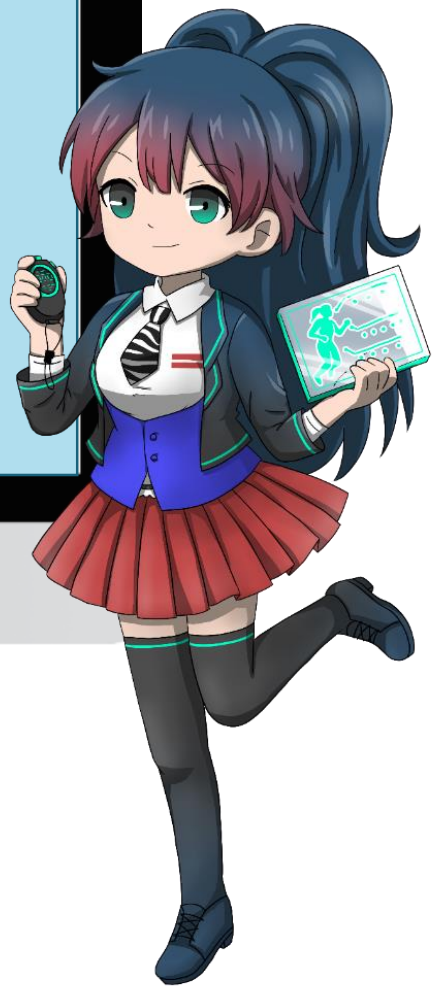
高雄女中新興科技區域推廣中心

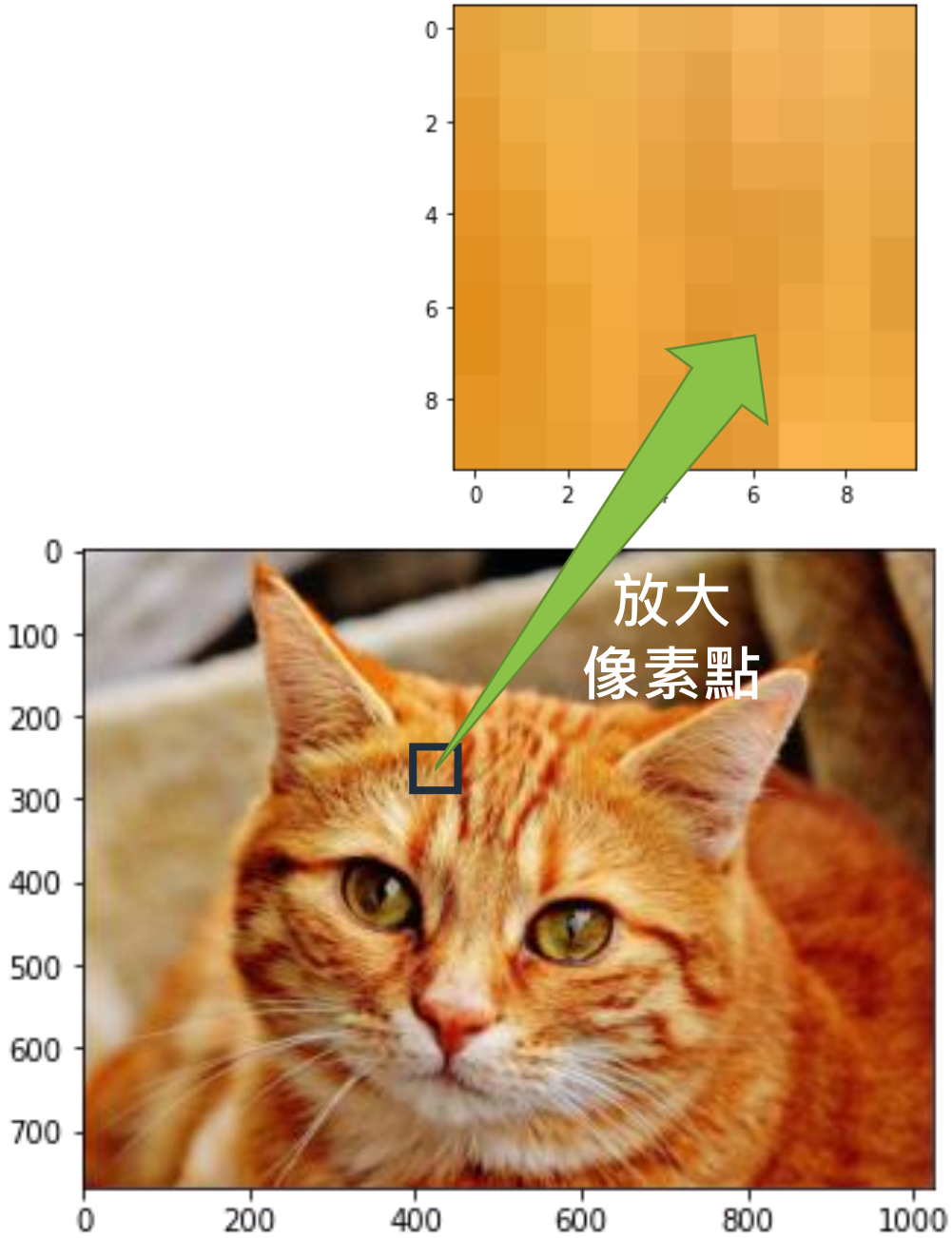
主題4

卷積神經網路 (CNN)



電腦影像的資料表示法

- 『點陣圖』：將整個圖像分割成為如棋盤式的方格點，每一個方格點稱為一個像素。
- 每一個像素可以單獨儲存顏色資訊，因此點陣圖任意構圖線條都能呈現，且顏色多變。
- 人工智慧處理的圖片一般是照片(人臉、汽車、、、)的點陣圖為主，了解點陣圖的資料架構，我們才能從中取得我們要的圖形特徵。
- 一張點陣圖平面上的所有像素點由一個二維矩陣組成，總像素等於長度的像素X長度的像素。
  例：Full HD的照片 其總像素=長1920像素×高1080像素
  =2073600像素(200萬)

