Image Processing Programming Assignment #2

310551067 吳岱容

- **Introduction**

  In assignment 2, we need to choose from the provided 3 task options to implement the methods from scratch. The three options are Canny edge detector, Hough transform, The SLIC superpixel algorithm. All of the above are related to segmentation or edge detection algorithms. In this assignment, I chose to implement Canny edge detector.

  Canny edge detector is a very famous algorithm to detect edges of an image, and it was proposed in 1986. This algorithm includes five steps: (1) Preprocess (2) Gradient magnitude (3) Non-maximum suppression (4) Double thresholding (5) Hysteresis.

- **Method**

  In this section I will briefly introduce how this algorithm works.

  - Preprocess

    The first step of this algorithm is to do the preprocess. We need to first turn the image into grayscale and do the blurring (in the implementation I do Gaussian blurring) in order to reduce the effect of noise.

  - Gradient Magnitude

    In this step, we need to calculate the magnitude of the gradient. We do this task by using the Sobel operator, and then we can get the gradient from different axes. Next, we need to calculate the magnitude at each pixel (L2 norm).

  - Non-maximum suppression

    Pixels that are close to the edge are usually featured with non-zero gradient magnitude, therefore, the task in this step is to find the pixel which has the maximum value compared to its neighbor, since it is very possible that it is a pixel on the edge.

  - Double Thresholding

    In double thresholding, we need to give two thresholds to decide the edge. There will be threshold 1 and threshold 2. If the value is larger that threshold 2, the pixel is definitely on edge (strong edge). Else if the value is between threshold 1 and 2, it will be decided in the next step (weak edge). If the value is smaller than threshold 1, it is not an edge.

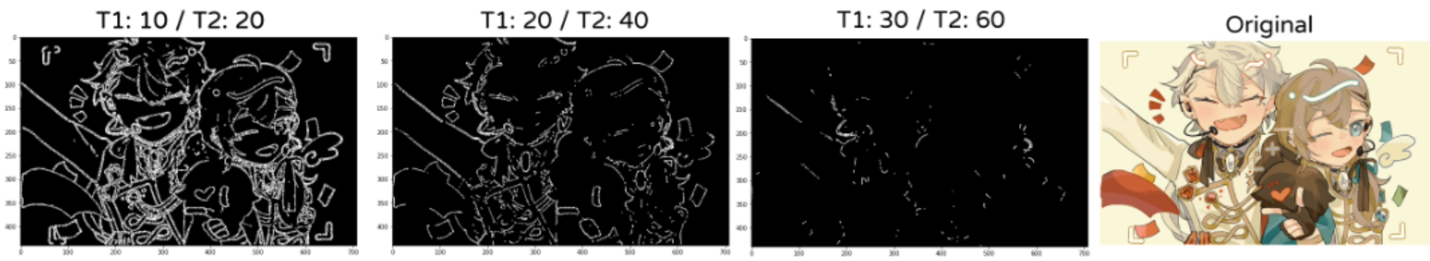
  - Hysteresis

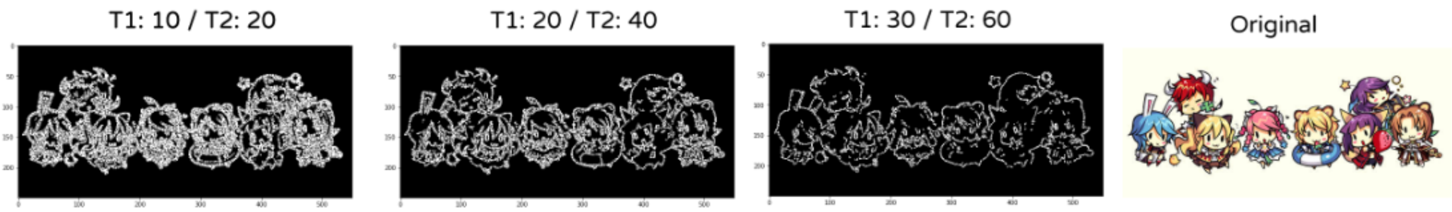
    In double thresholding, we have weak edges. In hysteresis, we need to decide if the weak edge is an edge or not. In order to do this, we need to test if weak edges can connect to strong edges.

