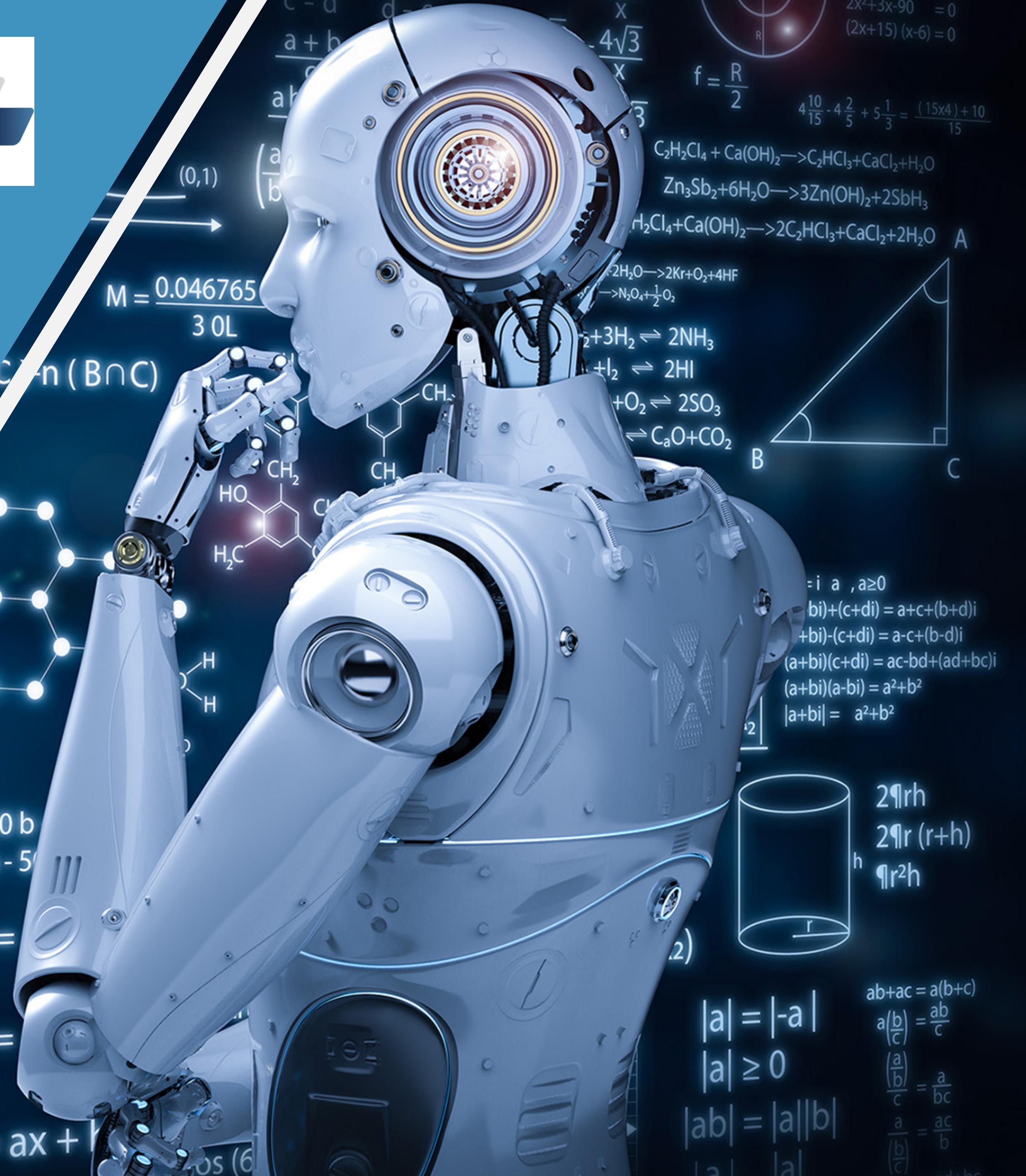




Day 08

深度學習與電腦視覺 學習馬拉松

Cupay 陪跑專家：楊鎮銘



基礎影像處理

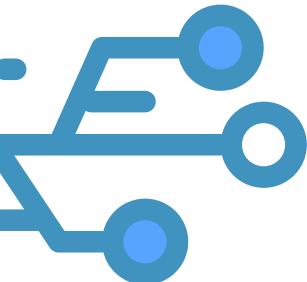
Filter 的概念與實作場景

(Sobel, Gaussian Blur)