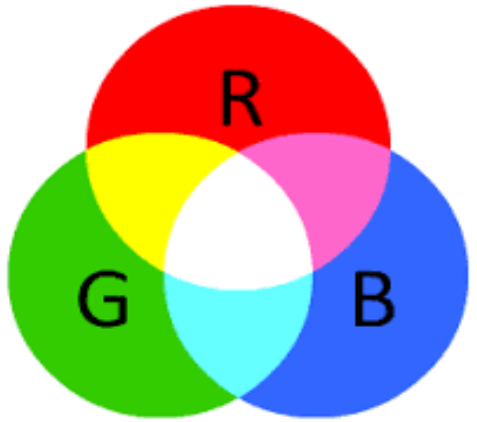


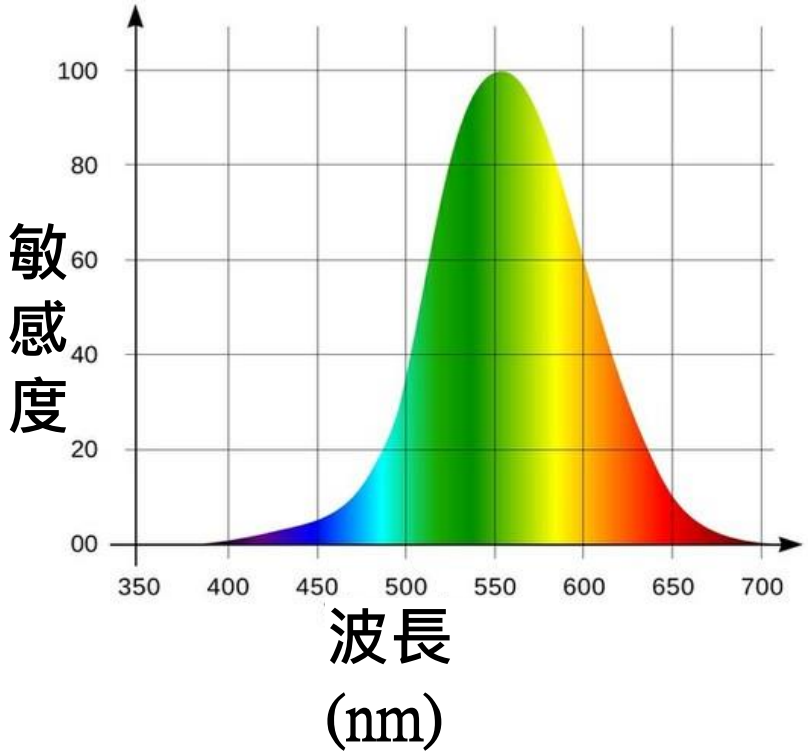
像素點的顏色資訊

- 如右上圖，像素的顏色資料是根據人類肉眼接受照光R(紅色Red)、G(綠色Green)、B(藍色Blue)三色不同強度混合而成彩色。

- 人的眼睛對不同波長的顏色其敏感度並不相同，下右圖為人眼對光線刺激敏感度與光波長的關係圖。

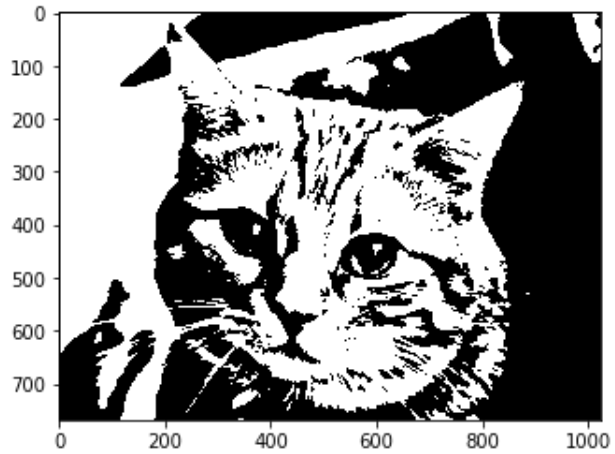
- 相同強度的R、G、B光線，其敏感度比值約為0.213:0.715: 0.072，故將R、G、B的強度對敏感度比值取線性加權總合，可以得到眼睛受光刺激的程度，而將圖片轉為灰階圖片。

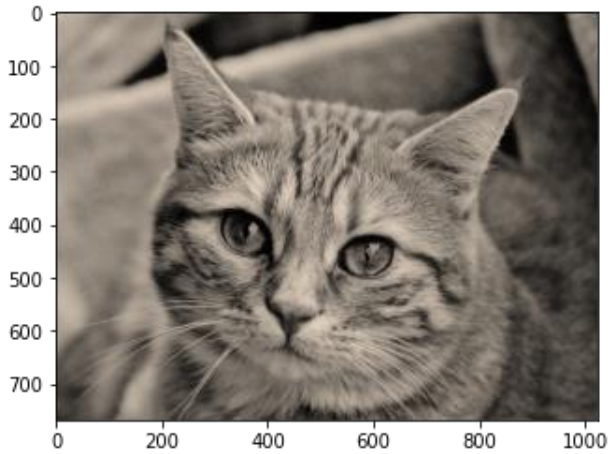


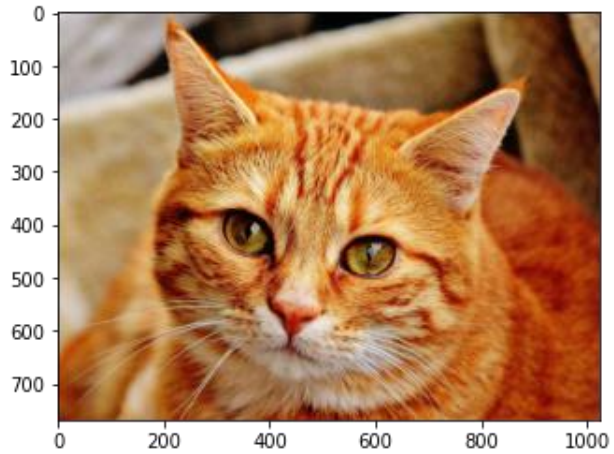


影像色彩的種類

- 黑白：即一個像素只有黑或白兩種情形，因此只需要一個位元便可以表示一個像素的顏色資訊。
- 灰階：還是黑白的，但由最黑到最白之間可有256種明亮度，每一個像素佔有的資料大小是一個位元組 (8個位元，1 byte) 。
- 全彩：最多可以有1677萬種顏色，每一個像素佔用3個位元組 (24位元，3 bytes) 。

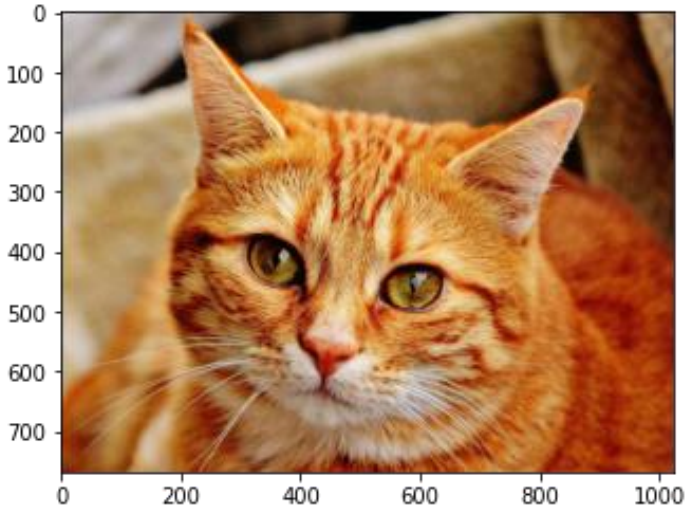


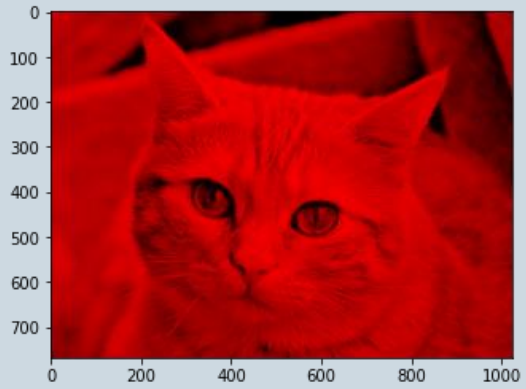
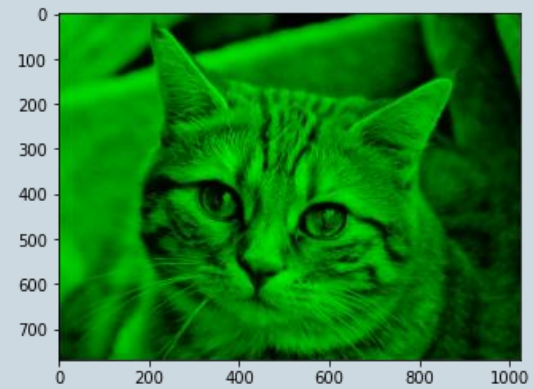
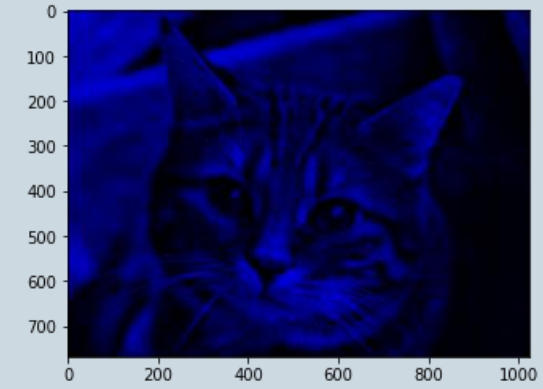
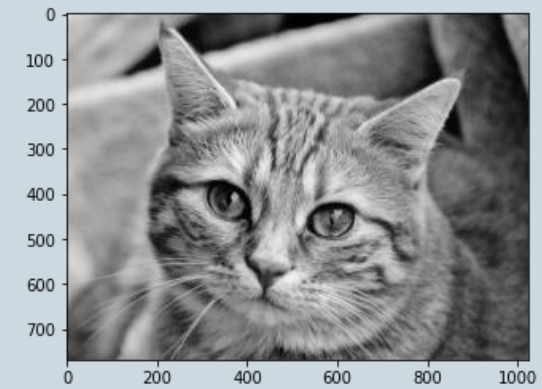




實作5-1 解析全彩圖片的資料組成結構

- 如右圖為檔名cat.png為全彩點陣圖，將其像素和顏色資料載入numpy array，解析其資料結構，包括高、寬的像素資料和R、G、B色頻的資料。
- 將圖片的R、G、B三原色分離顯示。
- 利用人眼對R、G、B三原色的敏感度，將圖片轉換為灰階圖案。