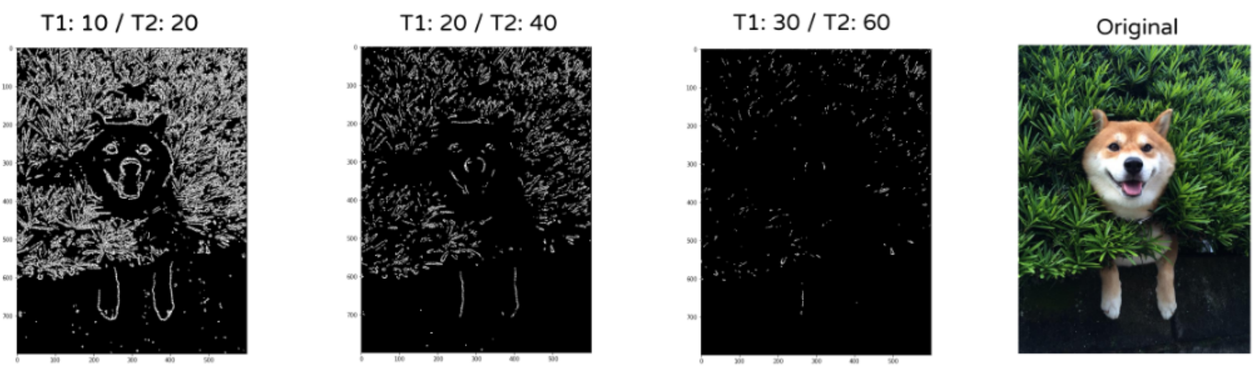
- **Experiment Results**

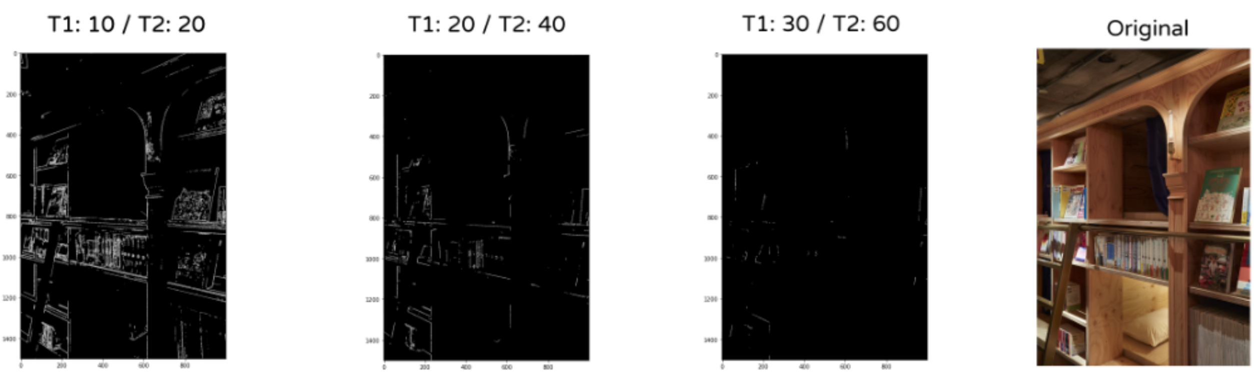
  In this part, I will show the result of the experiment. I used four pictures to do the experiment. Two of the pictures are cartoon style, and the other two are more realistic. First, I will show the results of different thresholds, and then I will compare the difference between opencv and the self implementation.

  - Different thresholds
    - Results









- Discussion

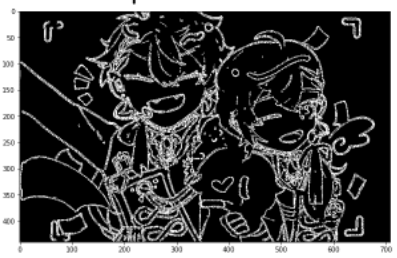
  As shown from the result, when the threshold becomes larger, the edge will be less, since the condition to be chosen to be edges is more strict.