重要知識點

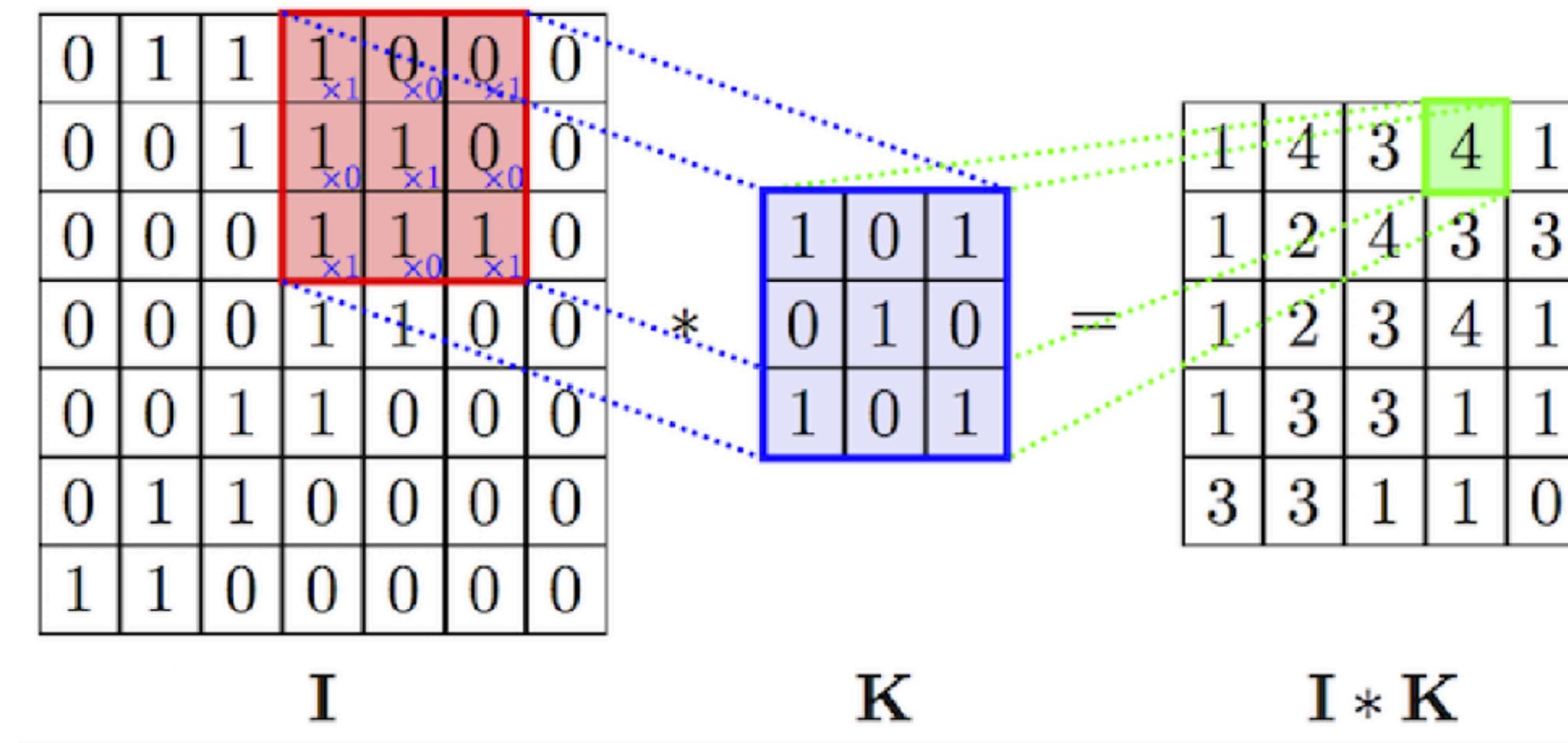
- 了解 Filter 的基本操作與其他影響最後結果的因素
- 了解模糊圖片跟邊緣偵測的概念與操作



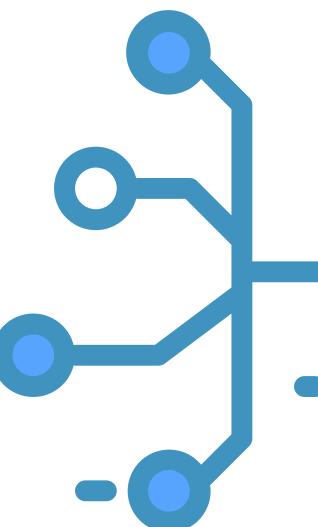
Filter 概念

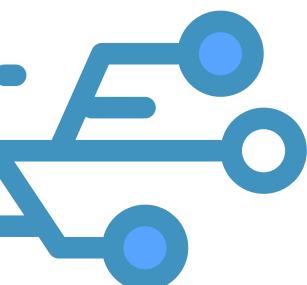


- Filter 又稱做 kernel，概念上是一個固定大小的矩陣，掃過圖片經過計算後取得新的圖片矩陣
- 假設以一個 3×3 的 filter 操作，每次運算都會把 9 個 pixel 的值相乘相加得到一個值，可以把這個操作想像為特徵提取



圖片來源：semanticscholar.org





Filter 操作 - Cross-Correlation vs Convolution



把 Filter 掃過圖片時我們要做兩個矩陣的運算

目前根據計算順序的不同主要分為 Cross-Correlation 跟 Convolution

a	b	c
d	e	f
g	h	i

f

0	0	0	0	0
0	A	B	C	0
0	D	E	F	0
0	G	H	I	0
0	0	0	0	0

g

Cross-correlation

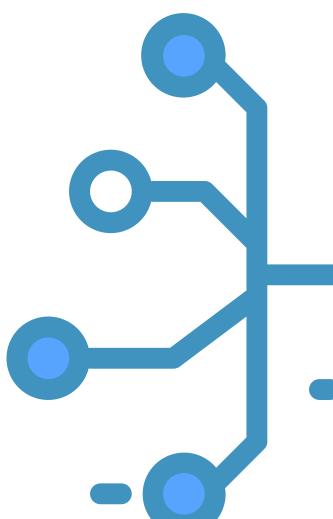
$$f \otimes g = a^*A + b^*B + c^*C + d^*D + e^*E + \dots$$

