ECE 661 – Homework 6

Ran Xu

xu943@purdue.edu

10/23/2018

1. Algorithms

1.1 Otsu's Algorithm

The purpose of Otsu's algorithm is to distinguish the foreground and background using the histogram of the feature in the image. The feature can be one of the R, G, and B channel or the texture features. The core design in Otsu's algorithm is to determine an optimal threshold k, where all pixels smaller than k are considered as background and those larger than k are considered foreground. However, this determination can be inversed depending on the relative intensity of background and foreground. The detailed steps are shown as follows,

Step 1: Pre-process the image and extract the feature channels. In this homework, I extract 6 channels in RGB-based segmentation, a.k.a, red channel (R), green channel (G), blue channel (B), 255 minus red channel (iR), 255 minus green channel (iG), and 255 minus blue channel (iB). I also extract 6 channels in texture-based segmentation, a.k.a, N=3 texture channel (N3), N=5 texture channel (N5), N=7 texture channel (N7), maximum value minus N=3 texture channel (iN3), maximum value minus N=5 texture channel (iN5), and maximum value minus N=7 texture channel (iN7).

Step 2: For each channel, setup 256 bins between the minimum value V_{min} and maximum value V_{max}, denoted $[V_{min} = V_0,\ V_1), [V_1, V_2), \dots, [V_{255}, V_{256} = V_{max}]$. I get the 256-long histogram vector $C = [C_0, C_1, \dots, C_{255}]$.

Step 3: Iteratively set k_{index} from 0 to 255 such that all pixels smaller than the threshold $k = V_{k_{index}+1}$ are background and those larger than the threshold are foreground. The frequency of background class is

$$\omega_0 = \sum_{i=0}^{k_{index}} C_i \Big/ \sum_{i=0}^{255} C_i$$

The frequency of foreground class is

$$\omega_1 = \sum_{i=k_{index}+1}^{255} C_i \Big/ \sum_{i=0}^{255} C_i$$

The mean of background class is

$$\mu_0 = \sum_{i=0}^{k_{index}} (C_i \cdot i) \Big/ \sum_{i=0}^{k_{index}} C_i$$

The mean of foreground class is

$$\mu_1 = \sum_{i=k_{index}+1}^{255} (C_i \cdot i) \Big/ \sum_{i=k_{index}+1}^{255} C_i$$

The optimization problem can be formulated as

$$k^*_{index} = \underset{k_{index}}{\operatorname{argmax}}\, \sigma_b^2 = \underset{k_{index}}{\operatorname{argmax}}\, \omega_0\omega_1(\mu_0 - \mu_1)^2$$

Step 4: Use the optimal value in this iteration $V_{k^*_{index}+1}$ to determine background and foreground as described in step 2. If the result is not satisfying, I replace V_{min} with $V_{k^*_{index}+1}$ and redo Step 2 to 4.

Step 5: The final mask (1 representing foreground, 0 representing background) can be generated using either element-wise AND or element-wise OR of the masks in each channels.

1.2 Texture-based Features

The purpose of texture-based features is to provide better features than RGB in image segmentation. The hypothesis in this approach is that background and foreground are different in terms of neighborhood variance. The neighborhood is defined as a 3x3, 5x5, 7x7 surrounding pixels depending on the size of kernel.

To be more in detail, I first convert RGB image to grey-scale image, denoted GSI. The resulting texture-based image is denoted TBI. Then the texture-based image can be computed as,

$$TBI[i,j] = Variance(GSI[i-1:i+2, i-1:i+2]), for\ 3x3\ kernel$$

$$TBI[i,j] = Variance(GSI[i-2:i+3, i-2:i+3]), for\ 5x5\ kernel$$

$$TBI[i,j] = Variance(GSI[i-3:i+4, i-3:i+4]), for\ 7x7\ kernel$$

Note that the boundary pixels of width (1, 2, and 3 in 3x3, 5x5, and 8x8 kernel) in TBI are set to be 0 to avoid undefined situation.

