

Python影像處理：邊界檢測 與影像判讀

矩陣運算

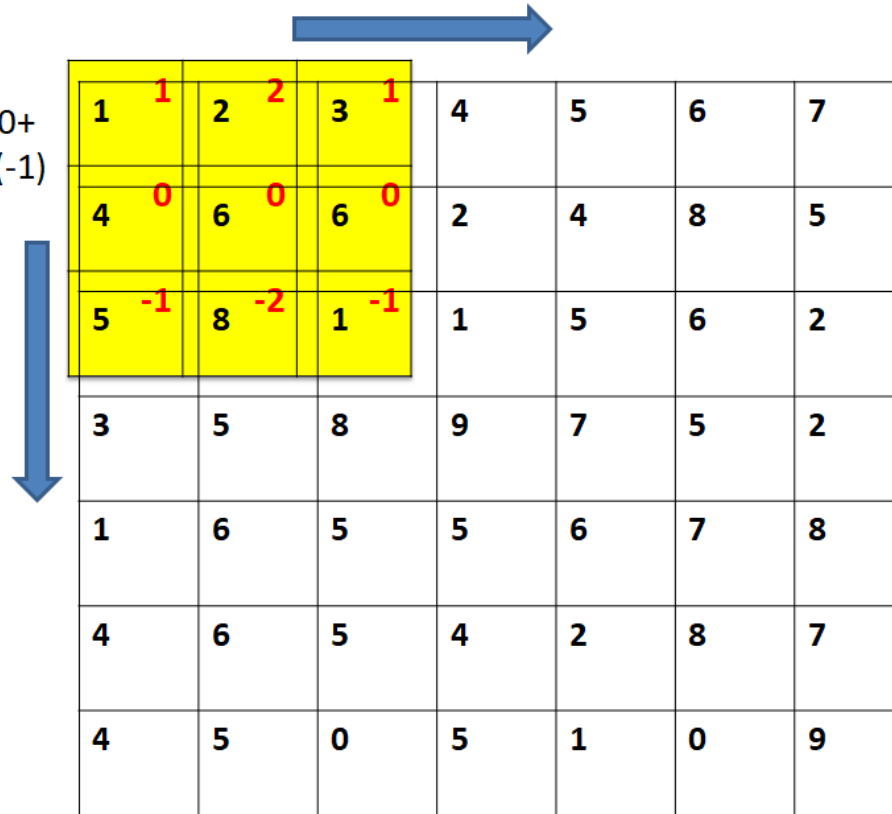
$$\begin{aligned} \mathbf{AB} &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix} \end{aligned}$$

一个具体例子:

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} -3 & 0 \\ 5 & 1/2 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} -7 & 2 \\ 4 & 6 \end{bmatrix} \\ \mathbf{AB} &= \begin{bmatrix} -3 & 0 \\ 5 & 1/2 \end{bmatrix} \begin{bmatrix} -7 & 2 \\ 4 & 6 \end{bmatrix} \\ &= \begin{bmatrix} (-3)(-7) + (0)(4) & (-3)(2) + (0)(6) \\ (5)(-7) + (1/2)(4) & (5)(2) + (1/2)(6) \end{bmatrix} \\ &= \begin{bmatrix} 21 & -6 \\ -33 & 13 \end{bmatrix} \end{aligned}$$

影像處理--二維卷積

$$1 \times 1 + 2 \times 2 + 3 \times 1 + 4 \times 0 + 6 \times 0 + 6 \times 0 + 5 \times (-1) + 8 \times (-2) + 1 \times (-1) = -14$$



1	2	3	4	5	6	7
4	6	6	2	4	8	5
5	8	1	1	5	6	2
3	5	8	9	7	5	2
1	6	5	5	6	7	8
4	6	5	4	2	8	7
4	5	0	5	1	0	9

OpenCV—二維卷積

如同課堂上所說，二維卷積在影像處理上有許多不同的應用，在OpenCV中提供了各種二維卷積的相關函數。其中一項基本功能就是使用自定義的乘積函數來對目標影像進行二維卷積，寫法如下：