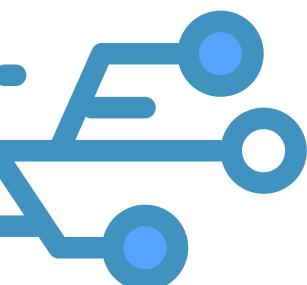
由左而右, 由上而下 looping

Convolution

$$f * g = a^*I + b^*H + c^*G + d^*F + e^*E + \dots$$

由右而左, 由下而上 looping



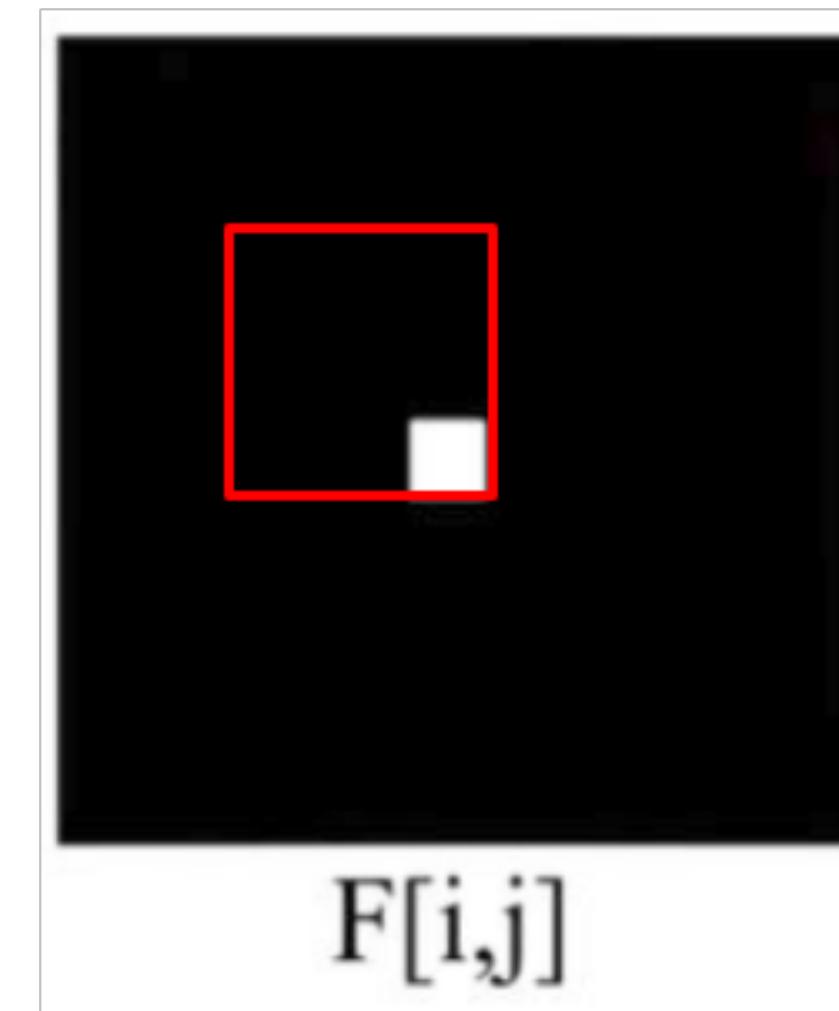


Filter 操作 - Cross-Correlation vs Convolution



觀察比較計算完後，左上角的值

黑色 = 0, 白色 = 255

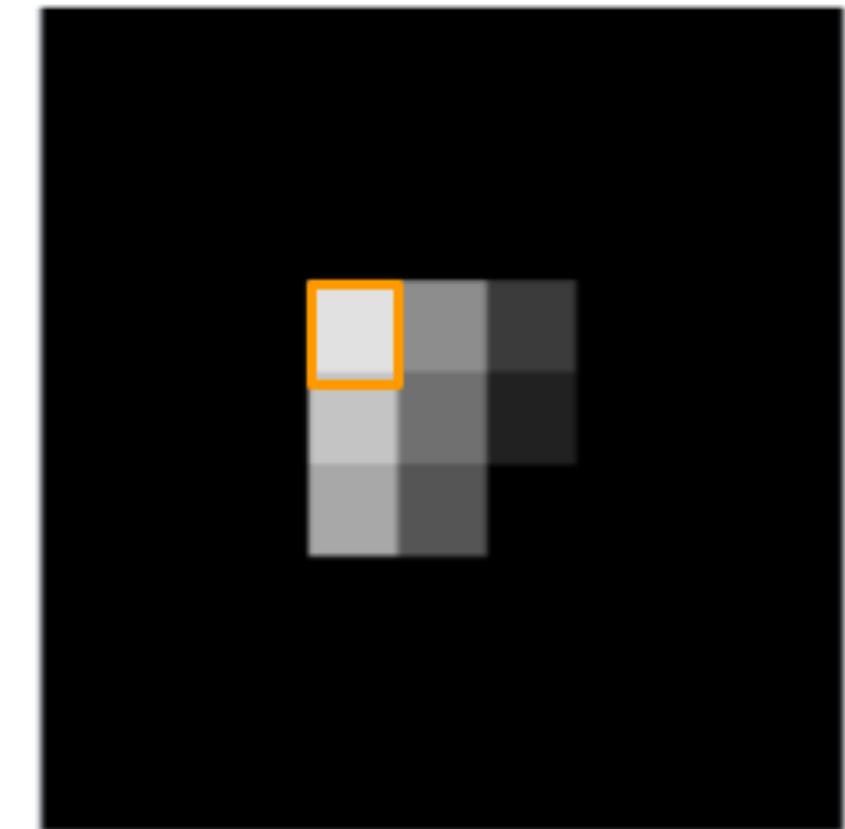


cross-correlation
 $255*255+0+0+...$

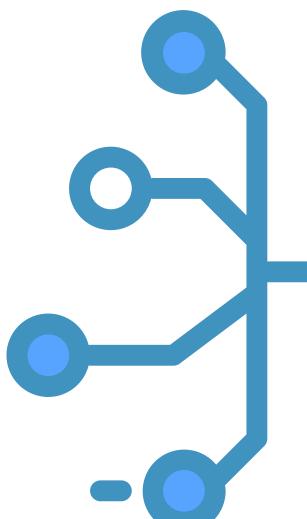
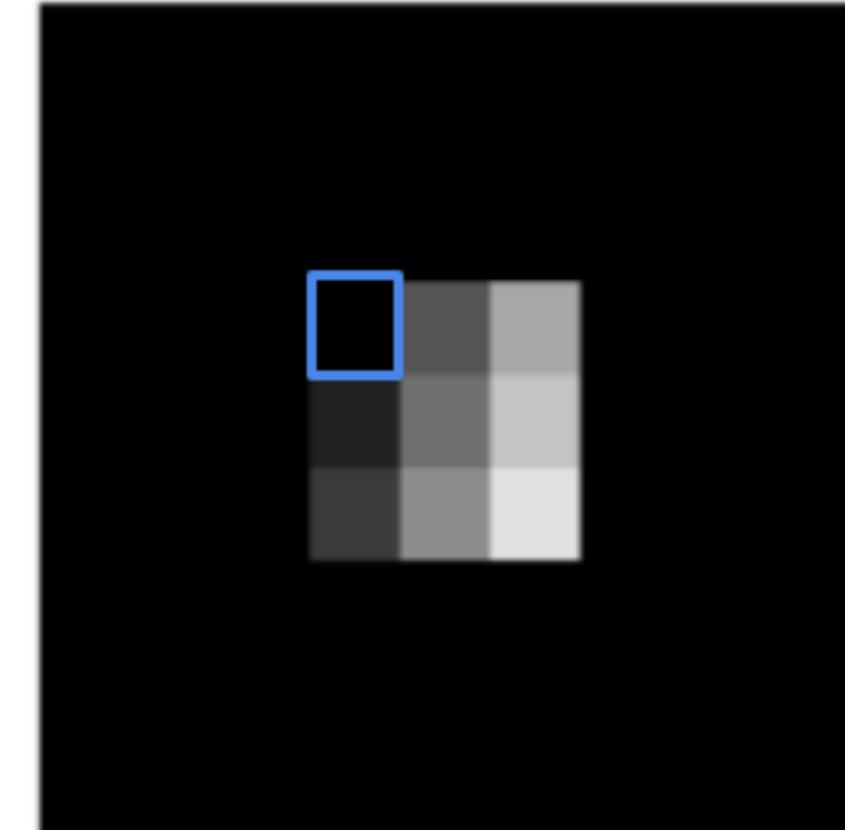