- Comparison with OpenCv

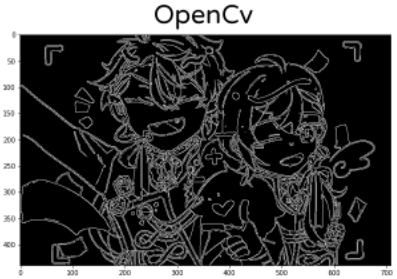
  For the self implementation part, I used 10 and 20 for thresholds, as for the OpenCv cases, I used 100 and 200.

  - Results

Implementation

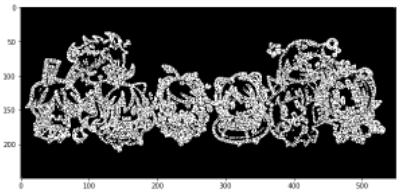


OpenCv

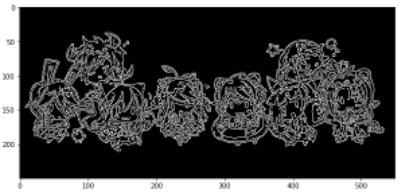


Original

Implementation

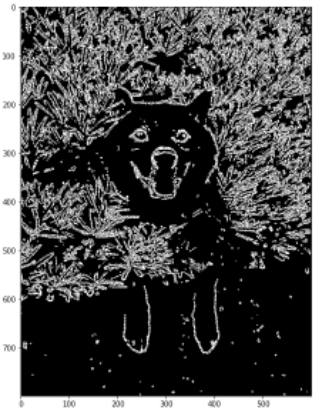


OpenCv

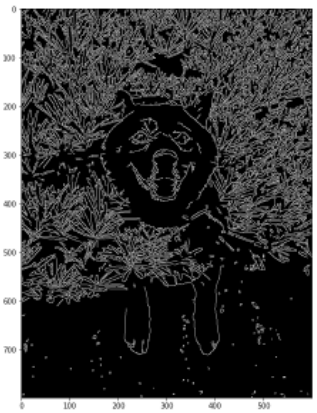


Original

Implementation

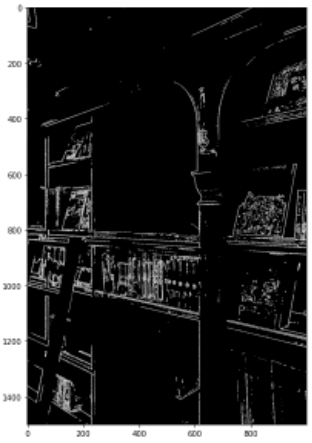


OpenCv

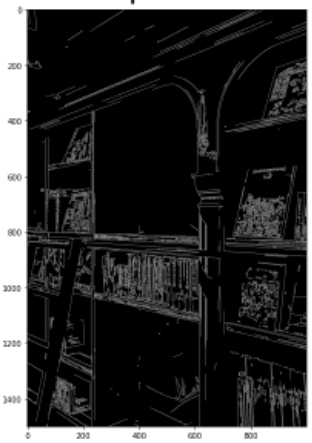


Original

Implementation



OpenCv



Original

- Discussion

  As the results show, I think the results from OpenCv are better, the edges are more stable and describe more details. The reason for the difference might be that some of the implementations are different. For the thresholds, I think there are already some that are very different. Also, the methods to do the hysteresis may also b different.

- **Improvement**

  An Improved Canny Edge Detection Algorithm Based on Type-2 Fuzzy Sets:

  https://www.sciencedirect.com/science/article/pii/S2212017312004136

  From the experiment results, we can say that it is very important to decide the thresholds. Furthermore, the thresholds may need to be decided according to different picture types. This paper proposed an algorithm called Type-2 Fuzzy Sets. The algorithm can automatically decide the thresholds in Canny edge detection.

