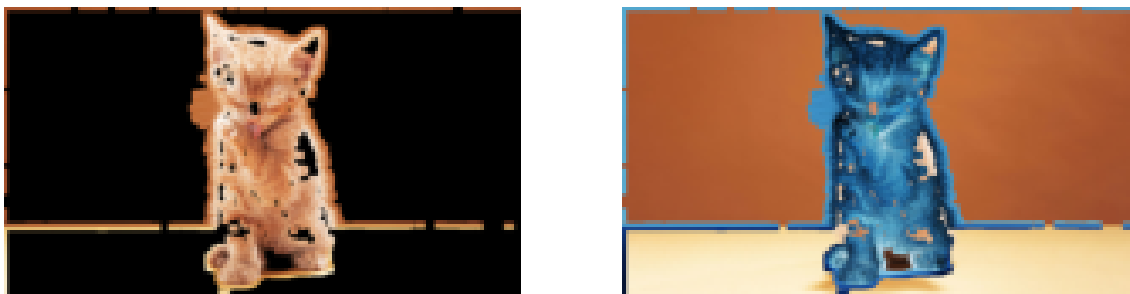


Figure 7: Foreground (left) and background (right) images



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

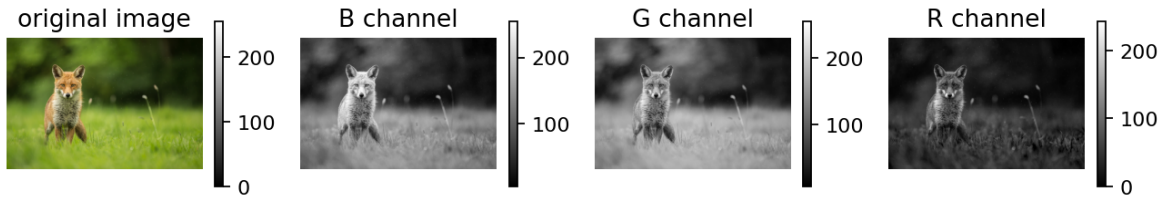


Figure 9: Original image and RGB channels

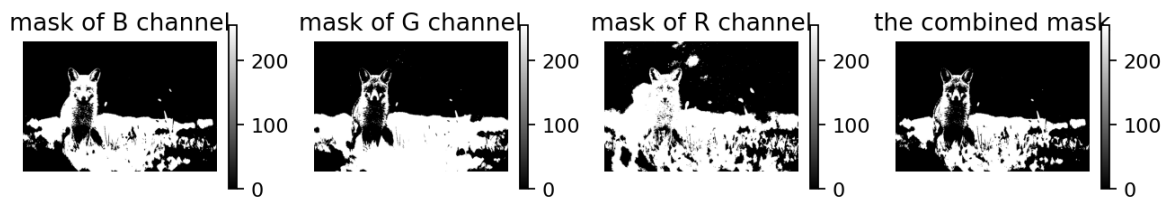


Figure 10: Generated masks of RGB channels

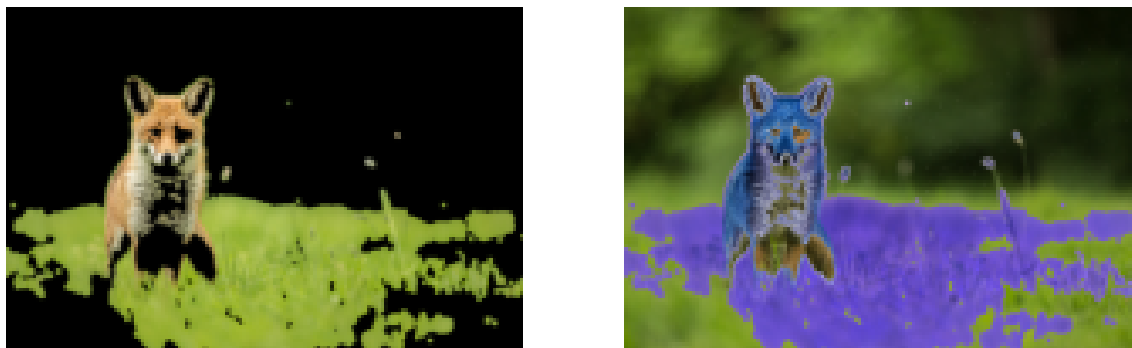


Figure 11: Foreground (left) and background (right) images



Figure 12: Extracted contour using RGB values

3.2.3 Texture-based Segmentation

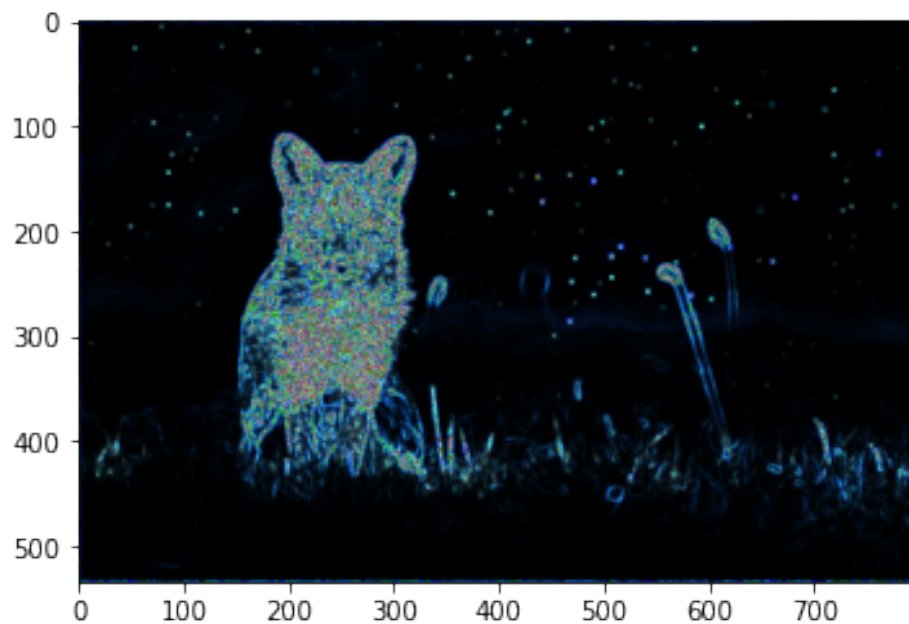


Figure 13: Texture image

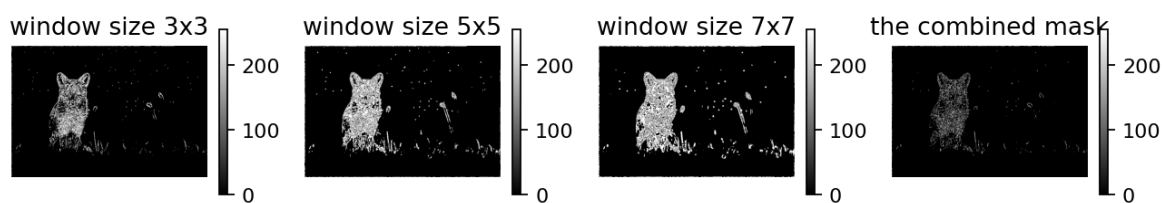


Figure 14: Generated masks of different window sizes

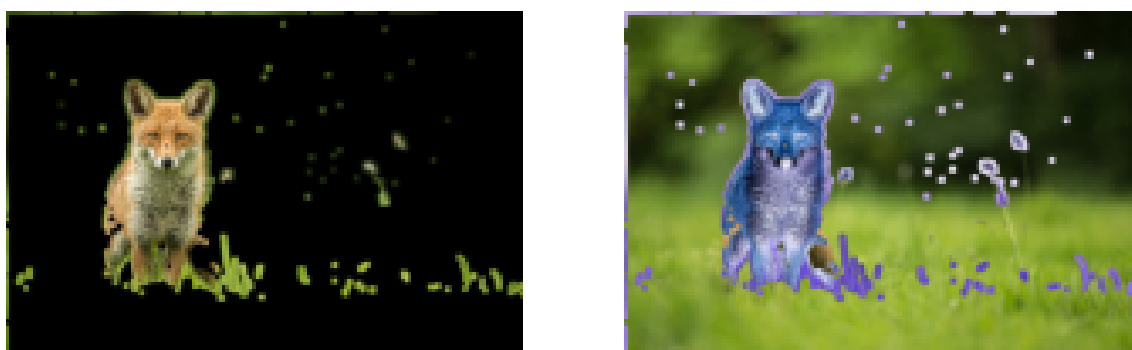


Figure 15: Foreground (left) and background (right) images

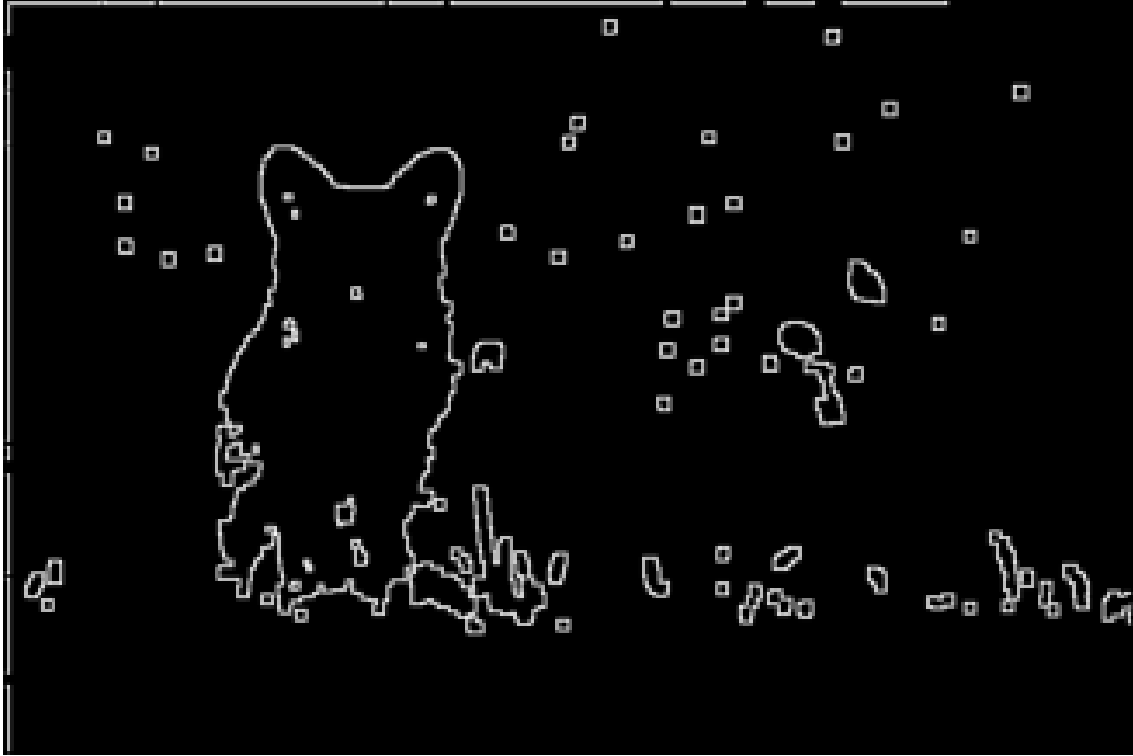


Figure 16: Extracted contour based on texture features

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

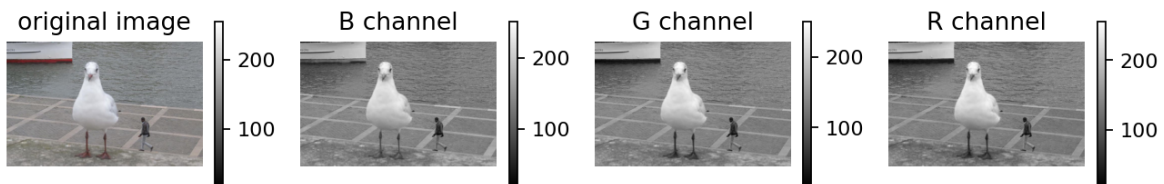


Figure 17: Original image and RGB channels

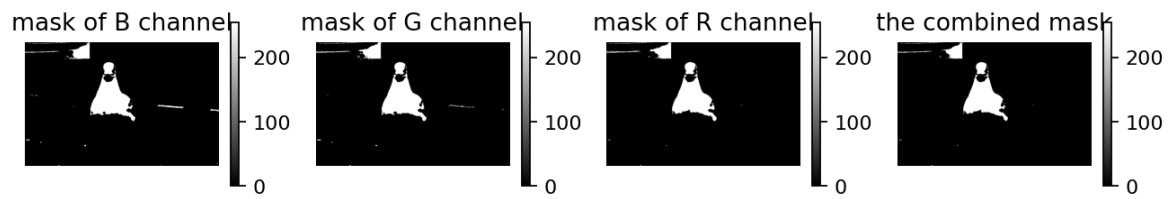


Figure 18: Generated masks of RGB channels



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



Figure 20: Extracted contour using RGB values

3.3.3 Texture-based Segmentation

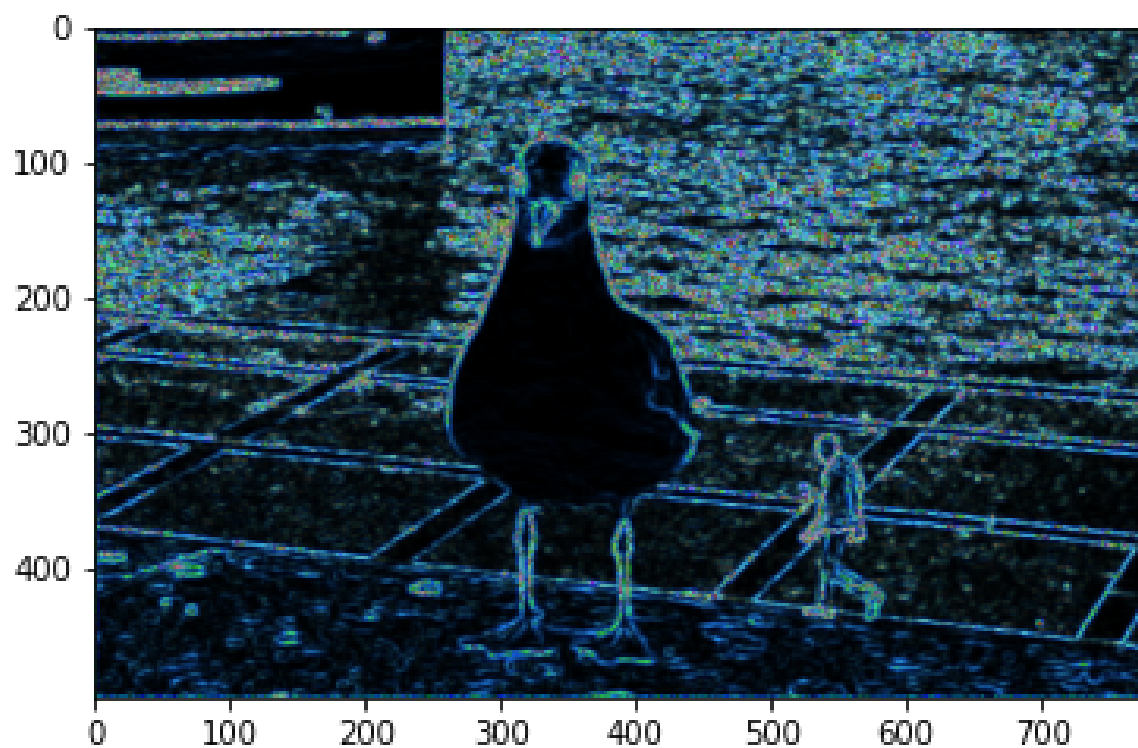


Figure 21: Texture image

