Image Processing (NYCU CS, Fall 2022) Programming Assignment #2

310581044 陳柏勳

實作目標 Task Option #1:Canny edge detector

介紹實作步驟

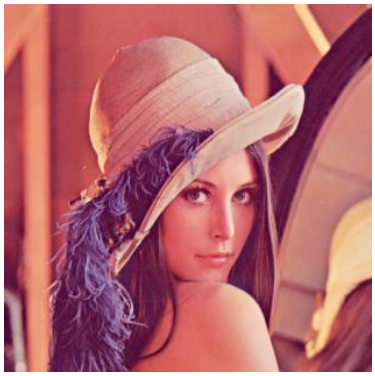
1. Noise reduction with gaussian filter
2.Calculating the intensity gradient of the image with Sobel filter
3.non_max_suppression
4.double_threshold
5.Edge tracking by hysteresis

Ablation study

1.effects of preprocessing such as smoothing and sharpening
2.choices of the two thresholds
3.effects with using nms or not
4.compare your results with the canny edge detection results from existing library/toolbox functions.

使用環境：用jupyter notebook跑

使用圖片和上次一樣從左上至右下分別命名為圖片一至四




0.change rgb to gray

在進到套用Canny edge detector前需要將rgb圖轉成灰階圖

首先對圖片做從彩色圖片轉成灰階的操作由於使用skimage.color.rgb2gray後所有圖片都會變成全黑看不情楚，但實際使用到sobel filter後就能可視化看到，故問題不大。顯示結果如下：

1. Noise reduction with gaussian filter

接著使用hw1中使用的filter對影像作模糊化，由於這裡轉成灰階後的圖片都是黑色看不清楚，所以這裡附上圖片一的中各pixel值

Ablation study

Gaussian filter寫法不同導致edge filter結果不同(以下皆為圖片一之數值)

1.Gaussian_Smooting_Filter以(3,3)kernel size在四邊補零

```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.47334598, 0.47024687, ..., 0.49741372, 0.47303531,
        0.        ],
       [0.        , 0.47142764, 0.46833839, ..., 0.46222404, 0.42750279,
        0.        ],
       ...,
       [0.        , 0.13075904, 0.1340358 , ..., 0.25113078, 0.26291757,
        0.        ],
       [0.        , 0.12749392, 0.13098654, ..., 0.26946659, 0.27802923,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

2.Gaussian_Filter_rb0以(3,3)kernel size會左上部分有值但右下從一排是0變成兩排式0

```
array([[0.47334598, 0.47024687, 0.46729302, ..., 0.47303531, 0.        ,
        0.        ],
       [0.47142764, 0.46833839, 0.46651984, ..., 0.42750279, 0.        ,
        0.        ],
       [0.46696999, 0.46442557, 0.46375763, ..., 0.31571234, 0.        ,
        0.        ],
       ...,
       [0.12749392, 0.13098654, 0.13153478, ..., 0.27802923, 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

在blur使用其他filter和Gaussian filter效果差別

1.median filter

```
array([[0.52702196, 0.60719529, 0.59368863, ..., 0.63184157, 0.52702196,
        0.        ],
       [0.60580353, 0.60719529, 0.60719529, ..., 0.63184157, 0.63184157,
        0.        ],
       [0.59595137, 0.60719529, 0.59790078, ..., 0.62877529, 0.52702196,
        0.        ],
       ...,
       [0.1651902 , 0.16965451, 0.17359882, ..., 0.32239333, 0.33209569,
        0.        ],
       [0.1553749 , 0.16965451, 0.16965451, ..., 0.3508702 , 0.35808745,
        0.        ],
       [0.        , 0.14645333, 0.15673608, ..., 0.3508702 , 0.35808745,
        0.        ]])
```

2.sharpen filter

```
[[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
  0.        ],
 [0.        , 0.06787874, 0.06883956, ..., 0.07577407, 0.08060688,
  0.        ],
 [0.        , 0.06781495, 0.07133821, ..., 0.09401451, 0.10282937,
  0.        ],
 ...,
 [0.        , 0.01567891, 0.01954932, ..., 0.0387681 , 0.04149647,
  0.        ],
 [0.        , 0.01349102, 0.02012619, ..., 0.04112357, 0.04390105,
  0.        ],
 [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
  0.        ]])
```