1.3 Contour Extraction

The purpose of contour extraction is to extract the boundary between foreground (pixel value = 1) and background (pixel value = 0). I use 8-neighborhood rule – if the 8 neighborhoods of a pixel are all 1s or all 0s, this pixel is not a boundary point and verse versa.

2. Observations

Firstly, the most important hypothesis of Otsu's algorithm is that the foreground and background can be determined just based on a hard threshold on the features. This hypothesis has the severe limitation that it cannot handle noise because the noisy pixels may have the similar features with the foreground. The Otsu's algorithm can be enhanced by locality-based post-processing approaches like dilation, erosion and so on.

Secondly, looking at the resulting mask of segmentation algorithm in each iteration, the human must determine whether to use initial color channel or the inversed channel and in which iteration the resulting mask looks the best. This step cannot be automated and thus requiring huge labor cost and expertise. Looking at the given three images, there is no hard number of iterations that gives the best resulting mask.

Finally, the texture-based features do provide another type of feature for image segmentation and is useful for grey-scale images. However, as mentioned in the first point, the noisy points may have the similar local variance with the foreground points and can never be distinguished using Otsu's algorithm. Post-processing approaches are stilled needed to remove those noise.

3. Results
3.1 RGB-based Segmentation and Contour Extraction

The resulting mask is (R channel in 10th iteration) AND (iG channel in 4th iteration) AND (iB channel in 3rd iteration).

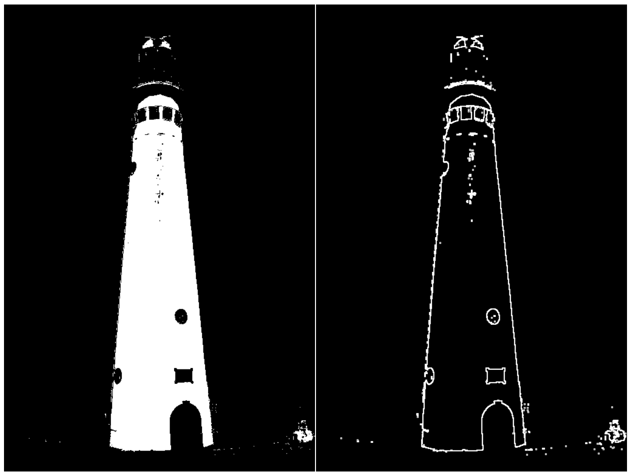

Figure 1: RGB-based segmentation (left) and contour extraction (right) of a lighthouse

The resulting mask is (iR channel in 2nd iteration) AND (iG channel in 2nd iteration) AND (iB channel in 2nd iteration).

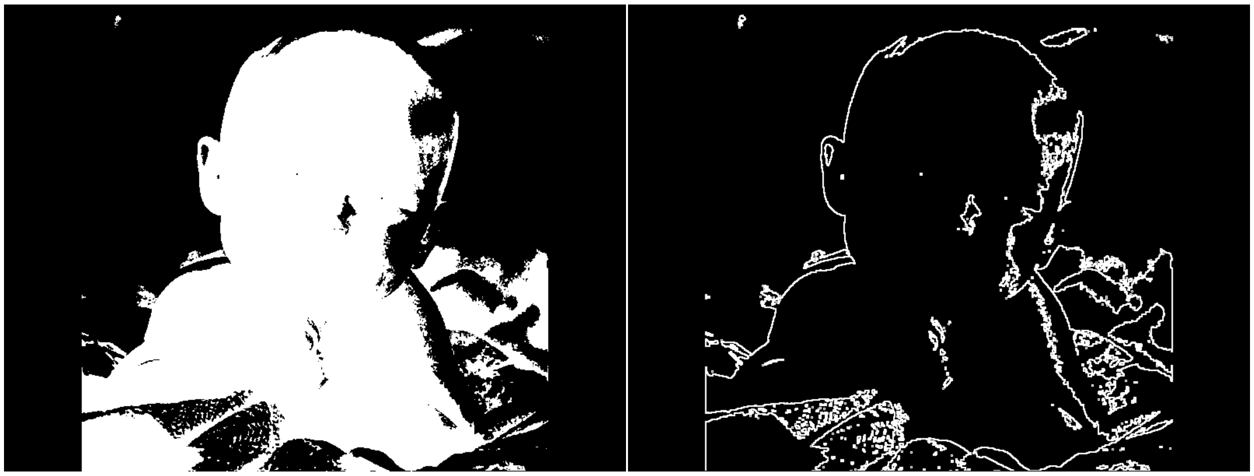

Figure 2: RGB-based segmentation (left) and contour extraction (right) of a baby

The resulting mask is iB channel in 10th iteration.

