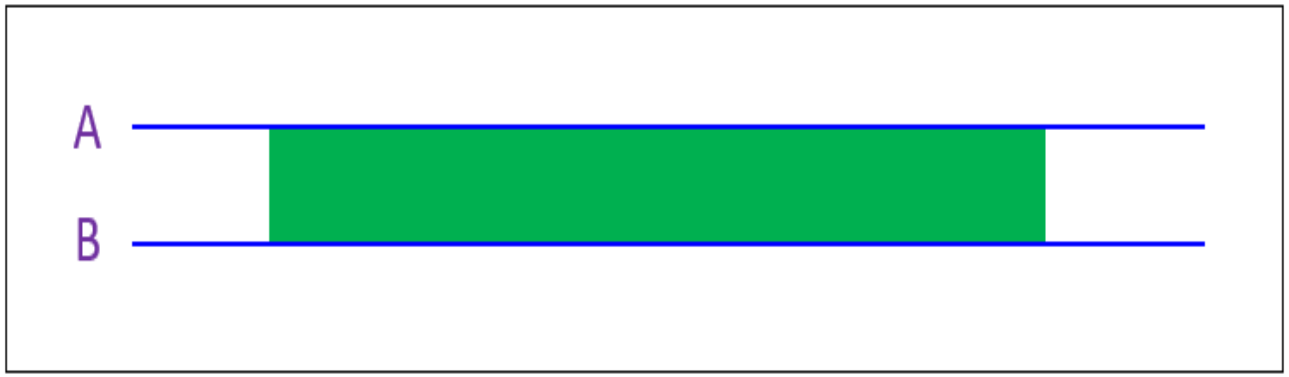
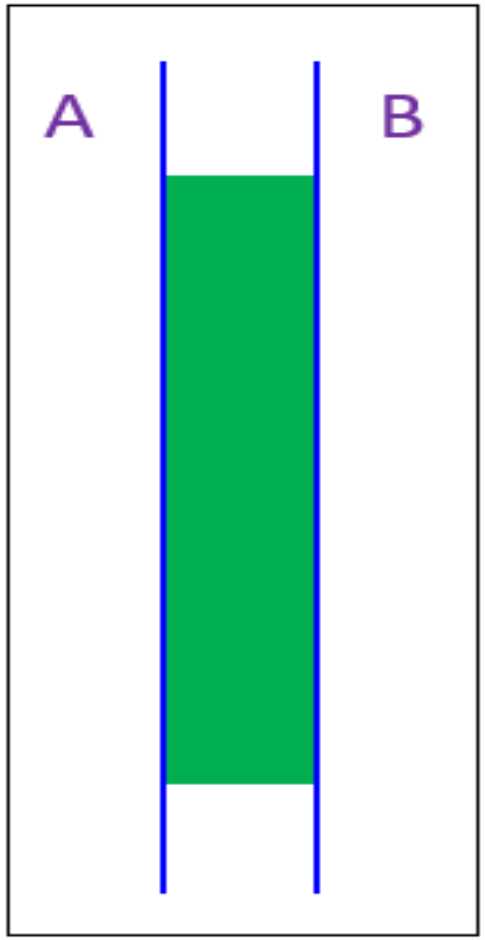
第13章
影像梯度與邊緣偵測

13.1：影像梯度的基礎觀念

- 13.1.1：直覺方法認識影像邊界

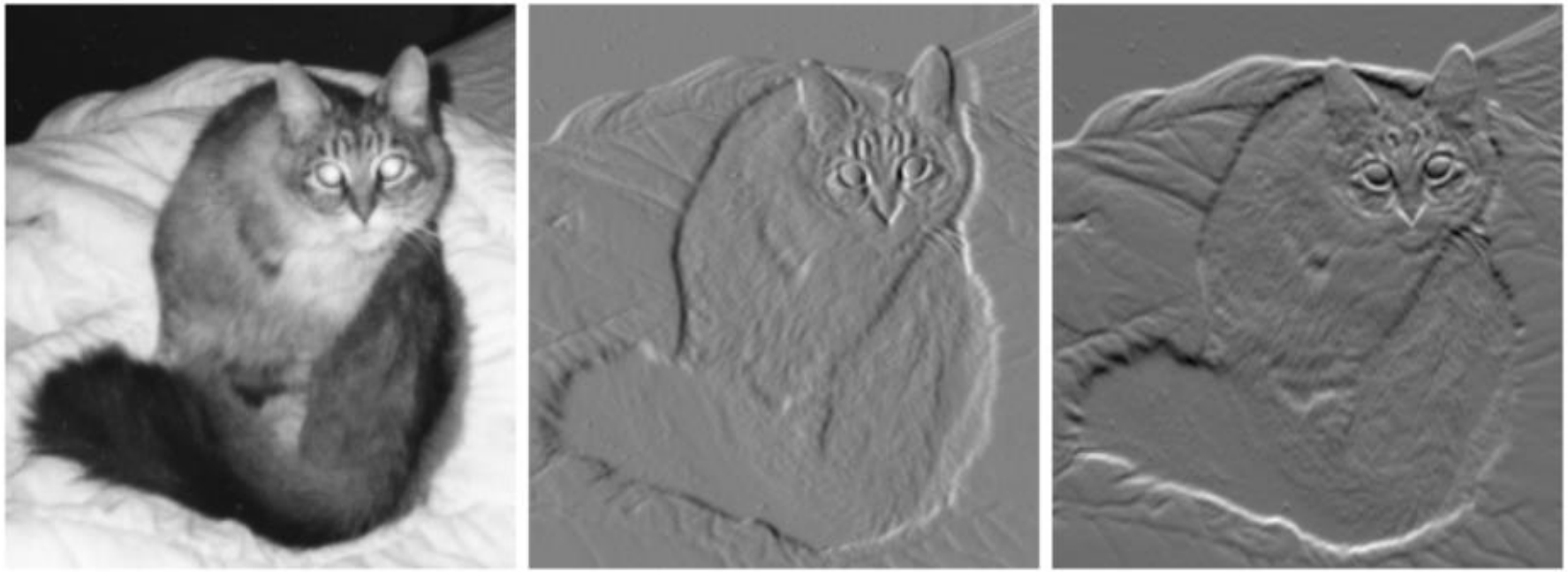




13.1.2：認識影像梯度

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

13.13：機器視覺



13.2：OpenCV函數Sobel( )

- 13.2.1：Sobel算子

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

用於 x 軸算子

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

用於 y 軸算子

13.22：使用Sobel算子計算x軸方向影像梯度

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A$$

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} p1 & p2 & p3 \\ p4 & p5 & p6 \\ p7 & p8 & p9 \end{bmatrix}$$

$$p5_x = (p3 - p1) + 2 * (p6 - p4) + (p9 - p7)$$

13.23：使用Sobel算子計算y軸方向影像梯度

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} p1 & p2 & p3 \\ p4 & p5 & p6 \\ p7 & p8 & p9 \end{bmatrix}$$

$$p5_y = (p7 - p1) + 2 * (p8 - p2) + (p9 - p3)$$

13.2.4：Sobel( )函數

- dst = cv2.Sobel(src, ddepth, dx, dy, ksize, scale, delta, borderType)

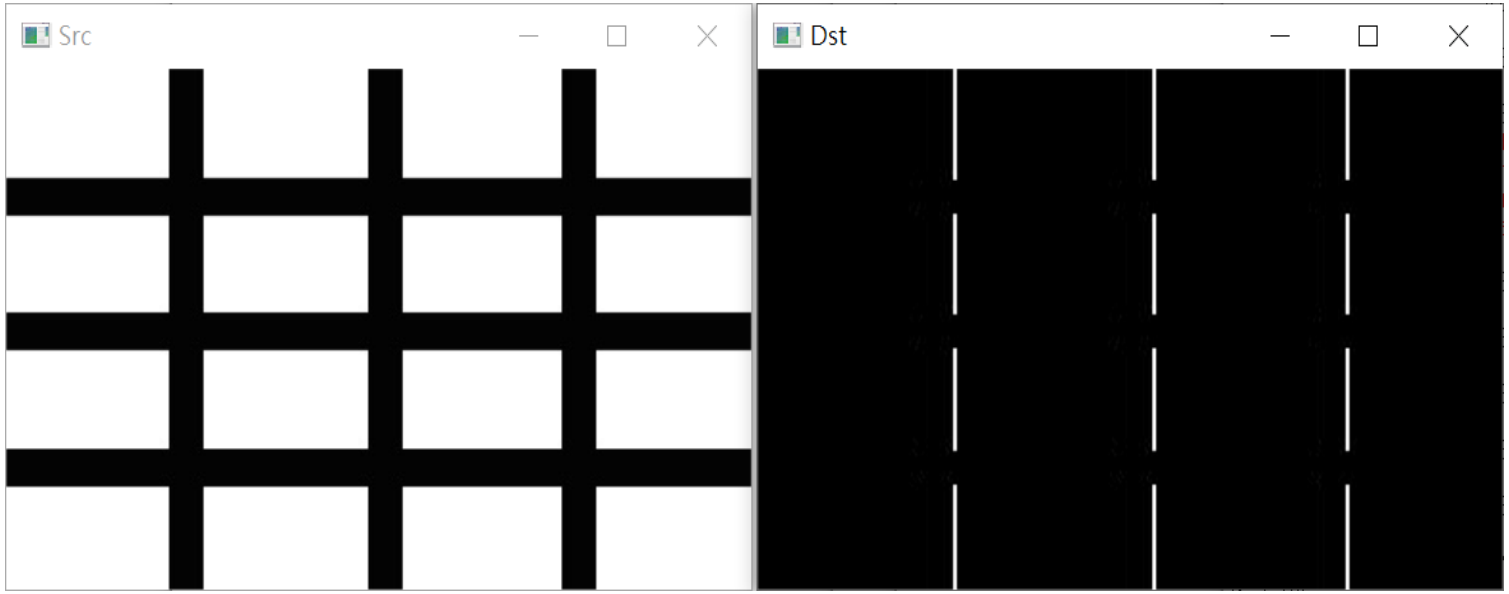
13.2.5：考量ddepth與取絕對值函數 convertScaleAbs( )

