ANN

CH8
Keras卷積神經網路(CNN)辨識手寫數字

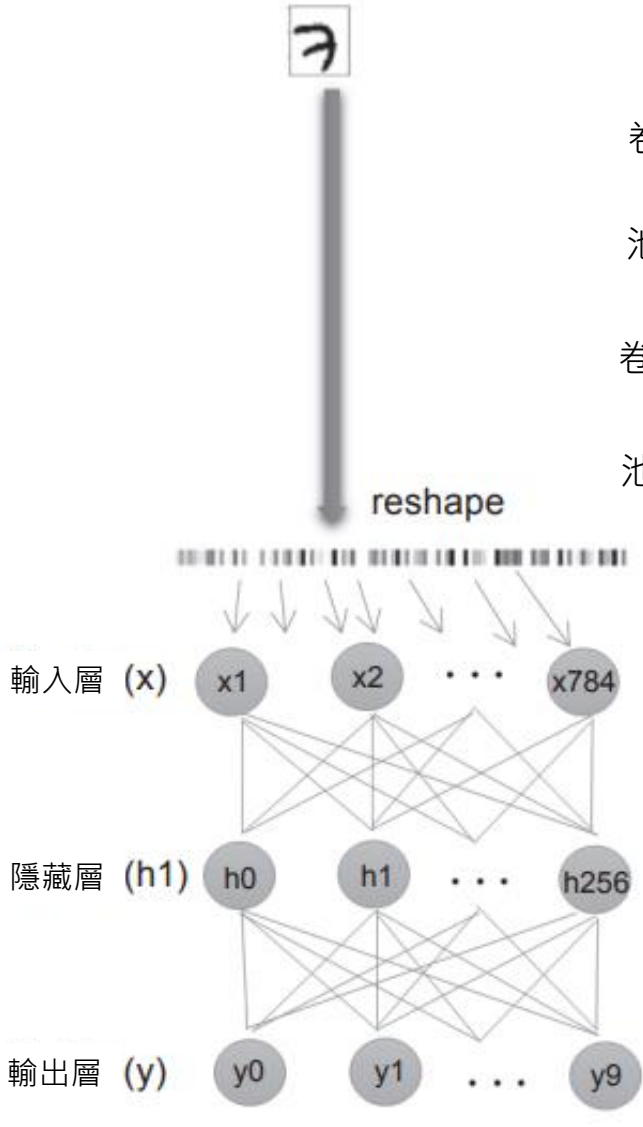
本章我們將介紹使用Keras建立卷積神經網路CNN (convolutional neural network)、然後訓練模型、評估模型準確率接近為99%、最後使用訓練完成的模型，辨識MNIST手寫數字。

8.1CNN卷積神經網路簡介

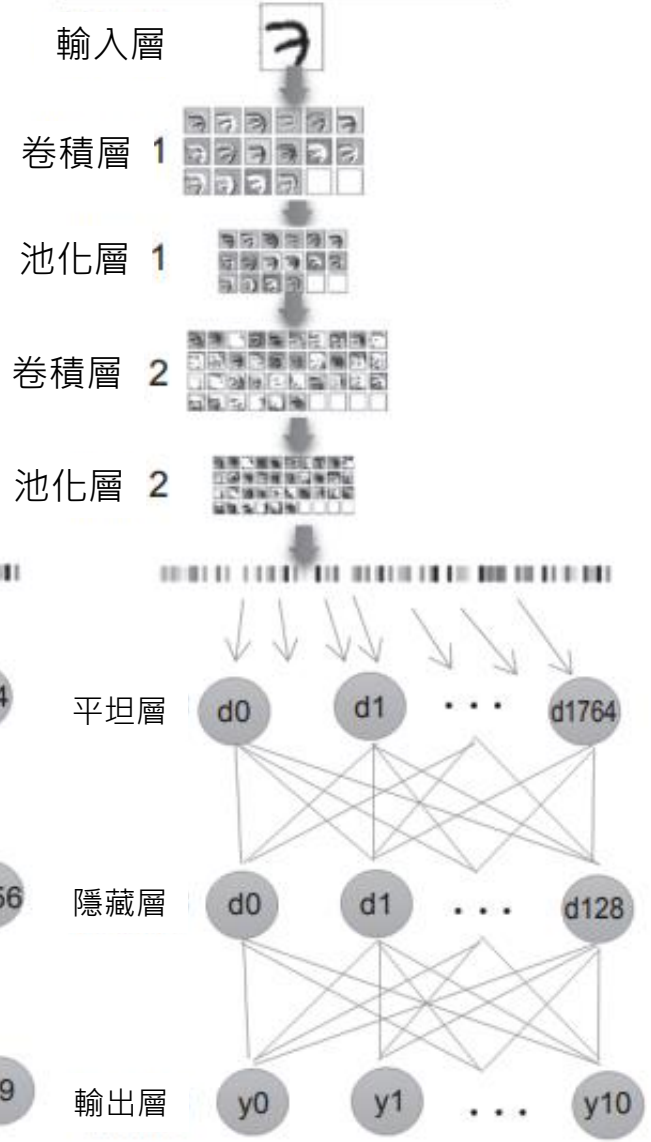
Step1.MLP多層感知器 vs. CNN卷積神經網路

主要的差異是提取特徵值的步驟(網路層)：
CNN增加了卷積層1、池化層1、卷積層2、池化層2來提取特徵

MLP多層感知器



CNN卷積神經網路



卷積層與池化層(提取影像特徵)

由多個卷積層與池化層所組成，能提取影像中很多的不同特徵，例如數字影像7能提取：橫線、斜線、轉折..等特徵。

全連接層(分類)

由平坦層、隱藏層、輸出層所組成，其功能是分類。

Step2. CNN 卷積神經網路介紹

CNN 可分為2大部分

1. **影像的特徵提取**
   透過卷積層1、池化層1、卷積層2、池化層2，提取影像的特徵

2. **完全連結神經網路**
   包含平坦層、隱藏層、輸出層所組成的類神經網路

卷積層與池化層：提取特徵(橫線)

影像(x)

