Cover Page

影像處理作業4 - Color edge detection

409410019 資工四 王郁誠

HW due : 6/15 00:00

HW handed in : 6/14

程式執行流程：

執行 `python hw4.py` 後，會先看到peppers.png的執行結果，接著取消註解46~47行，再重新執行，即可以看到其他圖片的執行結果。

Technical description

實作padding

```
def padding(img: np.ndarray):
    height, width = img.shape
    padded_img = np.zeros((height+2, width+2), dtype = np.uint8)
    # 四邊
    padded_img[1:-1, 1:-1] = img
    padded_img[1:-1,0] = img[:,0]
    padded_img[1:-1,-1] = img[:,-1]
    padded_img[0,1:-1] = img[0,:]
    padded_img[-1,1:-1] = img[-1,:]
    # 四角
    padded_img[0,0] = img[0,0]
    padded_img[0,-1] = img[0,-1]
    padded_img[-1,0] = img[-1,0]
    padded_img[-1,-1] = img[-1,-1]
    return padded_img
```

先開一個比input img四周各大1 pixel的padded_img，接著將img複製到padded_img中間，並取img外圍一圈作為padded_img四周邊界來進行padding，最後回傳。

實作 sobel operator卷積運算

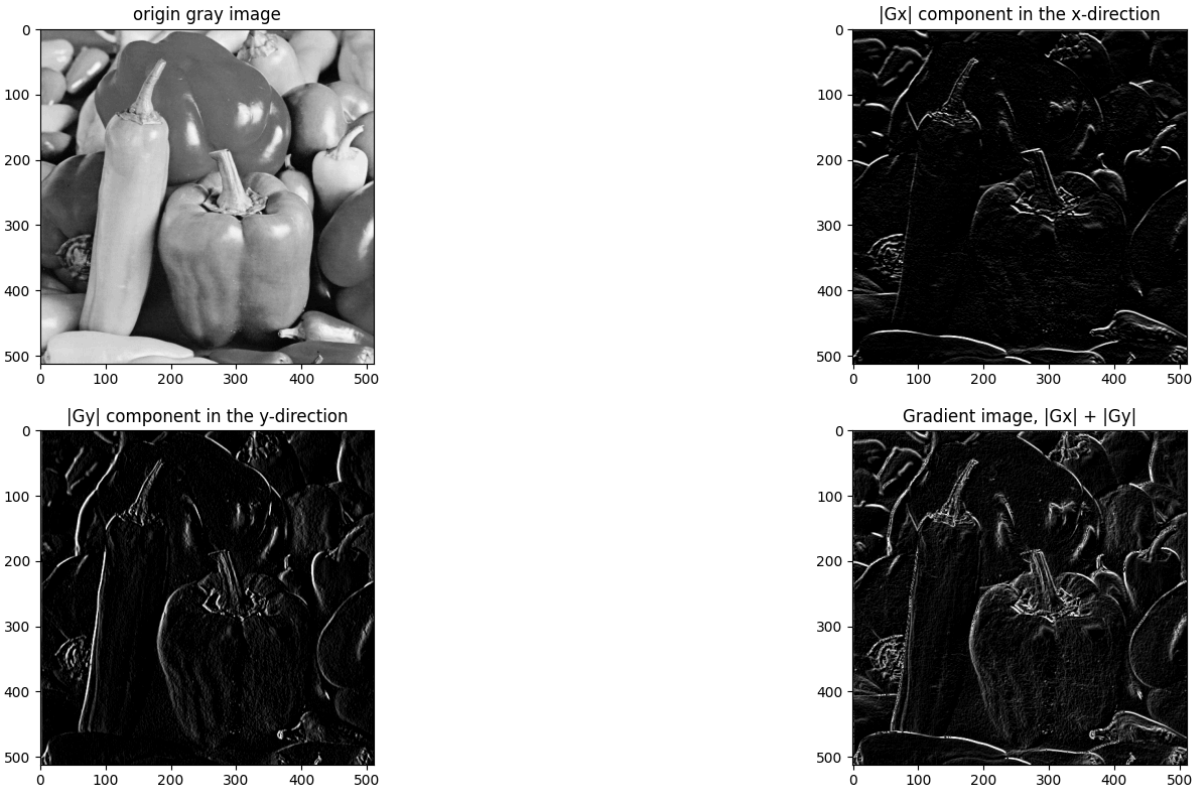
```
def sobel_filter_op(img: np.ndarray):
    height, width = img.shape
    img = img.astype(np.float32)
    filter_img_x = np.zeros((height, width), dtype = np.float32)
    filter_img_y = np.zeros((height, width), dtype = np.float32)
    for i in range(1,height-1):
        for j in range(1,width-1):
            filter_img_x[i,j] = ( - (img[i-1,j-1] + 2*img[i-1,j] + img[i-1,j+1]) + (img[i+1,j-1
            filter_img_y[i,j] = ( - (img[i-1,j-1] + 2*img[i,j-1] + img[i+1,j-1]) + (img[i-1,j+1
    result_img_x = filter_img_x[1:-1, 1:-1]
    result_img_y = filter_img_y[1:-1, 1:-1]
    result_img_x = np.clip(result_img_x, 0, 255)
    result_img_y = np.clip(result_img_y, 0, 255)
    return result_img_x.astype(np.uint8), result_img_y.astype(np.uint8)
```