- **Conclusion**

  This assignment aims to let us practice and implement the segmentation and edge detection methods. From the implementation, we can be more familiar with these famous methods.

```
[ ]  import cv2
     import numpy as np
     import matplotlib.pyplot as plt
     import math
```

```
     def sHalf(T, sigma):
         temp = -np.log(T) * 2 * (sigma ** 2)
         return np.round(np.sqrt(temp))

     def calculate_filter_size(T, sigma):
         return 2*sHalf(T, sigma) + 1
```

```
[ ]  def MaskGeneration(T, sigma):
         N = calculate_filter_size(T, sigma)
         shalf = sHalf(T, sigma)
         y, x = np.meshgrid(range(-int(shalf), int(shalf) + 1), range(-int(shalf), int(shalf) + 1))
         return x, y
```

```
]  def Gaussian(x,y, sigma):
       temp = ((x ** 2) + (y ** 2)) / (2 * (sigma ** 2))
       return (np.exp(-temp))

   def calculate_gradient_X(x,y, sigma):
       temp = (x ** 2 + y ** 2) / (2 * sigma ** 2)
       return -((x * np.exp(-temp)) / sigma ** 2)

   def calculate_gradient_Y(x,y, sigma):
       temp = (x ** 2 + y ** 2) / (2 * sigma ** 2)
       return -((y * np.exp(-temp)) / sigma ** 2)
```

```
def pad(img, kernel):
    r, c = img.shape
    kr, kc = kernel.shape
    padded = np.zeros((r + kr,c + kc), dtype=img.dtype)
    insert = np.uint((kr)//2)
    padded[insert: int(insert) + int(r), insert: int(insert) + int(c)] = img
    return padded

def smooth(img, kernel=None):
    if kernel is None:
        mask = np.array([[1,1,1],[1,1,1],[1,1,1]])
    else:
        mask = kernel
    i, j = mask.shape
    output = np.zeros((img.shape[0], img.shape[1]))
    image_padded = pad(img, mask)
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            output[x, y] = (mask * image_padded[x:x+i, y:y+j]).sum() / mask.sum()
    return output
```

```
def Create_Gx(fx, fy):
    gx = calculate_gradient_X(fx, fy, sigma)
    gx = (gx * 255)
    return np.around(gx)

def Create_Gy(fx, fy):
    gy = calculate_gradient_Y(fx, fy, sigma)
    gy = (gy * 255)
    return np.around(gy)
```

```
def ApplyMask(image, kernel):
    i, j = kernel.shape
    kernel = np.flipud(np.fliplr(kernel))
    output = np.zeros_like(image)
    image_padded = pad(image, kernel)
    for x in range(image.shape[0]):
        for y in range(image.shape[1]):
            output[x, y] = (kernel * image_padded[x:x+i, y:y+j]).sum()
    return output
```

```
def Gradient_Magnitude(fx, fy):
        mag = np.zeros((fx.shape[0], fx.shape[1]))
        mag = np.sqrt((fx ** 2) + (fy ** 2))
        mag = mag * 100 / mag.max()
        return np.around(mag)

def Gradient_Direction(fx, fy):
        g_dir = np.zeros((fx.shape[0], fx.shape[1]))
        g_dir = np.rad2deg(np.arctan2(fy, fx)) + 180
        return g_dir
```

```
def Digitize_angle(Angle):
    quantized = np.zeros((Angle.shape[0], Angle.shape[1]))
    for i in range(Angle.shape[0]):
        for j in range(Angle.shape[1]):
            if 0 <= Angle[i, j] <= 22.5 or 157.5 <= Angle[i, j] <= 202.5 or 337.5 < Angle[i, j] < 360:
                quantized[i, j] = 0
            elif 22.5 <= Angle[i, j] <= 67.5 or 202.5 <= Angle[i, j] <= 247.5:
                quantized[i, j] = 1
            elif 67.5 <= Angle[i, j] <= 122.5 or 247.5 <= Angle[i, j] <= 292.5:
                quantized[i, j] = 2
            elif 112.5 <= Angle[i, j] <= 157.5 or 292.5 <= Angle[i, j] <= 337.5:
                quantized[i, j] = 3
    return quantized
```

```
def Non_Max_Supp(qn, magni, D):
    M = np.zeros(qn.shape)
    a, b = np.shape(qn)
    for i in range(a-1):
        for j in range(b-1):
            if qn[i,j] == 0:
                if magni[i,j-1]< magni[i,j] or magni[i,j] > magni[i,j+1]:
                    M[i,j] = D[i,j]
                else:
                    M[i,j] = 0
            if qn[i,j]==1:
                if magni[i-1,j+1]<= magni[i,j] or magni[i,j] >= magni[i+1,j-1]:
                    M[i,j] = D[i,j]
                else:
                    M[i,j] = 0
            if qn[i,j] == 2:
                if magni[i-1,j]<= magni[i,j] or magni[i,j] >= magni[i+1,j]:
                    M[i,j] = D[i,j]
                else:
                    M[i,j] = 0
            if qn[i,j] == 3:
                if magni[i-1,j-1]<= magni[i,j] or magni[i,j] >= magni[i+1,j+1]:
                    M[i,j] = D[i,j]
                else:
                    M[i,j] = 0
    return M
```

```
def _double_thresholding(g_suppressed, low_threshold, high_threshold):
    g_thresholded = np.zeros(g_suppressed.shape)
    for i in range(0, g_suppressed.shape[0]):      # loop over pixels
        for j in range(0, g_suppressed.shape[1]):
            if g_suppressed[i,j] < low_threshold: # lower than low threshold
                g_thresholded[i,j] = 0
            elif g_suppressed[i,j] >= low_threshold and g_suppressed[i,j] < high_threshold:      # between thresholds
                g_thresholded[i,j] = 128
            else:                                 # higher than high threshold
                g_thresholded[i,j] = 255
    return g_thresholded
```

```
sigma = 0.5
T = 0.3
x, y = MaskGeneration(T, sigma)
gauss = Gaussian(x, y, sigma)
```

```
gx = -Create_Gx(x, y)
gy = -Create_Gy(x, y)
```

```
image = cv2.imread('original.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.figure(figsize = (8,8))
plt.imshow(gray, cmap='gray')
plt.show()
```

```
smooth_img = smooth(gray, gauss)
plt.figure(figsize = (8,8))
plt.imshow(smooth_img, cmap='gray')
```

```
fx = ApplyMask(smooth_img, gx)
plt.figure(figsize = (8,8))
plt.imshow(fx, cmap='gray')
```

```
fy = ApplyMask(smooth_img, gy)
plt.figure(figsize = (8,8))
plt.imshow(fy, cmap='gray')
```

```
mag = Gradient_Magnitude(fx, fy)
mag = mag.astype(int)
plt.figure(figsize = (8,8))
plt.imshow(mag, cmap='gray')
print('max', mag.max())
print('min', mag.min())
```

```
Angle = Gradient_Direction(fx, fy)
plt.figure(figsize = (5,5))
plt.imshow(Angle, cmap='gray')
print('max', Angle.max())
print('min', Angle.min())
```

```
quantized = Digitize_angle(Angle)
nms = Non_Max_Supp(quantized, Angle, mag)
plt.figure(figsize = (10,10))
plt.imshow(nms, cmap='gray')
print('max', nms.max())
print('min', nms.min())
```

```
threshold = _double_thresholding(nms, 30, 60)
cv2.imwrite('double_thresholded.jpg', threshold )
plt.figure(figsize = (10,10))
plt.imshow(threshold, cmap='gray')
```

```
hys = _hysteresis(threshold)
cv2.imwrite('Result.jpg', hys)
plt.figure(figsize = (10,10))
plt.imshow(hys, cmap='gray')
```

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

# read image
img = cv2.imread("Lenna.png", 0)
# Find edge with Canny edge detection
edges = cv2.Canny(img, 100, 200)

# display results
plt.figure(figsize = (10,10))
plt.imshow(edges, cmap='gray')
```