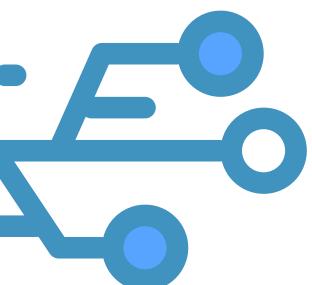
convolution
 $255*0+0+0+... = 0$

Cross-Correlation Output



Convolution Output



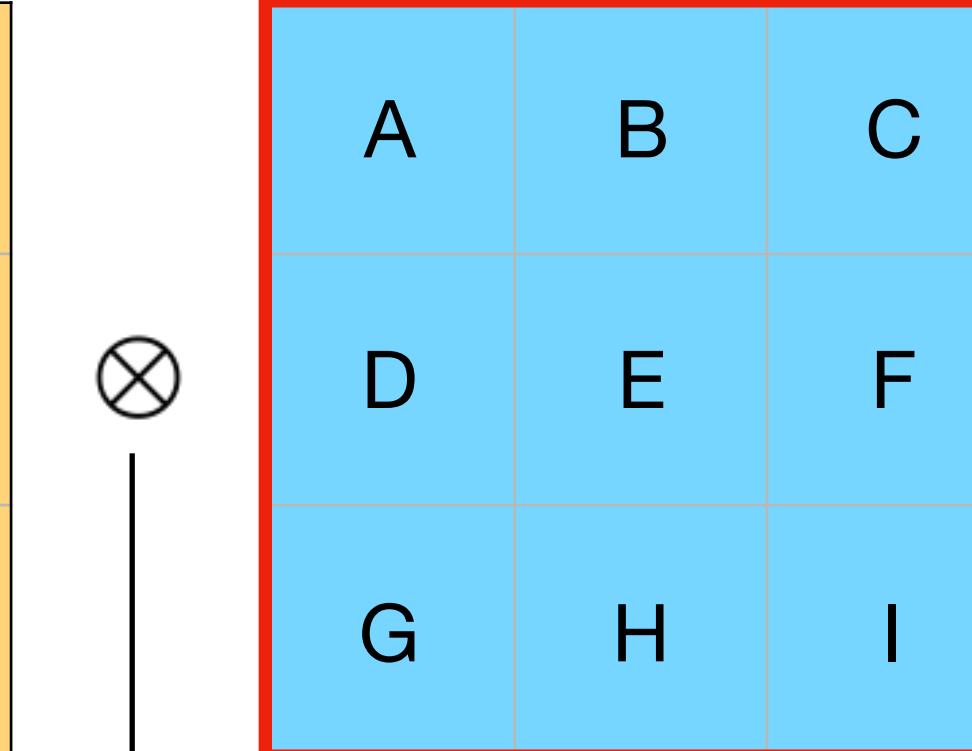


Filter 操作 - Padding



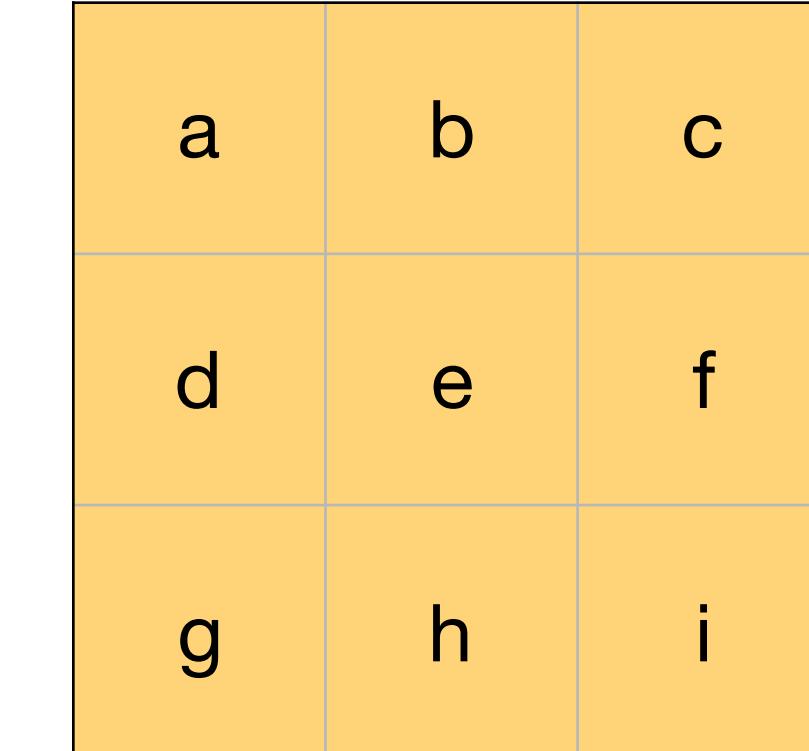
根據前面介紹的操作都是把 n 個值經過計算得到 1 個值，這樣圖片會變小，所以我們通常會在圖片周圍加上額外的值，確保運算完之後跟原圖大小一樣

a	b	c
d	e	f
g	h	i



1×1

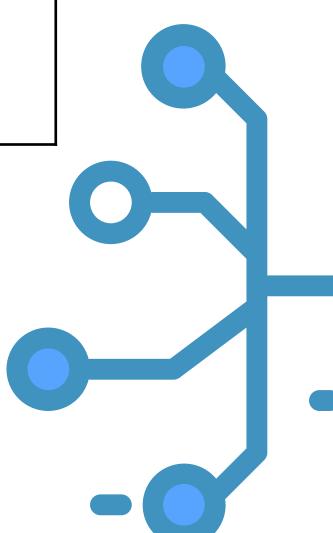
a	b	c
d	e	f
g	h	i

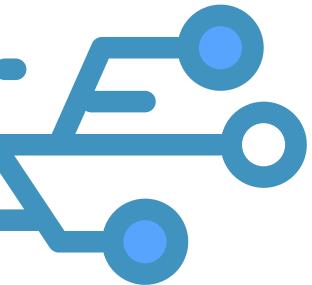


3×3

0	0	0	0	0
0	A	B	C	0
0	D	E	F	0
0	G	H	I	0
0	0	0	0	0

g





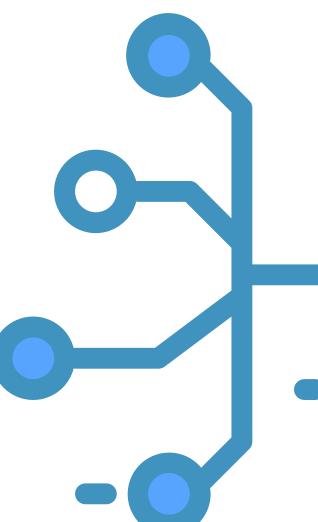