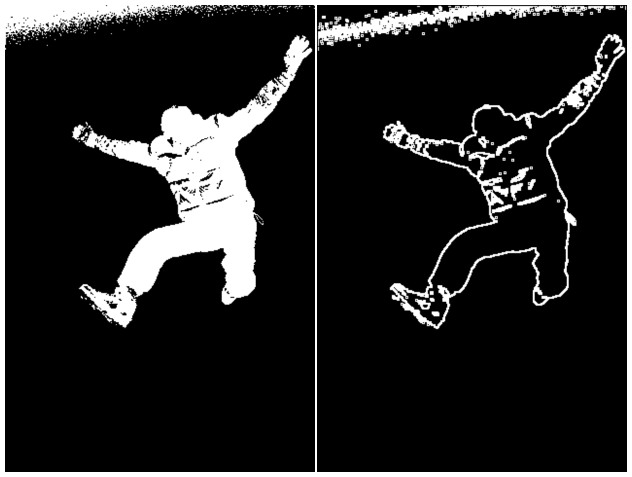
Figure 3: RGB-based segmentation (left) and contour extraction (right) of a jumping man

3.2 Texture-Based Segmentation and Contour Extraction

The resulting mask is (N3 channel in 1st iteration) OR (N5 channel in 3rd iteration) OR (N7 channel in 3rd iteration).

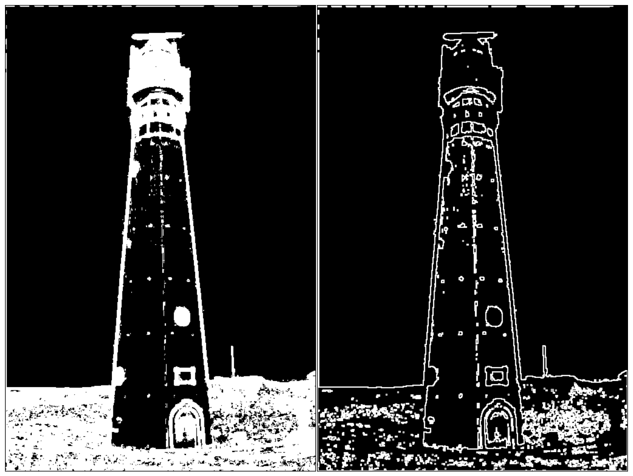
Figure 4: Texture-based segmentation (left) and contour extraction (right) of a lighthouse

The resulting mask is (N3 channel in 1st iteration) OR (N5 channel in 3rd iteration) OR (N7 channel in 4th iteration).