紅色(R)色頻圖	綠色(G)色頻圖	藍色(G)色頻圖	灰階圖
			

影像處理的原理

- 影像的資料結構就是一個二維矩陣，影像處理就是對這個矩陣內的像素資料進行改變。
  - 特徵濾鏡：用一個二維的矩陣，用來跟原影像的局部或全部對應位置的像素做數學運算後，轉換出來一個新的影像。
  - 不同的濾鏡能萃取不同的影像特徵，例如下圖中我們分別利用Sobel濾波器萃取垂直邊緣特徵和水平邊緣特徵。

萃取水平邊緣

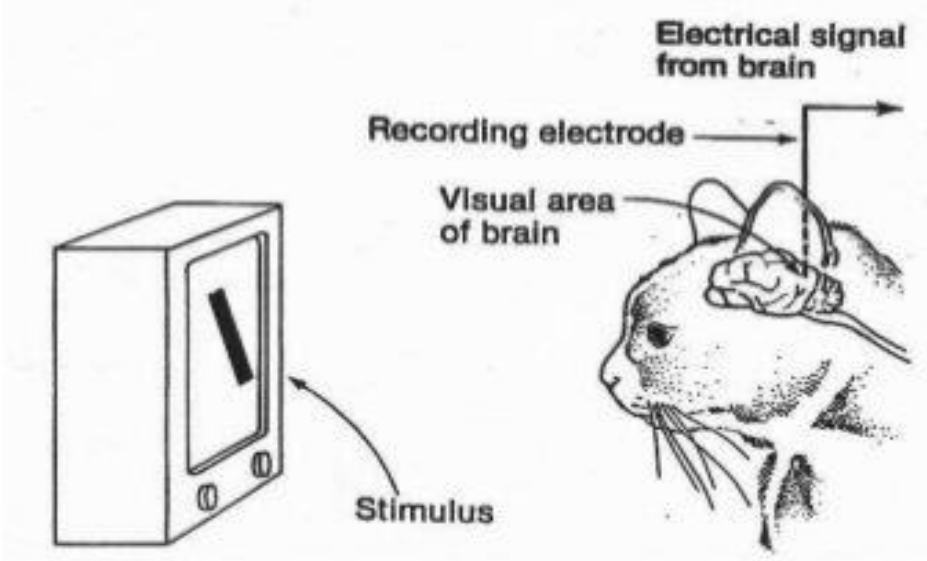
濾鏡 $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

萃取垂直邊緣

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

影像辨識的原理

- Hubel和Wiesel在研究動物大腦皮層的視覺神經元時發現：(獲得1981諾貝爾獎)
  - 每個視覺神經元只負責處理局部視覺信號，有的神經元對簡單圖像敏感(例如橫條、轉角)，有的對複雜圖像敏感(例如矩形、網狀)。
  - 而且處理複雜圖像的神經元其輸入並不是原始圖像，而是來自處理簡單圖像的神經元的輸出。
  - 視覺神經元是有層次的，高層神經元會將低層神經元檢測到的訊息整理抽象出更複雜的特徵，然後交給更高層的神經元處理。



影像辨識的原理

- 仿照動物的圖像處理結構，類神經網路在圖片上要辨識出是否有某一個物件的影像，必須要設計幾個萃取該物件關鍵影像特徵的濾鏡(Feature Detector)。
- 由於物體的關鍵特徵，可能出現在影像的任何地方，故濾鏡以掃描的方式檢測整張影像的，標示關鍵特徵出現的位置和特徵符合程度。
- 第一層的濾鏡只要找簡單的線條即可，慢慢地尋找由各種簡單線條所構成的複雜圖案，因此尋找關鍵特徵的動作會重複進行多次。
- 若有累積足夠符合的關鍵特徵，用以判斷物件是否出現在影像中。

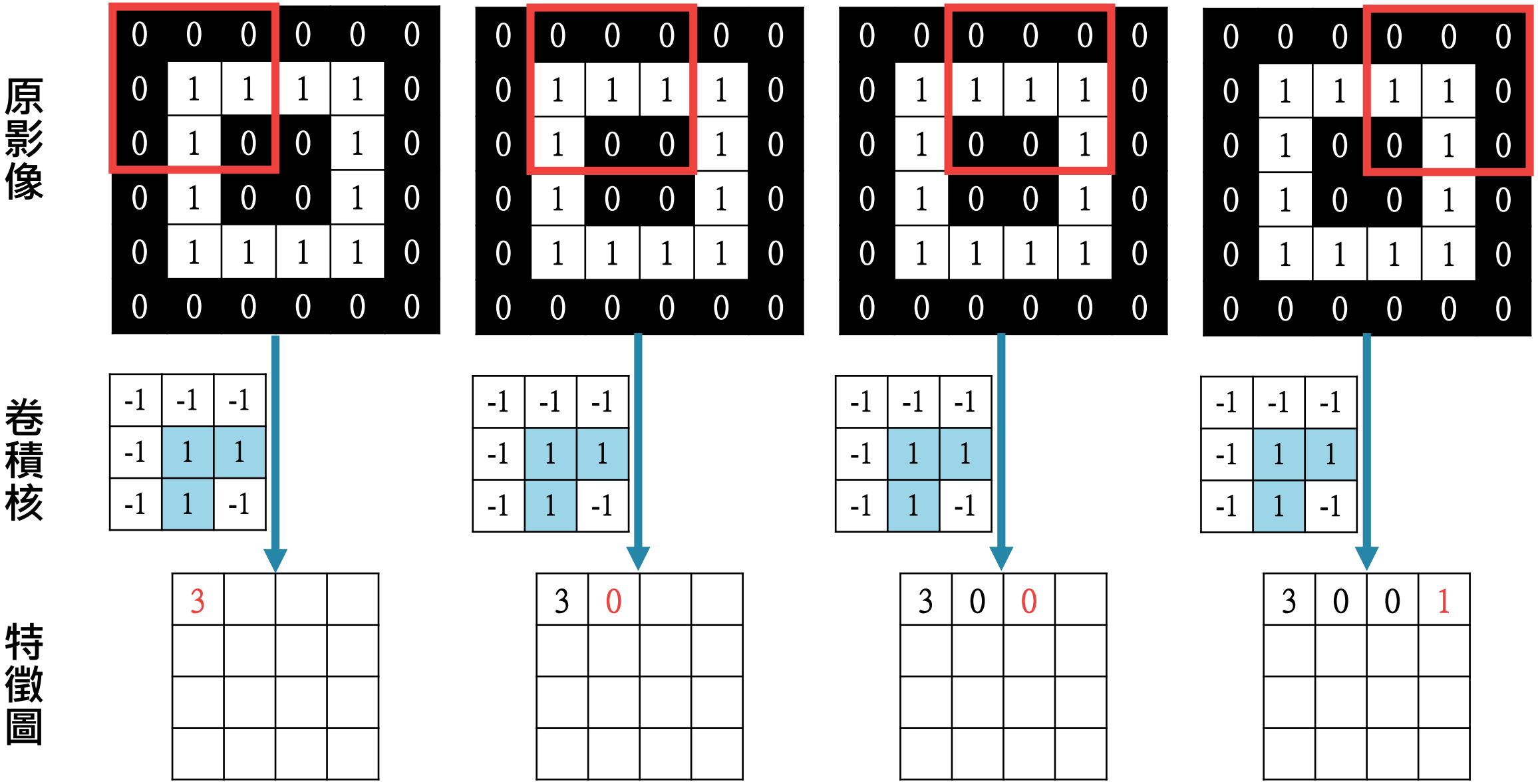
卷積(Convolution)運算

- 萃取該物件關鍵影像特徵的濾鏡(Feature Detector)稱為卷積核(Convolution Kernel)，為一個正方形的矩陣所構成。
- 卷積的基本原理：
  - 想像卷積核是一個採集器，由影像的左上角開始，由左而右，由上而下，將掃描區域內的像素值和卷積核上對應位置的元素值相乘再加總稱為卷積值，將卷積值放在一個新矩陣，直到掃完整張影像。
  - 原照片取出的像素矩陣X為 $\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$