卷積層

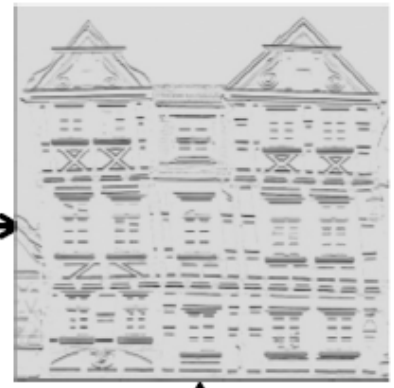
1	2	1
0	0	0
-1	-2	-1

提取橫線特徵的濾鏡

Feature map 特徵圖

池化層

Feature map特徵圖



1.影像：有各種橫線、直線、斜線等特徵

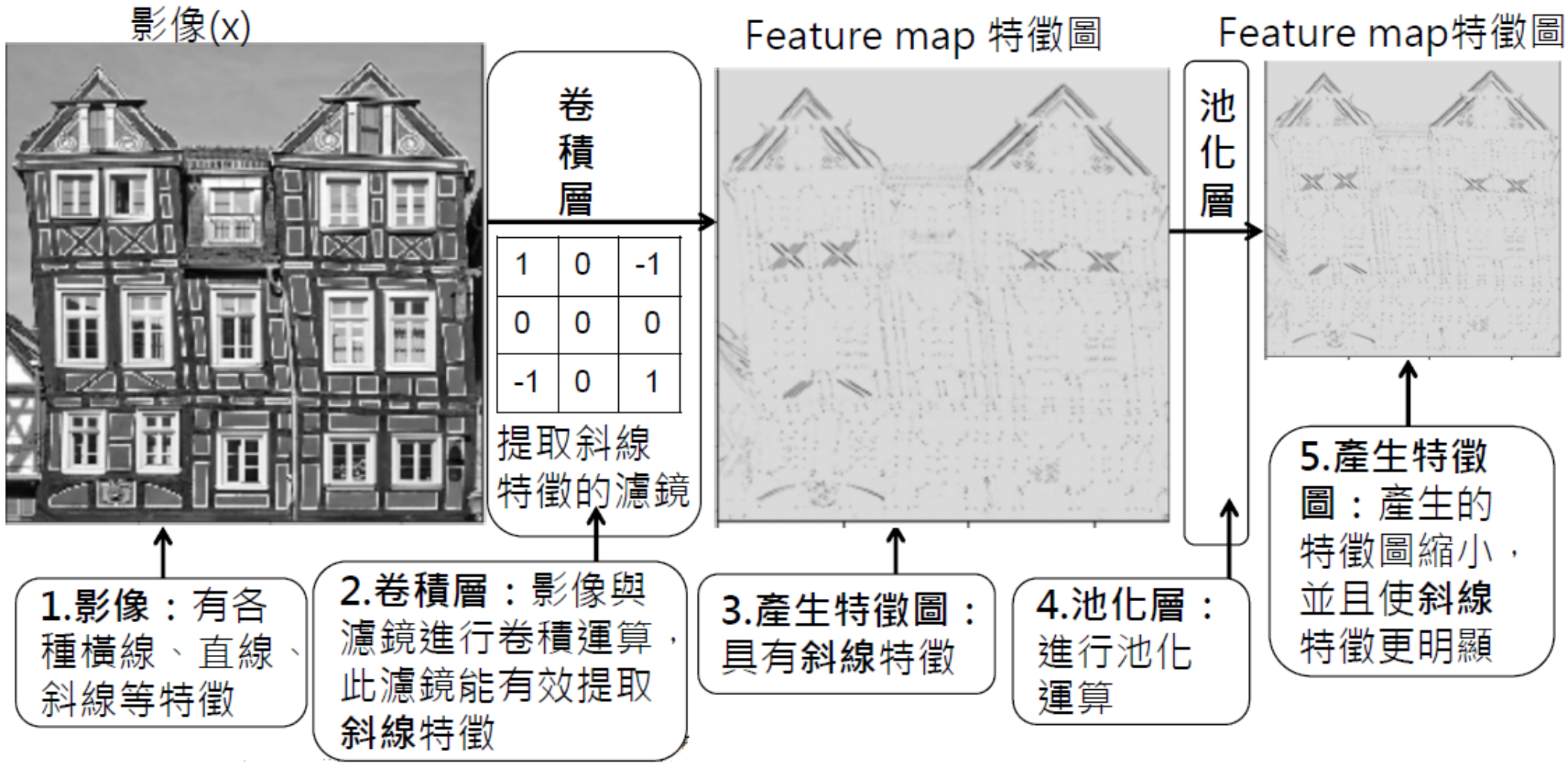
2.卷積層：影像與濾鏡進行卷積運算，此濾鏡能提取**橫線**特徵

3.產生特徵圖：具有**橫線**特徵

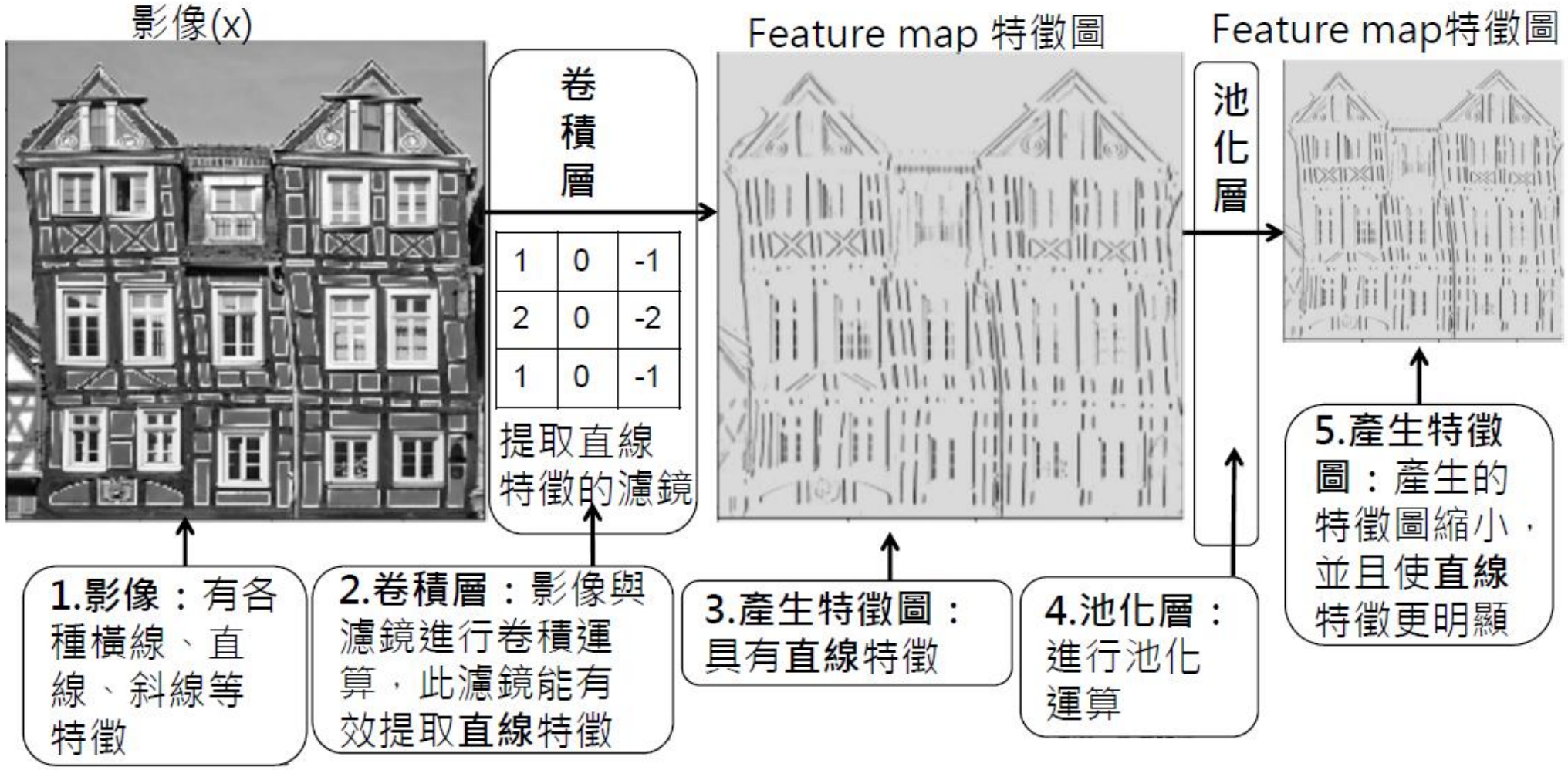
4.池化層：進行池化運算

5.產生特徵圖：產生的特徵圖縮小，並且使**橫線**特徵更明顯

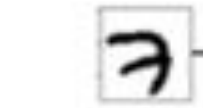
卷積層與池化層：提取特徵(斜線)



卷積層與池化層：提取特徵(直線)