- dst = cv2.convertScaleAbs(src, alpha, beta)
- 程式實例ch13_1.py：使用convertScaleAbs( )函數將一個含負數的矩陣，全部轉為正值。

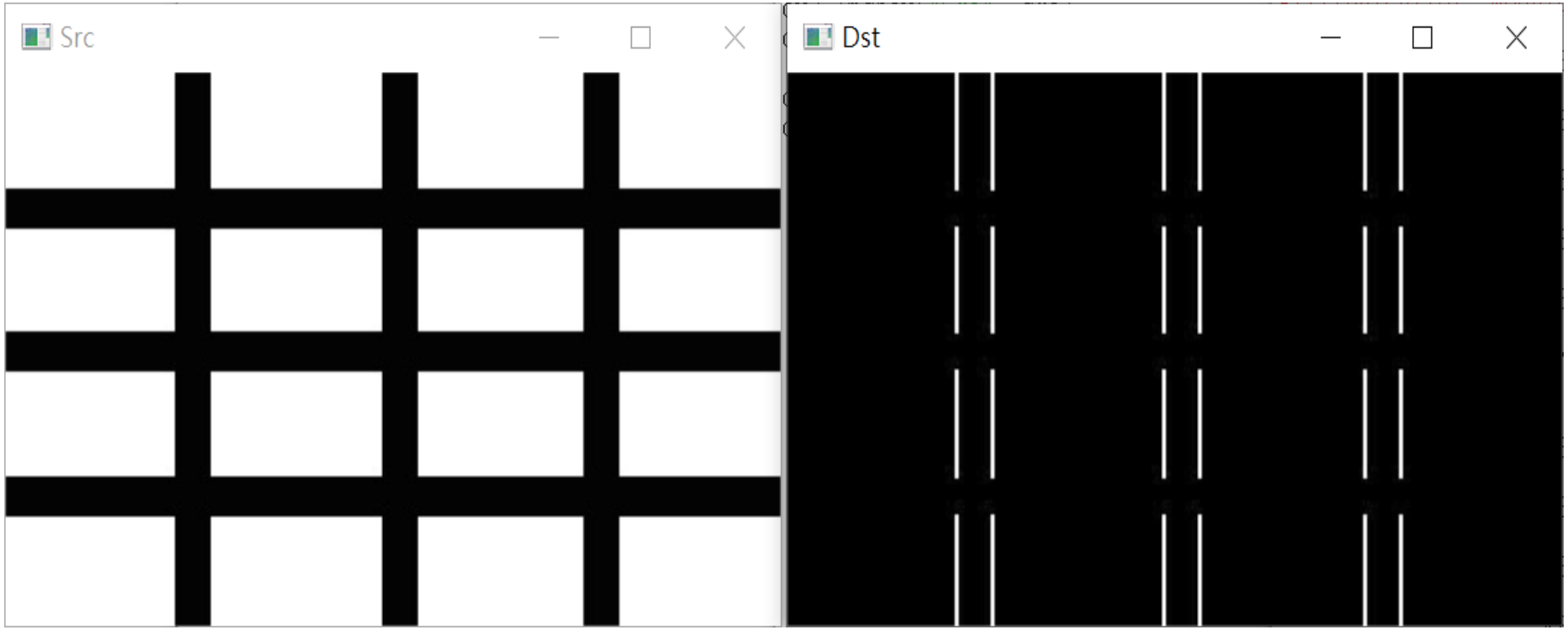
```
================== RESTART: D:/OpenCV_Python/ch13/ch13_1.py ==================
src =
 [[ -53 -175  -96  130 -100]
 [  48    5 -223   33   57]
 [  17 -244  137  -51 -142]]
dst =
 [[ 53 175  96 130 100]
 [ 48   5 223  33  57]
 [ 17 244 137  51 142]]
```

13.2.6：x軸方向的影像梯度

- dst = cv2.Sobel(src, ddepth, 1, 0)　　# 設定1階導數，x軸的影像梯度
- 程式實例ch13_2.py：設定ddepth = -1，繪製x軸方向的影像梯度。