會在最後附上以不同filter做blur後的canny edge filter結果

2.Calculating the intensity gradient of the image

使用計算gradient較為簡單易懂的Sobel filter其kernel為[[1, 2, 1], [0, 0, 0], [-1, -2, -1]]和[[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]的sobel filter來偵測出x方向和y方向的edge direction 和 edge intensity，gradients計算方法如下：

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{and} \quad \mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * \mathbf{A}$$

接著再將x y 方向gradient合併求出總gradient和角度，方法如下：

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}, \quad \mathbf{\Theta} = \operatorname{atan2}(\mathbf{G}_y, \mathbf{G}_x)$$

以下四張圖的sobel結果可以看到，圖片的邊緣經由取gradient已經能基本看出輪廓但有些不是邊緣的線也被取了進來


3.non_max_suppression

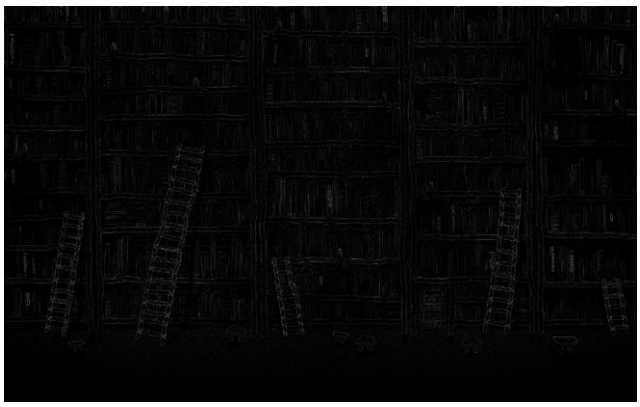
查看pixel時將其四周的pixel分為以下四種：

- 0 degrees => horizontal direction
- 45 degrees => positive diagonal
- 90 degrees => vertical direction
- 135 degrees => negative diagonal

也就是說對於pixel x y 可以根據角度查詢其四個方向的鄰近pixel，同時把此pixel和四周的pixel比較大小值，保留較大的就可以得到更細的線，分類方式如下：

- between [0, 22.5) and [157.5, 180] => looking at pixel (x, y-1) and pixel (x, y+1)
- between [22.5 , 67.5) => looking at pixel (x-1, y-1) and pixel (x+1, y+1)
- between [67.5, 112.5) => looking at pixel (x-1, y) and pixel (x+1, y)
- between [112.5, 157.5) => looking at pixel (x+1, y-1) and pixel (x-1, y+1)

以下為使用nms後的四張圖片，相比上一步驟，所取圖片的線明顯少了許多，在較暗的圖片甚至難以看出取了那些線



4.double_threshold

這裡是Canny edge detector的核心思想，同時也是最能影響能否成功偵測的關鍵，設定兩個值threshold值，將所有pixel依據threshold分成strong(高於大threshold) weak(高於小threshold)和捨棄值(低於小threshold)可看到根據threshold的strong point會被選到，相比上一階段更好看出圖片邊緣的對比度

Ablation study

對圖片一使用不同threshold值結果以(lowThreshold，highThreshold)代表

(0.03,0.07) (0.03,0.12) (0.03, 0.2)

Interesting point

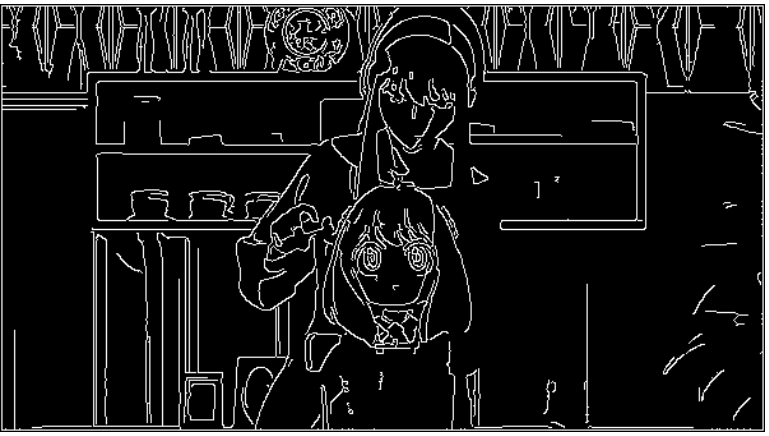
似乎影響圖片邊緣的是strong threshold影響較多，在strong threshold固定情況weak threshold上升下降不太有明顯差異但微調strong threshold會造成線條差異很大