Filter 操作 - Padding



但因為操作關係，周圍填甚麼值會影響最後結果

這邊根據不同情況常見的操作有以下幾種：

- 補零
- 補鄰近 pixel 值
- 補整張圖片 pixel 值的 mean
- 鏡射



Filter 概念

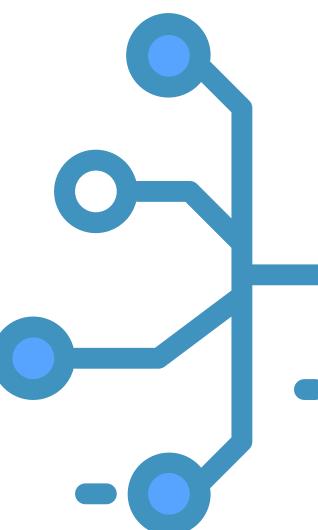


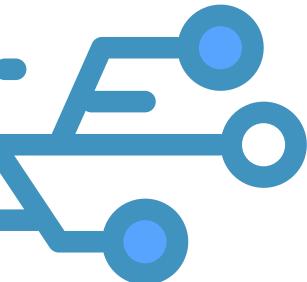
根據 Filter 中設定的係數不同，會有不同的結果
後面我們會透過不同的目標任務來熟悉 Filter 的概念與實作

模糊 Gaussian Blur



邊緣偵測 Sobel



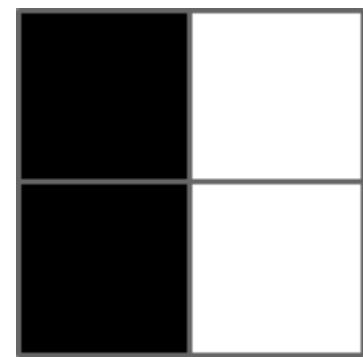


模糊 - 概念

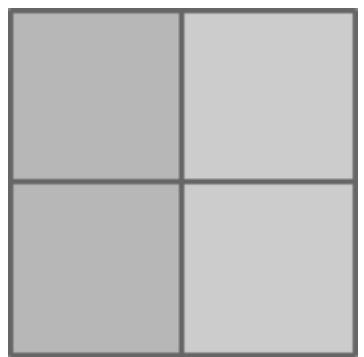


影像模糊從另一個觀點解釋可以說是邊緣不明顯
而邊緣的物理意義可以簡單代表兩側的顏色差別大

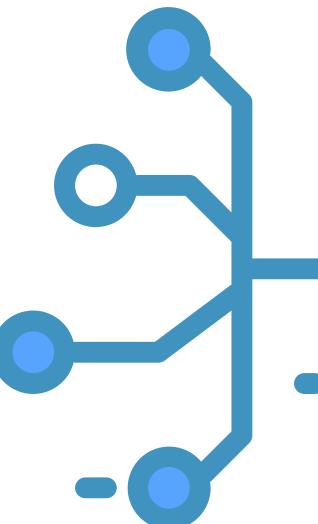
因此圖片要變模糊主要是物體邊界會受影響
我們也可以把模糊想像成是要讓顏色的差異變小