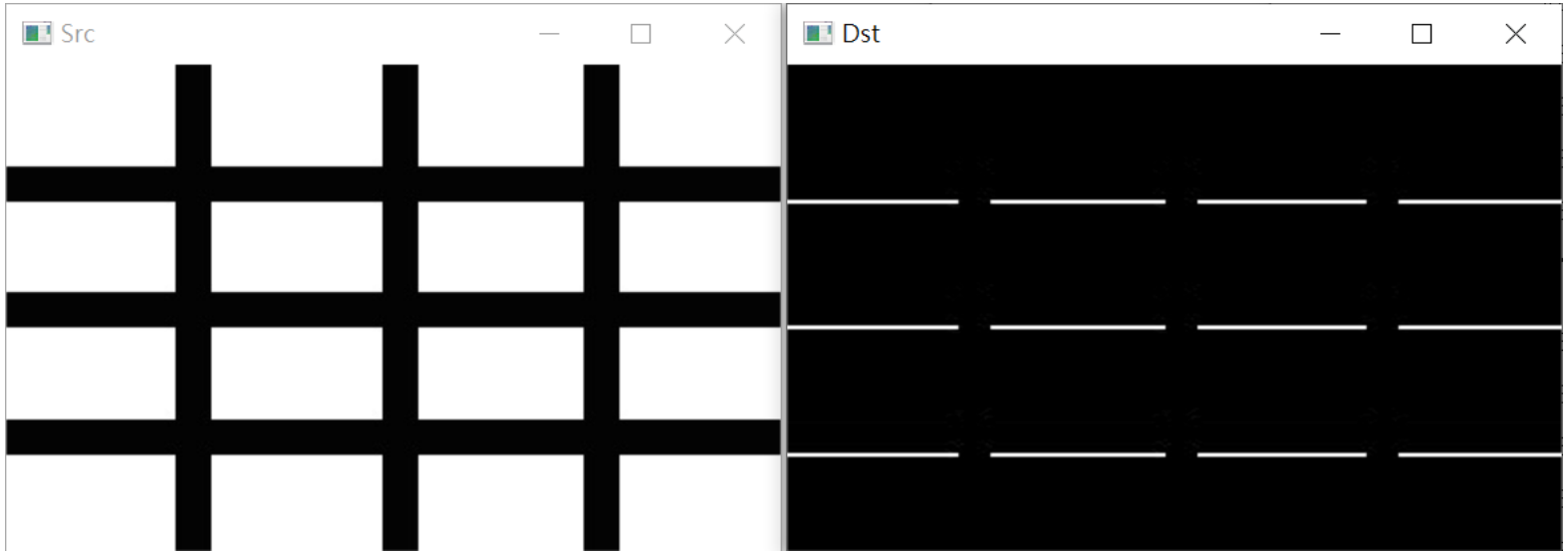


- 程式實例ch13_3.py：使用convertScaleAbs( )函數將負值的梯度改為正值，重新設計ch13_2.py。

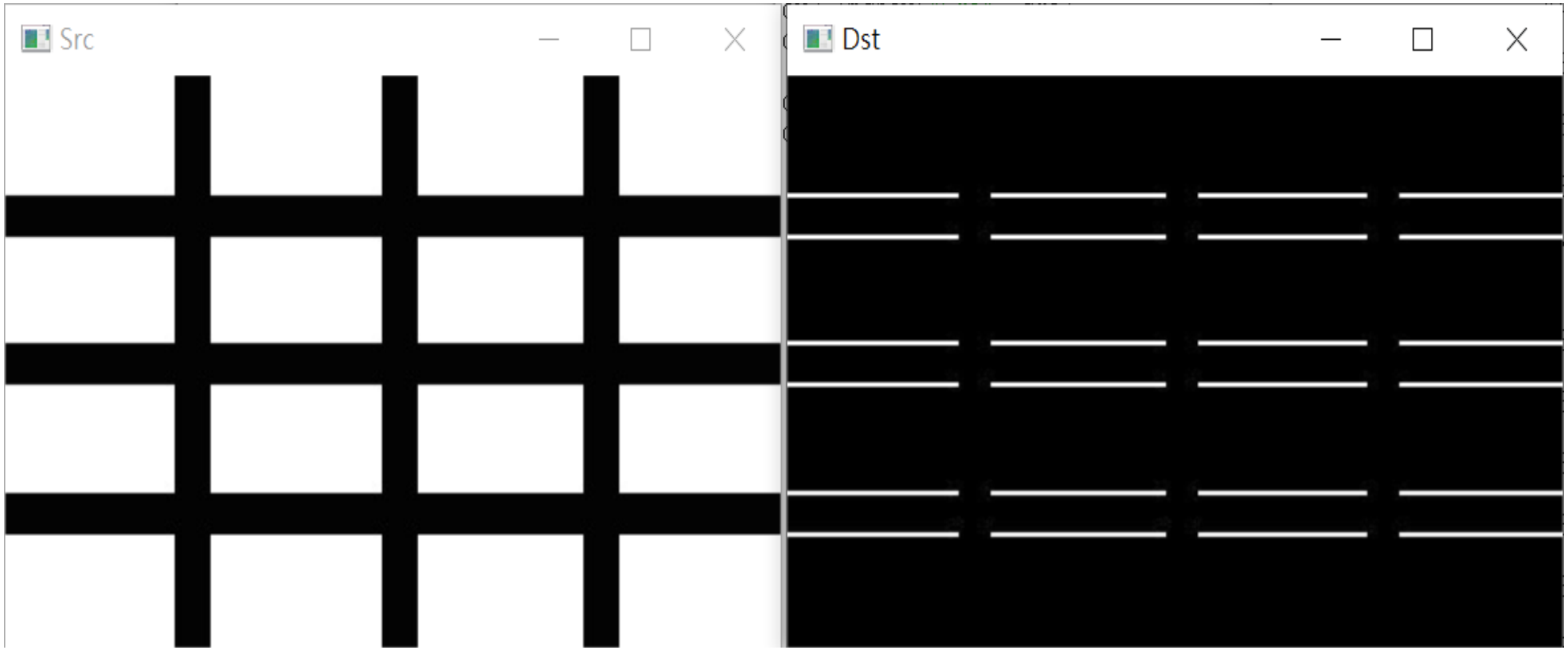


13.2.7：y軸方向的影像梯度

- dst = cv2.Sobel(src, ddepth, 0, 1)　　# 設定1階導數，y軸的影像梯度
- 程式實例ch13_4.py：設定ddepth = -1，繪製y軸方向的影像梯度。