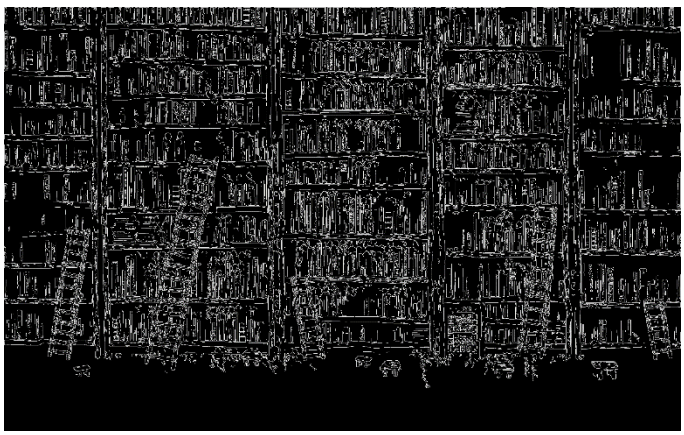
5.Edge tracking by hysteresis

最後將以區分成strong和weak的pixel進行挑選，選取全部strong pixel以及和strong相鄰的weak pixel，可以看到有些在上一步驟刪去的圖片線條被加回來了，將原本有些中間斷掉的線條成功的連成線

以下是以treshold值lowThresholdRatio=0.03, highThresholdRatio=0.12

四張圖片通過上述步驟形成的canny edge detector成果





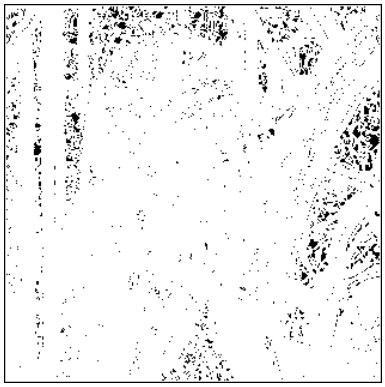
Interesting point

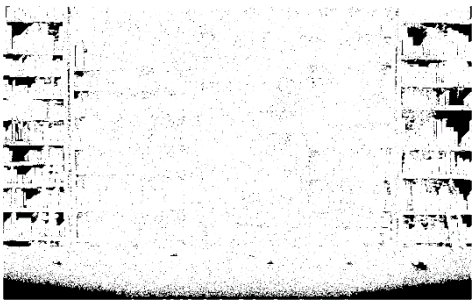
對於整體亮度較低的圖片，由於其pixel數值多數也較低，故對於threshold選擇也需要調低，否則會和圖四一樣不太能把重要邊給過濾出來