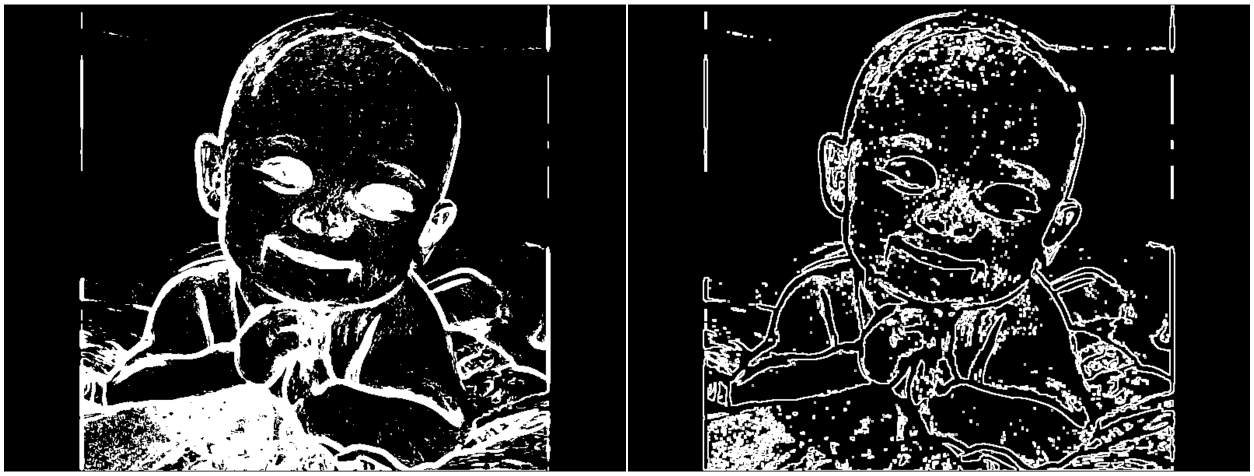

Figure 5: Texture-based segmentation (left) and contour extraction (right) of a baby

The resulting mask is (N3 channel in 1st iteration) OR (N5 channel in 2nd iteration) OR (N7 channel in 2nd iteration).

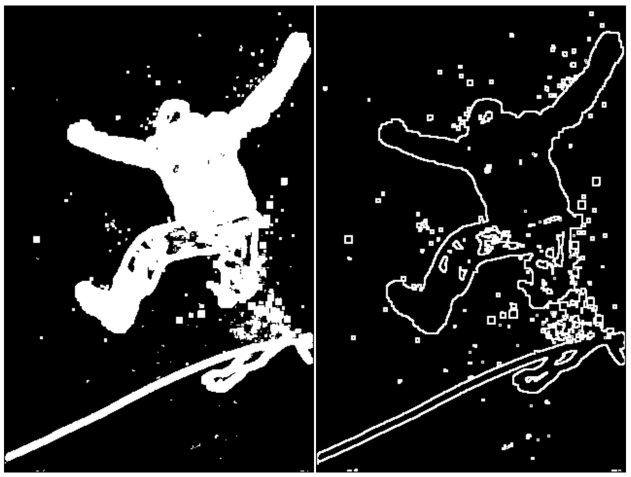

Figure 6: Texture-based segmentation (left) and contour extraction (right) of a jumping man

4. Source code

4.1 RGB-Based Segmentation and Contour Extraction

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
def Otsu(img, rounds = 1): # img (768H, 512W)
    Masks, k_opts, min_th = ([], [], 0)
    for rd in range(rounds):
        cnt, _ = np.histogram(img.flatten(), bins = np.arange(min_th-0.5, 255.6))

        freq, sigma_b_square = (np.zeros((256,)), np.zeros((256,)))
        freq[min_th:256] = cnt / np.sum(cnt)
        for k in range(min_th, 256): # k = min_th,min_th+1,...,255
            # Background class: [min_th,k], omega0, mu0
            # Foreground class: [k+1, 255], omega1, mu1
            omega0 = np.sum(freq[min_th:k+1])
            omega1 = np.sum(freq[k+1:256])
            mu0 = np.sum(freq[min_th:k+1] * np.arange(min_th, k+1))
            mu1 = np.sum(freq[k+1:256] * np.arange(k+1, 256))
            sigma_b_square[k] = omega0 * omega1 * (mu0-mu1) * (mu0-mu1)
        k_opt = np.argmax(sigma_b_square)
        min_th = k_opt
        Masks.append(((img > k_opt)*255).astype("uint8"))
        k_opts.append(k_opt)
        #plt.figure(); plt.subplot(211); plt.plot(freq);
        #plt.subplot(212); plt.plot(sigma_b_square); plt.xlabel("k_opt = %d" %k_opt)
    return Masks, k_opts
def Contour(img): # img (768H, 512W) binary image, 3x3 kernal
    img = img.astype("int")
    moment1 = np.zeros((img.shape[0]-2, img.shape[1]-2))
    for idx_h in range(3):
        for idx_w in range(3):
            if idx_h ==1 and idx_w == 1:
                continue
            moment1 = moment1 + img[idx_h:idx_h+img.shape[0]-2,
                                    idx_w:idx_w+img.shape[1]-2]
    ContourImg = np.zeros(img.shape)
    ContourImg[1:-1, 1:-1] = np.logical_and(moment1 > 0, moment1 < 8)
    return ContourImg
```