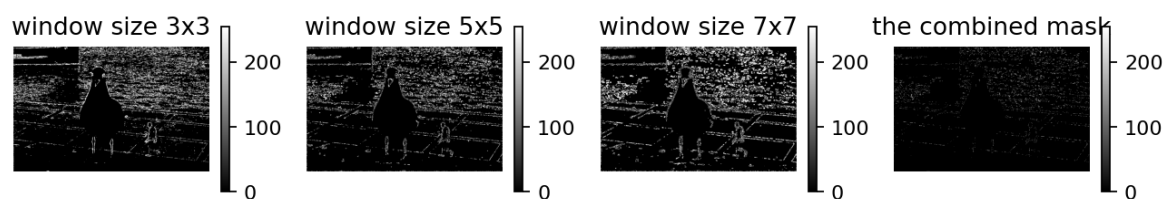


Figure 22: Generated masks of different window sizes

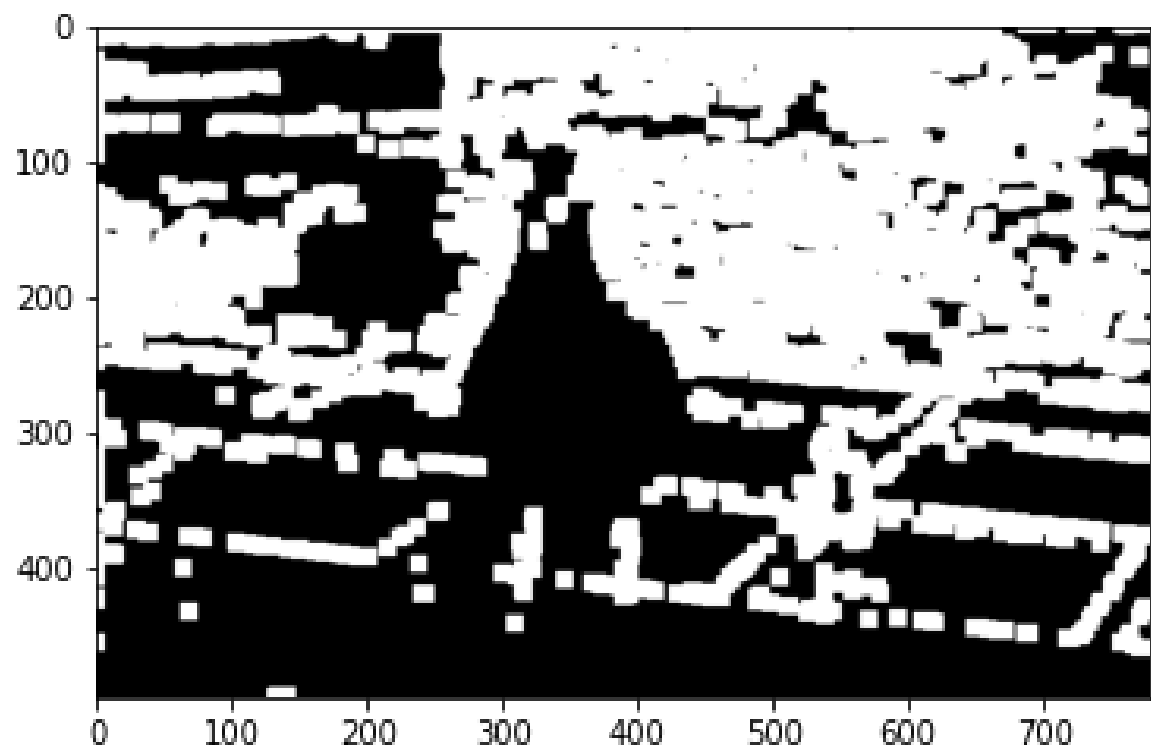


Figure 23: Denoised mask

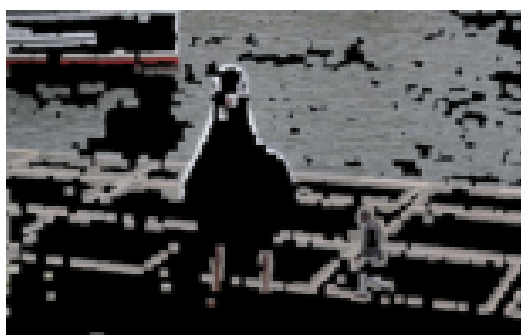


Figure 24: Foreground (left) and background (right) images



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

```
1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from matplotlib.pyplot import figure
1004 from skimage import io
1005
1006 def compute_thres(img_mono):
1007     """
1008     Compute the threshold that maximizes the between class variance
1009     """
1010     img = img_mono.flatten()
1011     hist, bin_edges = np.histogram(img, bins=256, range=(0, 256), density=True)
1012     plt.hist(hist, bins=256)
1013     plt.show()
1014     plt.clf()
1015     pr_hist = hist * np.arange(256)
1016     mean_total = np.sum(pr_hist)
1017     optimum = -1e3
1018     for i in range(256):
1019         omega = np.sum(hist[:i+1])
1020         mu = np.sum(pr_hist[:i+1])
1021         if ((mean_total*omega - mu) **2 / (omega * (1-omega))) > optimum and (omega
1022 !=0) and (omega!=1):
1023             optimum = (mean_total*omega - mu) **2 / (omega * (1-omega))
1024             thres = i
1025
1026     return thres
1027
1028 def otsu_mask(img_mono, num_iter):
1029     """
1030     Otsu Algorithm
1031     """