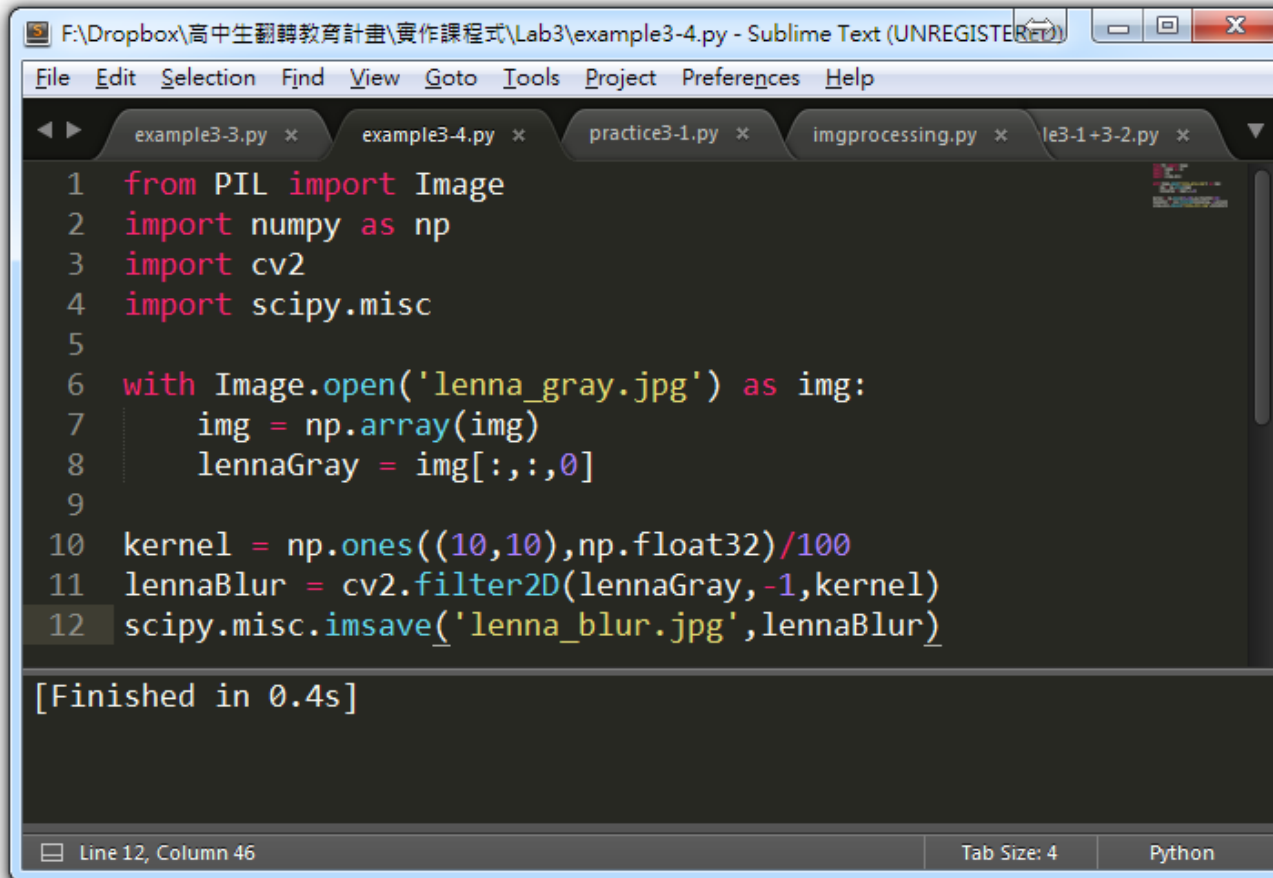
`Cv2.filter2D(src, ddepth, kernel)`

`src`: 欲卷積的影像。

`ddepth`: 輸出之影像格式，-1為與原始影像相同。

`kernel`: 自定義的乘積函數

範例四：套用OpenCV進行模糊



```
F:\Dropbox\高中生翻轉教育計畫\實作課程\Lab3\example3-4.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
example3-3.py x example3-4.py x practice3-1.py x imgprocessing.py x le3-1+3-2.py x
1 from PIL import Image
2 import numpy as np
3 import cv2
4 import scipy.misc
5
6 with Image.open('lenna_gray.jpg') as img:
7     img = np.array(img)
8     lennaGray = img[:, :, 0]
9
10 kernel = np.ones((10,10), np.float32)/100
11 lennaBlur = cv2.filter2D(lennaGray, -1, kernel)
12 scipy.misc.imsave('lenna_blur.jpg', lennaBlur)

[Finished in 0.4s]
Line 12, Column 46 Tab Size: 4 Python
```