Ablation study

有無使用nms之canny edge filter結果

依據猜想，沒有使用nms的canny edge filter會留有冗餘的邊，以下是四張圖片未加入nms的canny edge filter結果：



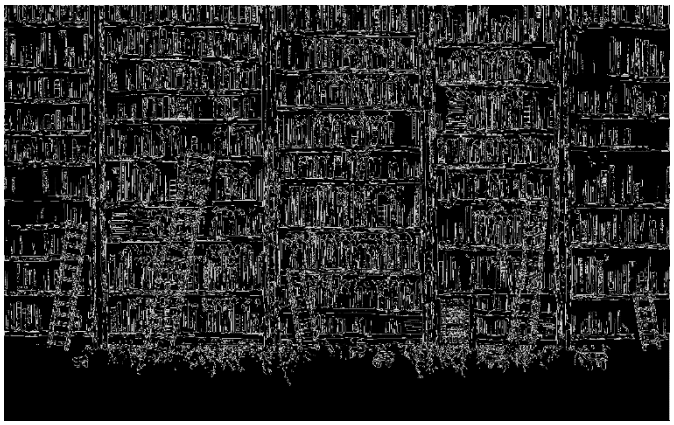


可以看到和預想的不一樣，出來的結果只有黑白區域代表可能nms其實是把白色區域縮成一條線，後面才能把白色邊線條檢測出來。

Ablation study

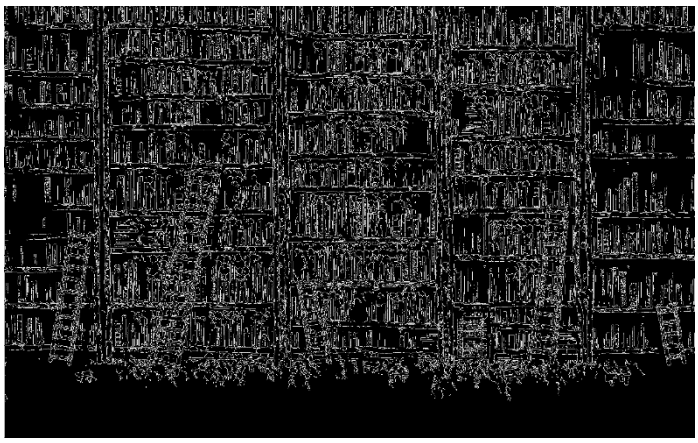
此處為第一步驟根據不同的filter得到最終結果如下

1. Gaussian_Filter_rb0



觀察主要是圖片四周的東西都不太檢測得出來， 或許是因為寫法不同造成連續0太過於集中導致圖片邊緣都是0無法有效套用canny來偵測

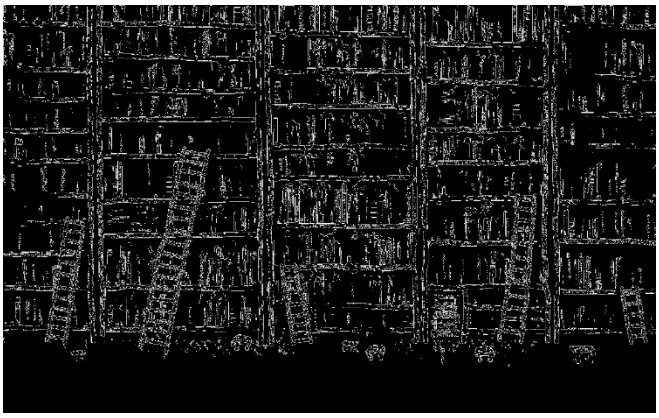
2. median filter

可以看到median效果類似於strong treshhold值設定較小時的情況，且似乎median讓圖片有些錯誤判斷的雜訊產生，我想或許是原圖片同一顏色區域大時，用median會使那區值都相近像是直線般，造成有detector錯誤判斷有直線邊緣

3. sharpen filter



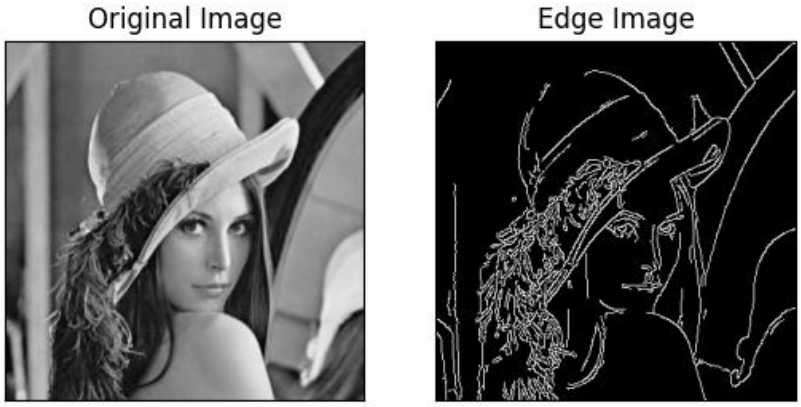



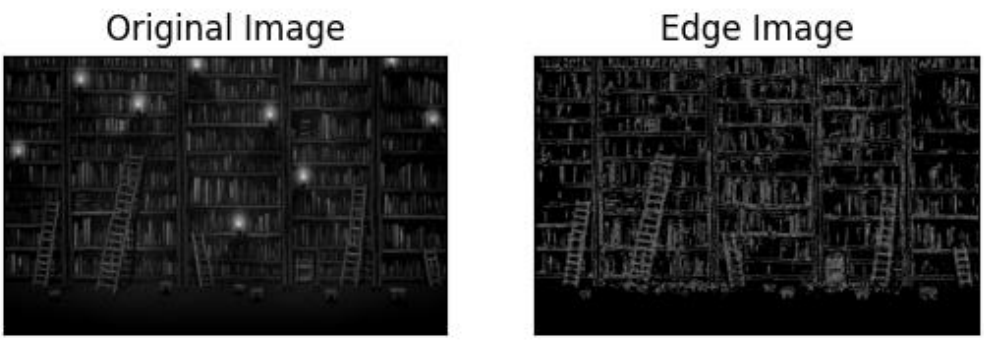
可以看到sharpen是效果最差的，對於邊只能選到靠近邊緣的點而沒在一條直線上，故形成像毛邊的效果