1032     mask = np.ones(img_mono.shape, dtype=np.uint8)
1033     for i in range(num_iter):
1034         thres = compute_thres(img_mono[np.nonzero(mask)])
1035         # print(thres)
1036         ret, mask = cv2.threshold(img_mono, thres, 255, cv2.THRESH_BINARY)
1037
1038     return mask
1039
1040 def otsu_rgb(img, num_iter):
1041     """
1042     Get mask using Otsu algorithm using RGB values
1043     """
1044     b, g, r = cv2.split(img)
1045     b_iter, g_iter, r_iter = num_iter[0], num_iter[1], num_iter[2]
1046     mask_b = otsu_mask(b, b_iter)
1047     mask_g = otsu_mask(g, g_iter)
1048     mask_r = otsu_mask(r, r_iter)
1049     mask = cv2.bitwise_and(cv2.bitwise_and(mask_b, mask_g), mask_r)
1050     # Plot
1051     figure(num=None, figsize=(10, 1.5), dpi=160, facecolor='w', edgecolor='k')
1052     plt.subplot(141)
1053     plt.imshow(mask_b, cmap='gray')
1054     plt.colorbar()
1055     plt.axis('off')
1056     plt.title('mask of B channel')
```

```

1060 plt.subplot(142)
1061 plt.imshow(mask_g, cmap='gray')
1062 plt.colorbar()
1063 plt.axis('off')
1064 plt.title('mask of G channel')
1065 plt.subplot(143)
1066 plt.imshow(mask_r, cmap='gray')
1067 plt.colorbar()
1068 plt.axis('off')
1069 plt.title('mask of R channel')
1070 plt.subplot(144)
1071 plt.imshow(mask, cmap='gray')
1072 plt.colorbar()
1073 plt.axis('off')