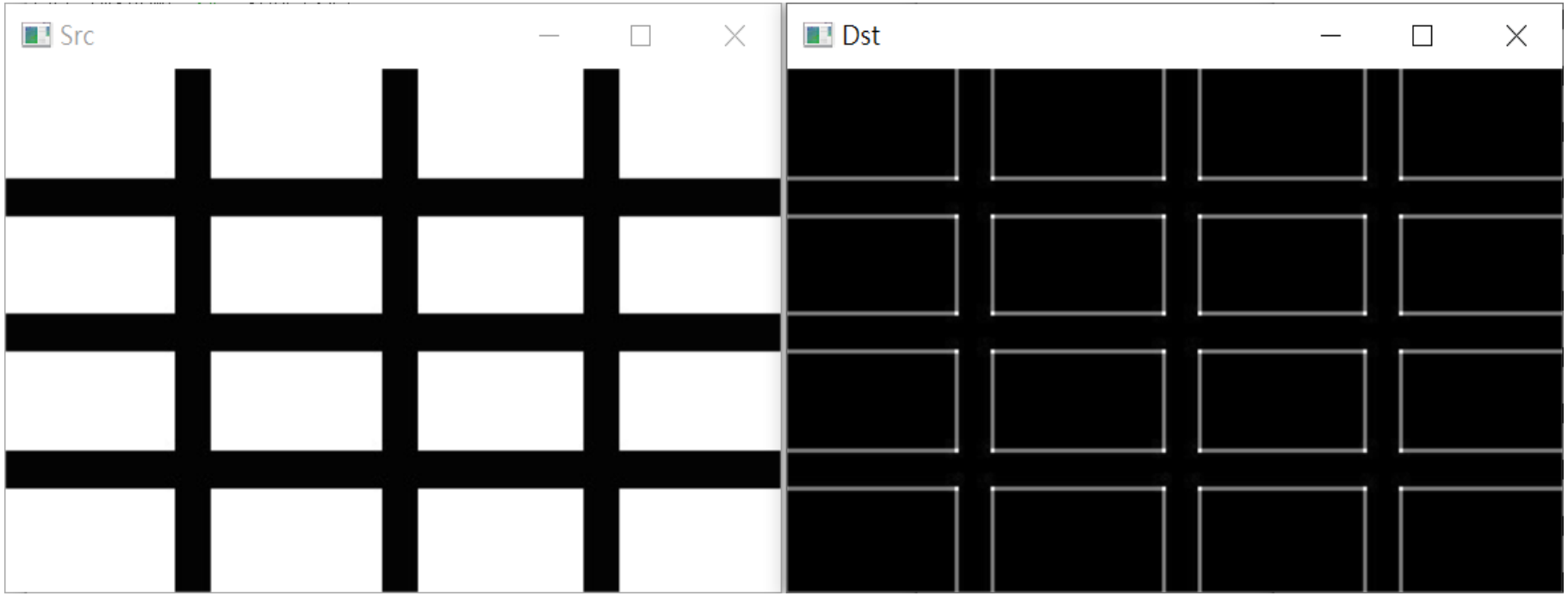


- 程式實例ch13_5.py：使用convertScaleAbs( )函數將負值的梯度改為正值，重新設計ch13_4.py。

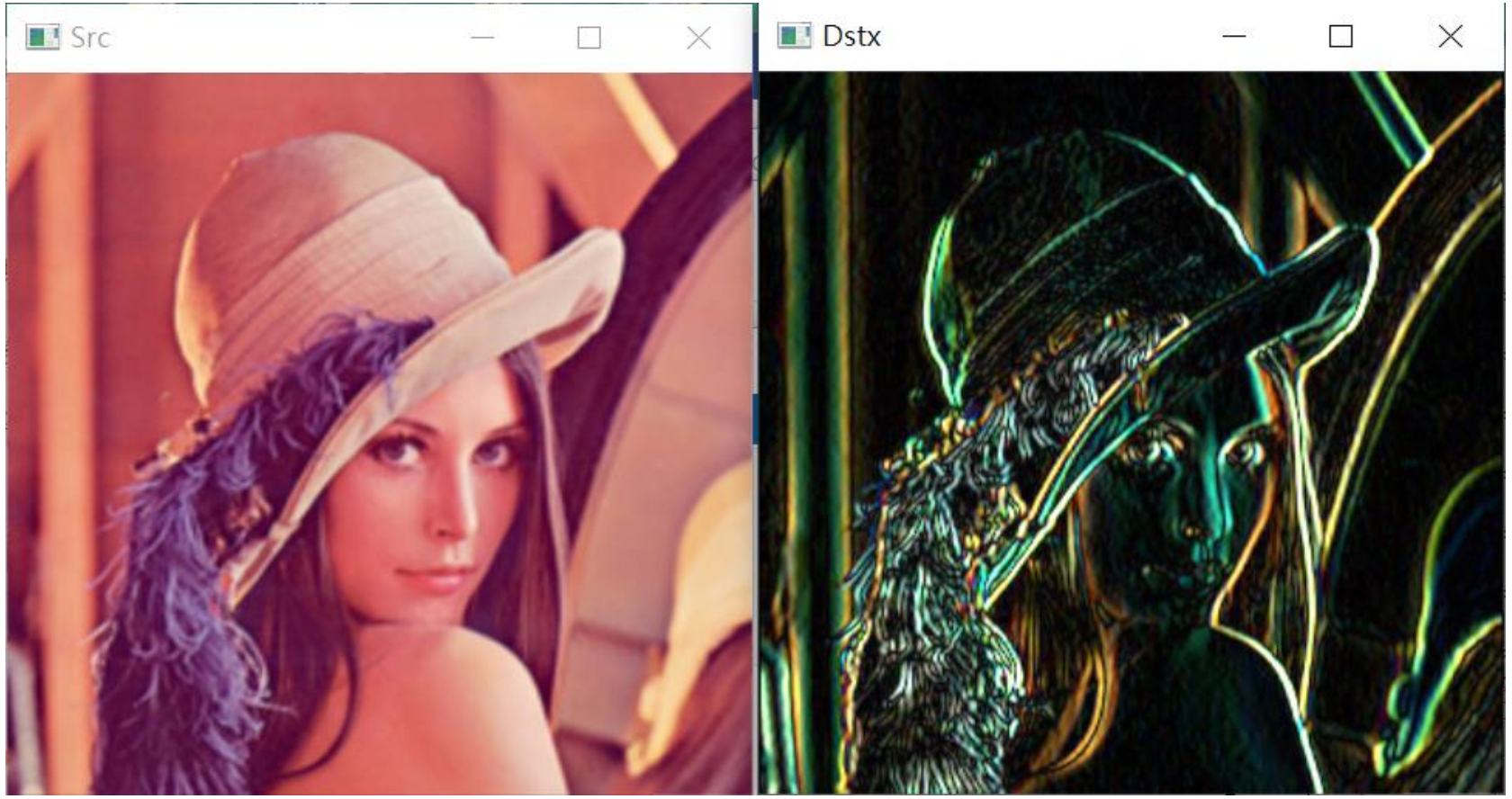


13.2.8：x軸和y軸影像梯度的融合

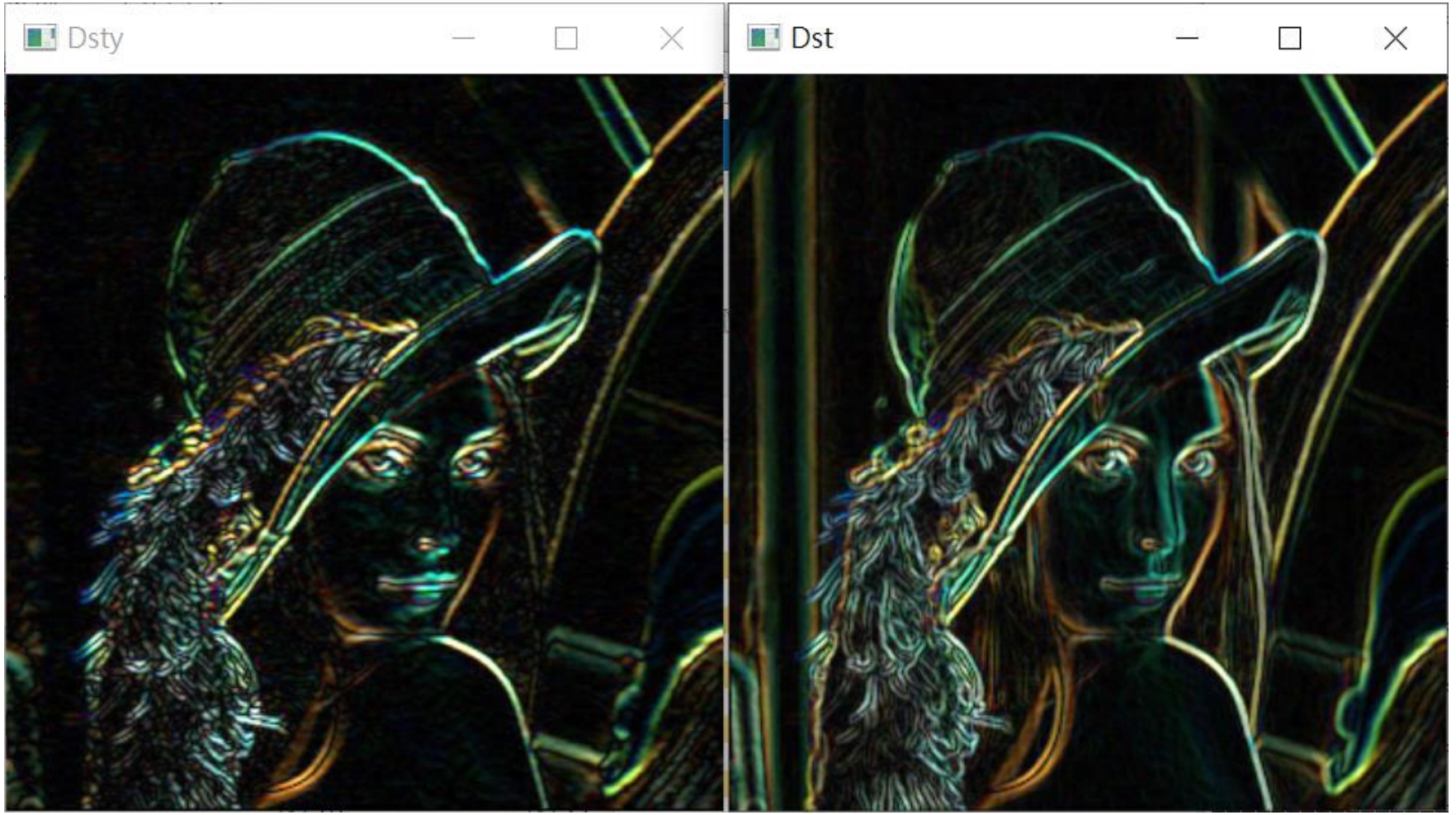
- 程式實例ch13_6.py：將ch13_3.py與ch13_5.py的影像融合。



- 程式實例ch13_7.py：將Sobel( )函數應用到實際影像，繪製原始影像、x軸影像梯度、y軸影像梯度、融合x軸和y軸的影像梯度。



- 下列是y軸梯度影像，x軸和y軸梯度融合的影像。



13.3：OpenCV函數Scharr( )