Ablation study

使用opencv內建之canny()測試四張照片結果如下：

感覺和自己寫的比起來更加平滑一些，且背景的邊能夠較晚整的檢測到，我想或許主因是在第一步驟的filter寫法不同，和treshhold值設定不同引起的差距





Code部分

```
import numpy as np
import skimage
import math
def Gaussian_Smooting_Filter(image,size = (3,3), sigma=1 ):
    output = np.zeros(size)
    h = (size[0]-1)/2
    w = (size[1]-1)/2
    for i in range(size[0]):
        for j in range(size[1]):
            output[i,j] = (1/(2*math.pi*(sigma**2)))*math.exp(-((i-
h)**2+(j-w)**2)/(2*sigma**2))
    image_size = image.shape
    new = np.zeros([image_size[0], image_size[1]])
    for i in range(1, image_size[0] - 1):
        for j in range(1, image_size[1] -1):
            new[i, j] = np.sum(np.multiply(image[i - 1 : i +2, j - 1 : j
+2], output))

    return new
def Gaussian_Filter_rb0(data,k_size = (3,3), sigma = 1):

    k_height = (k_size[0]-1)/2
    k_width = (k_size[1]-1)/2
    mask = np.zeros(k_size)
    for i in range(k_size[0]):
        for j in range(k_size[1]):
            mask[i,j] = (1/(2*math.pi*(sigma**2)))*math.exp(-((i-
k_height)**2+(j-k_width)**2)/(2*sigma**2))

    img= np.array(data)
    height, width = img.shape

    img_new = np.zeros([height, width])
    for i in range(height-k_size[0]+1):
        for j in range(width-k_size[1]+1):
            img_new[i,j] = np.sum(img[i:i+k_size[0],j:j+k_size[1]]*mask)
```

```
    return img_new
def median_filter(image, filter_size):
    temp = []
    indexer = filter_size // 2
    data_final = []
    data= np.array(image)
    data_final = np.zeros((len(data),len(data[0])))
    for i in range(len(data)):

        for j in range(len(data[0])):

            for z in range(filter_size):
                if i + z - indexer < 0 or i + z - indexer > len(data) -
1:
                    for c in range(filter_size):
                        temp.append(0)
                else:
                    if j + z - indexer < 0 or j + indexer > len(data[0])
- 1:
                        temp.append(0)
                    else:
                        for k in range(filter_size):
                            temp.append(data[i + z - indexer][j + k -
indexer])

            temp.sort()
            data_final[i][j] = temp[len(temp) // 2]
            temp = []


    return data_final
def sharpen_filter(data):
    img= np.array(data)

    height, width = img.shape

    # Develop Averaging filter(3, 3) mask
```

```
    mask = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
    mask = mask / 9

    # Convolve the 3X3 mask over the image
    img_new = np.zeros([height, width])

    for i in range(1, height-1):
        for j in range(1, width-1):
            temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0,
1]+img[i-1, j + 1]*mask[0, 2]+img[i, j-1]*mask[1, 0]+ img[i, j]*mask[1,
1]+img[i, j + 1]*mask[1, 2]+img[i + 1, j-1]*mask[2, 0]+img[i + 1,
j]*mask[2, 1]+img[i + 1, j + 1]*mask[2, 2]

            img_new[i, j]= temp



    return img_new
def sobel_filter(data):
    img= np.array(data)

    height, width = img.shape

    # Develop Averaging filter(3, 3) mask
    masky = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    masky = masky / 9
    maskx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    maskx = maskx / 9


    convolved = np.zeros(img.shape)
    # Convolve the 3X3 mask over the image
    img_newx = np.zeros([height, width])
    img_newy = np.zeros([height, width])

    for i in range(1, height-1):
        for j in range(1, width-1):
```

```
            tempx = img[i-1, j-1]*maskx[0, 0]+img[i-1, j]*maskx[0,
1]+img[i-1, j + 1]*maskx[0, 2]+img[i, j-1]*maskx[1, 0]+ img[i,