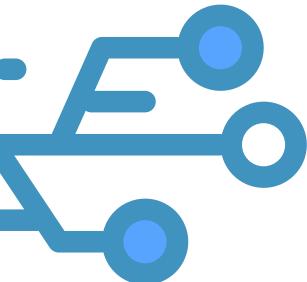


邊緣明顯，影像清楚



邊緣不明顯，影像模糊
(左右顏色不一樣)





模糊 - Averaging, Gaussian Blur



最簡單的概念是只要把周遭的 pixel 全部平均就好
但比較常使用的是 Gaussian Filter，認為中心點的資訊還是最重要的

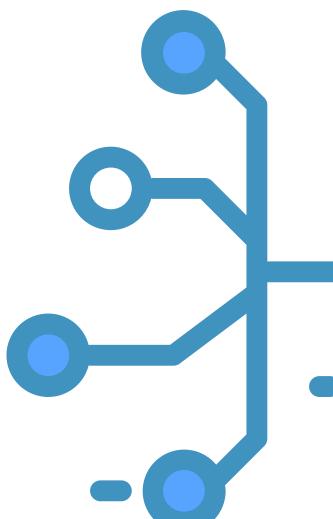
$$K = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

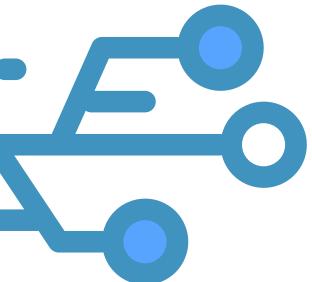
假設 filter 是 3*3 大小
每個位置的權重都一樣，再除以 9
把這個區域內的值做平均

$$K = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

根據高斯函數產生的值，可以根據
不同的 mean 跟 variance 產生不同
的 Gaussian Filter

Gaussian Blur 有時候也會有去雜訊的效果





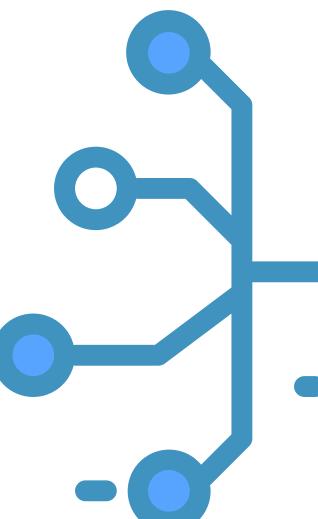
模糊 - 實作

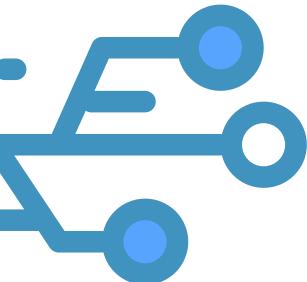


```
# 取得模糊過的圖片，Gaussian Filter size 設為 3*3  
img_blur = cv2.GaussianBlur(img, (3, 3), 0)
```

sigmaX: X 軸的標準差，設為 0 會根據 filter size 自己算
(sigmaY default = 0)