輸入層
input
28X28影像共1層



輸入的數字影像28×28大小，例如:數字7的影像。

第1次卷積

卷積層1
28X28影像
共16層

第1次卷積運算：輸入的數字影像28×28大小，會產生16個影像，卷積運算並不會改變影像大小，所以仍然是28×28大小。

第1次縮減取樣

池化層1
14X14影像
共16層

第1次縮減取樣:將16個28X28影像,縮小為16個14X14的影像。

第2次卷積

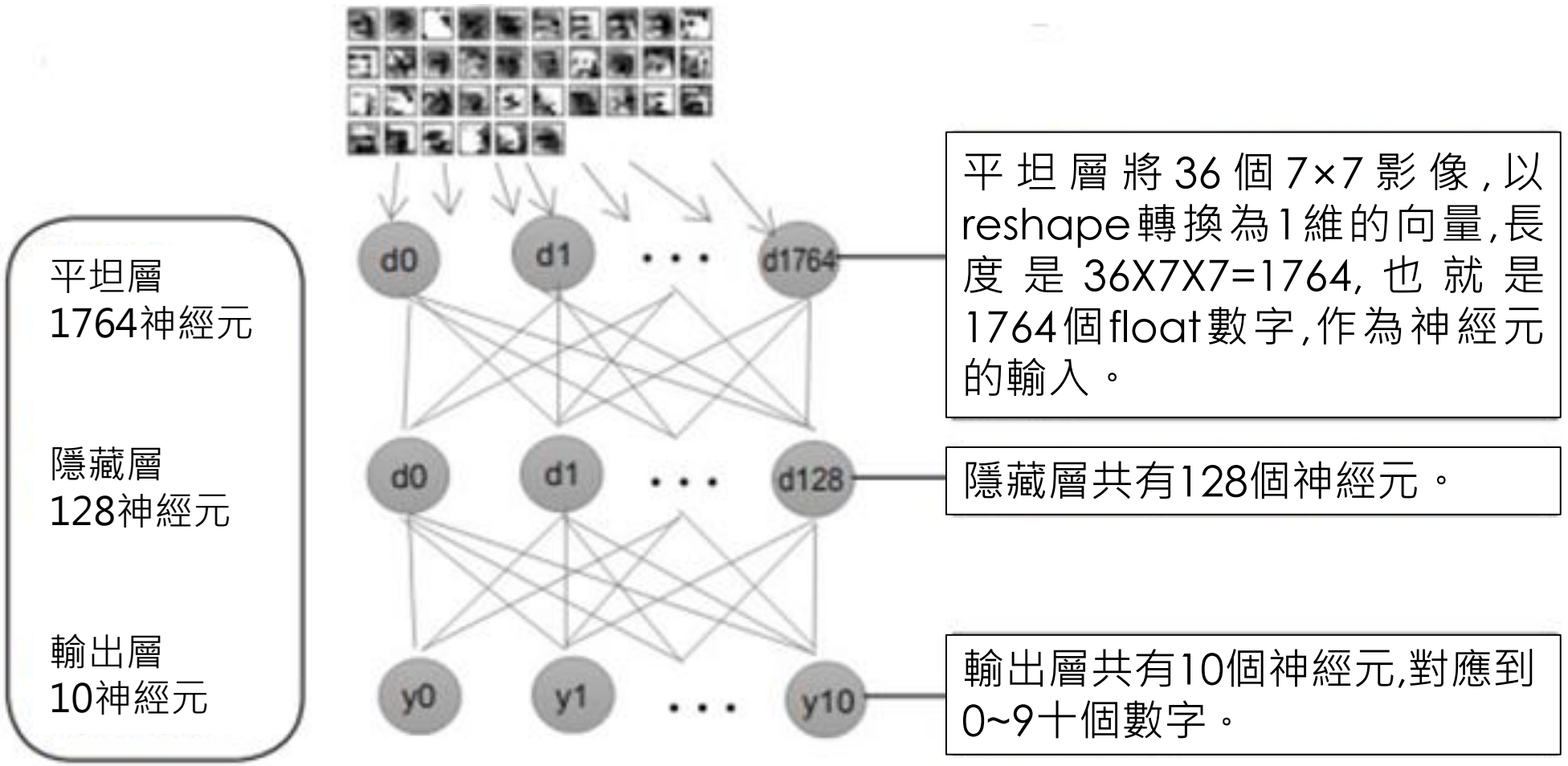
卷積層2
14X14影像
共66層

第2次卷積運算：將原本16個的影像，轉換為36個影像，卷積運算不會改變影像大小，所以仍然是大小14X14

第2次縮減取樣

池化層2
7X7影像
共36層

第2次縮減取樣：將36個14X14影像，縮小為36個7×7的影像

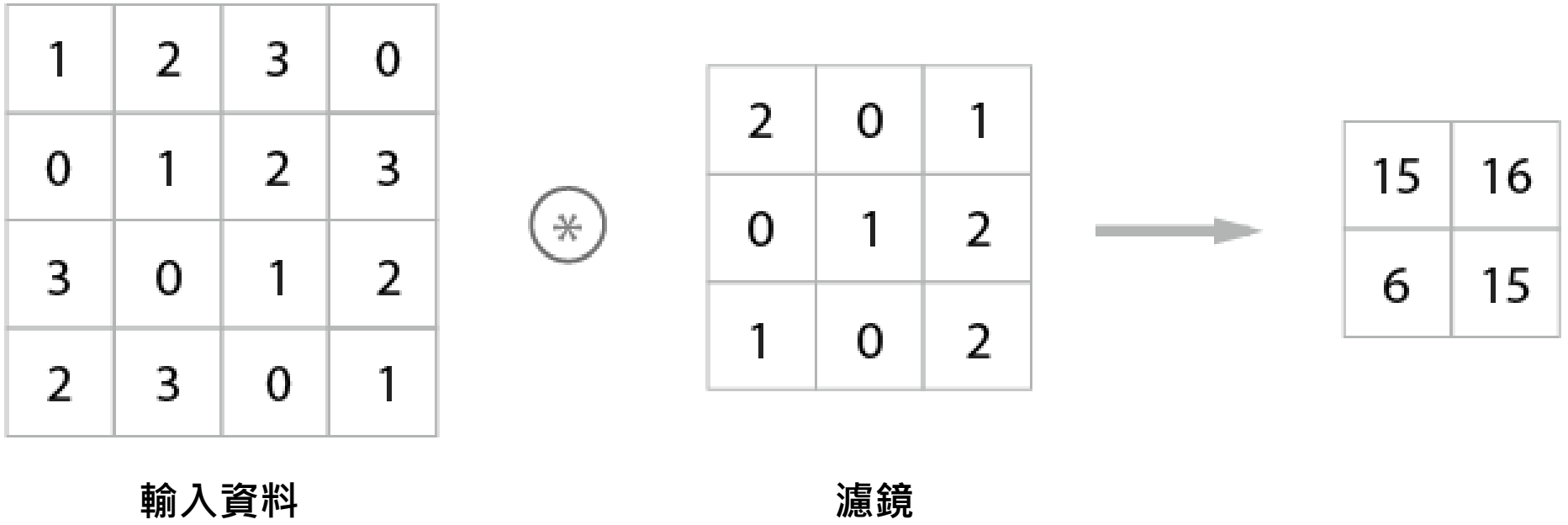


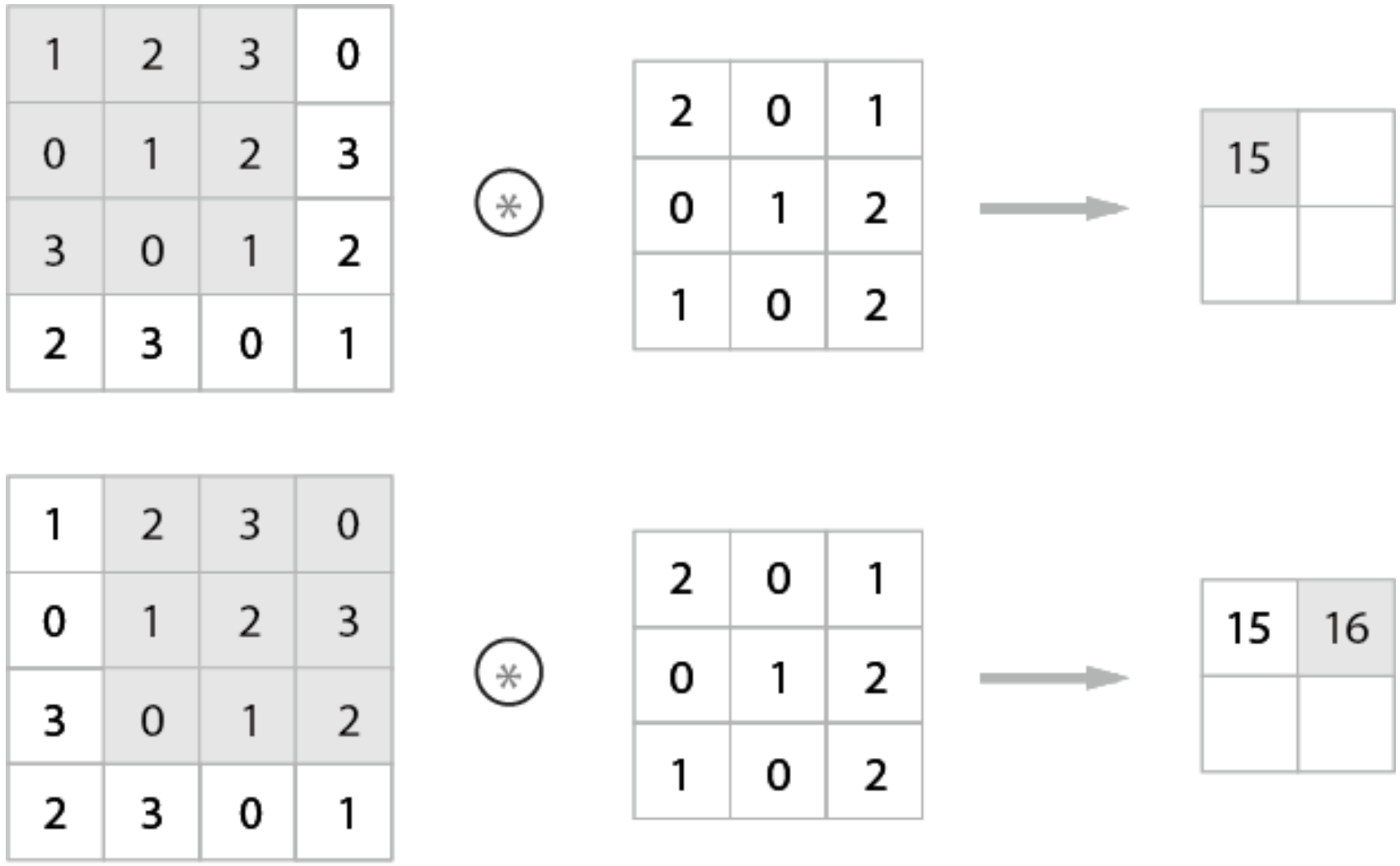
以上你可以看到這些影像擷取了「7」的影像的特徵,卷積運算的效果很類似濾鏡效果，擷取了不同的特徵。

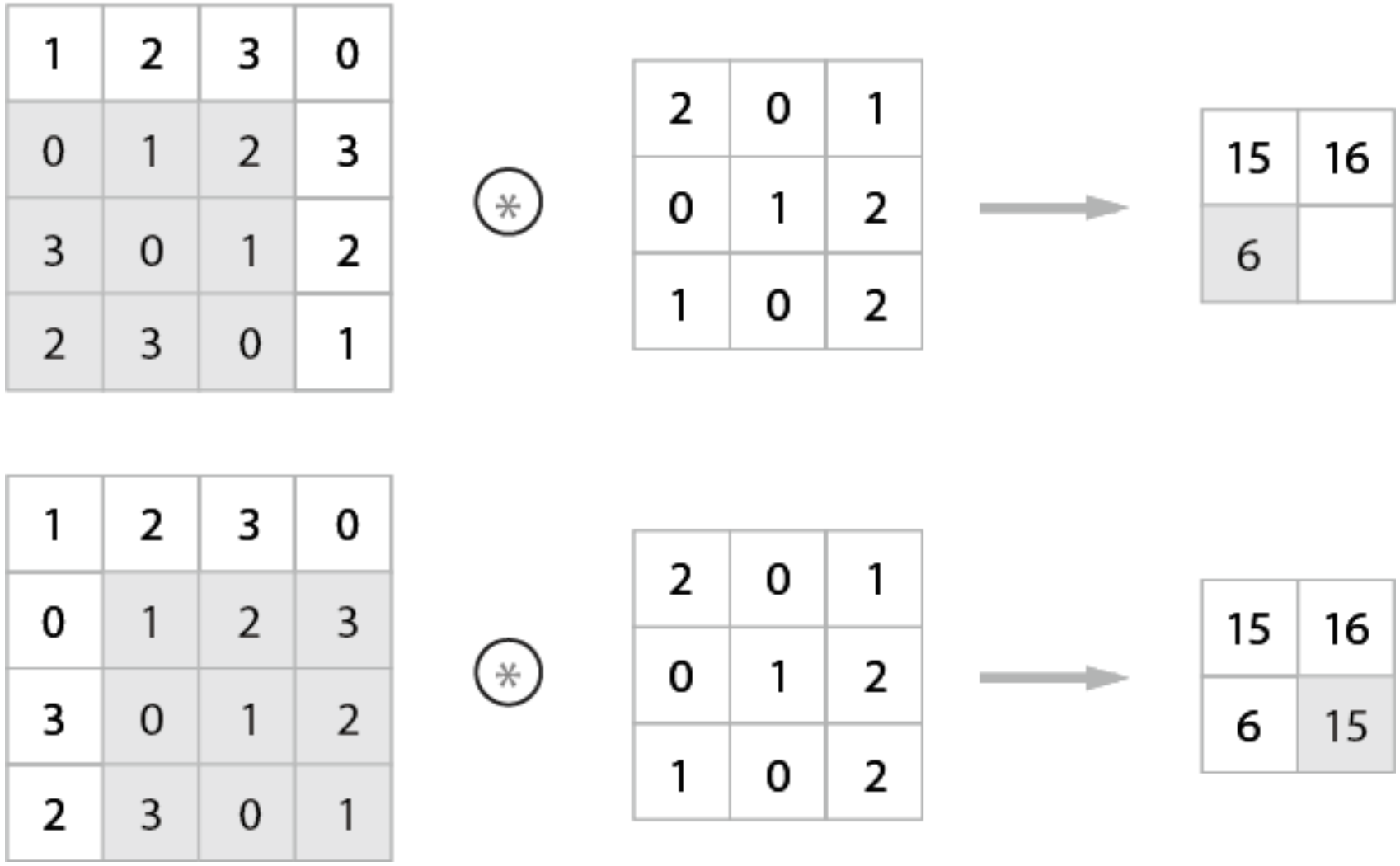
卷積運算介紹

卷積層的意義是將原本一個影像，經過卷積運算(濾鏡)，產生多個影像，就好像將相片卷積起來

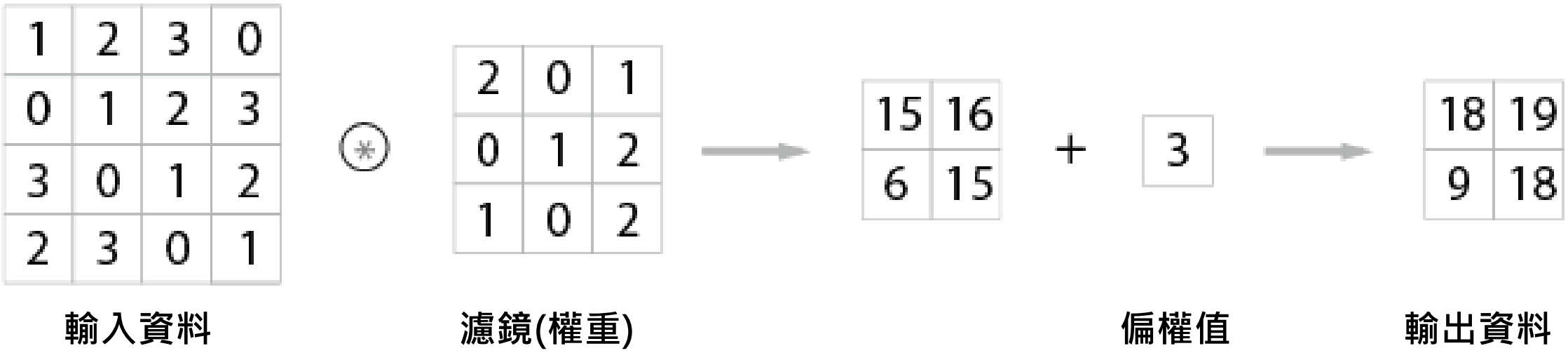
執行矩陣與濾鏡的對應元素相乘，並計算總合(也稱做**積和**運算)