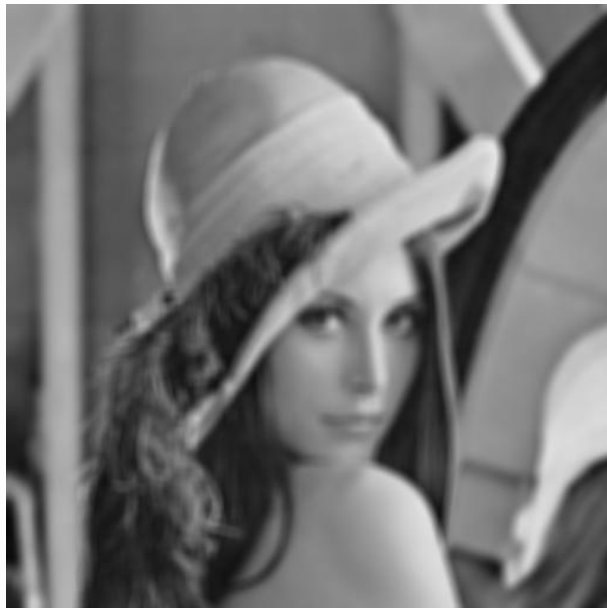
Line 10: 建立一個10x10，值皆為0.01的kernel。預設的檔案格式為整數，注意需加上np.float32。

Line 11: 進行二維卷積，輸出的影像格式和原圖一樣所以填入-1。

模糊前/後比較



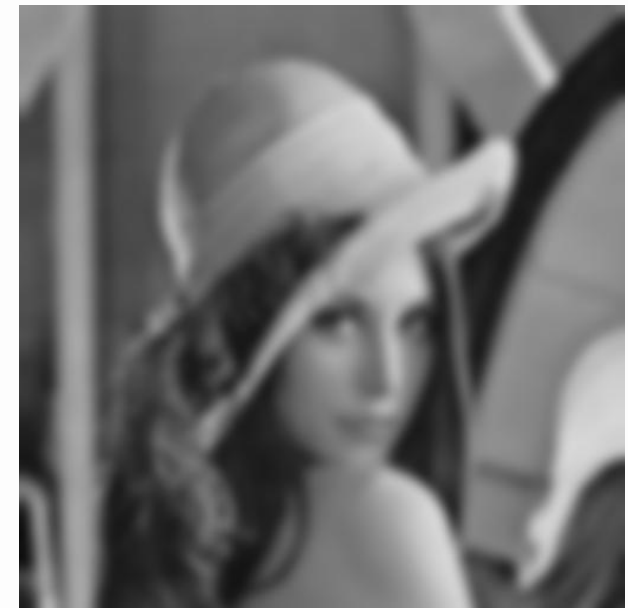
Lenna



Lenna_blur1



Lenna_blur2



Lenna_blur3

重複進行模糊四次

小練習~將矩陣改寫成 50×50



Lenna



Lenna_blur4

邊界檢測—Laplacian operator

Laplacian operator源自數學上的Laplace operator，它可以用來計算一個空間中的場的梯度。簡而言之，一個場(比如等高線圖、位能、或是這邊要處理的灰階影像)在經過Laplace運算以後，變化大的區域會得到較大的值，平緩區域則會得到接近零的值。

在影像中變化大的地方就是圖形的邊界，所以Laplacian operator可以拿來做邊界檢測。以影像處理的應用來說，Laplacian operator常被近似為下列兩種形式：

$$L_1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$L_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