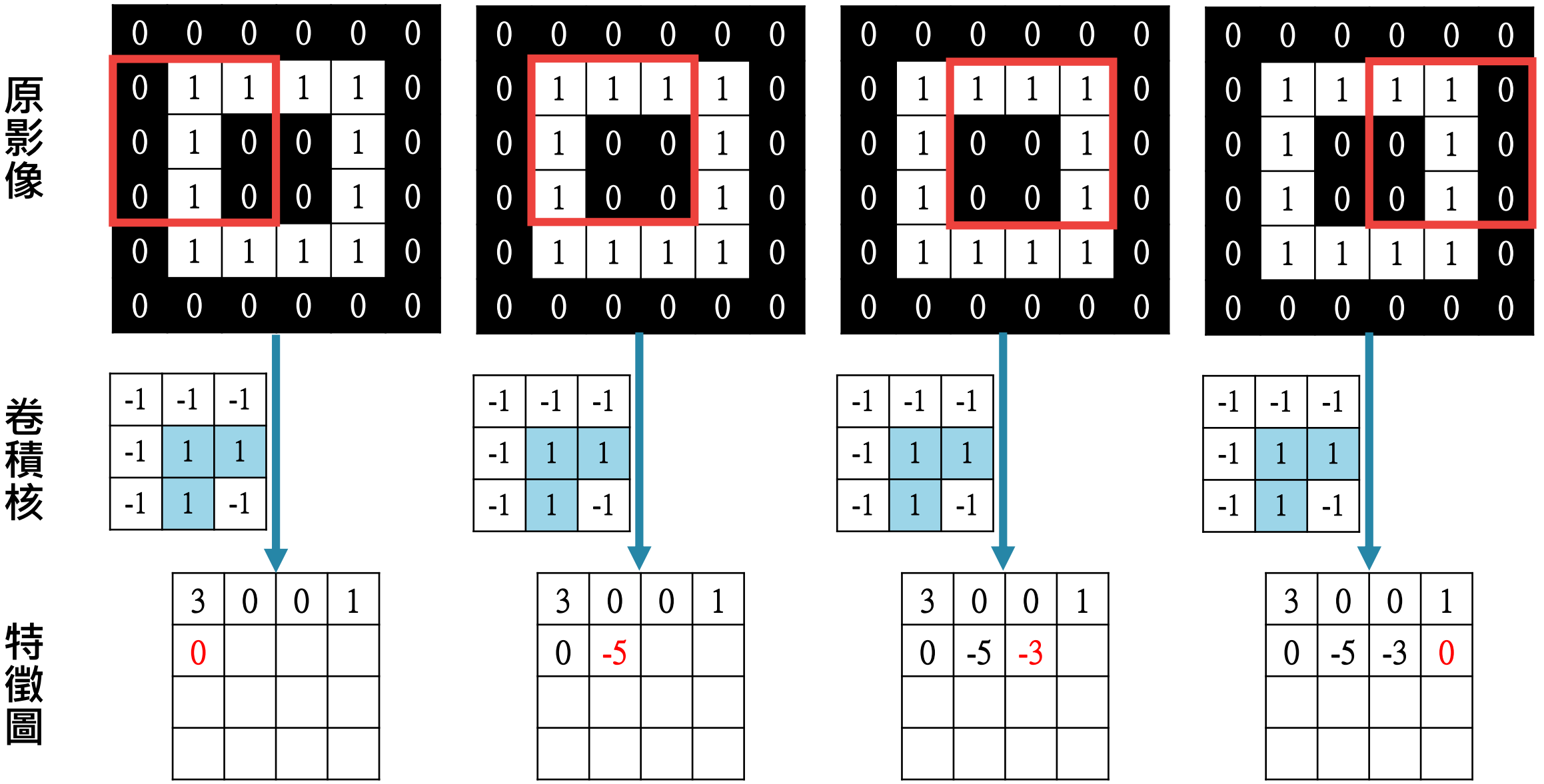
    卷積核矩陣W為 $\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$
  - 運算後的卷積值 $=\sum w_{ij}x_{ij}$

$$=w_{11}x_{11}+w_{12}x_{12}+w_{13}x_{13}+w_{21}x_{21}+w_{22}x_{22}+w_{23}x_{23}+w_{31}x_{31}+w_{32}x_{32}+w_{33}x_{33}$$