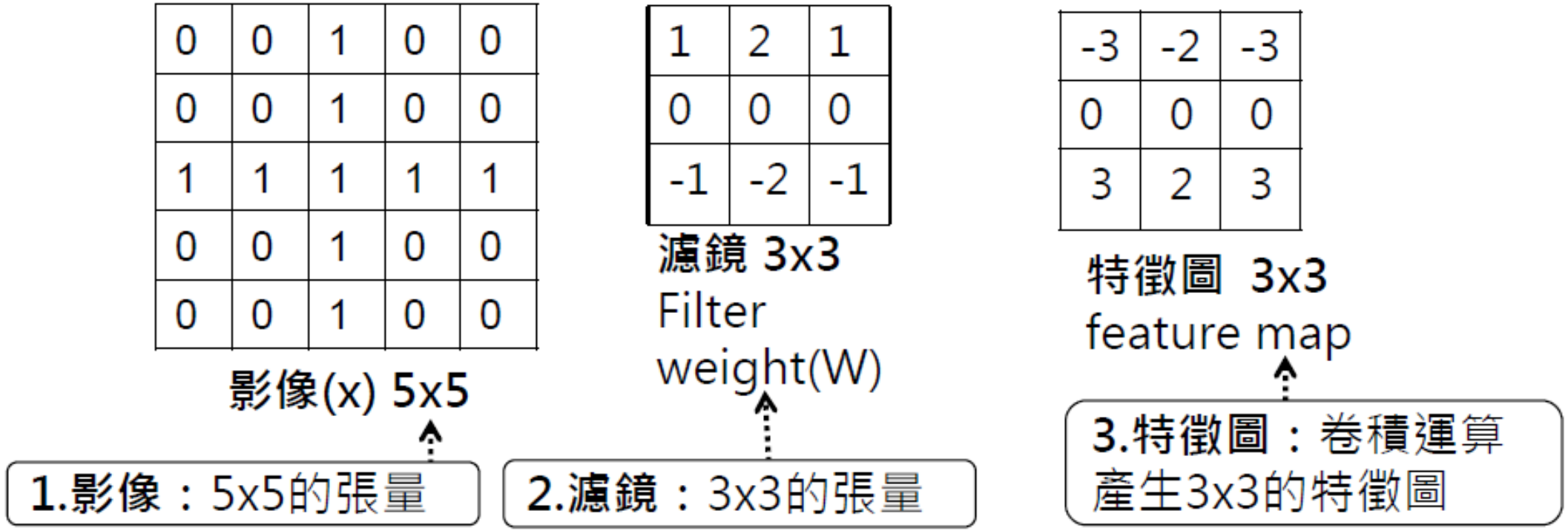


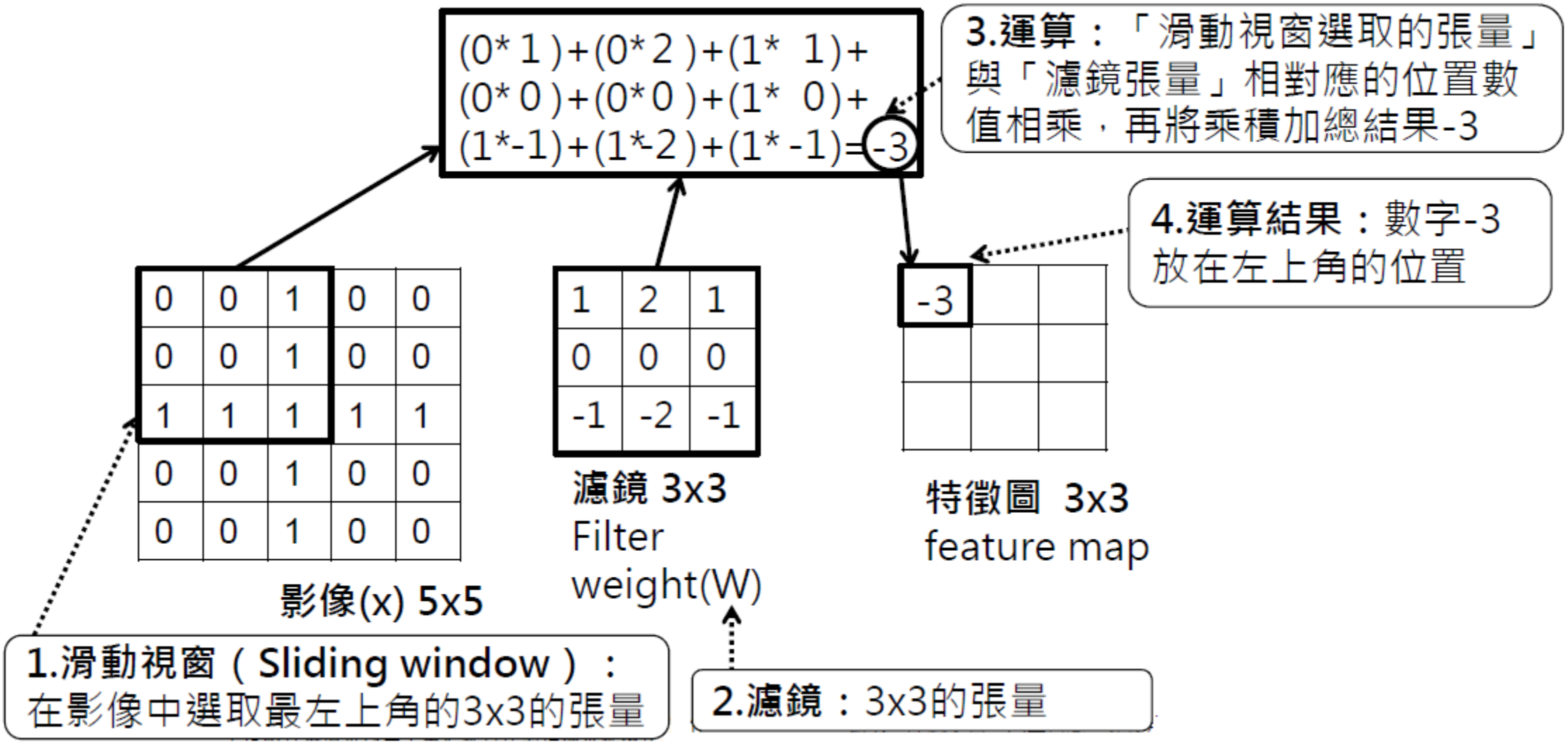

1 2 3 0
0 1 2 3
3 0 1 2
2 3 0 1
*
2 0 1
0 1 2
1 0 2
15
1 2 3 0
0 1 2 3
3 0 1 2
2 3 0 1
*
2 0 1
0 1 2
1 0 2
15 16

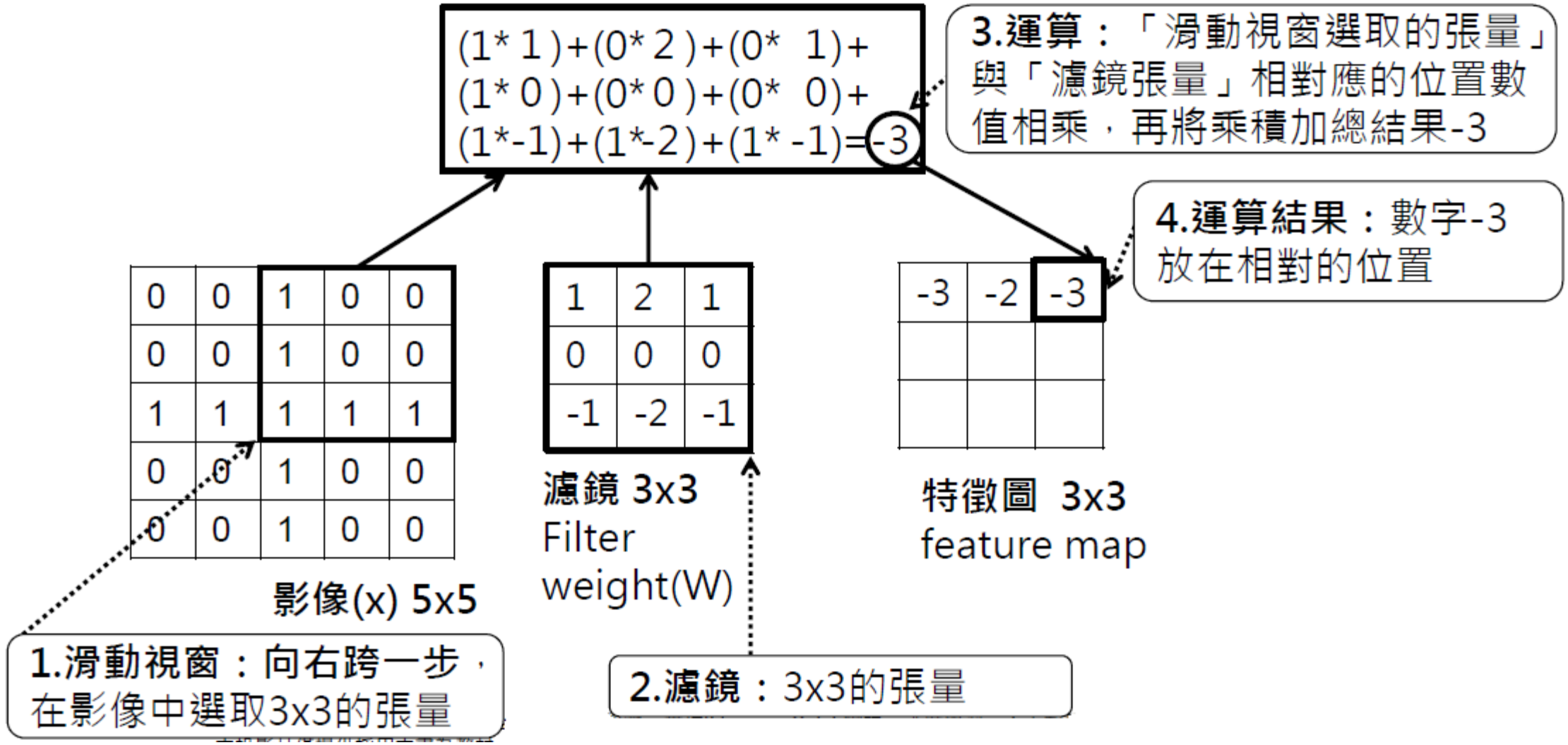

1 2 3 0
0 1 2 3
3 0 1 2
2 3 0 1
*
2 0 1
0 1 2
1 0 2
15 16
6
1 2 3 0
0 1 2 3
3 0 1 2
2 3 0 1
*
2 0 1
0 1 2
1 0 2
15 16
6 15