邊界檢測—Sobel operator

除了Laplacian operator之外，另一個常用的邊界檢測算法會用到Sobel operator。Sobel operator分為檢測X方向邊界與Y方向邊界兩種運算子，分別卷積完成後再將兩個方向的值平方相加開根號。這個步驟可以理解成類似計算直角三角形斜邊的概念，只是把兩股長改成兩個垂直方向的梯度。X和Y方向的Sobel operator如下：

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

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

練習一：邊界檢測

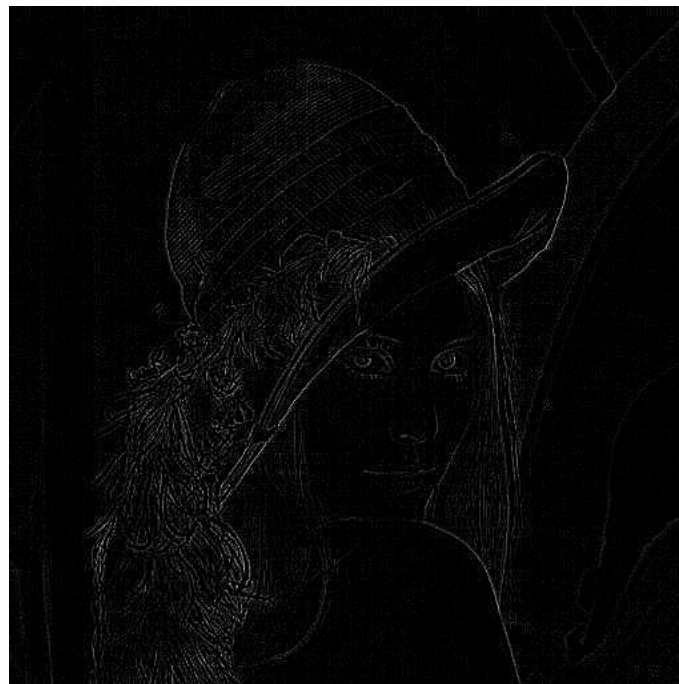
分別使用Laplacian operator和Sobel operator進行邊界檢測，輸出的檔案要有五個: 兩種Laplacian operator卷積出來的影像、Sobel operator的X方向與Y方向卷積出來的影像及兩方向合併後的影像。