先傳入padding過邊界的灰階圖，將傳入的圖型態轉成float32操作，接著先開兩個空白img做為卷積運算的結果圖。接著遍歷img中所有pixel，根據PPT ch10 p.15給的filter mask(有x,y兩種方向)，對img進行卷積運算(公式太長，截圖只擷取部分)，並將結果分別存入x,y的結果圖。接著再將原本padding的邊界給拿掉，大小才會跟原圖一樣。最後再檢查pixel value，將值clip到合理範圍(0~255)，再以uint8作為data type傳回。

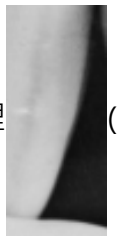
Experimental results

執行程式後，會看到一個視窗"result"，裡面由上至下,由左至右分別為<原圖, x方向邊緣偵測圖, y方向邊緣偵測圖, 整體邊緣偵測圖>。

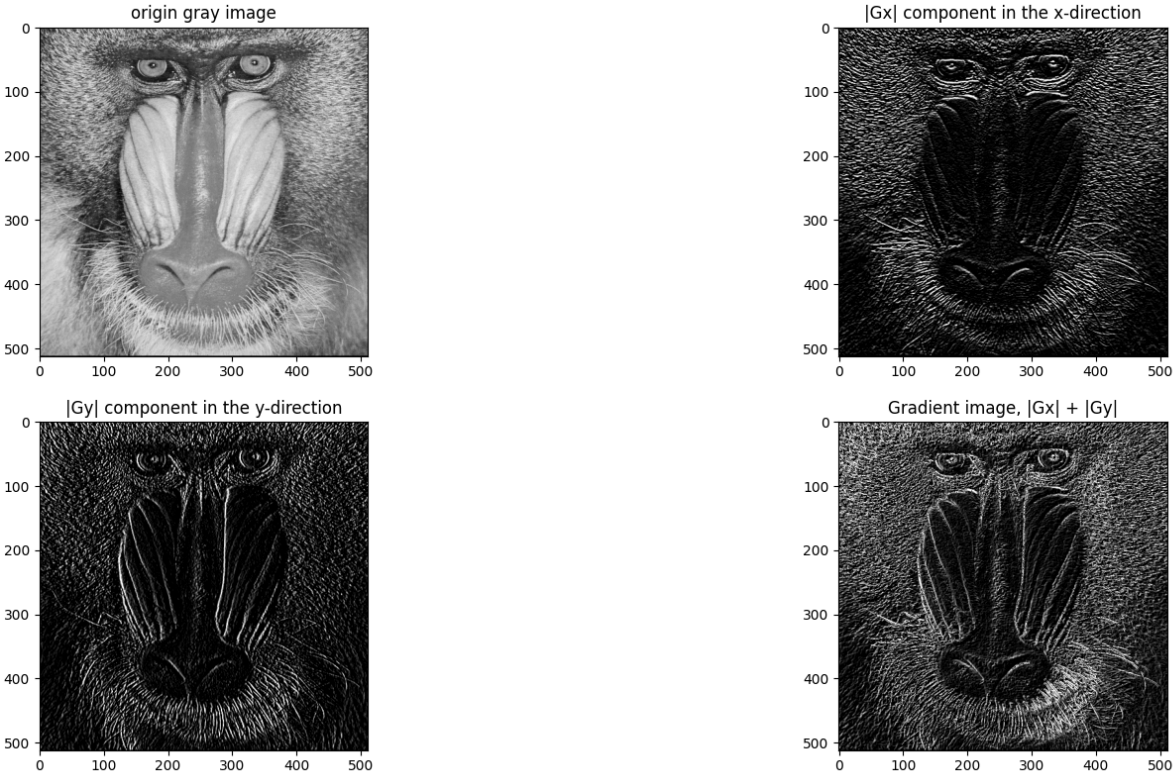
- peppers.png



這case可以明顯的看出x方向的邊緣和y方向的邊緣，兩者結合得出的最終邊緣偵測

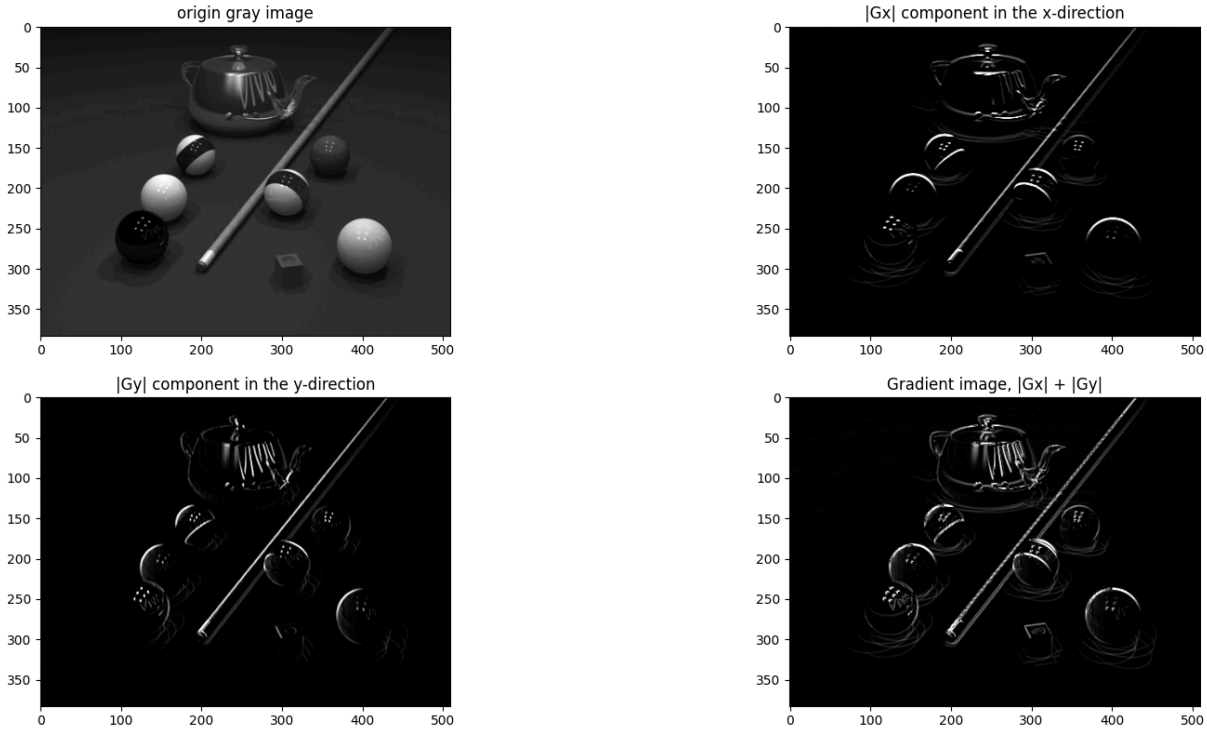
效果也不錯，但有一點我覺得很怪，就是左下角這裡  (約在原圖400*150的位置)，我認為這裡是明顯的邊緣，但是在y方向的邊緣偵測卻沒有偵測出來，之後有用cv2.filter2D()來做卷積運算，出來也是沒被偵測，感覺有可能是sobel operator本身的問題。