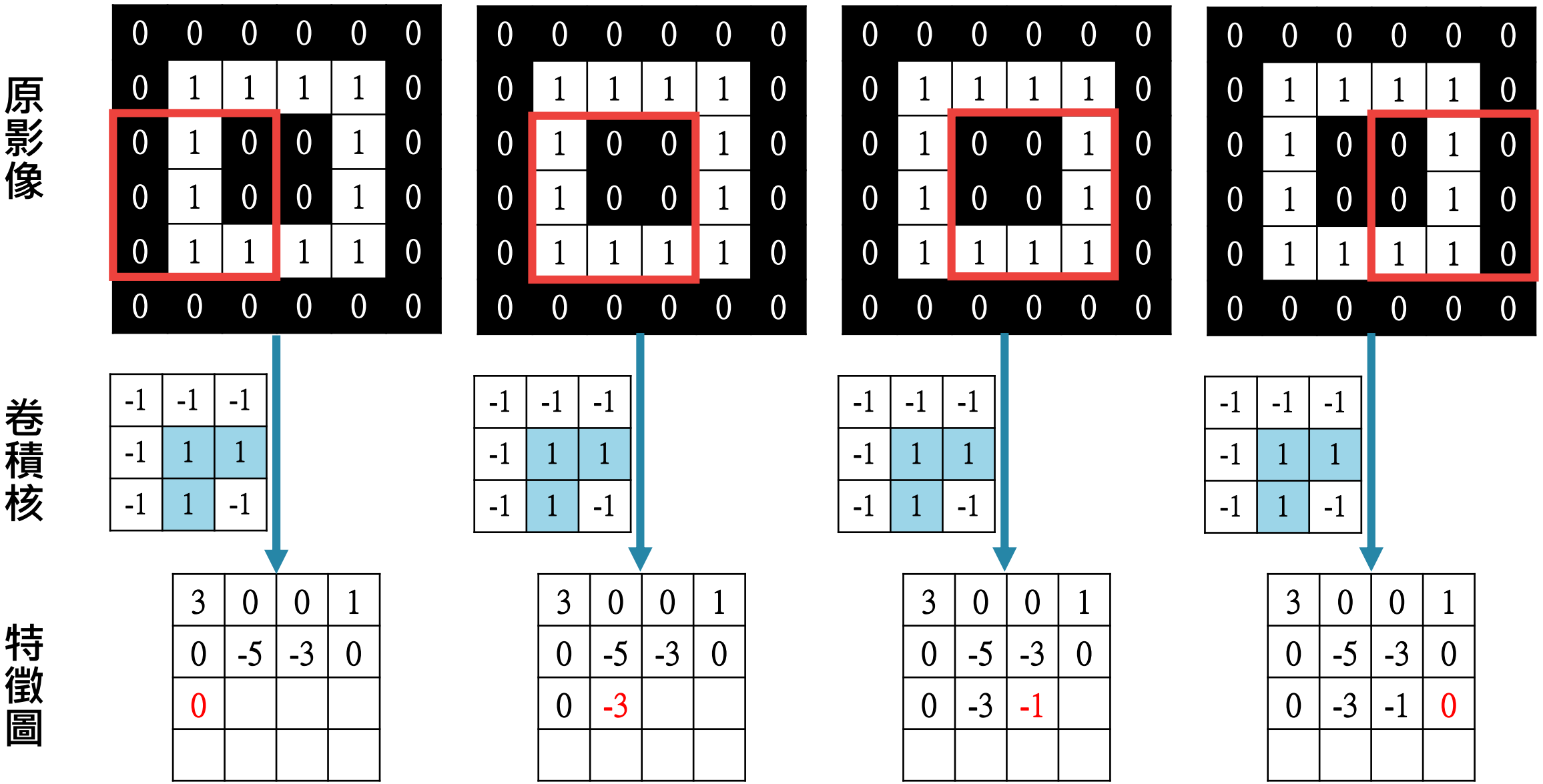
卷積(Convolution)運算



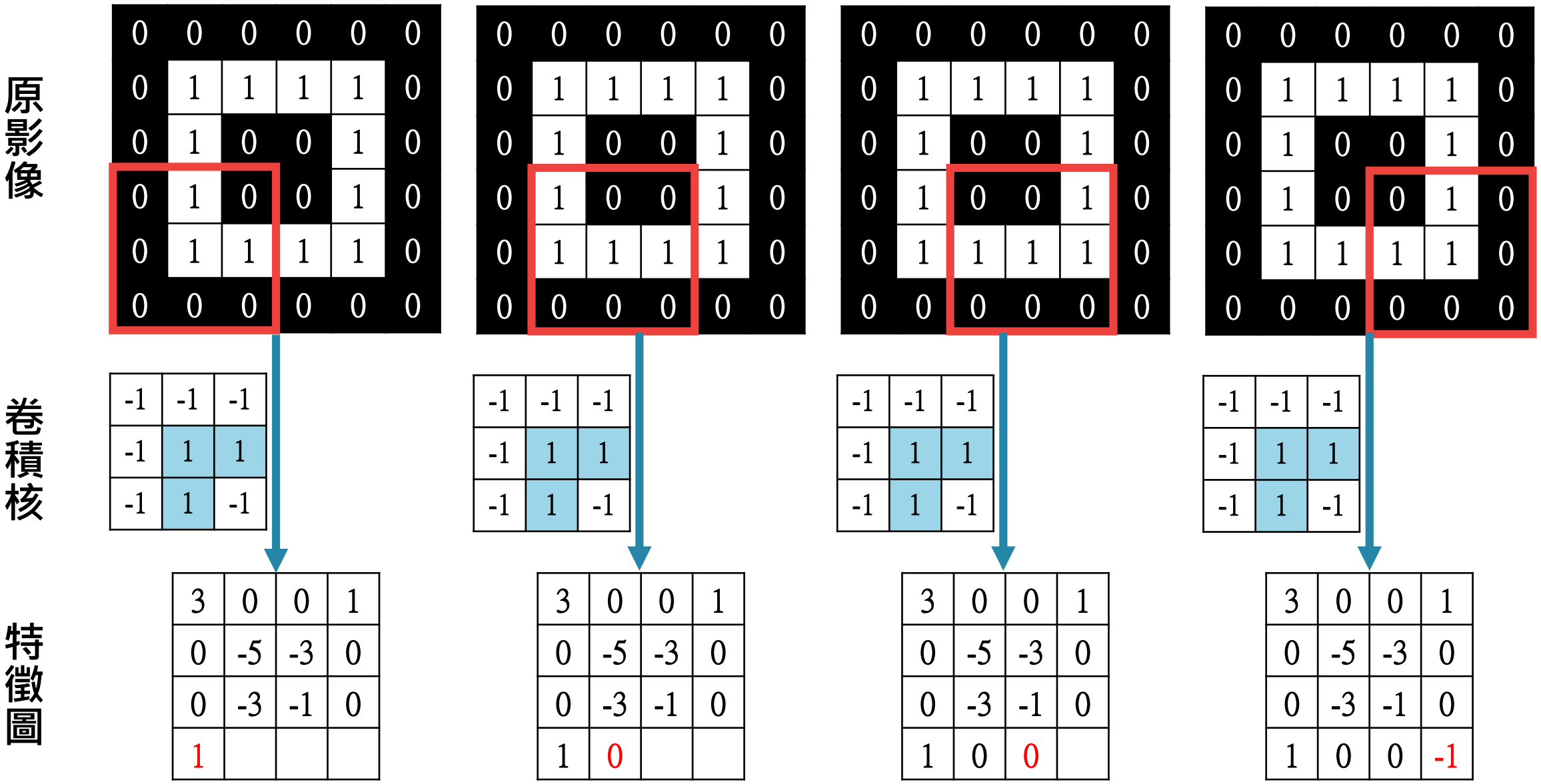
卷積(Convolution)運算



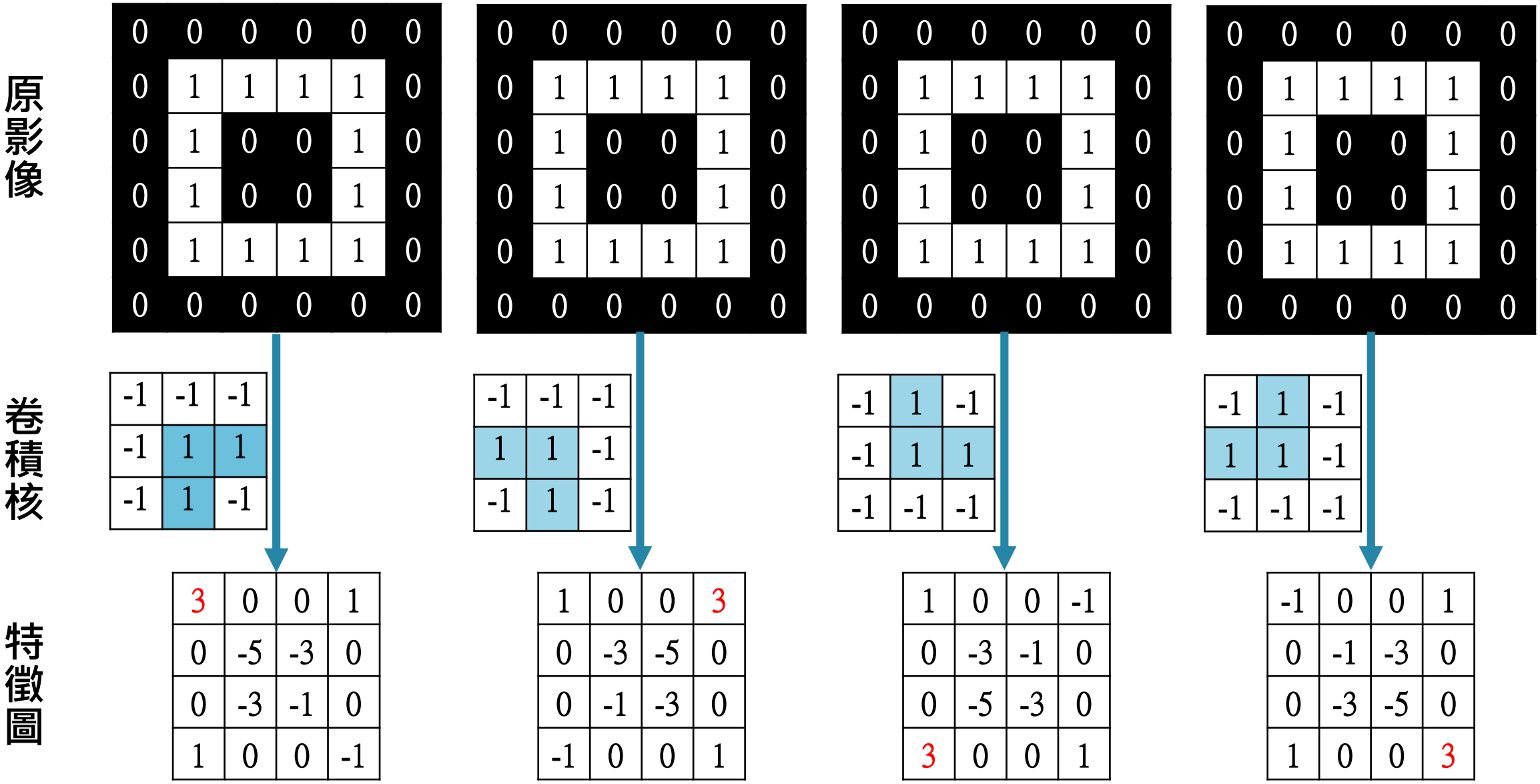
卷積(Convolution)運算



卷積(Convolution)運算



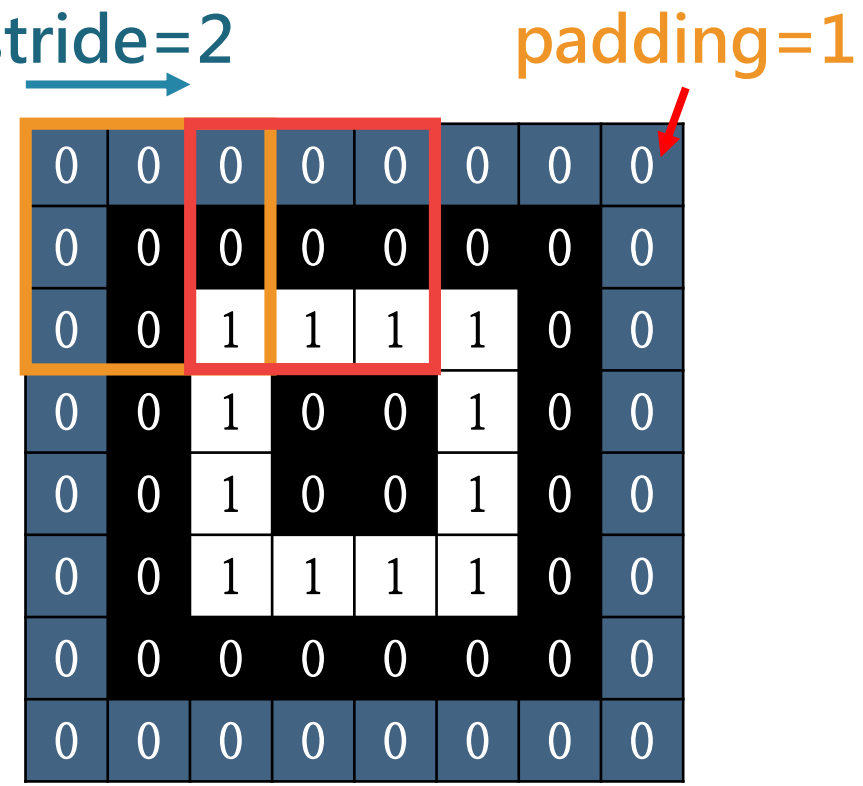
卷積(Convolution)運算

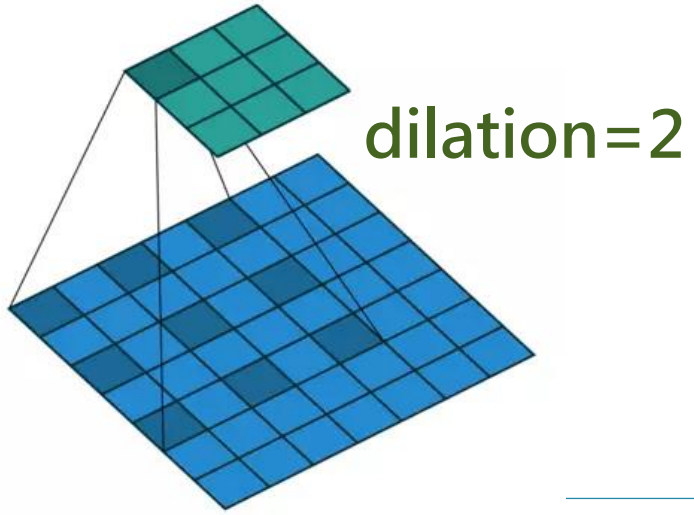


卷積(Convolution)運算

- 掃描時不一定要只平移1個像素，可以調整適合的步長(stride)。
- 若考慮關鍵特徵在影像邊緣，避免卷積運算時被忽略，可以在卷積處理前，在原影像周圍填補空白像素(padding)。
- 原影像經卷積運算處理後的得到的新矩陣，稱為特徵圖(Feature Map)，特徵圖上數值越高的位置，代表影像的局部和關鍵特徵的相符程度越高。
- 擴充卷積：卷積核間的步長(dilation) 。
- 特徵圖輸出尺寸計算公式：

$$W_{out} = \left[\frac{W_{in} - (\text{dilation} \times (\text{kernel\_size} - 1) + 1)}{\text{stride}}\right] + 1$$





torch.nn. Conv2d

- pytorch可以利用torch.nn. Conv2d()執行卷積運算。
- 參數說明：
  - in_channels：輸入的圖片通道數，第一層卷積會是圖片的顏色狀況，例如灰階圖片=1，全採圖片=3；第二層卷積要設成前一層卷積的out_channels。
  - out_channels：輸出的通道數，這邊會變成下一層的 in_channels。
  - kernel_size：卷積核尺寸，可用邊長或(高,寬)表示。
  - stride=1：設定每次卷積移動的步長，預設會是 1。
  - padding=0：設定原圖四周填補厚度，預設會是 0。
  - dilation=1：擴充卷積。
  - groups=1：分組卷積。
  - bias=True：卷積後是否加偏值。
  - padding_mode=“zeros”：填補模式。
- 輸入格式：(批次圖片數， in_channels ，圖片高度，圖片寬度)
- 輸出格式：(批次圖片數， out_channels ，圖片高度，圖片寬度)