```python
ImgPath = "HW6Pics/lighthouse.jpg"
img = cv2.imread(ImgPath)    # (768H, 512W, 3) uint8 np ndarray
(B, G, R) = (img[:,:,0], img[:,:,1], img[:,:,2])
(iB, iG, iR) = (255-B, 255-G, 255-R)
plt.figure(figsize=(10,20))
for c_idx, channel in enumerate([B, iB, G, iG, R, iR]):
    Masks, k_opts= Otsu(channel, rounds = 10)
    for idx, Mask in enumerate(Masks):
        plt.subplot(10,6, idx*6+c_idx+1)
        plt.imshow(Mask, cmap='gist_gray') # 0 = black, 255 = white
        plt.xticks([])
        plt.yticks([])
plt.subplot(10,6,1); plt.title("B");
plt.subplot(10,6,2); plt.title("iB");
plt.subplot(10,6,3); plt.title("G");
plt.subplot(10,6,4); plt.title("iG");
plt.subplot(10,6,5); plt.title("R");
plt.subplot(10,6,6); plt.title("iR");
for x in range(1,11):
    plt.subplot(10,6,6*x-5); plt.ylabel("%d" %x, rotation=0);
plt.savefig("lighthouse_RGBdetail.png")
```

```python
# Choose iB @ 3rd, iG @ 4th, R@10th round
Masks_iB, _ = Otsu(iB, rounds = 10)
Masks_iG, _ = Otsu(iG, rounds = 10)
Masks_R, _ = Otsu(R, rounds = 10)
Mask = np.logical_and(Masks_iB[2]>0, Masks_iG[3]>0)
Mask = np.logical_and(Masks_R[3]>0, Mask)

plt.figure()
plt.imshow(Mask, cmap = "gist_gray")
cv2.imwrite("lighthouse_RGB.png", (Mask*255).astype("uint8"))

ContourImg = Contour(Mask)
plt.figure()
plt.imshow(ContourImg, cmap = "gist_gray")
cv2.imwrite("lighthouse_RGBContour.png", (ContourImg*255).astype("uint8"))
```

```python
ImgPath = "HW6Pics/baby.jpg"
img = cv2.imread(ImgPath)    # (768H, 512W, 3) uint8 np ndarray
(B, G, R) = (img[:,:,0], img[:,:,1], img[:,:,2])
(iB, iG, iR) = (255-B, 255-G, 255-R)
plt.figure(figsize=(10,6)) # (w, h)
for c_idx, channel in enumerate([B, iB, G, iG, R, iR]):
    Masks, k_opts= Otsu(channel, rounds = 3)
    for idx, Mask in enumerate(Masks):
        plt.subplot(3,6, idx*6+c_idx+1)
        plt.imshow(Mask, cmap='gist_gray') # 0 = black, 255 = white
        plt.xticks([])
        plt.yticks([])
plt.subplot(3,6,1); plt.title("B");
plt.subplot(3,6,2); plt.title("iB");
plt.subplot(3,6,3); plt.title("G");
plt.subplot(3,6,4); plt.title("iG");
plt.subplot(3,6,5); plt.title("R");
plt.subplot(3,6,6); plt.title("iR");
for x in range(1,3+1):
    plt.subplot(3,6,6*x-5); plt.ylabel("%d" %x, rotation=0);
plt.savefig("baby_RGBdetail.png")
```

```python
# Choose iB @ 2nd, iG @ 2nd, iR@ 2nd round
Masks_iB, _ = Otsu(iB, rounds = 3)
Masks_iG, _ = Otsu(iG, rounds = 3)
Masks_iR, _ = Otsu(iR, rounds = 3)
Mask = np.logical_and(Masks_iB[1]>0, Masks_iG[1]>0)
Mask = np.logical_and(Masks_iR[1]>0, Mask)
plt.figure()
plt.imshow(Mask, cmap = "gist_gray")
cv2.imwrite("baby_RGB.png", (Mask*255).astype("uint8"))