提示: 使用`np.hypot(array1,array2)`將兩個array的個別數值進行平方相加開根號。

輸出影像--Laplacian operator



L₁



L₂

輸出影像--Sobel operator



S_x



S_y

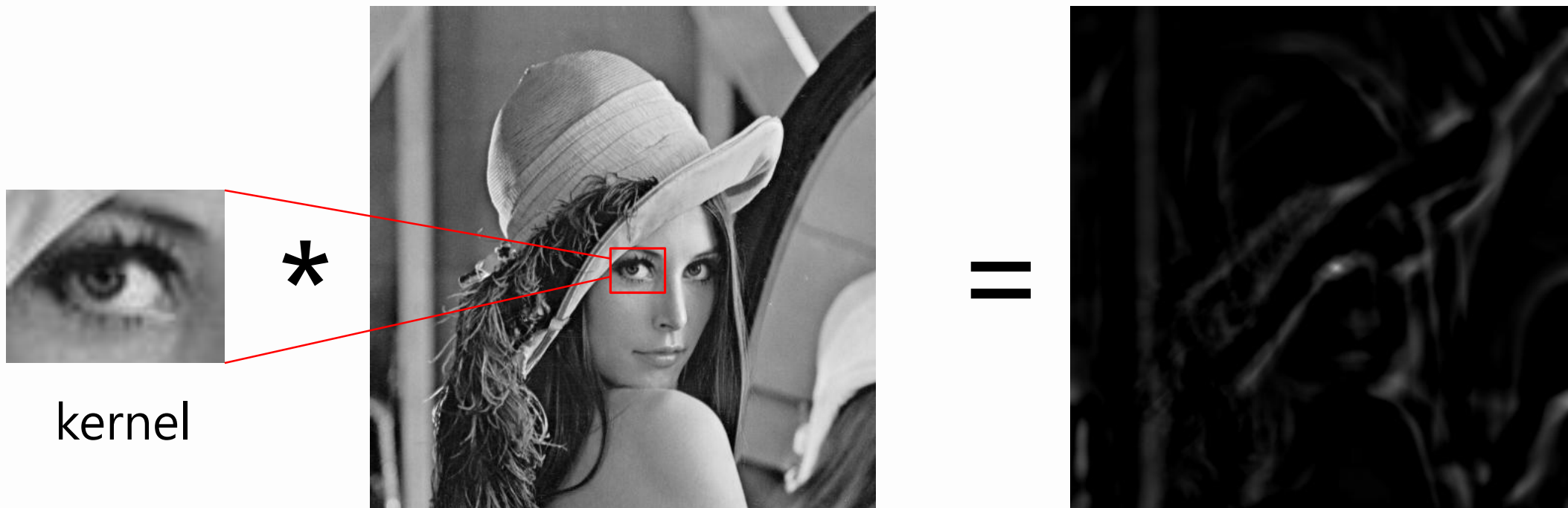


$S_x + S_y$

二維卷積的應用—尋找物件

若以特定物件的外觀特徵做成kernel進行二維卷積，則在卷積通過特徵相符的物體時，會得到相對較大的值。以前面用的圖片為例，簡單的做個小實驗：將原圖中人物的眼睛部分做為kernel來對原圖做二維卷積，來看看會得到什麼影像。

二維卷積的應用—尋找物件



試著執行find_eye.py

練習二：用二維卷積辨識條紋寬度

1. 將先前實驗拍下的rolling shutter effect影像轉成灰階影像。
2. 用小畫家簡單估計一下亮/暗條紋的平均寬度 w (像素數)。
3. 建立一個1與-1以 w 個相間，另一個維度大小為10的kernel，如下圖示意。
4. 將此kernel與影像進行二維卷積，觀察得到的結果；若任意調整1與-1的循環週期，會得到怎樣的結果？這個現象該如何解釋？

$$\begin{array}{c} \text{w個} \left\{ \begin{array}{l} \begin{array}{c} \text{10個} \\ \begin{bmatrix} 1 & 1 & \dots & \dots & 1 & 1 \\ 1 & 1 & \dots & \dots & 1 & 1 \\ \dots & & & & & \dots \\ 1 & 1 & \dots & \dots & 1 & 1 \\ -1 & -1 & \dots & \dots & -1 & -1 \\ -1 & -1 & \dots & \dots & -1 & -1 \\ \dots & & & & & \dots \\ -1 & -1 & \dots & \dots & -1 & -1 \end{bmatrix} \end{array} \end{array} \right. \end{array}$$

練習二：用二維卷積辨識條紋寬度

Hint:

利用for or while迴圈改變w值

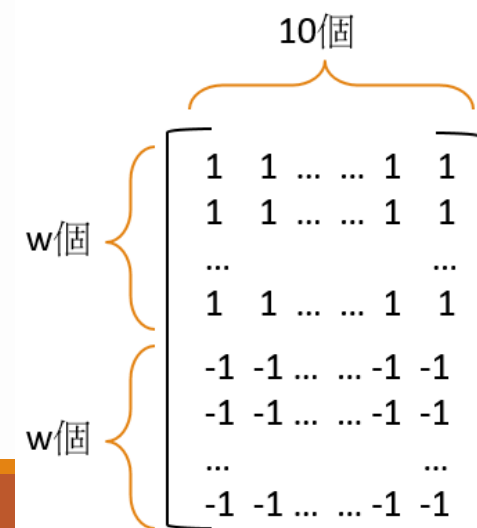
建構a*b都是一矩陣: `one=np.ones((a,b),np.float32)`

建構a*b都是負一矩陣 `minusone=one*-1`

矩陣組合 `lap=np.vstack((one,minusone))/test/10` (疊三層)

進行卷積

列印出答案



練習三：比較 w 與卷積結果的關係

承練習二，輸出一個圖表，橫軸為 w 的大小，縱軸為整個畫面中卷積結果的最大值，結果如下圖：