實作5-2 設計一個卷積核尋找影像關鍵特徵位置

- 觀察檔名為star1.jpg的影像檔，星空中最亮的一顆星為幾個白色像素點組成+形狀。
- 請你設計一個3x3矩陣的卷積核，能夠萃取上述最亮的星星的關鍵特徵。
- 先將這個影像的像素轉成灰階的numpy array，利用二維迴圈將卷積核在像素陣列中掃描做卷積運算，並將運算結果存到新的numpy array。
- 尋找卷積運算後的最大值，並由其位置推出原影像中星星的座標。

什麼是池化(Pooling)

- 池化是把原來的影像劃分為許多小區塊(比如2x2)，在將每個小區塊中的像素值，濃縮成一個數值。
- 池化的目的：
  - 壓縮特徵圖的大小，把相鄰區域的資訊進一步壓縮，去蕪存菁，減少運算量。
  - 關鍵特徵在原影像上佔有較大的區域，經過池化後變成較小的特徵圖上的一個點，從一個點看到整個原影像蘊含的資訊，也謂「見微知著，凸顯特徵」。
  - 若原圖上的關鍵特徵有扭曲、旋轉、平移，經過池化可以保持特徵不變性。

輸入的影像	池化種類	輸出影像
3 -2 -1 1 0 -5 -3 -1 -1 -3 -1 0 1 -1 -2 -1	Max Pooling	3 1 1 0
	Average Pooling	-1 -1 -1 -1

池化的種類

- Max Pooling (最大池化)
  - 選擇最大的
  - 優點：能保留更多紋理邊緣訊息。
- Average Pooling (平均池化)
  - 相加後平均
  - 優點：能保留更多圖像的背景訊息。

輸入的影像	池化種類	輸出影像
3 -2 -1 1 0 -5 -3 -1 -1 -3 -1 0 1 -1 -2 -1	Max Pooling	3 1 1 0
	Average Pooling	-1 -1 -1 -1

實作5-3 利用卷積層及池化層擷取圖片特徵

- 觀察檔名為star.jpg的影像檔，星空中閃亮的20顆星中有7顆星為幾個白色像素點組成x形狀，其他為+形狀。
- 請你設計一個5x5矩陣的卷積核，能夠萃取上述的7顆星星的關鍵特徵。
- 先將這個影像的像素轉成灰階的numpy array，利用二維迴圈將卷積核在像素陣列中掃描做卷積運算，並將運算結果存到新的numpy array為卷積特徵圖。

實作5-3 利用卷積層及池化層擷取圖片特徵

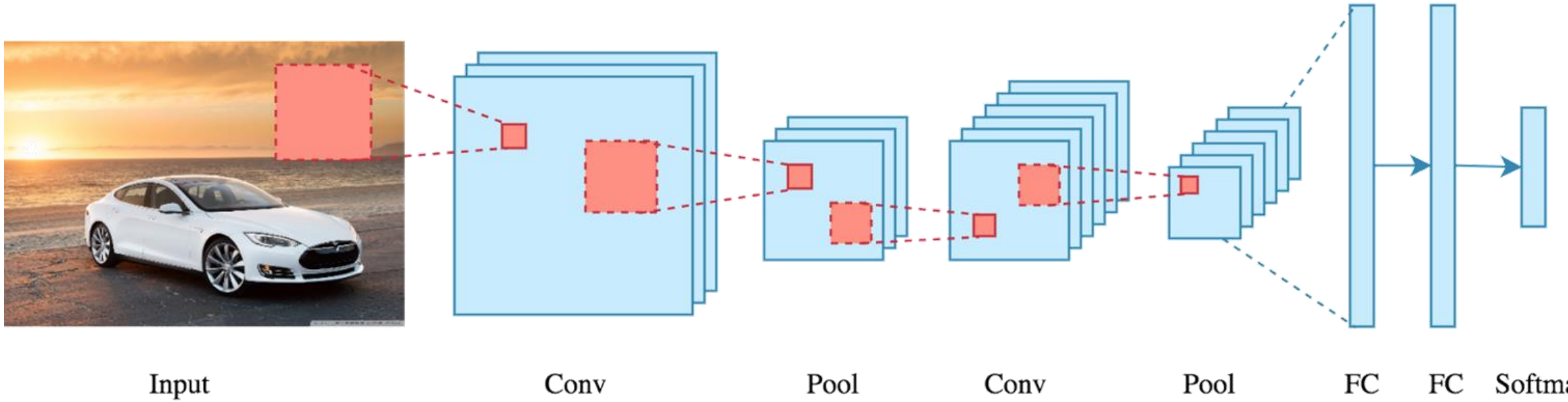
- 以2 × 2的像素方塊將上述卷積特徵圖作Max Pooling池化，並將運算結果存到新的numpy array為池化特徵圖。