j]*maskx[1, 1]+img[i, j + 1]*maskx[1, 2]+img[i + 1, j-1]*maskx[2,
0]+img[i + 1, j]*maskx[2, 1]+img[i + 1, j + 1]*maskx[2, 2]
            img_newx[i, j]= tempx

            tempy = img[i-1, j-1]*masky[0, 0]+img[i-1, j]*masky[0,
1]+img[i-1, j + 1]*masky[0, 2]+img[i, j-1]*masky[1, 0]+ img[i,
j]*masky[1, 1]+img[i, j + 1]*masky[1, 2]+img[i + 1, j-1]*masky[2,
0]+img[i + 1, j]*masky[2, 1]+img[i + 1, j + 1]*masky[2, 2]
            img_newy[i, j]= tempy


    G = np.hypot(img_newx, img_newy)
    G = G / G.max() * 255


    theta = np.arctan2(img_newy, img_newx)

    return (G, theta)
def non_max_suppression(img, D):
    M, N = img.shape
    Z = np.zeros((M,N), dtype=np.int32)
    angle = D * 180. / np.pi
    angle[angle < 0] += 180


    for i in range(1,M-1):
        for j in range(1,N-1):
            try:
                q = 255
                r = 255

               #angle 0
                if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <=
180):
                    q = img[i, j+1]
                    r = img[i, j-1]
```

```
                #angle 45
                elif (22.5 <= angle[i,j] < 67.5):
                    q = img[i+1, j-1]
                    r = img[i-1, j+1]
                #angle 90
                elif (67.5 <= angle[i,j] < 112.5):
                    q = img[i+1, j]
                    r = img[i-1, j]
                #angle 135
                elif (112.5 <= angle[i,j] < 157.5):
                    q = img[i-1, j-1]
                    r = img[i+1, j+1]

                if (img[i,j] >= q) and (img[i,j] >= r):
                    Z[i,j] = img[i,j]
                else:
                    Z[i,j] = 0

            except IndexError as e:
                pass

    return Z
def threshold(img, lowThresholdRatio=0.05, highThresholdRatio=0.09):

    highThreshold = img.max() * highThresholdRatio;
    lowThreshold = highThreshold * lowThresholdRatio;

    M, N = img.shape
    res = np.zeros((M,N), dtype=np.int32)

    weak = np.int32(25)
    strong = np.int32(255)

    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)

    weak_i, weak_j = np.where((img <= highThreshold) & (img >=
lowThreshold))
```

```
    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak

    return (res, weak, strong)
def hysteresis(img, weak, strong=255):

    M, N = img.shape

    for i in range(1, M-1):
        for j in range(1, N-1):
            if (img[i,j] == weak):
                try:
                    if ((img[i+1, j-1] == strong) or (img[i+1, j] ==
strong) or (img[i+1, j+1] == strong)
                        or (img[i, j-1] == strong) or (img[i, j+1] ==
strong)
                        or (img[i-1, j-1] == strong) or (img[i-1, j] ==
strong) or (img[i-1, j+1] == strong)):
                        img[i, j] = strong
                    else:
                        img[i, j] = 0
                except IndexError as e:
                    pass

    return img
from PIL import Image
images = []


images.append(Image.open('lena.jpg'))
images.append(Image.open('giant.jpg'))
images.append(Image.open('spy.jpg'))
images.append(Image.open('dark_book.jpg'))

images[3]
img_gray= []
img_gray0= skimage.color.rgb2gray(images[0])
img_gray1= skimage.color.rgb2gray(images[1])
```

```
img_gray2= skimage.color.rgb2gray(images[2])
img_gray3= skimage.color.rgb2gray(images[3])
output = Image.fromarray(img_gray3.astype('uint8'))
output
from scipy import misc
import numpy as np

from scipy.ndimage.filters import convolve
def Canny_detector(img):
    """ Your implementation instead of skimage """
    img_filtered = Gaussian_Smooting_Filter(img,size = (3,3), sigma = 1)