如果有偏權值，處理流程如下

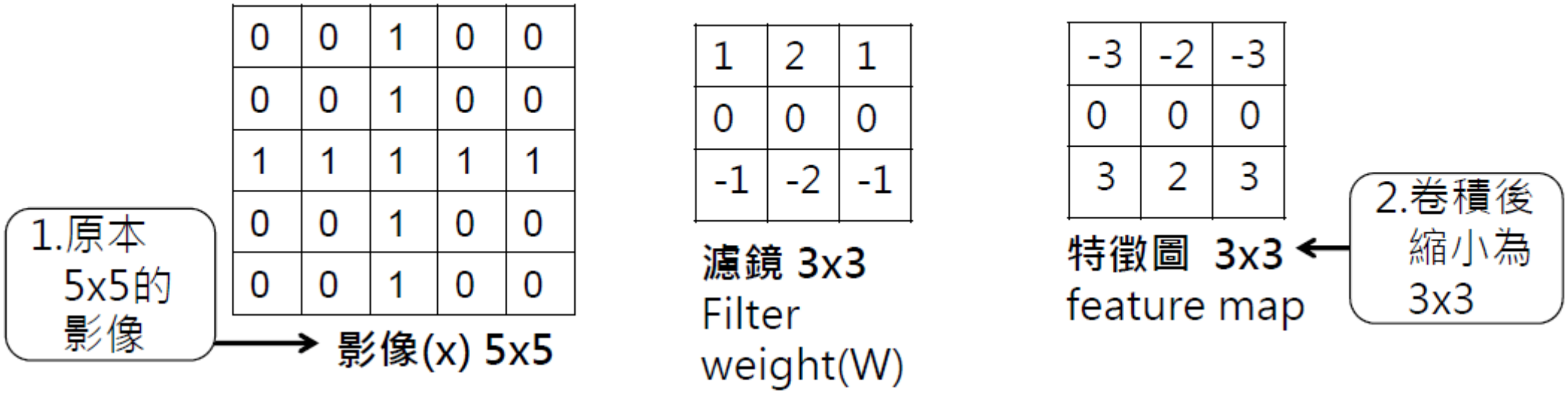



0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
影像(x) 5x5
1 2 1
0 0 0
-1 -2 -1
濾鏡 3x3
Filter
weight(W)
-3 -2 -3
0 0 0
3 2 3
特徵圖 3x3
feature map
1.影像：5x5的張量
2.濾鏡：3x3的張量
3.特徵圖：卷積運算
產生3x3的特徵圖


(0*1)+(0*2)+(1*1)+
(0*0)+(0*0)+(1*0)+
(1*-1)+(1*-2)+(1*-1)=-3
3.運算：「滑動視窗選取的張量」與「濾鏡張量」相對應的位置數值相乘，再將乘積加總結果-3
4.運算結果：數字-3放在左上角的位置
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
影像(x) 5x5
1 2 1
0 0 0
-1 -2 -1
濾鏡 3x3
Filter
weight(W)
-3
特徵圖 3x3
feature map
1.滑動視窗（Sliding window）：
在影像中選取最左上角的3x3的張量
2.濾鏡：3x3的張量


(1*1)+(0*2)+(0*1)+
(1*0)+(0*0)+(0*0)+
(1*-1)+(1*-2)+(1*-1)=-3
3.運算：「滑動視窗選取的張量」與「濾鏡張量」相對應的位置數值相乘，再將乘積加總結果-3
4.運算結果：數字-3放在相對的位置
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
影像(x) 5x5
1 2 1
0 0 0
-1 -2 -1
濾鏡 3x3
Filter
weight(W)
-3 -2 -3
特徵圖 3x3
feature map
1.滑動視窗：向右跨一步，在影像中選取3x3的張量
2.濾鏡：3x3的張量

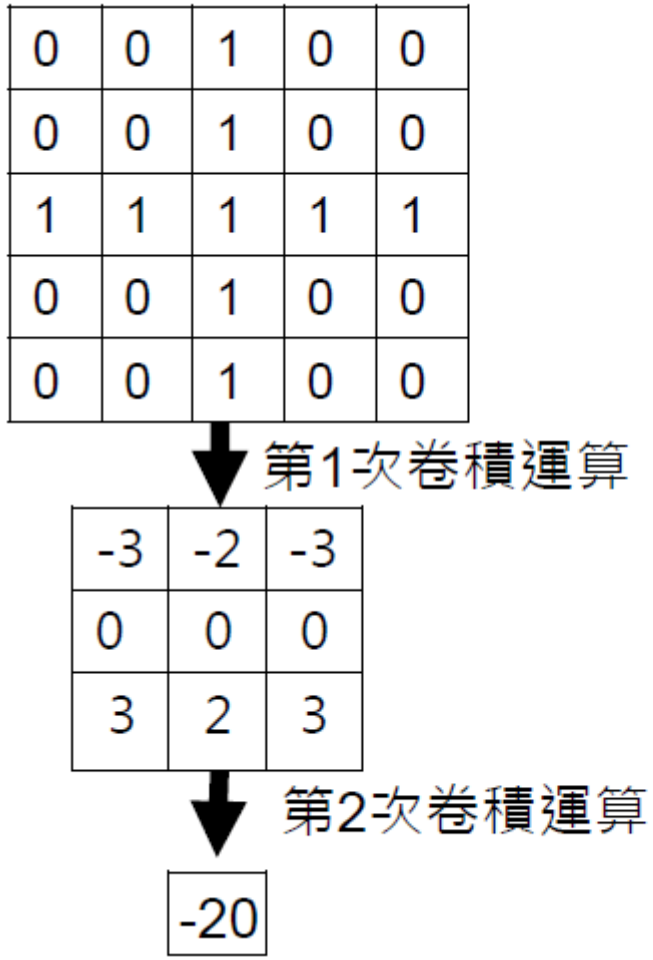
卷積運算處理後，縮小特徵圖的問題



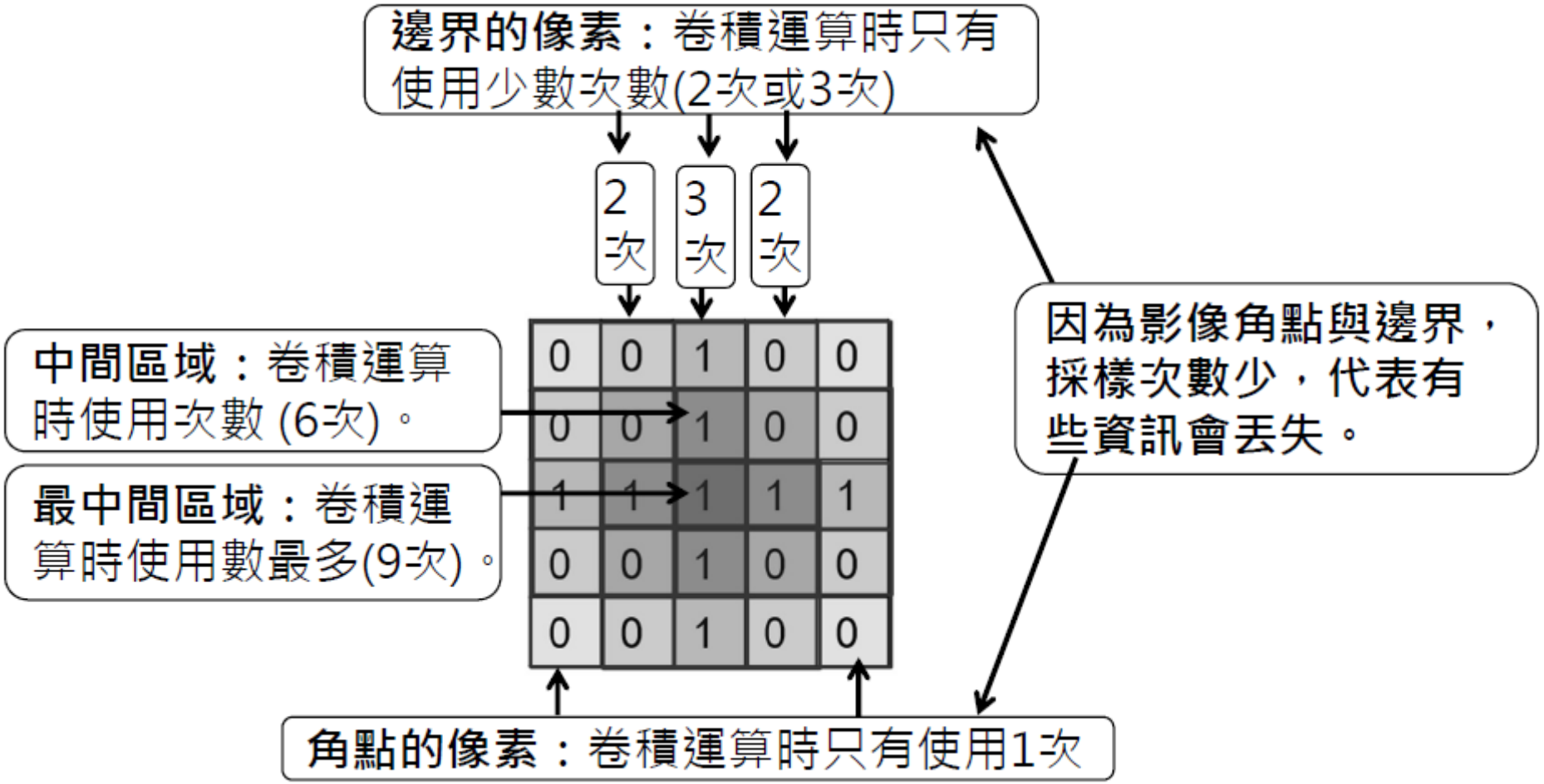
我們進行卷積運算，主要是希望提取特徵，而不是縮小。
上一節介紹卷積運算後，產生的特徵圖會縮小，這會有什麼問題呢?
主要以下二點：