- 將池化特徵圖顯示出來，比較灰階圖、卷積特徵圖和池化特徵圖的差別。

灰階圖	卷積特徵圖	池化特徵圖
		

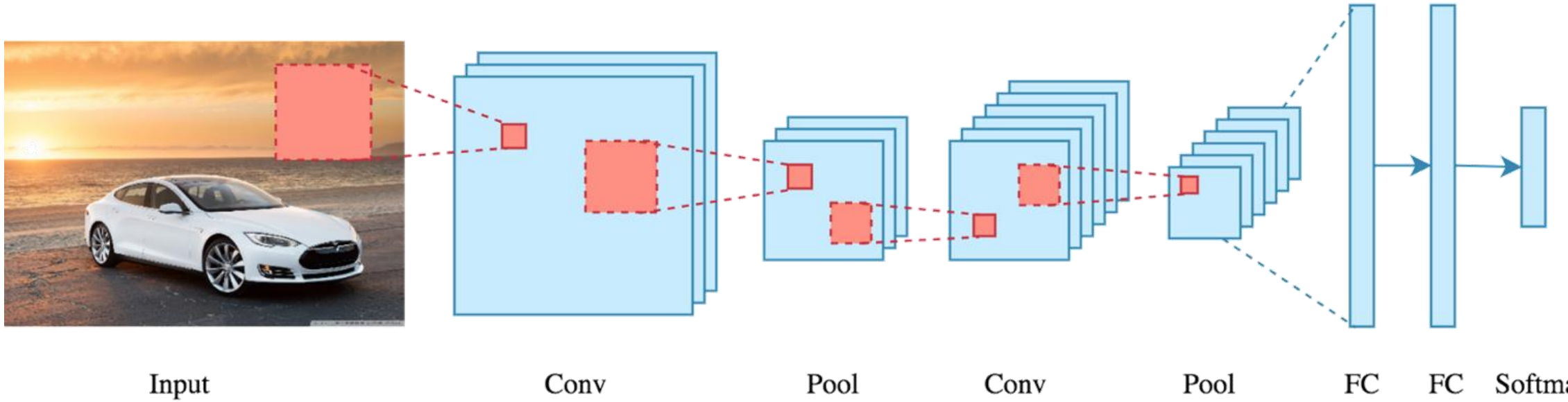
卷積神經網路(CNN)

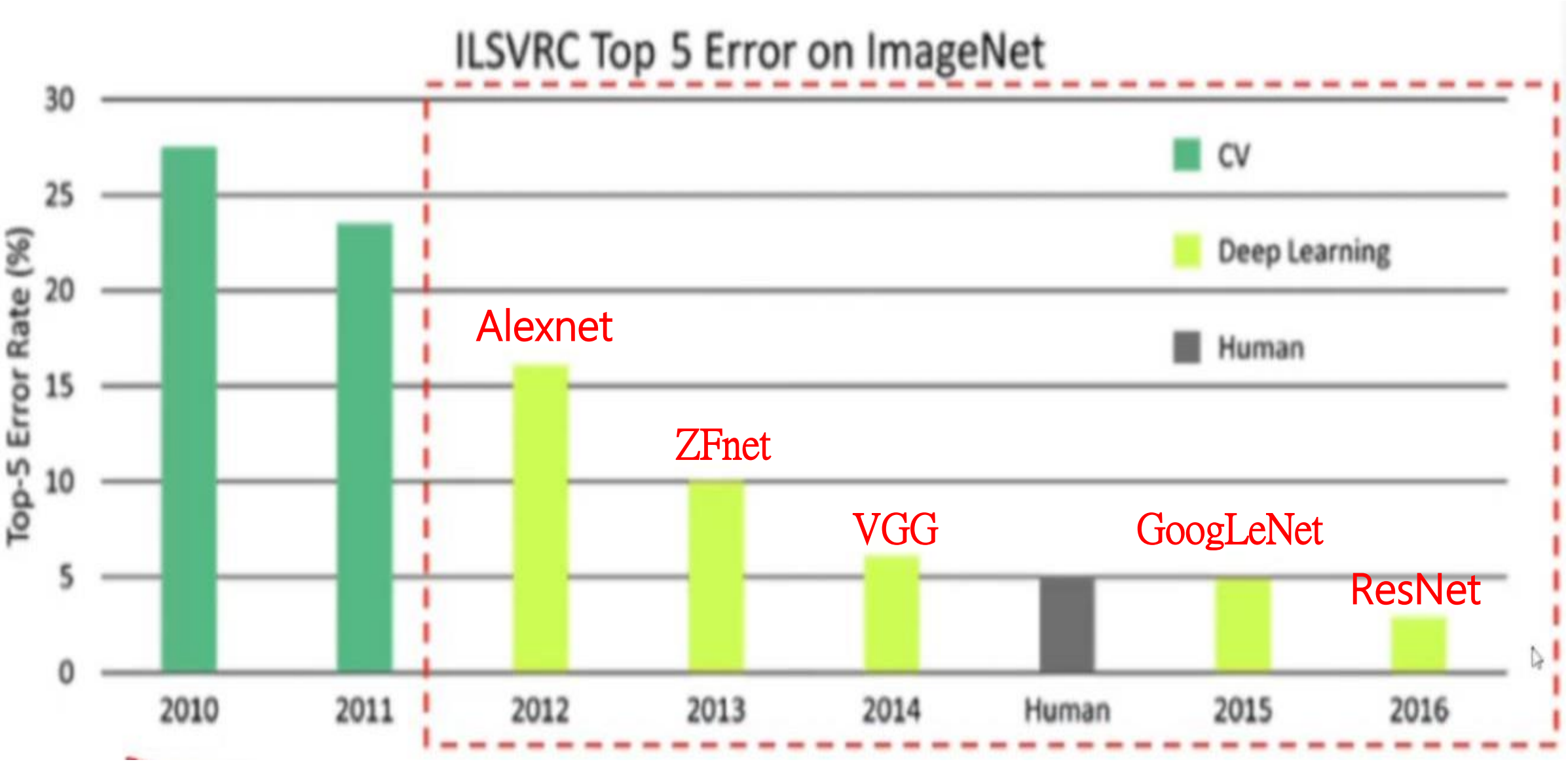
- 典型的卷積神經網路(Convolutional Neural Network) 由3個部分構成：
  - 卷積層(Convolution Layer)：負責提取圖像中的局部特徵
  - 池化層(Pooling Layer)：大幅降低參數量級(降維)
  - 全連接層(Fully Connected Layer)：傳統神經網路的部分，用來輸出想要的結果。



卷積神經網路(CNN)

- 通常一層卷積層和一層池化層為一組，會經過多組以萃取特徵，視為特徵工程。
- 最後一組的卷積和池化後輸出的特徵圖，會收集大量的圖片特徵，會將其拉平後接上全連階層。
- 全連接層通常使用深度神經網路，組合更深層的關鍵特徵。
- 卷積運算中，卷積核內的權重也是透過反向傳播的過程中，利用梯度下降法不斷調整而得到的。





經典卷積神經網路-LeNet

- LeNet是最早的分類卷積網路，被設計用於手寫數字辨識，1998年由Yann Lecun提出。
- 因為其理論解釋性較差，並且效果不如處理人工特徵的SVM，沒有得到重視。
- 經過幾代的改良，LeNet-5用於辨識MNIST資料集結構如圖。

實作5-4 以CNN 作fashion MNIST物品影像辨識分類

- MNIST 是一個灰階的物品圖像資料集，每個圖片大小(長 x 寬)是 28 x 28 像素。

- 檔案 fashion_train.csv 包含訓練資料(60000 筆)，檔案 fashion_valid.csv 包含驗證資料(10000 筆)。
- 檔案內一列一筆資料，每一筆資料逗號分開 785 欄位
  - 第 1 欄為 0、1、2、3、4、5、6、7、8、9 的數字標識，分別代表下列物品：

0	1	2	3	4	5	6	7	8	9
短袖圓領 T 恤	褲子	套衫	連衣裙	外套	涼鞋	襯衫	運動鞋	包	短靴

  - 第 2 欄~785 欄為灰階圖片的 28X28=784 個像素
  - 灰階像素由 0~255 依序代表黑到白的深淺程度

實作5-4 以CNN 作fashion MNIST物品影像辨識分類

- 建立一個卷積神經網路LeNet-5，來訓練模型：

- 利用訓練完成的卷積神經網路(CNN)，以驗證資料集驗證其準確度。

經典卷積神經網路-AlexNet

- AlexNet主要提出兩點改進：
  - Dropout方法：在訓練過程中，會以一定機率讓神經網路節點失去活性，這樣訓練出來的神經網路能夠得到類似多模型整合的效果，緩解了模型的過擬合問題。
  - 資料增強：資料增強過程相當於增加了樣本的多樣性，使模型具有更強的泛化能力。

