Hw2_report

學號 : 410410043

姓名 : 林秉燁

全部作業都是使用捲積運算所做

步驟 :

1. 讀取圖片
2. 獲得捲積核
3. 做捲積運算獲得新圖
4. 輸出圖片

> 若是題目 a, b 還會再計算 PSNR

題目 a & b

mask 製作

針對a, b 都是使用 gaussian filter 來達到 blur 的效果
因此我將 製作mask 都包裝成一個函式來使用

```
def get_mask(size: int, sigma: float):
    minX = minY = -(size // 2)
    maxX = maxY = (size // 2)
    x, y = np.mgrid[minX : maxX + 1, minY : maxY + 1]
    kernel = np.exp((-(x**2 + y**2))/(2 * (sigma ** 2)))
    kernel = kernel / kernel.sum()
    return kernel
```

將其中的 size 改變即可完成第一題
改變 sigma 即可完成第二題

> 也就是利用以下公式得出捲積核，由於這公式所轉出的捲積核總和並不是1，因此最後在除總和即可
>
> $e^{\frac{-(x^2+y^2)}{2\sigma^2}}$

捲積運算

捲積的方式則是先按照我們捲積核的大小，建立一個比原圖略大的矩陣，再將原圖複製到矩陣的中心，藉此來獲得一張原圖周圍有數圈0的新圖，再開始做捲積。

但彩色圖還要先將其的 r, g, b轉出，因此所有矩陣的建立都是依靠 numpy 來達成，這樣會更便利的得出 r, g, b的矩陣，不過opencv 的圖片讀取顏色排列是 b, g, r，因此我用以下方式來得出。

```
tmp1 = new_img[i:i+mask_size, j:j+mask_size]
b = tmp1[:,:,0]
g = tmp1[:,:,1]
r = tmp1[:,:,2]
b = b * blur_mask
g = g * blur_mask
r = r * blur_mask
final_img[i][j] = [int(b.sum()), int(g.sum()), int(r.sum())]
```

這樣一來，就有最後的圖片了，再將其輸出

計算PSNR

最後利用 opencv 內建的函式得出 PSNR值

```
# cv2.PSNR(原圖, 新圖)
cv2.PSNR(cv2.imread(input_name), cv2.imread(output_name))
```

PSNR

用來表示訊號最大可能功率和影響它表示精度的破壞性雜訊功率的比值

簡單來說，就是看圖片處理後有沒有失真，與原圖差距的一個衡量方法

若測出來的值越高 則代表越不失真，反之，則越偏離原圖

a 3x3 PSNR : 39.65107717143949

a 7x7 PSNR : 39.6252052267915

a 11x11 PSNR : 39.62528001588494

b σ 1 PSNR : 32.69518187239329

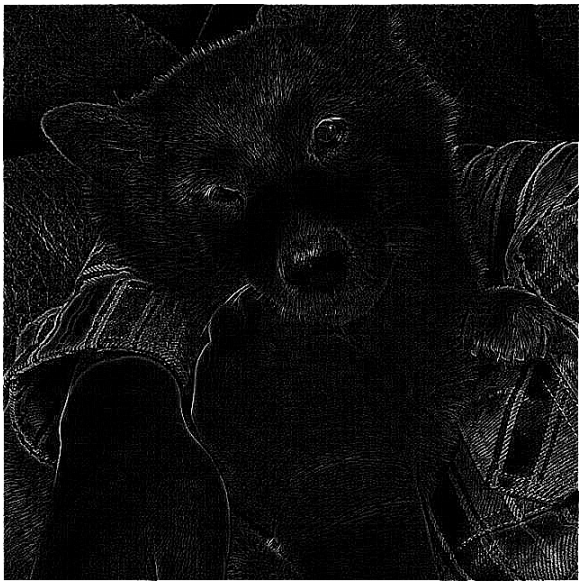
b σ 10 PSNR : 31.29912330122936

b σ 30 PSNR : 31.29772486223824

在 a 題裡，我所設定的σ是0.5，不確定是否有關係，但PSNR卻差不多

但在 b 題裡， 有隨著σ升高而下降

題目 d



捲積運算的方式 與上面兩題一樣

但是為了做這題的edge detection，要先將圖片改成灰階

方式是在讀取的時候透過opencv 讀圖片的參數來實現的

```
# cv2.imread(image, 0)
# 第二個參數為0時，會得到灰階圖片
```

因為這次沒有rgb，因此可以直接去做捲積，不用再去多做處理