ContourImg = Contour(Mask)
plt.figure()
plt.imshow(ContourImg, cmap = "gist_gray")
cv2.imwrite("baby_RGBContour.png", (ContourImg*255).astype("uint8"))
```

```python
ImgPath = "HW6Pics/ski.jpg"
img = cv2.imread(ImgPath)    # (768H, 512W, 3) uint8 np ndarray
(B, G, R) = (img[:,:,0], img[:,:,1], img[:,:,2])
(iB, iG, iR) = (255-B, 255-G, 255-R)
plt.figure(figsize=(10,20))
for c_idx, channel in enumerate([B, iB, G, iG, R, iR]):
    Masks, k_opts= Otsu(channel, rounds = 10)
    for idx, Mask in enumerate(Masks):
        plt.subplot(10,6, idx*6+c_idx+1)
        plt.imshow(Mask, cmap='gist_gray') # 0 = black, 255 = white
        plt.xticks([])
        plt.yticks([])
plt.subplot(10,6,1); plt.title("B");
plt.subplot(10,6,2); plt.title("iB");
plt.subplot(10,6,3); plt.title("G");
plt.subplot(10,6,4); plt.title("iG");
plt.subplot(10,6,5); plt.title("R");
plt.subplot(10,6,6); plt.title("iR");
for x in range(1,11):
    plt.subplot(10,6,6*x-5); plt.ylabel("%d" %x, rotation=0);
plt.savefig("ski_RGBdetail.png")
```

```
# Choose iB @ 10th, iG @ 1st, iR@ 1st round
Masks_iB, _ = Otsu(iB, rounds = 10)
Masks_iG, _ = Otsu(iG, rounds = 10)
Masks_iR, _ = Otsu(iR, rounds = 10)
Mask = Masks_iB[9]>0
plt.figure()
plt.imshow(Mask, cmap = "gist_gray")
cv2.imwrite("ski_RGB.png", (Mask*255).astype("uint8"))