假如覺得模糊效果不足，可以對同一張圖片多做幾次
或是調整 filter size 決定模糊範圍大小





邊緣檢測 - Sobel

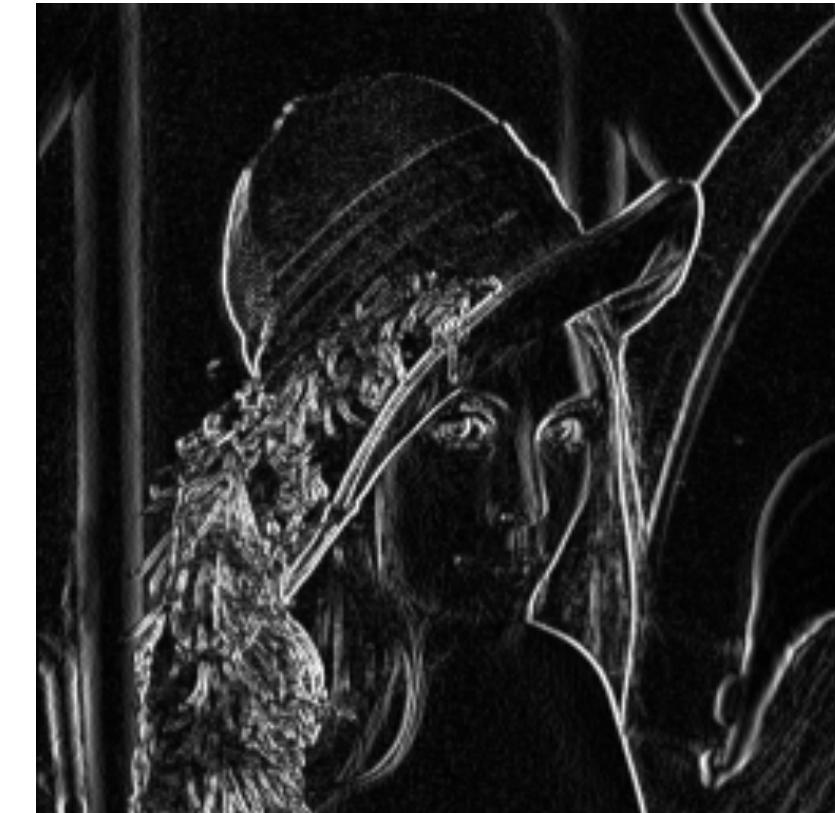


如同我們前面說的，邊緣的特性是兩側的顏色差別很大
跟模糊不一樣的地方是邊緣檢測是要加強邊緣特性

- 通常會用 Gray Scale 圖來做邊緣檢測
- 基本的邊緣檢測：Sobel filter

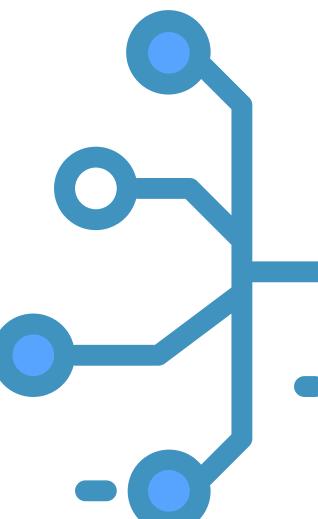
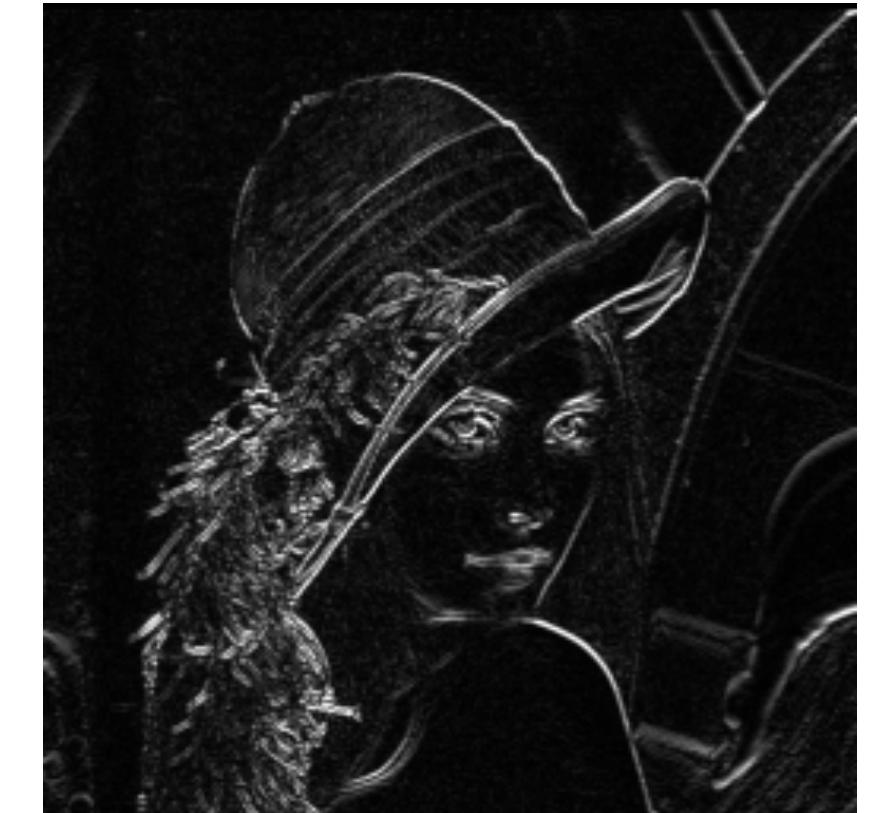
水平方向邊緣檢測

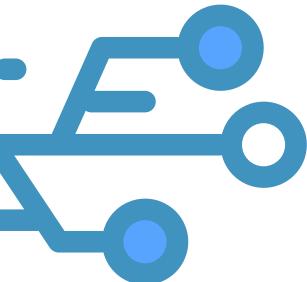
$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix}$$



垂直方向邊緣檢測

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





邊緣檢測 - 實作

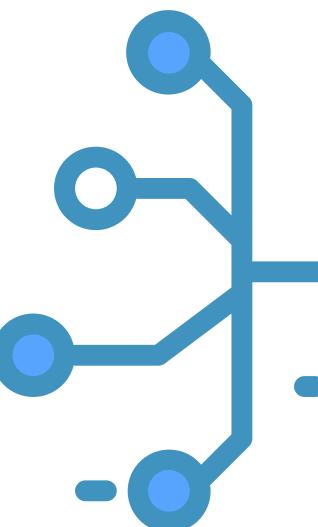


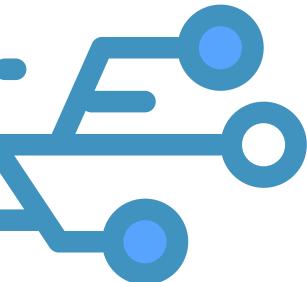
尋找邊緣正確的說法應該是計算圖片的導數 (derivatives)
我們可以透過 Sobel 個別拿到 x 與 y 方向的邊緣後再結合

```
img_x = cv2.Sobel(img_grey,  
                   cv2.CV_16S,  
                   dx=1, dy=0,  
                   ksize=3)  
  
img_y = ...
```