- 13.3.1：Scharr算子

$$\begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

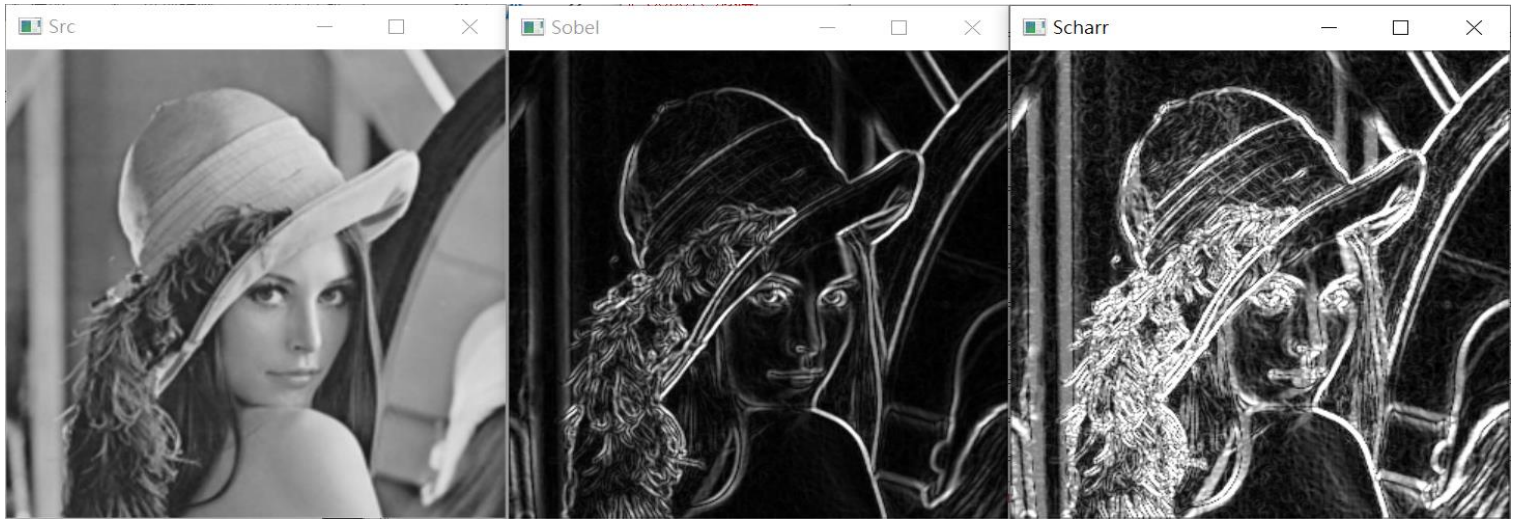
用於 x 軸算子

$$\begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

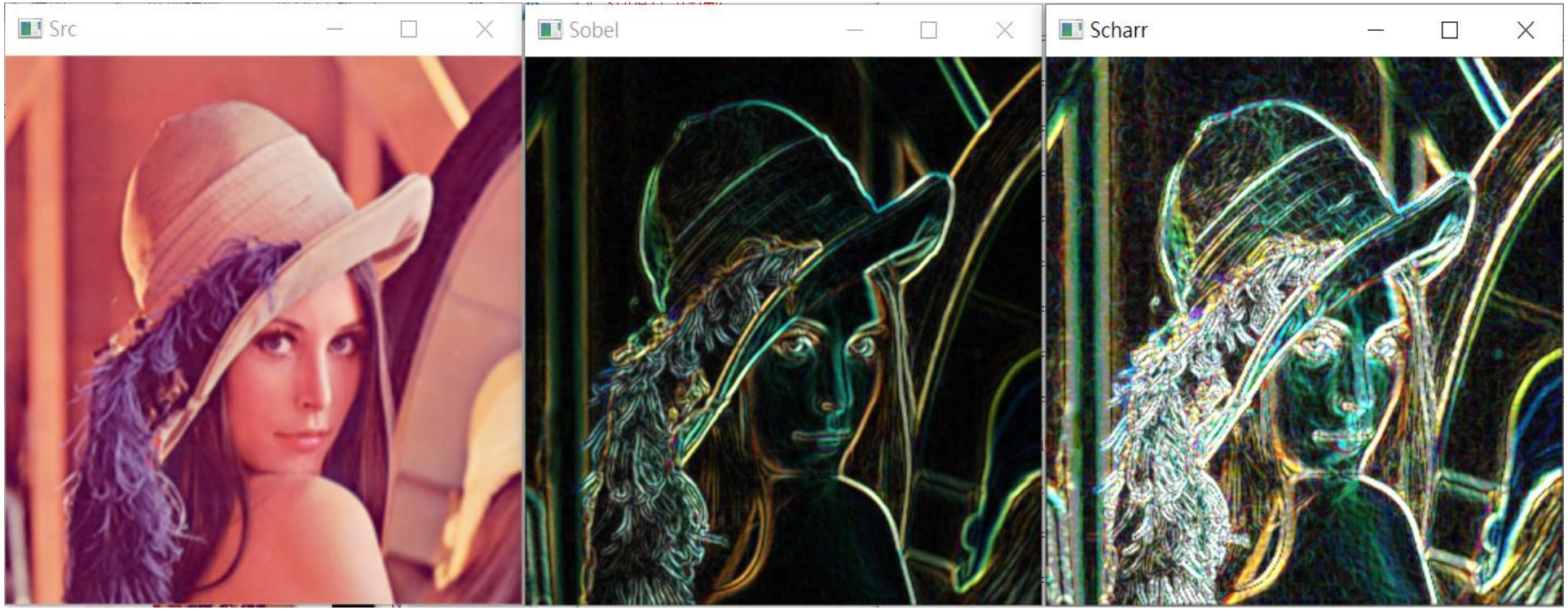
用於 y 軸算子

13.3.2：Scharr( )函數

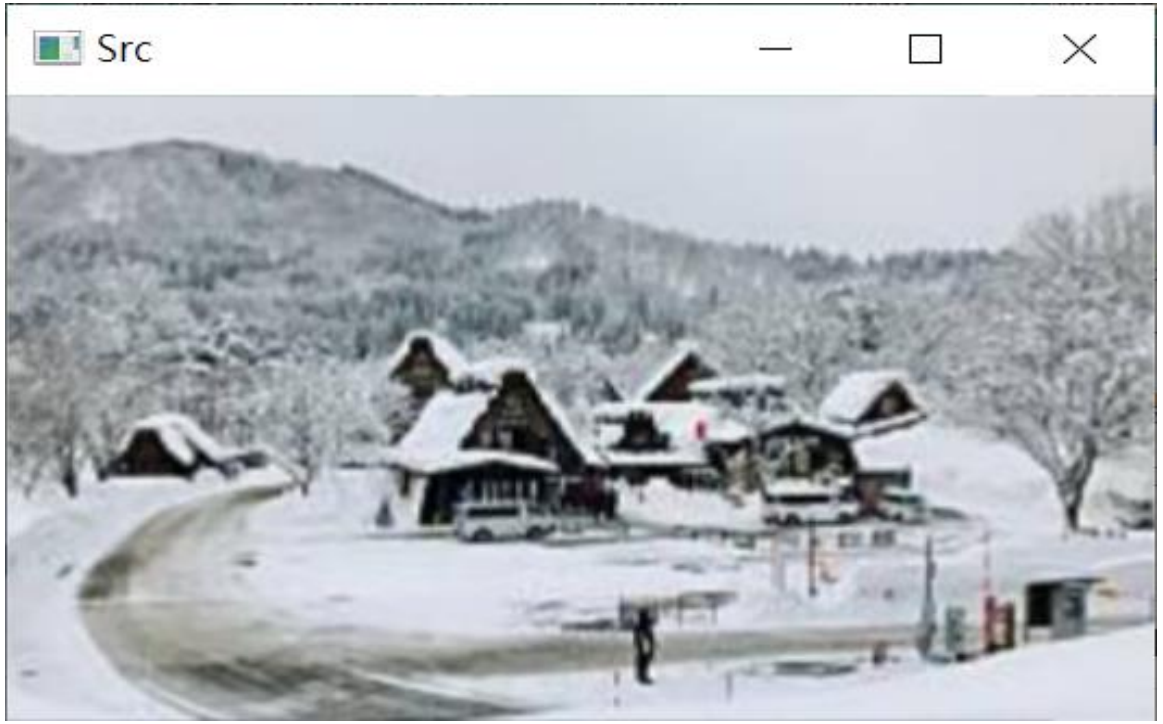
- dst = cv2.Scharr(src, ddepth, dx, dy, ksize, scale, delta, borderType)
- 程式實例ch13_8.py：使用灰階讀取lena.jpg影像，然後更改設計ch13_7.py，最後將Sobel( )和Scharr( )函數所獲得的影像邊緣做比較。



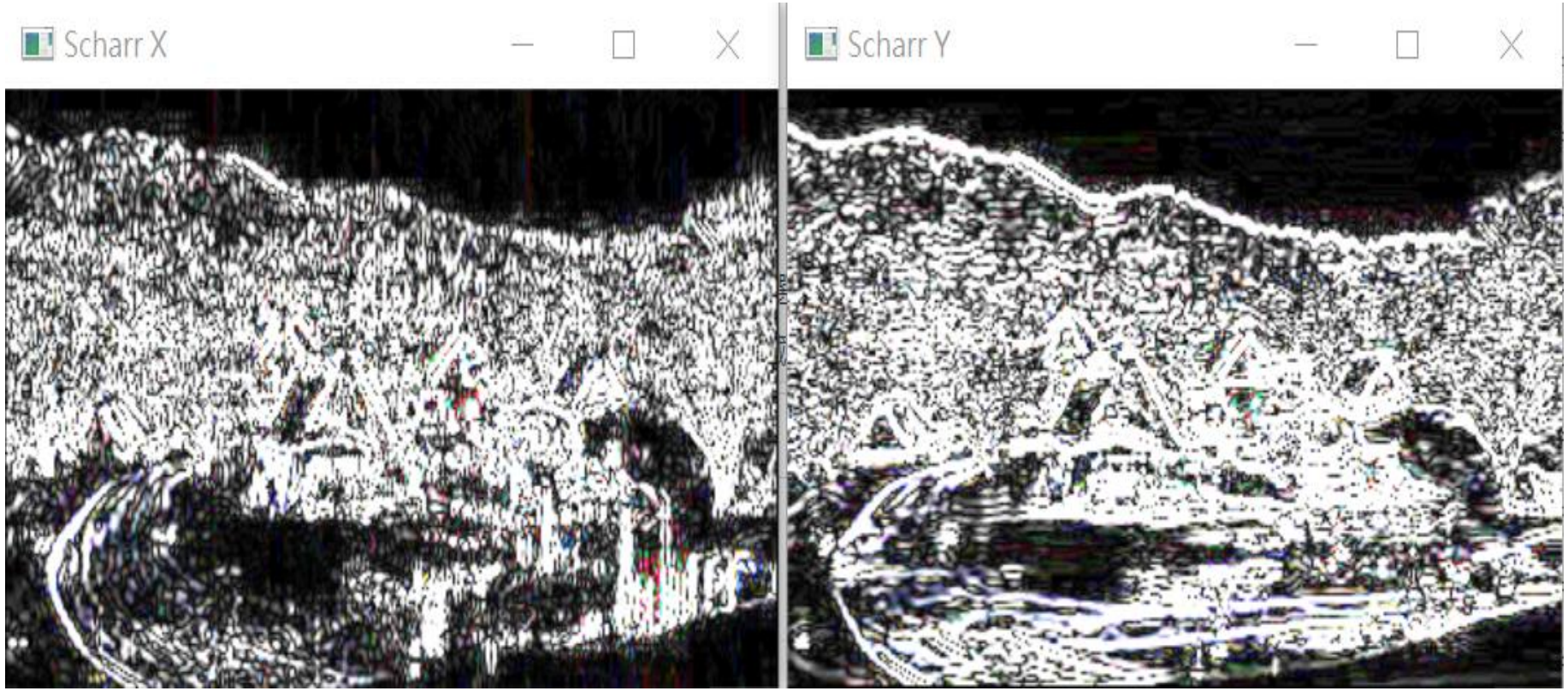
- 程式實例ch13_8_1.py：更改ch13_8.py，使用彩色讀取，讀者可以比較執行結果。