ContourImg = Contour(Mask)
plt.figure()
plt.imshow(ContourImg, cmap = "gist_gray")
cv2.imwrite("ski_RGBContour.png", (ContourImg*255).astype("uint8"))
```

4.2 Texture-Based Segmentation and Contour Extraction

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def Otsu(img, rounds = 1): # img (768H, 512W)
    Masks, th_opts, min_th, max_th = ([], [], np.min(img), np.max(img))

    for rd in range(rounds):
        # Construct 257 bin edges, 256 bin widths, 255 bin widths = max-min
        step = (max_th - min_th)/255
        Otsu_bins = np.arange(min_th-step/2, max_th+step/2+1e-3, step)

        cnt, _ = np.histogram(img.flatten(), bins = Otsu_bins)
        freq, sigma_b_square = (np.zeros((256,)), np.zeros((256,)))
        freq = cnt / np.sum(cnt)
        for k in range(0, 256): # k = 0,1,...,255
            # Background class: [0,k], omega0, mu0
            # Foreground class: [k+1, 255], omega1, mu1
            omega0 = np.sum(freq[0:k+1])
            omega1 = np.sum(freq[k+1:256])
            mu0 = np.sum(freq[0:k+1] * np.arange(0, k+1))
            mu1 = np.sum(freq[k+1:256] * np.arange(k+1, 256))
            sigma_b_square[k] = omega0 * omega1 * (mu0-mu1) * (mu0-mu1)
        k_opt = np.argmax(sigma_b_square)
        min_th = Otsu_bins[k_opt+1]
        Masks.append(((img > min_th)*255).astype("uint8"))
        th_opts.append(min_th)
        #plt.figure(); plt.subplot(211); plt.plot(freq);
        #plt.subplot(212); plt.plot(sigma_b_square); plt.xlabel("k_opt = %d" %k_opt)
    return Masks, th_opts
```

```
def TextOpt(img): # img (768H, 512W)
    # moment1 = sum(x)
    # moment2 = sum(x^2)
    # Var = moment2/(N**2) - (moment1/N/N)**2
    img = img.astype("float")
    Ns = [3,5,7]
    re = np.zeros((img.shape[0], img.shape[1], 3))
    for idx_ch, N in enumerate(Ns):
        Diff = int((N-1)/2)
        moment1 = np.zeros((img.shape[0]-N+1, img.shape[1]-N+1))
        moment2 = np.zeros((img.shape[0]-N+1, img.shape[1]-N+1))
        for idx_h in range(N):
            for idx_w in range(N):
                moment1 = moment1 + img[idx_h:idx_h+img.shape[0]-N+1,
                                        idx_w:idx_w+img.shape[1]-N+1]
                moment2 = moment2 + img[idx_h:idx_h+img.shape[0]-N+1,
                                        idx_w:idx_w+img.shape[1]-N+1]**2
        re[Diff:-Diff, Diff:-Diff, idx_ch] = moment2/N/N - (moment1/N/N)**2
    return re
def Contour(img): # img (768H, 512W) binary image, 3x3 kernal
    img = img.astype("int")
    moment1 = np.zeros((img.shape[0]-2, img.shape[1]-2))
    for idx_h in range(3):
        for idx_w in range(3):
            if idx_h ==1 and idx_w == 1:
                continue
            moment1 = moment1 + img[idx_h:idx_h+img.shape[0]-2,
                                    idx_w:idx_w+img.shape[1]-2]
    ContourImg = np.zeros(img.shape)
    ContourImg[1:-1, 1:-1] = np.logical_and(moment1 > 0, moment1 < 8)
    return ContourImg
```

```
ImgPath = "HW6Pics/lighthouse.jpg"
img = cv2.imread(ImgPath, cv2.IMREAD_GRAYSCALE)   # (768H, 512W) uint8 np ndarray
img = TextOpt(img)
(B, G, R) = (img[:,:,0], img[:,:,1], img[:,:,2])
(iB, iG, iR) = (np.max(img)-B, np.max(img)-G, np.max(img)-R)
plt.figure(figsize=(10,4 * 2))
for c_idx, channel in enumerate([B, iB, G, iG, R, iR]):
    Masks, _ = Otsu(channel, rounds = 4)
    for idx, Mask in enumerate(Masks):
        plt.subplot(4,6, idx*6+c_idx+1)
        plt.imshow(Mask, cmap='gist_gray') # 0 = black, 255 = white
        plt.xticks([])
        plt.yticks([])
plt.subplot(4,6,1); plt.title("B");
plt.subplot(4,6,2); plt.title("iB");
plt.subplot(4,6,3); plt.title("G");