1074 plt.title('the combined mask')
1075 plt.show()
1076 plt.clf()
1077 return mask_b, mask_g, mask_r, mask
1078
1079 def plot_rgb(img):
1080     """
1081     Plot the original image and the RGB channels
1082     """
1083     b, g, r = cv2.split(img)
1084     figure(num=None, figsize=(10, 1.5), dpi=160, facecolor='w', edgecolor='k')
1085     plt.subplot(141)
1086     plt.imshow(img, cmap='gray')
1087     plt.colorbar()
1088     plt.axis('off')
1089     plt.title('original image')
1090     plt.subplot(142)
1091     plt.imshow(b, cmap='gray')
1092     plt.colorbar()
1093     plt.axis('off')
1094     plt.title('B channel')
1095     plt.subplot(143)
1096     plt.imshow(g, cmap='gray')
1097     plt.colorbar()
1098     plt.axis('off')
1099     plt.title('G channel')
1100     plt.subplot(144)
1101     plt.imshow(r, cmap='gray')
1102     plt.colorbar()
1103     plt.axis('off')
1104     plt.title('R channel')
1105     plt.show()
1106     plt.clf()
1107
1108 def refine_mask(mask, kernel_size, num_iter, method):
1109     """
1110     Denoise mask
1111     :return:
1112     """
1113     kernel = np.ones((kernel_size, kernel_size), dtype=np.uint8)
1114     if method == 'dilation' or 'dilate':
1115         output = cv2.dilate(mask, kernel, iterations=num_iter)
1116     elif method == 'erosion' or 'erose':