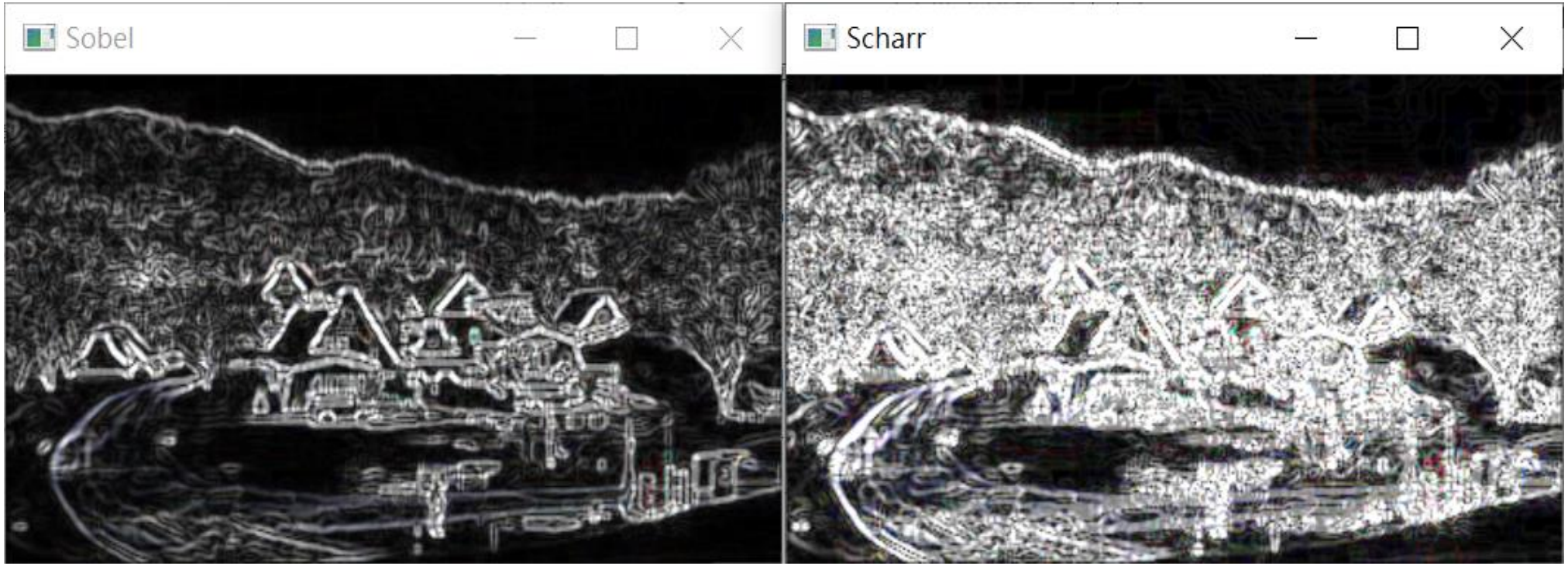
- 程式實例ch13_9.py：原始影像是snow.jpg，請使用Scharr( )函數建立此影像的x軸和y軸影像梯度，同時使用Sobel( )函數和Scharr( )函數建立此完整影像梯度，最後比較結果。



- 下列是Scharr( )函數建立的x軸和y軸影像梯度。



- 下列左邊是Sobel( )函數建立的邊緣影像，右邊是Scharr( )函數建立的邊緣影像。



13.4：OpenCV函數Laplacian( )

- 13.4.1：二階微分

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$

- 13.4.2：Laplacian算子

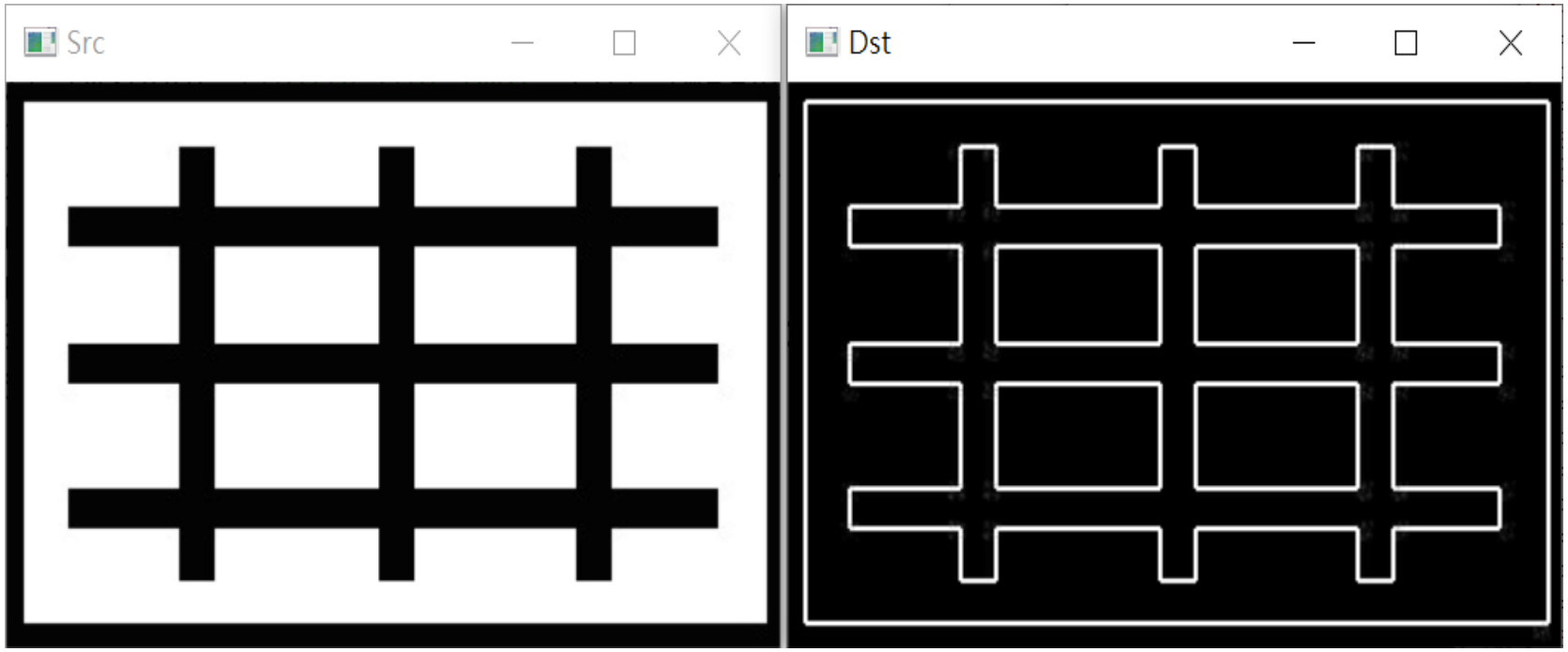
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) - f(x,y-1) - 4f(x,y)$$