plt.subplot(4,6,4); plt.title("iG");
plt.subplot(4,6,5); plt.title("R");
plt.subplot(4,6,6); plt.title("iR");
for x in range(1,4+1):
    plt.subplot(4,6,6*x-5); plt.ylabel("%d" %x, rotation=0);
plt.savefig("lighthouse_txtdetail.png")
```

```
# Choose B @ 1st, G @ 3rd, R@3rd round
Masks_B, _ = Otsu(B, rounds = 3)
Masks_G, _ = Otsu(G, rounds = 3)
Masks_R, _ = Otsu(R, rounds = 3)
Mask = np.logical_or(Masks_B[0]>0, Masks_G[2]>0)
Mask = np.logical_or(Masks_R[2]>0, Mask)
```

```
plt.figure()
plt.imshow(Mask, cmap = "gist_gray")
cv2.imwrite("lighthouse_text.png", (Mask*255).astype("uint8"))

ContourImg = Contour(Mask)
plt.figure()
plt.imshow(ContourImg, cmap = "gist_gray")
cv2.imwrite("lighthouse_textContour.png", (ContourImg*255).astype("uint8"))
```

```
ImgPath = "HW6Pics/baby.jpg"
img = cv2.imread(ImgPath, cv2.IMREAD_GRAYSCALE)    # (768H, 512W) uint8 np ndarray
img = TextOpt(img)
(B, G, R) = (img[:,:,0], img[:,:,1], img[:,:,2])
(iB, iG, iR) = (np.max(img)-B, np.max(img)-G, np.max(img)-R)
plt.figure(figsize=(10,5*2))
for c_idx, channel in enumerate([B, iB, G, iG, R, iR]):
    Masks, _ = Otsu(channel, rounds = 5)
    for idx, Mask in enumerate(Masks):
        plt.subplot(5,6, idx*6+c_idx+1)
        plt.imshow(Mask, cmap='gist_gray') # 0 = black, 255 = white
        plt.xticks([])
        plt.yticks([])
plt.subplot(5,6,1); plt.title("B");
plt.subplot(5,6,2); plt.title("iB");
plt.subplot(5,6,3); plt.title("G");
plt.subplot(5,6,4); plt.title("iG");
plt.subplot(5,6,5); plt.title("R");
plt.subplot(5,6,6); plt.title("iR");
for x in range(1,5+1):
    plt.subplot(5,6,6*x-5); plt.ylabel("%d" %x, rotation=0);
plt.savefig("baby_txtdetail.png")
```

```
# Choose B @ 1st, G @ 3rd, R@4th round
Masks_B, _ = Otsu(B, rounds = 4)
Masks_G, _ = Otsu(G, rounds = 4)
Masks_R, _ = Otsu(R, rounds = 4)
Mask = np.logical_or(Masks_B[0]>0, Masks_G[2]>0)
Mask = np.logical_or(Masks_R[3]>0, Mask)
plt.figure()
plt.imshow(Mask, cmap = "gist_gray")
cv2.imwrite("baby_text.png", (Mask*255).astype("uint8"))

ContourImg = Contour(Mask)
plt.figure()
plt.imshow(ContourImg, cmap = "gist_gray")
cv2.imwrite("baby_textContour.png", (ContourImg*255).astype("uint8"))
```

```
ImgPath = "HW6Pics/ski.jpg"
img = cv2.imread(ImgPath, cv2.IMREAD_GRAYSCALE)    # (768H, 512W) uint8 np ndarray
img = TextOpt(img)
(B, G, R) = (img[:,:,0], img[:,:,1], img[:,:,2])
(iB, iG, iR) = (np.max(img)-B, np.max(img)-G, np.max(img)-R)
plt.figure(figsize=(10,3*2))
for c_idx, channel in enumerate([B, iB, G, iG, R, iR]):
    Masks, _ = Otsu(channel, rounds = 3)
    for idx, Mask in enumerate(Masks):
        plt.subplot(3,6, idx*6+c_idx+1)
        plt.imshow(Mask, cmap='gist_gray') # 0 = black, 255 = white
        plt.xticks([])
        plt.yticks([])
```

```python
plt.subplot(3,6,1); plt.title("B");
plt.subplot(3,6,2); plt.title("iB");
plt.subplot(3,6,3); plt.title("G");
plt.subplot(3,6,4); plt.title("iG");
plt.subplot(3,6,5); plt.title("R");
plt.subplot(3,6,6); plt.title("iR");
for x in range(1,3+1):
    plt.subplot(3,6,6*x-5); plt.ylabel("%d" %x, rotation=0);
plt.savefig("ski_txtdetail.png")
```

```python
# Choose B @ 1st, G @ 2nd, R@2nd round
Masks_B, _ = Otsu(B, rounds = 2)
Masks_G, _ = Otsu(G, rounds = 2)
Masks_R, _ = Otsu(R, rounds = 2)
Mask = np.logical_or(Masks_B[0]>0, Masks_G[1]>0)
Mask = np.logical_or(Masks_R[1]>0, Mask)
plt.figure()
plt.imshow(Mask, cmap = "gist_gray")
cv2.imwrite("ski_text.png", (Mask*255).astype("uint8"))

ContourImg = Contour(Mask)
plt.figure()
plt.imshow(ContourImg, cmap = "gist_gray")
cv2.imwrite("ski_textContour.png", (ContourImg*255).astype("uint8"))
```