1117         output = cv2.erode(mask, kernel, iterations=num_iter)
1118     # plt.subplot(121)
1119     # plt.imshow(mask, cmap='gray')
1120     # plt.axis('off')
1121     # plt.title('before')

```



```

1124 # plt.subplot(122)
1125 # plt.imshow(output, cmap='gray')
1126 # plt.axis('off')
1127 # plt.title('after')
1128 # plt.show()
1129 # plt.clf()
1130 return output

1131
1132 def get_contour(mask, window_size):
1133     """
1134     Obtain the contour
1135     """
1136     output = np.zeros(mask.shape, dtype=np.uint8)
1137     for i in range(window_size, mask.shape[0]-window_size):
1138         for j in range(window_size, mask.shape[1]-window_size):
1139             if np.min(mask[i - window_size:i+window_size+1, j-window_size:j+
1140 window_size+1])==0:
1141                 output[i, j] = 255
1142
1143     output[mask == 0] = 0
1144     plt.imshow(output, cmap='gray')
1145     plt.axis('off')
1146     plt.show()
1147     plt.clf()
1148     return output
1149
1150 def get_texture(img, win_size = [3, 5, 7], Niter):
1151     """
1152     Get mask from Otsu algorithm based on texture features
1153     """
1154     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
1155     mask = np.zeros(gray.shape, dtype=np.uint8)
1156     texture_img = np.zeros(img.shape, dtype=np.uint8)
1157     masks = np.zeros(img.shape, dtype=np.uint8)