- **無法加上很多卷積層**
- **原始影像角點與邊界丟失資訊**

卷積運算縮小特徵圖：無法加上很多卷積層

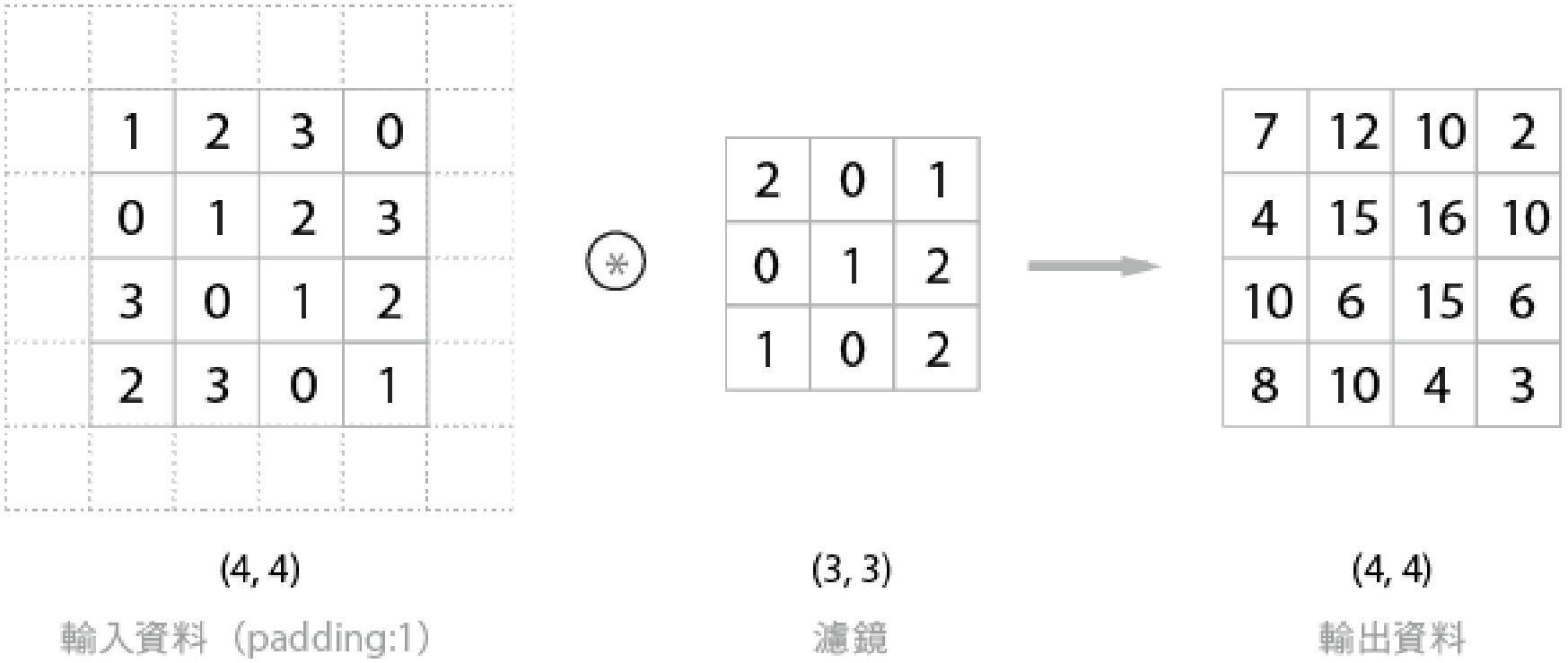


卷積運算縮小特徵圖：影像角點與邊界丟失資訊



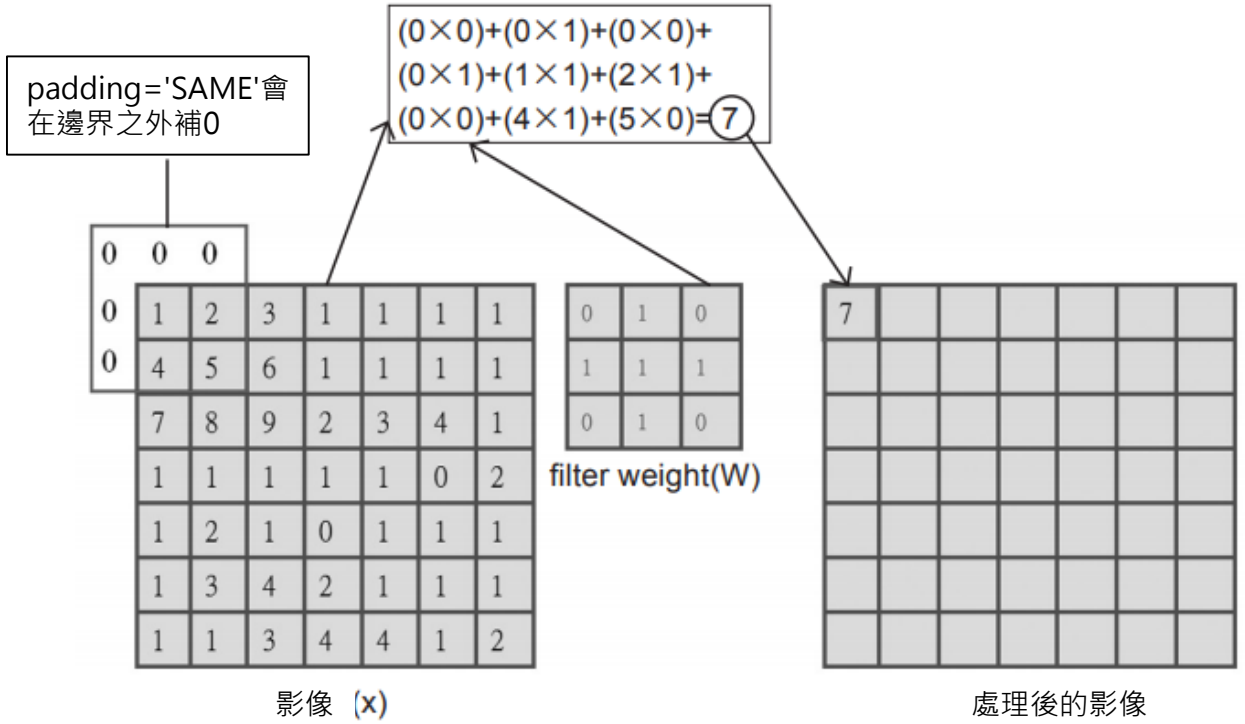
卷積運算 – 填補(padding)

對於 4×4 的矩陣進行寬度 1 的填補(周圍增加一像素的 0 補滿)

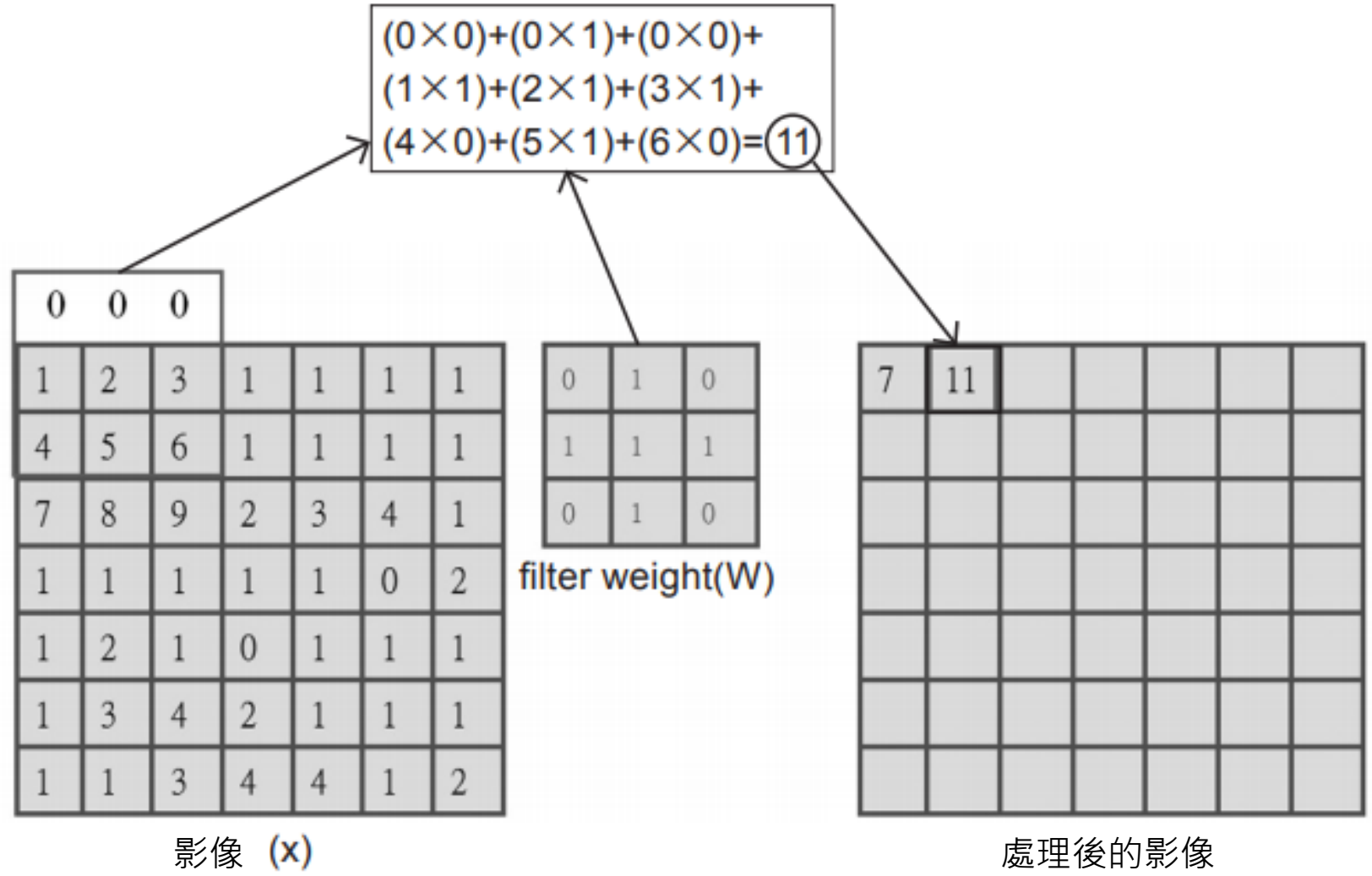


- **卷積運算:**

1. 先以隨機的方式產生 filter weight ，大小是 3x3
2. 要轉換的影像由左而右、由上而下，依序選取 3x3 的矩陣
3. 影像選取的矩陣 (3x3) 與 filter weight(3x3)，計算產生第1列、第1行的數字