- 5層卷積和3個全連接層，總體參數達6000萬個。
- 2012年的 ImageNet 圖像分類競賽中，以錯誤率少10.9%贏過第二名，至此 CNN 開始受到研究者的强烈關注，之後的冠軍一直是 CNN。

經典卷積神經網路-VGGNet

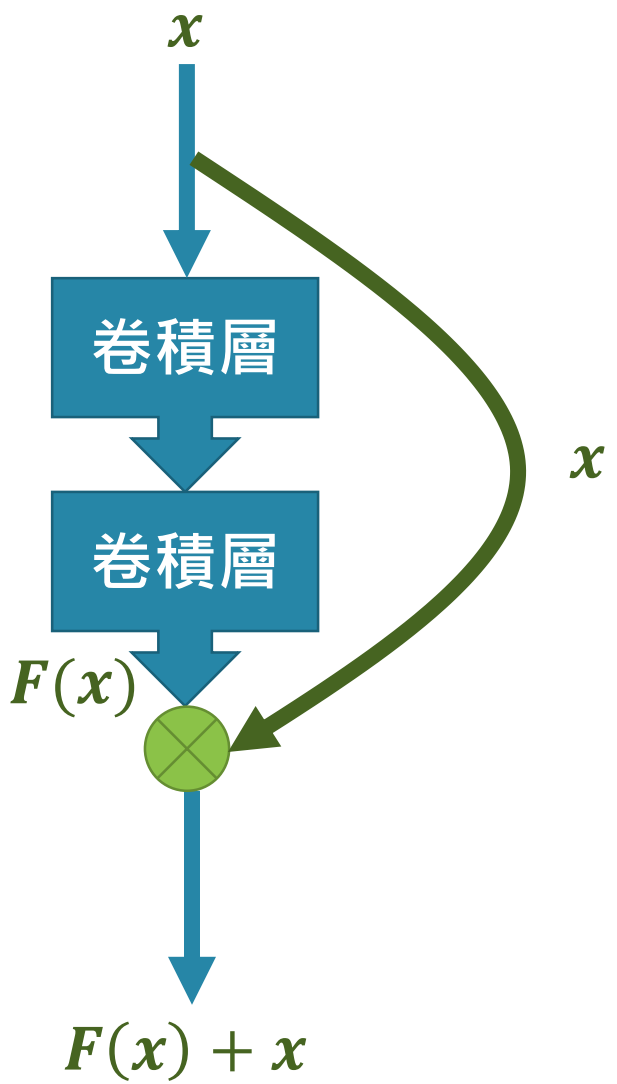
- VGGNet看做是一個加深版的AlexNet，模型總參數達到1.3億個，主要有兩點貢獻：
  - 證明小尺寸卷積核並增加網路深度，能有效提高模型效果。
    例：3層3x3卷積後的特徵圖和1層7x7卷積相同，但參數比較少，準確度也比較高。
  - 層數增加到19層時，無法靠增加層數來提高準確度。

經典卷積神經網路-GoogleNet

- GoogleNet改變依靠網路結構深度的想法，改變網路結構的廣度，主要有下列改進：
  - Inception：多個卷積或池化操作，放在一起組裝成一個網路模組，設計神經網路時以模組為單位去組裝整個網路結構，讓網路去選擇適合的卷積或池化。
  - 增加廣度後，使用1x1的卷積幫助降維，整體參數量只有AlexNet的1/12和VGGNet的1/25。

經典卷積神經網路-ResNet

- ResNet獨創的殘差結構，能夠有效的緩解梯度彌散問題，使網路層數達到100層時，仍能有效訓練，主要有下列改進：
  - 殘差模組：
  - ResNet 其實就是VGGNet的改良，在其多個路層間加入殘差模組。



遷移式學習(Transfer Learning)

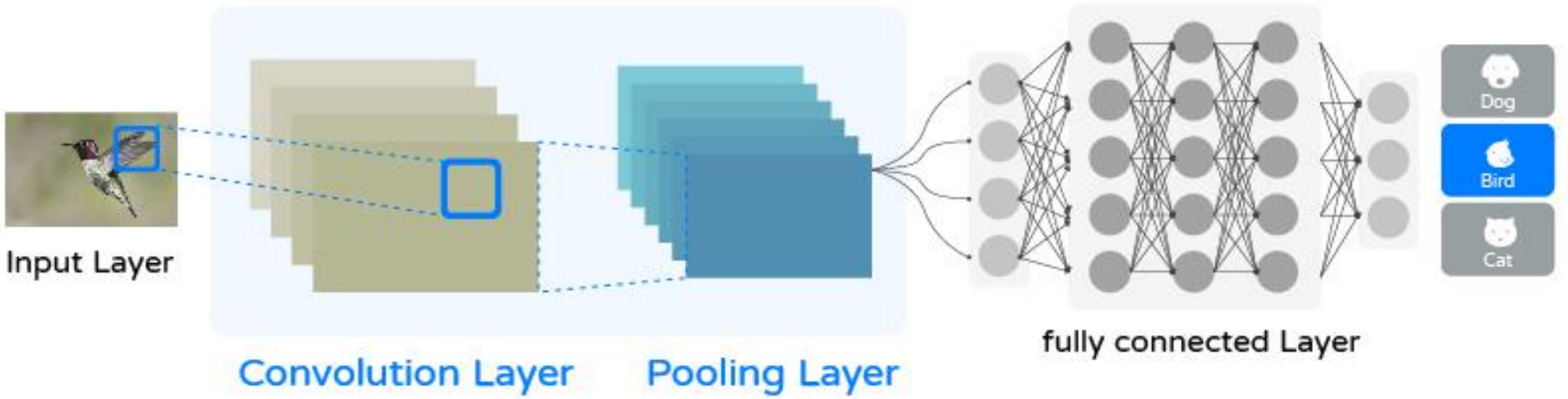
- 訓練一個良好的神經網路是一件花錢費時的工作：
  - 訓練前要收集大量的可用資料費，並做預處理。
  - 訓練時要搭建複雜的神經網路，更要有高效的硬體運算設備，才能節省訓練時間。
  - 訓練後要反覆調整網路架構，和訓練方式。

- 如果現實上沒錢沒時間的人，但又需要這一些良好的神經網路和訓練結果，可以藉找到相似的模型(稱為預訓練模型)來修改，以達到我們所需的功能，即為遷移學習式(Transfer Learning) 。

遷移式學習的原理

- 神經網路的最後一層輸出層是為了用途(迴歸、分類)而設計，扣除這層以外的每一層都是特徵工程的其中一步驟，都在找某一種代表特徵。

- 樣本資料輸入神經網路後，每一層節點的各數值都可以用來代表這個樣本的特徵向量，這個特徵向量可以代表樣本輸入自建的神經網路。



遷移式學習的應用方式

- 我們只需將預訓練模型的架構和找特徵的方法(個層的權重和偏值)，運用在我們的某層神經網路之內，或是對原始模型進行微調就能的到一個很好的結果。
- 遷移學習的應用方式分為兩種方式，分別是基於特徵(feature-based)與微調(fine-tuen)的方法。

- 基於特徵(feature-based):這一種做法是使用一個經過大數據訓練過的模型結果(通常是特徵值)，套入到模型的其中一層，並且通過自己的資料集，不斷的調整與學習資料。

- 微調(fine-tuen):這種方法會保留原始模型，讓我們新增資料去調整這個原始模型各階層之間的權重，之後通過一個額外的接口(fine-tunr中主要是訓練這個接口)，來實現各種不同的下游任務。

PyTorch 提供的預訓練模型

- PyTorch 官方將常見的神經網路模型整合在函式庫中，並且提供預訓練模型，供開發者直接利用。
- 預訓練模型網路架構包含 VGG、ResNet、、、，相關資訊可以再 https://pytorch.org/vision/0.8/models.html查看。
- 使用方法：

```
import torchvision.models as models  # 載入套件
# 只載入模型架構
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
# 載入模型架構和預訓練參數
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
```

PyTorch 提供的預訓練模型

- 預訓練模型都期望輸入圖像以相同的方式歸一化，即形狀為 (3 x H x W) 的 3 通道 RGB 圖像的小批量，其中 H 和 W 預計至少為 224。
- 圖像必須加載到 [0, 1] 的範圍內，然後使用 mean = [0.485, 0.456, 0.406] 和 std = [0.229, 0.224, 0.225] 進行歸一化。您可以使用以下轉換進行標準化：

```
normalize = transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
)
```