1158     for n in range(len(win_size)):
1159         N = win_size[n]
1160         temp = np.zeros((mask.shape[0] + 2 * int((N - 1) / 2), mask.shape[1] + 2 * int
1161 ((N - 1) / 2)), dtype=np.uint8)
1162         temp[int((N - 1) / 2):temp.shape[0] - int((N - 1) / 2), int((N - 1) / 2):temp.
1163 shape[1] - int((N - 1) / 2)] = gray
1164         for i in range(mask.shape[0]):
1165             for j in range(mask.shape[1]):
1166                 x = i + int((N - 1) / 2)
1167                 y = j + int((N - 1) / 2)
1168                 window = temp[x - int((N - 1) / 2): x + int((N - 1) / 2), y - int((N -
1169 1) / 2): y + int((N - 1) / 2)]
1170                 mask[i, j] = np.var(window)
1171                 texture_img[:, :, n] = mask
1172                 masks[:, :, n] = otsu_mask(mask, Niter[n])
1173     mask = cv2.bitwise_and(cv2.bitwise_and(masks[:, :, 0], masks[:, :, 1]), masks[:,
1174 :, 2])
1175     figure(num=None, figsize=(10, 1.5), dpi=160, facecolor='w', edgecolor='k')
1176     plt.subplot(141)
1177     plt.imshow(masks[:, :, 0], cmap='gray')
1178     plt.colorbar()
1179     plt.axis('off')
1180     plt.title('window size 3x3')
1181     plt.subplot(142)
1182     plt.imshow(masks[:, :, 1], cmap='gray')
1183     plt.colorbar()
1184     plt.axis('off')
1185     plt.title('window size 5x5')