    #img_filtered = Gaussian_Filter_rb0(img,k_size = (3,3), sigma = 1)
    #img_filtered = median_filter(img, filter_size = 3)
    #img_filtered = sharpen_filter(img)
    grad, theta = sobel_filter(img_filtered)
    img_nms = non_max_suppression(grad, theta)
    img_thresh, weak, strong = threshold(img_nms,
lowThresholdRatio=0.03, highThresholdRatio=0.12)
    img_final = hysteresis(img_thresh, weak, strong=strong)
    #img_new = np.clip(img_final, 0, 255)
    output = Image.fromarray(img_final.astype('uint8'))
    return output
def Canny_detector2(img):
    """ Your implementation instead of skimage """
    img_filtered = Gaussian_Smooting_Filter(img,size = (3,3), sigma = 1)
    #img_filtered = Gaussian_Filter_rb0(img,k_size = (3,3), sigma = 1)
    #img_filtered = median_filter(img, filter_size = 3)
    #img_filtered = sharpen_filter(img)
    grad, theta = sobel_filter(img_filtered)
    #img_nms = non_max_suppression(grad, theta)
    img_thresh, weak, strong = threshold(grad, lowThresholdRatio=0.03,
highThresholdRatio=0.12)
    #img_thresh, weak, strong = threshold(img_nms,
lowThresholdRatio=0.03, highThresholdRatio=0.12)
    img_final = hysteresis(img_thresh, weak, strong=strong)
    #img_new = np.clip(img_final, 0, 255)
    output = Image.fromarray(img_final.astype('uint8'))
    return output
```

```
img_filtered = median_filter(img_gray0,3)
#output = Image.fromarray(img_filtered.astype('uint8'))
#output
img_filtered
img_filtered = sharpen_filter(img_gray0)
#output = Image.fromarray(img_filtered.astype('uint8'))
img_filtered
img_filtered = Gaussian_Smooting_Filter(img_gray0,size = (3,3),
sigma=1 )
img_filtered
img_filtered = Gaussian_Filter_rb0(img_gray0,k_size = (3,3), sigma=1 )
img_filtered
img_filtered1 = Gaussian_Smooting_Filter(img_gray1,size = (3,3),
sigma=1 )
#img_filtered1
img_filtered2 = Gaussian_Smooting_Filter(img_gray2,size = (3,3),
sigma=1 )
#img_filtered2
img_filtered3 = Gaussian_Smooting_Filter(img_gray3,size = (3,3),
sigma=1 )
#img_filtered3
grad,theta= sobel_filter(img_filtered)
output = Image.fromarray(grad.astype('uint8'))
output
#grad
#theta
grad1,theta1= sobel_filter(img_filtered1)
output = Image.fromarray(grad1.astype('uint8'))
output
grad2,theta2= sobel_filter(img_filtered2)
output = Image.fromarray(grad2.astype('uint8'))
output
grad3,theta3= sobel_filter(img_filtered3)
output = Image.fromarray(grad3.astype('uint8'))
output
img_nms = non_max_suppression(grad, theta)
output1 = Image.fromarray(img_nms.astype('uint8'))
output1
```

```
img_nms1 = non_max_suppression(grad1, theta1)
output1 = Image.fromarray(img_nms1.astype('uint8'))
output1
img_nms2 = non_max_suppression(grad2, theta2)
output1 = Image.fromarray(img_nms2.astype('uint8'))
output1
img_nms3 = non_max_suppression(grad3, theta3)
output1 = Image.fromarray(img_nms3.astype('uint8'))
output1
img_thresh, weak, strong = threshold(img_nms, lowThresholdRatio=0.03,
highThresholdRatio=0.12)
output2 = Image.fromarray(img_thresh.astype('uint8'))
output2
img_final = hysteresis(img_thresh, weak, strong=strong)
output = Image.fromarray(img_final.astype('uint8'))
output
Canny_detector(img_gray0)
Canny_detector(img_gray1)
Canny_detector(img_gray2)
Canny_detector(img_gray3)
Canny_detector2(img_gray0)
Canny_detector2(img_gray1)
Canny_detector2(img_gray2)
Canny_detector2(img_gray3)
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('dark_book.jpg',0)
edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
plt.show()


#images.append(Image.open('lena.jpg'))
#images.append(Image.open('giant.jpg'))
#images.append(Image.open('spy.jpg'))
```

```
#images.append(Image.open('dark_book.jpg'))
```