原本的圖片是 uint8 (非負整數) 格式
觀察 Sobel filter 可以發現有負數
所以我們希望運算過程中用 int16

代表求導次數，0 代表不用求導





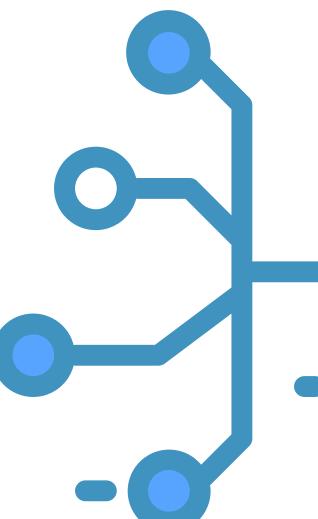
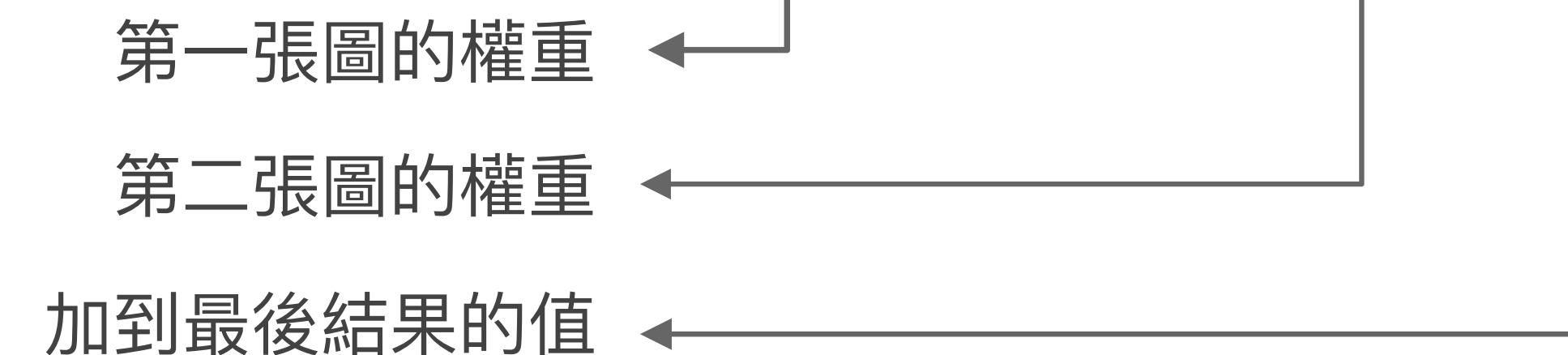
邊緣檢測 - 實作

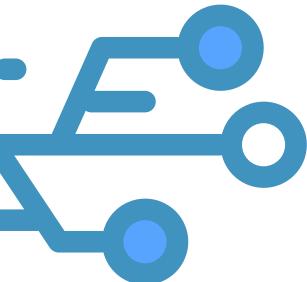


個別計算完 xy 方向的邊緣之後再接著處理

- 處理計算完的負數部份，做正規化變成非負整數再改為 uint8
- 合併 xy 邊緣的圖

```
img_x = cv2.convertScaleAbs(img_x) # 處理變為非負整數  
img_y = cv2.convertScaleAbs(img_y)  
img_sobel = cv2.addWeighted(img_x, 0.5, img_y, 0.5, 0)
```



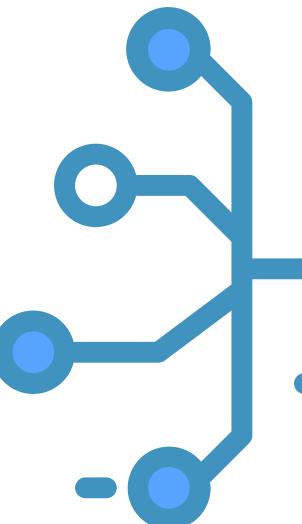


邊緣檢測 - 實作 (optional)



根據前面提到的一些問題可以印出來看看效果

1. 直接以非負整數操作 Sobel 會有數值截斷的問題
2. Sobel 調整 dx, dy 看求導次數變多會有甚麼效果



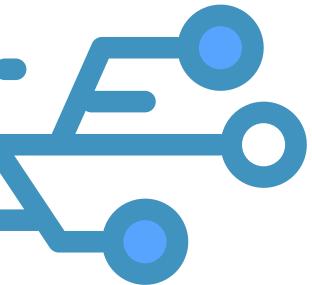
知識點回顧

- 了解 Filter 的基本操作
 - * Convolution 與 Cross-Correlation 基本上只有順序不一樣
 - * 直接對圖片做 Convolution 會使得最後圖片變小，所以通常會加上 padding
- 了解模糊與邊緣的操作都是對周遭 pixel 值的操作
 - * 周遭顏色愈相近即為模糊
 - * 轉為灰階圖片後讓周遭顏色差異愈大即為邊緣偵測

範例

- 實作模糊與邊緣檢測
 - 透過 Gaussian Filter 實作模糊操作
 - 透過 Sobel Filter 實作邊緣檢測





推薦延伸閱讀



原创 OpenCV-Python教程（6、Sobel算子）

2013-06-27 15:46:16 Daetalus 阅读数 57114 更多

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：<https://blog.csdn.net/sunny2038/article/details/9170013>

本篇文章介绍如何用OpenCV-Python来使用Sobel算子。

提示：

- 转载请详细注明原作者及出处，谢谢！
- 本文介绍使用OpenCV-Python实现基本的滤波处理
- 本文不介详细的理论知识，读者可从其他资料中获取相应的背景知识。笔者推荐清华大学出版社的《图像处理与计算机视觉算法及应用(第2版)》。

Sobel算子

原型

Sobel算子依然是一种过滤器，只是其是带有方向的。在OpenCV-Python中，使用Sobel的算子的函数原型如下：

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

函数返回其处理结果。

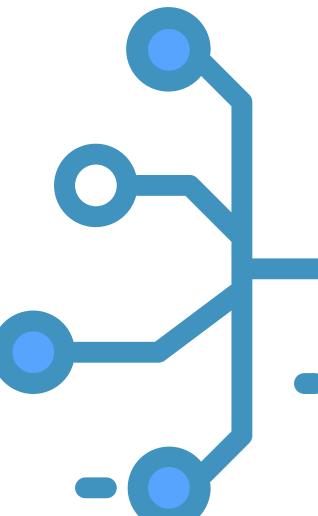
前四个是必须的参数：

- 第一个参数是需要处理的图像；
- 第二个参数是图像的深度，-1表示采用的是与原图像相同的深度。目标图像的深度必须大于等于原图像的深度；

CSDN Sobel

用中文詳細解釋 OpenCV 使用 Sobel 來做邊緣檢測的相關問題，
包含參數與資料型態。

連結



解題時間 Let's Crack It



請跳出 PDF 至官網 Sample Code & 作業開始解題