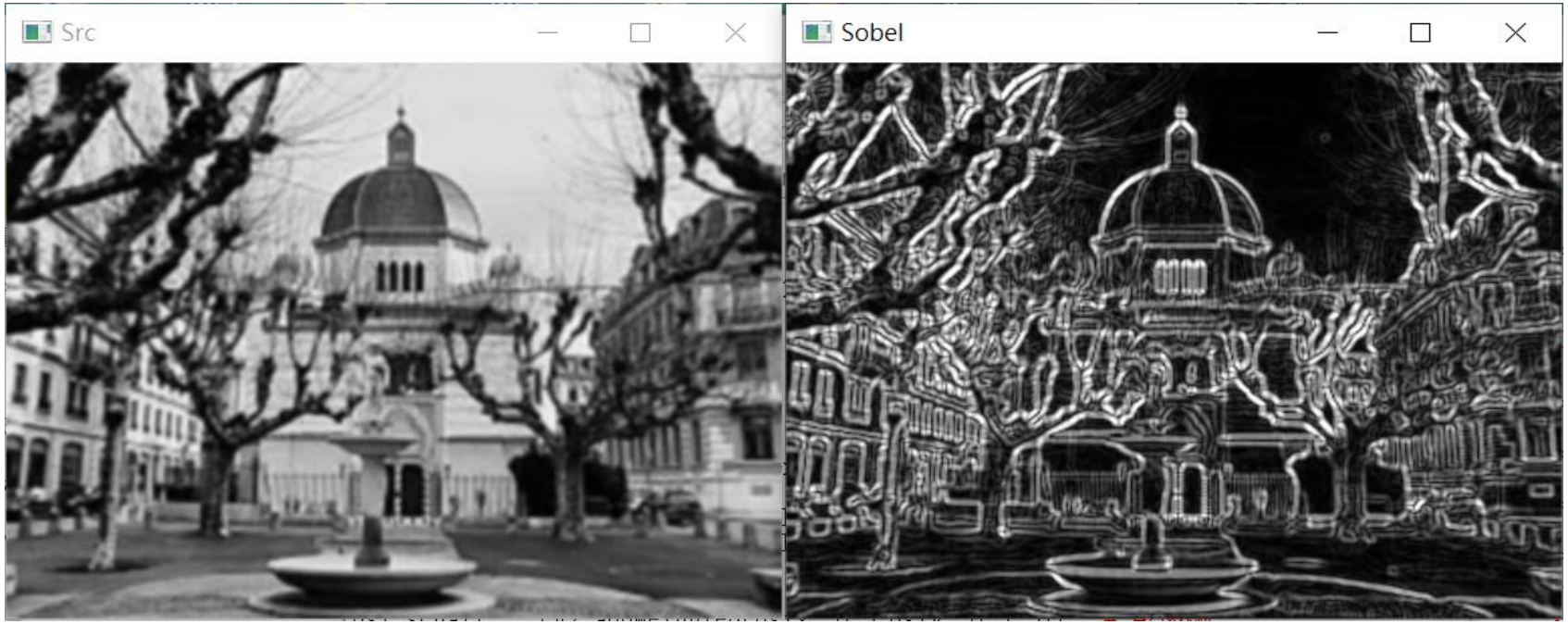
$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

13-4-3：Laplacian( )函數

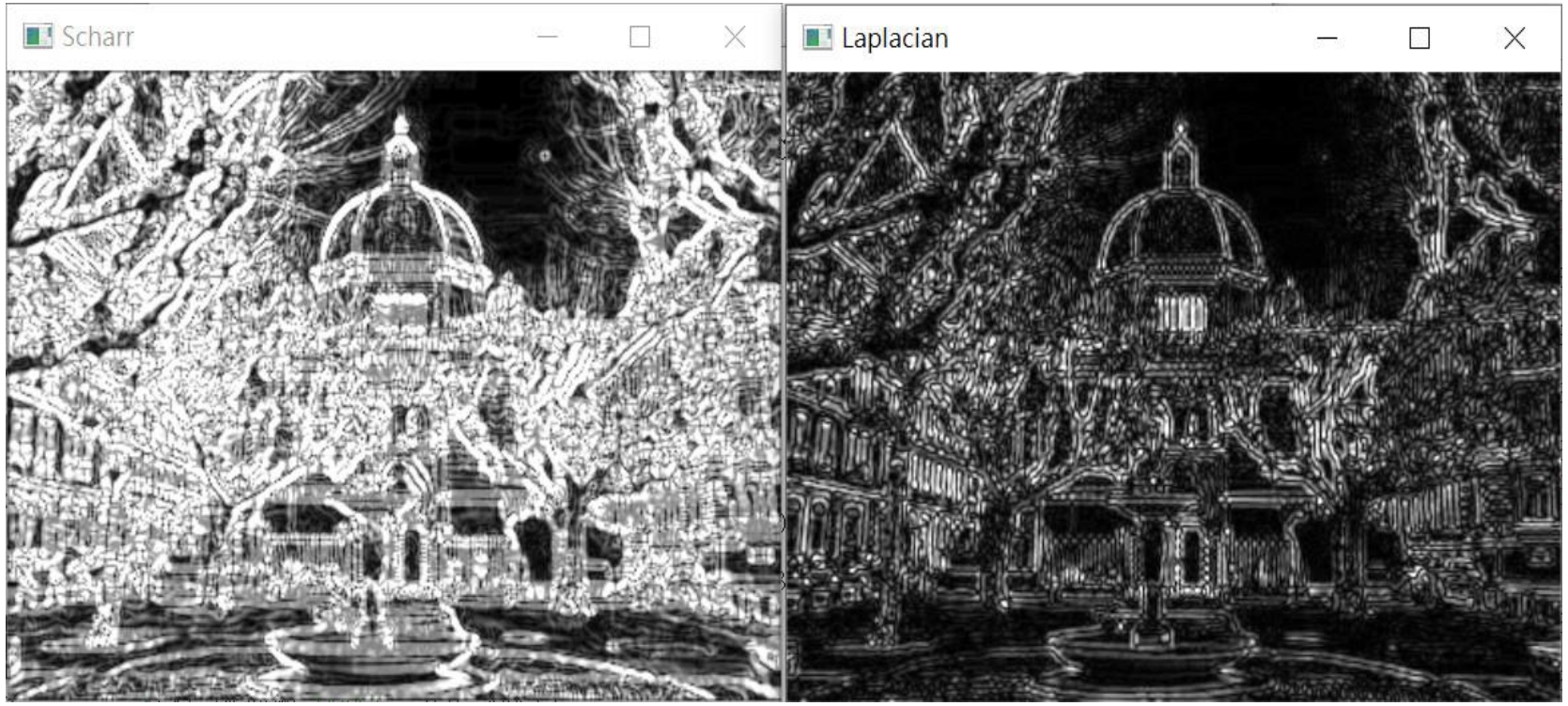
- dst = cv2.Laplacian(src, ddepth, ksize, scale, delta, borderType)
- 程式實例ch13_10.py：使用Laplacian( )函數偵測影像邊緣的應用。



- 程式實例ch13_11.py：瑞士日內瓦建築物的邊緣偵測，這個程式會分別列出原始影像、Sobel( )、Scharr( )和Laplacian( )函數的執行結果。
- 下列是原始影像和Sobel( )函數的執行結果。