- baboon.png



這case我覺得效果也不錯，x方向邊緣偵測出眼窩、鼻孔、嘴角；y方向邊緣偵測出鼻子紋路等等，皆表現不錯。合併出來的整體邊緣偵測圖同樣也表現不錯。

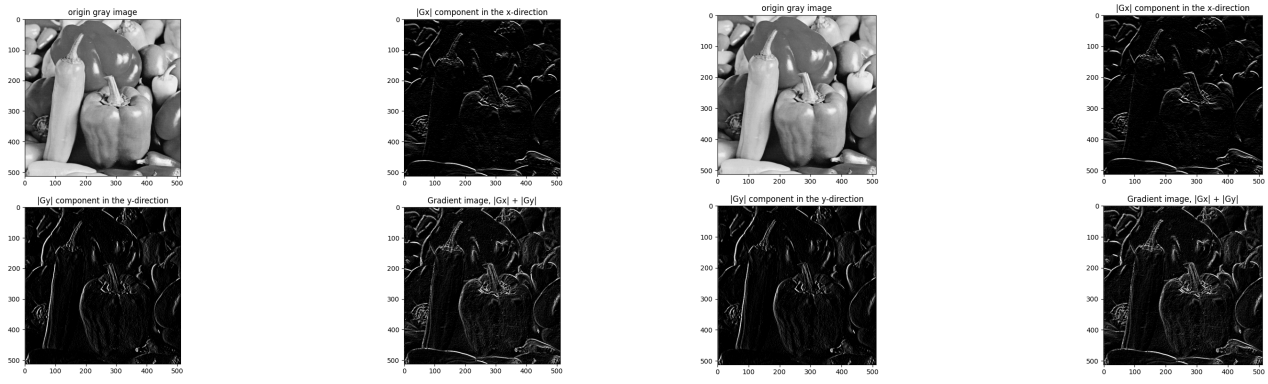
- pool.png



我覺得這張效果也不賴，同樣x,y方向邊緣偵測都有偵測出東西來，合併起來的圖片也挺好看。

Discussions & Appendix

在hw2中，我自己的卷積運算function跟cv2.filter2D()做出來的不一樣，我那時做的挺糟的，但之後發現是因為我那時沒有把處理完的圖片clip到合理範圍。在這次作業中，我就有做clip的動作，使得我自己function做出來的效果和cv2.filter2D()差不多，下面為以peppers.png為例，我自己卷積運算的function跟cv2.filter2D()比較：



基本可以說是一模一樣，實作出有用且正確的功能，挺有成就感的。

接著是我這次作業遇到的問題，其實這次整體實作過程都沒什麼問題，但就是peppers.png那張圖的左下角那裏(在實驗結果那邊有講過的點)，那裏很明顯就是邊緣吧，但不知道為什麼就是偵測不出來，這是我這次作業唯一覺得疑惑的地方。

不知不覺也過完一個學期了，透過這門課我學到了許多影像處理的概念和知識，也謝謝助教認真地幫我們檢查作業~。