```

```

1182     plt.subplot(143)
1183     plt.imshow(masks[:, :, 2], cmap='gray')
1184     plt.colorbar()
1185     plt.axis('off')
1186     plt.title(r'window size 7x7')
1187     plt.subplot(144)
1188     plt.imshow(mask, cmap='gray')
1189     plt.colorbar()
1190     plt.axis('off')
1191     plt.title(r'the combined mask')
1192     plt.show()
1193     plt.clf()
1194
1195     return mask, texture_img

```

utils.py

5.2 main.py

```

1000 import numpy as np
1001 import cv2
1002 import matplotlib.pyplot as plt
1003 from matplotlib.pyplot import figure
1004 from skimage import io
1005 from utils import *
1006
1007 # Read images
1008 img1 = io.imread('/hw6_images/cat.jpg')
1009 img2 = io.imread('/hw6_images/Red-Fox_.jpg')
1010 img3 = io.imread('/hw6_images/pigeon.jpeg')
1011 #
1012 # =====
1013
1014 # img1
1015 img = img1
1016 plot_rgb(img)
1017 iters = [1, 1, 1]
1018 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
1019 # b, g, r = cv2.split(img)
1020 # img = cv2.merge((b,g,r))
1021 # RGB-based
1022 mask_b, mask_g, mask_r, final_mask = otsu_rgb(img, num_iter=iters)
1023 final_mask = refine_mask(final_mask, kernel_size=3, num_iter=1, method='dilation')
1024 final_mask = refine_mask(final_mask, kernel_size=3, num_iter=1, method='erosion')
1025
1026 foreground_img = cv2.bitwise_and(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
1027 background_img = cv2.bitwise_xor(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
1028 # plt.subplot(121)
1029 # plt.imshow(foreground_img, cmap='gray')
1030 # plt.axis('off')
1031 # plt.subplot(122)
1032 # plt.imshow(background_img, cmap='gray')
1033 # plt.axis('off')
1034 # plt.show()
1035 # plt.clf()
1036
1037 contour_img = get_contour(final_mask, window_size=1)
1038 plt.imshow(contour_img, cmap='gray')
1039 plt.axis('off')
1040 plt.show()
1041 plt.clf()
1042 # Texture-based
1043 Niter = [1, 1, 1]