- 下列是Scharr( )和Laplacian( )函數的執行結果。



13.5：Canny邊緣檢測

- **13.5.1： 認識Canny邊緣檢測**
- Canny的目標是找到一個最好的邊緣檢測演算法，這個方法必須有下列要件。

❑好的檢測：演算法盡可能多地標出影像的實際邊緣。

❑好的定位：標出的邊緣要與實際影像相符。

❑最小響應：影像的邊緣只標出一次，同時雜訊不被標示為邊緣。

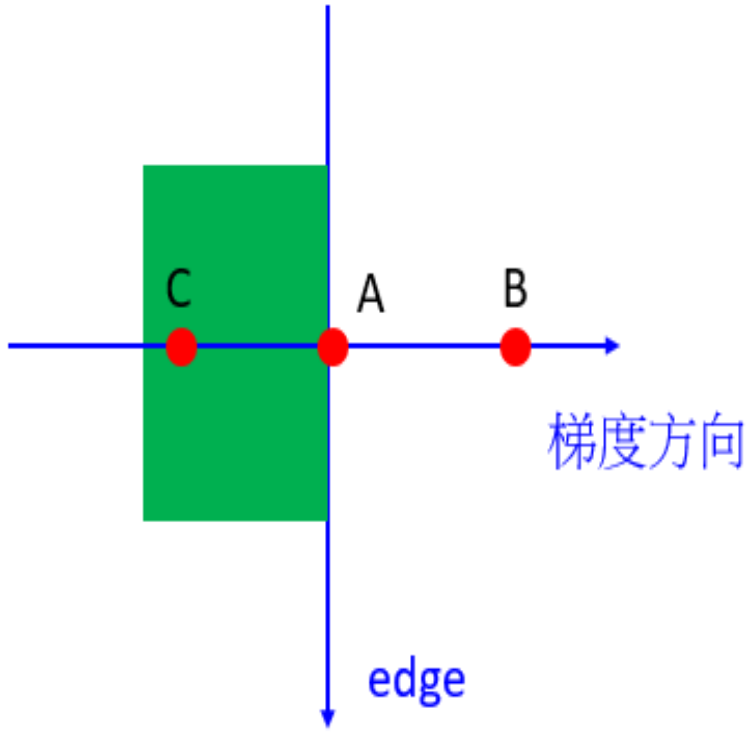
13.5.2：Canny演算法的步驟

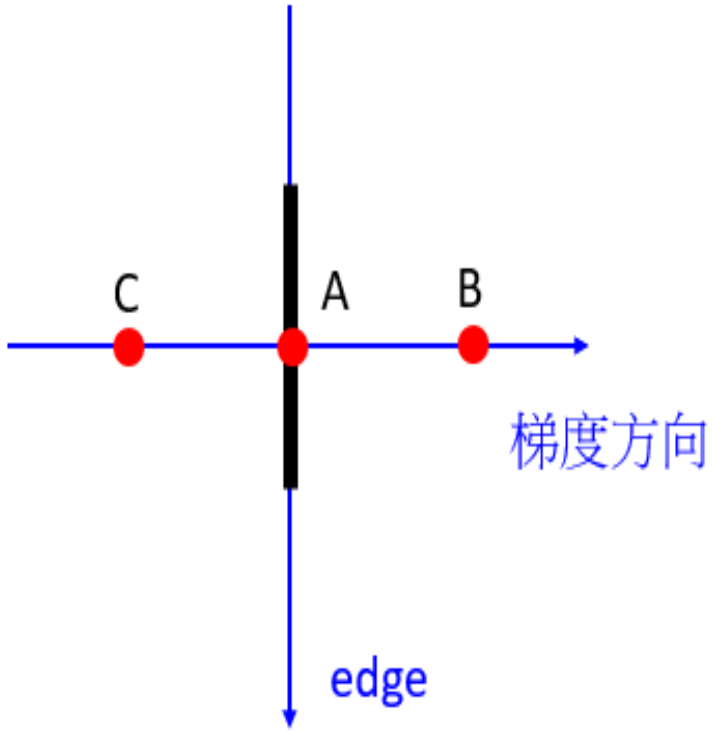
- 在Canny演算法的步驟分別如下：
- 1：降低噪音(Noise Reduction)
- 2：找尋影像的亮度梯度(Find Intensity Gradient of the Image)

$$Edge\_Gradient(G) = \sqrt{G_x^2 + G_y^2}$$

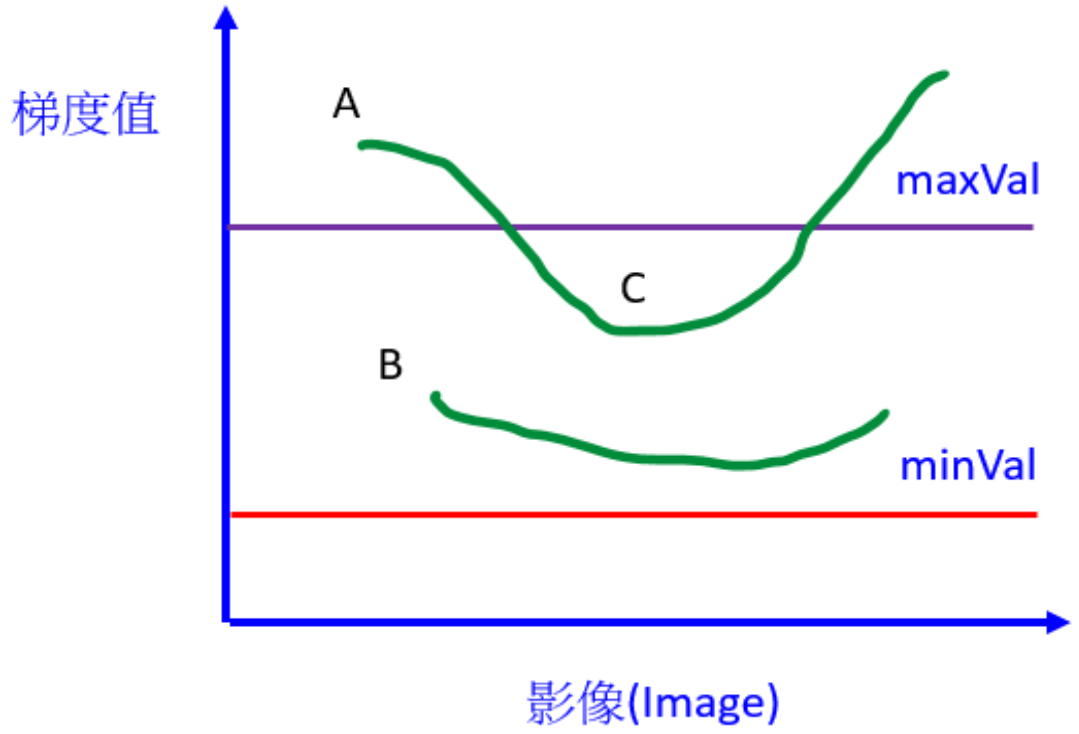
$$Angle(\theta) = tan^{-1}\left(\frac{G_y}{G_x}\right)$$

- 3：非最大值則抑制(Non-maximum Suppression)





- 4：滯後閾值(Hysteresls Thresholding)



- 決定像素是否為邊緣的方式如下：

❑ 如果像素點梯度值大於maxVal一定是邊緣。

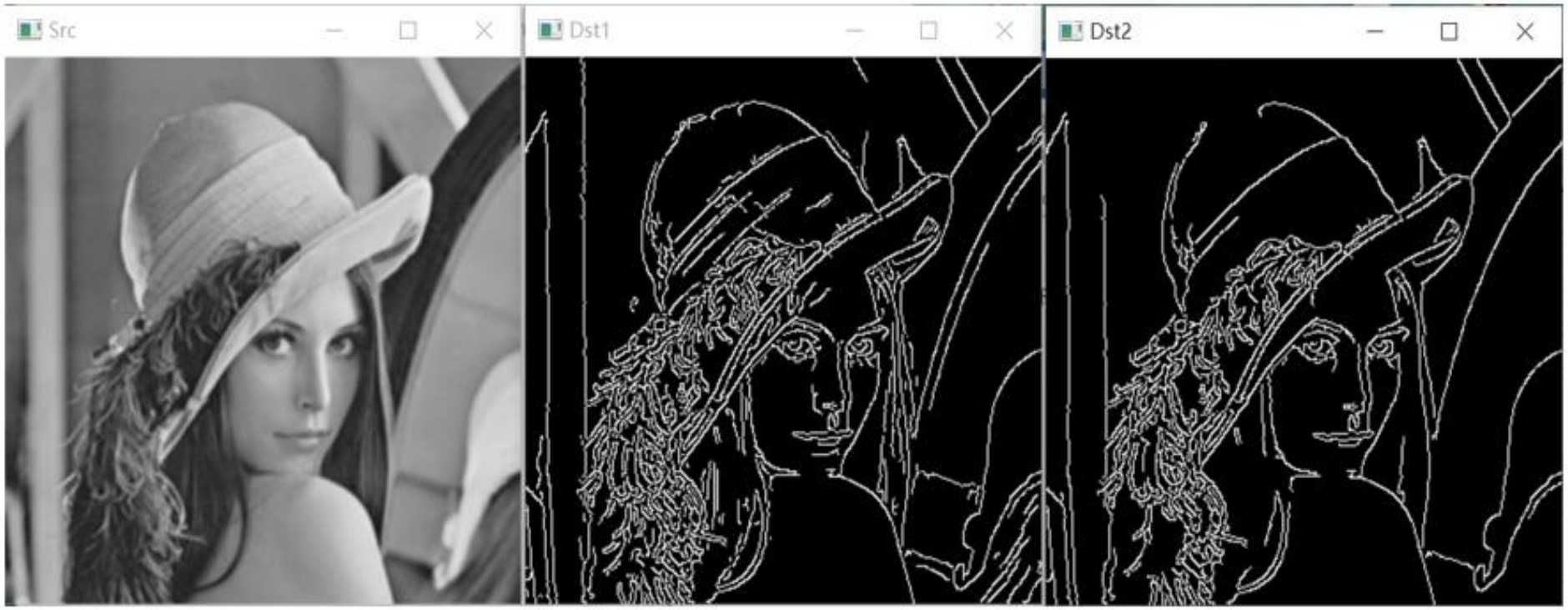
❑ 如果像素點梯度值小於minVal一定不是邊緣，可以拋棄。

❑ 像素點梯度值介於maxVal和minVal之間，如果他們連接到確定邊緣像素，則被認為是邊緣，否則拋棄。

13.5.3：Canny( )函數

- dst = cv2.Laplacian(image, edges, threshold1, threshold2, apertureSize = 3,
- L2gradient = False)

- 程式實例ch13_12.py：使用minVal = 50和maxVal分別是100, 200，偵測lena.jpg的影像邊緣。



原始影像　　minVal=50, maxVal=100　　minVal=50, maxVal=200

- 程式實例ch13_13.py：重新設計ch13_11.py，增加使用Canny檢測方法，最後列出4種方法的結果並做比較。
- 下列左邊是Canny( )和右邊是Sobel( )函數的執行結果。

- 下列是Scharr( )和Laplacian( )函數的執行結果。