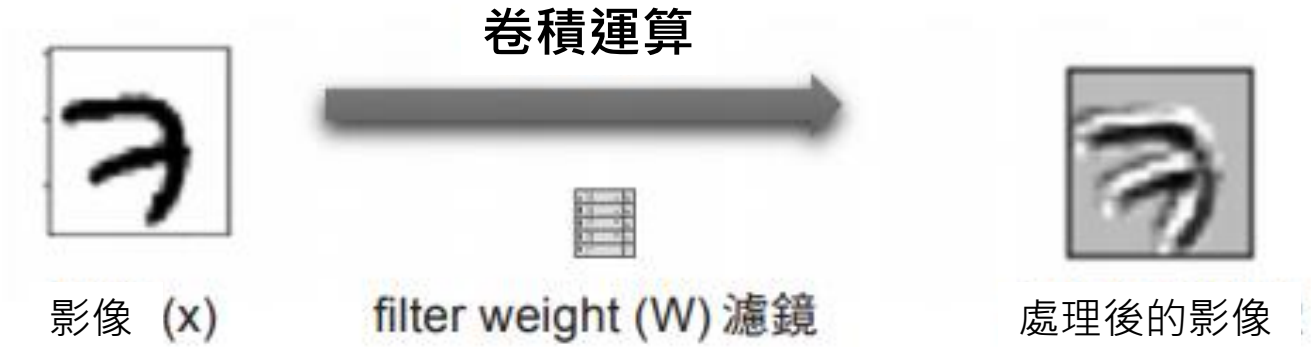
再以相同的方式，計算第1列、第2行的數字。



依照上列相同的方式，依序完成所有的運算，就完成影像的處理。

Step4. 使用單一filter weight卷積運算產生影像

如下圖，將數字影像7是28X28大小，使用隨機產生5X5的filter weight(W)濾鏡，進行卷積運算。



卷積運算並不會改變影像大小，所以處理後的影像，仍然是28X28大小。卷積運算後的效果，很類似濾鏡效果。這可以幫助提取輸入的不同特徵，例如：邊緣、線條和角等。

原圖

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	1	1	1	1	1	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0

卷積後的特徵圖

0	-1	-2	-1	0
-3	-3	-2	-3	-3
0	0	0	0	0
3	3	2	3	3
0	1	2	1	0

加入bias後的結果

-1	-2	-3	-2	-1
-4	-4	-3	-4	-4
0	0	0	0	0
2	2	1	2	2
-1	0	1	0	-1

ReLU轉換後的特徵圖
能提取橫線特徵

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
2	2	1	2	2
0	0	1	0	0

濾鏡
(可訓練參數)

1	2	1
0	0	0
-1	-2	-1

卷積運算

卷積層

+ -1 bias
(可訓練參數)

ReLU(激活函數)
轉換

1.卷積運算：
原圖片與濾鏡(filter weight)進行卷積運算

2. +bias 偏差:卷積後的特徵圖，每一個數值加入bias偏差，調整影像(例如bias=-1，所有數值都減1)

3.ReLU激活函數轉換:
轉換後負數會變成0，正數不變，可以進一步提取特徵

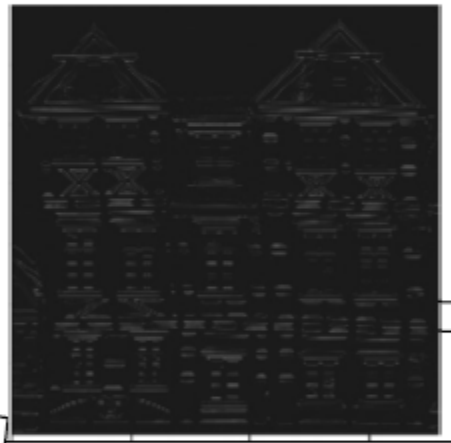
1.卷積運算提取橫線的特徵：
提取橫線特徵，不過不太明顯

2. +bias 偏差調整影像：
每一個數值加入bias偏差，調整影像

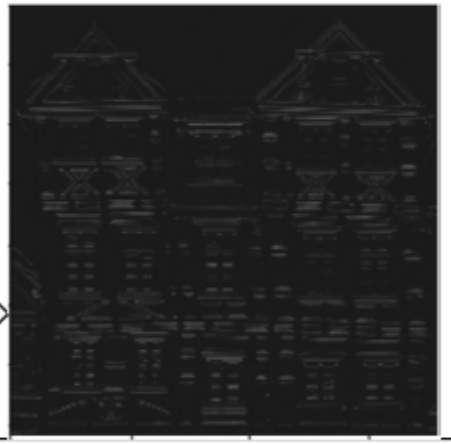
3.ReLU激活函數轉換:
轉換後負數會變成0，正數不變。讓橫線的特徵更明顯

原圖

卷積後的特徵圖



加入bias後的結果



ReLU轉換後的特徵圖

濾鏡
(可訓練參數)

1	2	1
0	0	0
-1	-2	-1

卷積運算

卷積層

+ -1 bias
(可訓練參數)

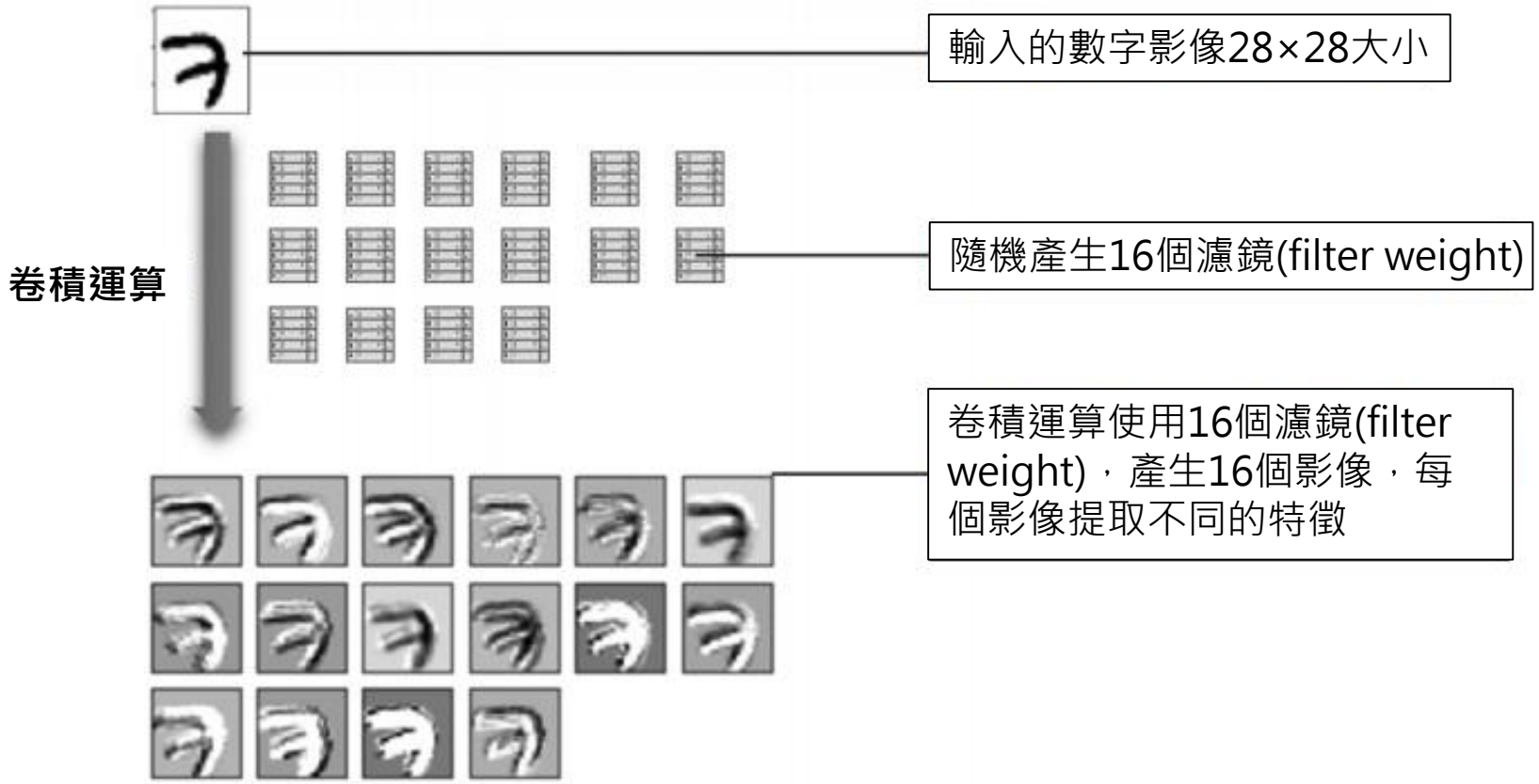
ReLU
轉換

(激活函數)