```

```

1042 mask, texture_img = get_texture(img, win_size=[3, 5, 7], Niter=Niter)
1043 plt.imshow(texture_img, cmap='gray')
1044 plt.show()
1045 plt.clf()
1046
1047 mask = refine_mask(mask, kernel_size=7, num_iter=1, method='dilation')
1048 mask = refine_mask(mask, kernel_size=9, num_iter=1, method='erosion')
1049
1050 foreground_img = cv2.bitwise_and(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
1051 background_img = cv2.bitwise_xor(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
1052 # plt.subplot(121)
1053 # plt.imshow(foreground_img, cmap='gray')
1054 # plt.axis('off')
1055 # plt.subplot(122)
1056 # plt.imshow(background_img, cmap='gray')
1057 # plt.axis('off')
1058 # plt.show()
1059 # plt.clf()
1060
1061 contour_img = get_contour(mask, window_size=1)
1062 plt.imshow(contour_img, cmap='gray')
1063 plt.axis('off')
1064 plt.show()
1065 plt.clf()
1066
1067 #
1068 # =====
1069
1070 # img2
1071 img = img2
1072 plot_rgb(img)
1073 # b, g, r = cv2.split(img)
1074 # img = cv2.merge((b,g,r))
1075 # RGB-based
1076 mask_b, mask_g, mask_r, final_mask = otsu_rgb(img, num_iter=iters)
1077 final_mask = refine_mask(final_mask, kernel_size=5, num_iter=1, method='dilation')
1078 final_mask = refine_mask(final_mask, kernel_size=3, num_iter=1, method='erosion')
1079
1080 foreground_img = cv2.bitwise_and(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
1081 background_img = cv2.bitwise_xor(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
1082 # plt.subplot(121)
1083 # plt.imshow(foreground_img, cmap='gray')
1084 # plt.axis('off')
1085 # plt.subplot(122)
1086 # plt.imshow(background_img, cmap='gray')
1087 # plt.axis('off')
1088 # plt.show()
1089 # plt.clf()
1090
1091 contour_img = get_contour(final_mask, window_size=1)
1092 plt.imshow(contour_img, cmap='gray')
1093 plt.axis('off')
1094 plt.show()
1095 plt.clf()
1096 # Texture-based
1097 Niter = [1, 1, 1]
1098 mask, texture_img = get_texture(img, win_size=[3, 5, 7], Niter=Niter)
1099 plt.imshow(texture_img, cmap='gray')
1100 plt.show()
1101 plt.clf()

```

```

1104 mask = refine_mask(mask, kernel_size=7, num_iter=1, method='dilation')
1105 mask = refine_mask(mask, kernel_size=3, num_iter=1, method='erosion')
1106
1107 foreground_img = cv2.bitwise_and(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
1108 background_img = cv2.bitwise_xor(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
1109 # plt.subplot(121)
1110 # plt.imshow(foreground_img, cmap='gray')
1111 # plt.axis('off')
1112 # plt.subplot(122)
1113 # plt.imshow(background_img, cmap='gray')
1114 # plt.axis('off')
1115 # plt.show()
1116 # plt.clf()
1117
1118 contour_img = get_contour(mask, window_size=1)
1119 plt.imshow(contour_img, cmap='gray')
1120 plt.axis('off')
1121 plt.show()
1122 plt.clf()
1123
1124 #
1125 # =====
1126 # img3
1127 img = img3
1128 plot_rgb(img)
1129 iters = [2, 2, 2]
1130 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
1131 # b, g, r = cv2.split(img)
1132 # img = cv2.merge((b,g,r))
1133 # RGB-based
1134 mask_b, mask_g, mask_r, final_mask = otsu_rgb(img, num_iter=iters)
1135 final_mask = refine_mask(final_mask, kernel_size=3, num_iter=2, method='dilation')
1136 final_mask = refine_mask(final_mask, kernel_size=3, num_iter=1, method='erosion')
1137
1138 foreground_img = cv2.bitwise_and(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
1139 background_img = cv2.bitwise_xor(img, cv2.cvtColor(final_mask, cv2.COLOR_GRAY2BGR))
1140 # plt.subplot(121)
1141 # plt.imshow(foreground_img, cmap='gray')
1142 # plt.axis('off')
1143 # plt.subplot(122)
1144 # plt.imshow(background_img, cmap='gray')
1145 # plt.axis('off')
1146 # plt.show()
1147 # plt.clf()
1148
1149 contour_img = get_contour(final_mask, window_size=1)
1150 plt.imshow(contour_img, cmap='gray')
1151 plt.axis('off')
1152 plt.show()
1153 plt.clf()
1154 # Texture-based
1155 Niter = [1, 2, 2]
1156 mask, texture_img = get_texture(img, win_size=[3, 5, 7], Niter=Niter)
1157 plt.imshow(texture_img, cmap='gray')
1158 plt.show()
1159 plt.clf()
1160
1161 mask = refine_mask(mask, kernel_size=9, num_iter=1, method='dilation')
1162 mask = refine_mask(mask, kernel_size=3, num_iter=2, method='erosion')
1163
1164 foreground_img = cv2.bitwise_and(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))
1165 background_img = cv2.bitwise_xor(img, cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR))

```

```

1166 # plt.subplot(121)
      # plt.imshow(foreground_img, cmap='gray')
1168 # plt.axis('off')
      # plt.subplot(122)
1170 # plt.imshow(background_img, cmap='gray')
      # plt.axis('off')
1172 # plt.show()
      # plt.clf()
1174
      contour_img = get_contour(mask, window_size=1)
1176 plt.imshow(contour_img, cmap='gray')
      plt.axis('off')
1178 plt.show()
      plt.clf()

```

main.py