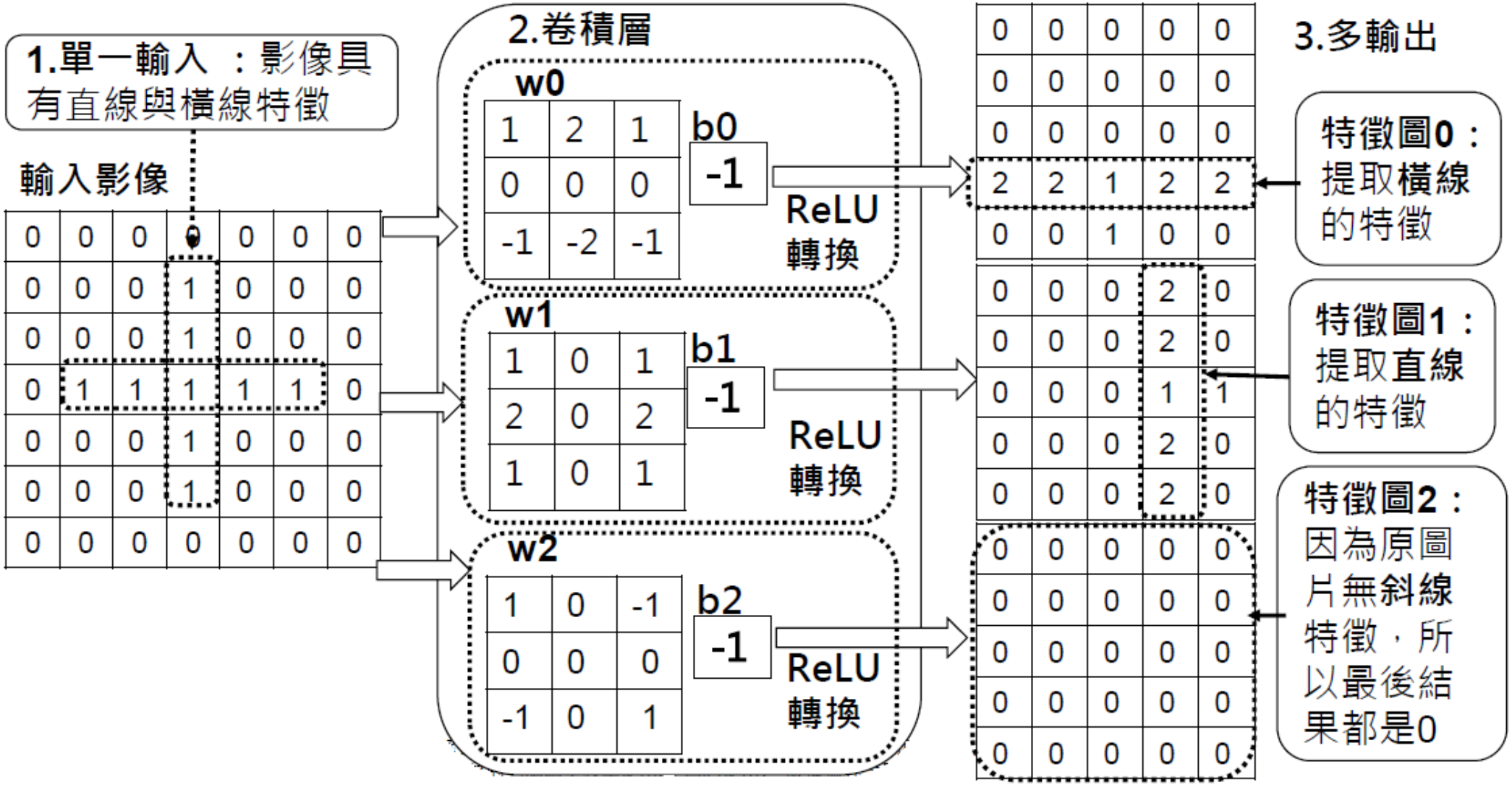
Step5. 使用多個filter weight卷積運算產生多個影像

接下來，我們將隨機產生16個(filter weight)，也就是16個濾鏡。

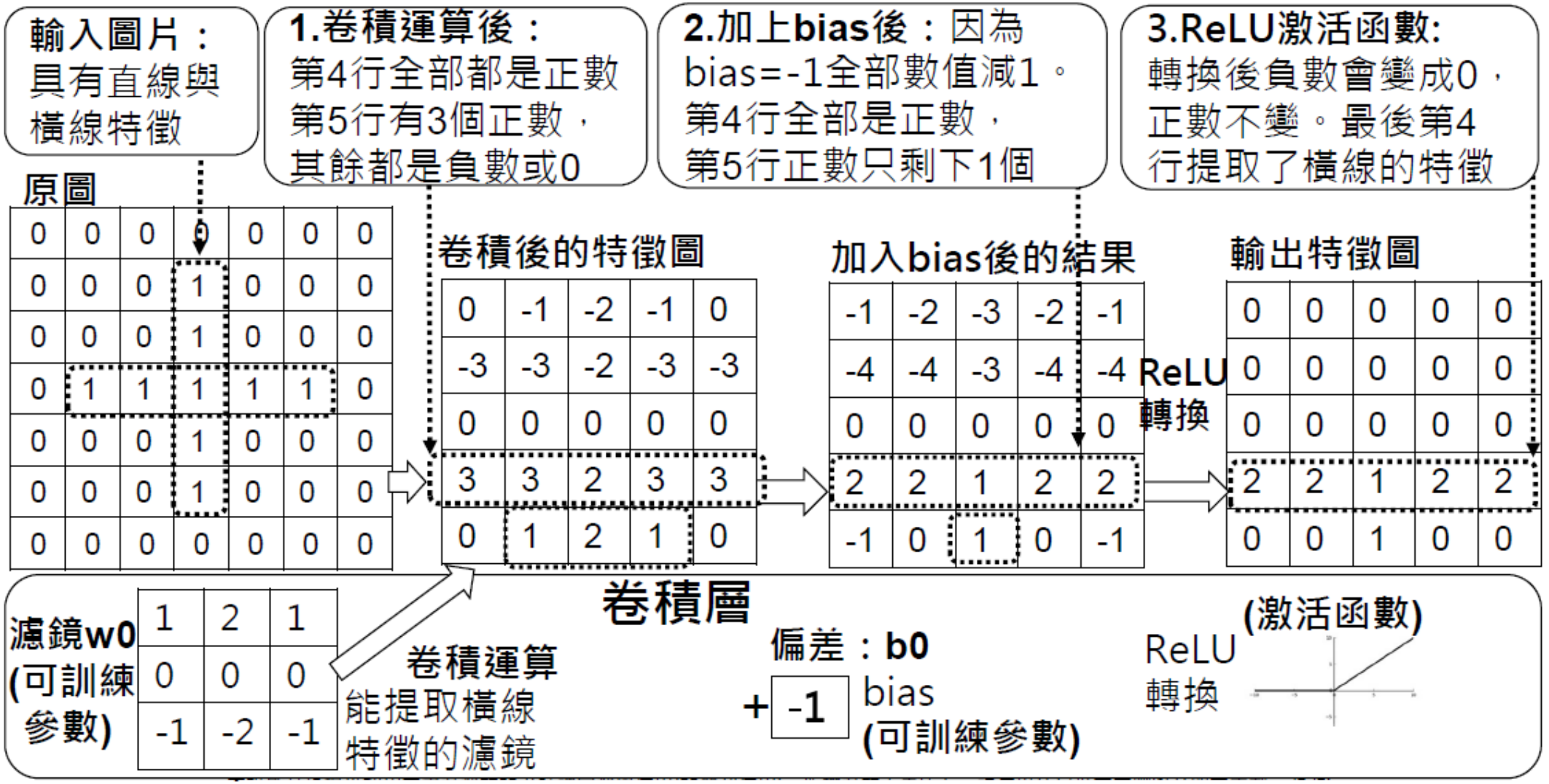
卷積運算使用16個濾鏡(filter wight)，產生16個影像，每個影像提取不同的特徵。



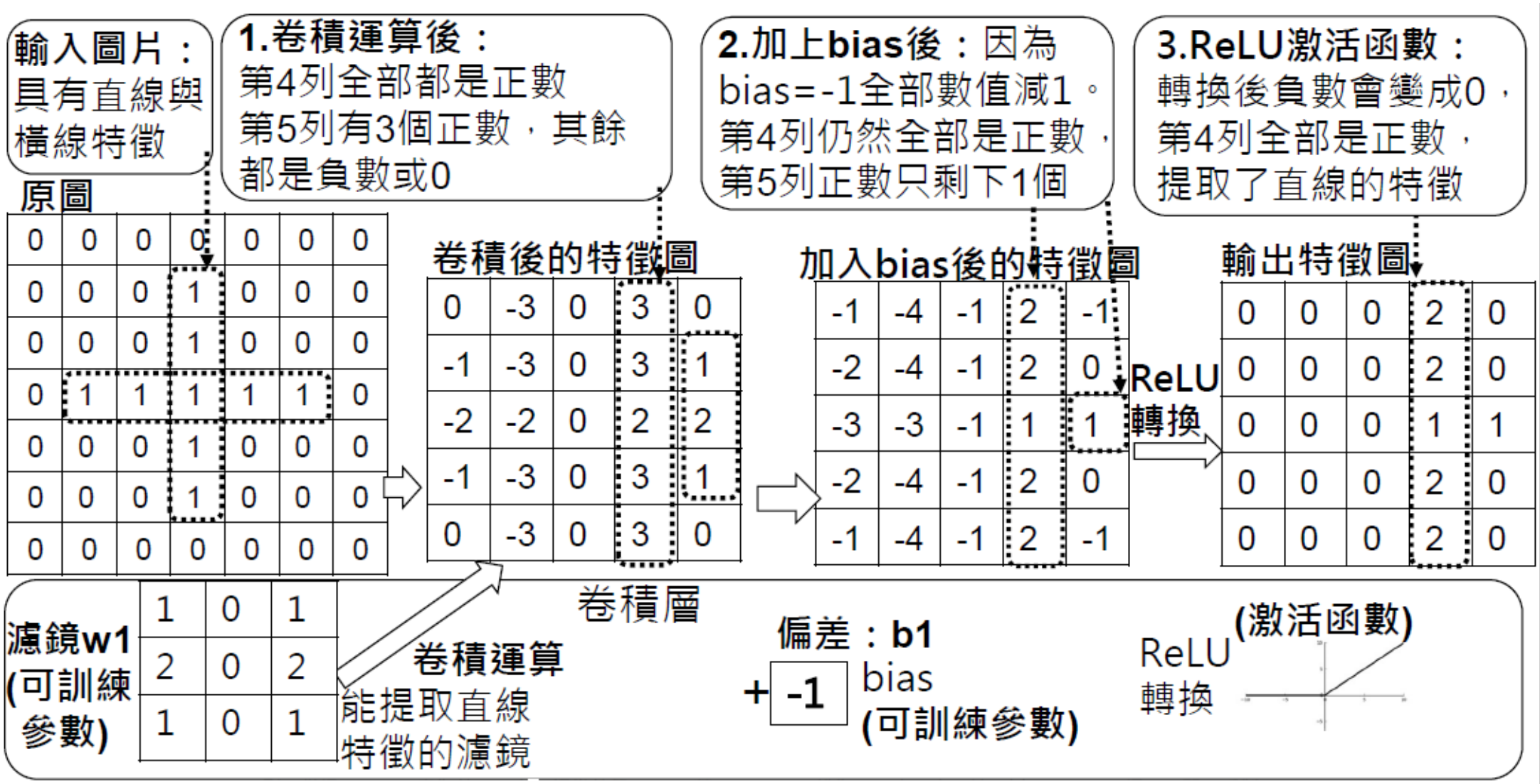
卷積運算：使用多個濾鏡，產生多個影像，提取多個特徵



卷積運算：使用多個濾鏡，產生多個影像，提取多個特徵 (橫線)



卷積運算：使用多個濾鏡，產生多個影像，提取多個特徵 (直線)



卷積運算：使用多個濾鏡，產生多個影像，提取多個特徵 (斜線)

輸入圖片：
無斜線特徵

1.卷積運算後：
全部數值是1或0或-1。

2.加上bias後：
因bias=-1全部數值減1，全部數值變成負數

3.ReLU激活函數:
轉換後負數會變成0，
因為原圖片無斜線特徵，
所以最後結果都是0

輸入圖片

0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	1	1	1	1	1	0
0	0	0	1	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0

卷積後的特徵圖

0	1	0	-1	0
1	-1	0	1	-1
0	0	0	0	0
-1	1	0	-1	1
0	-1	0	1	0

加入bias後的特徵圖

-1	0	-1	-2	-1
0	-2	-1	0	-2
-1	-1	-1	-1	-1
-2	0	-1	-2	0
-1	-2	-1	0	-1

ReLU轉換

輸出特徵圖

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

濾鏡w2
(可訓練參數)

1	0	-1
0	0	0
-1	0	1

卷積運算
能提取斜線特徵的濾鏡

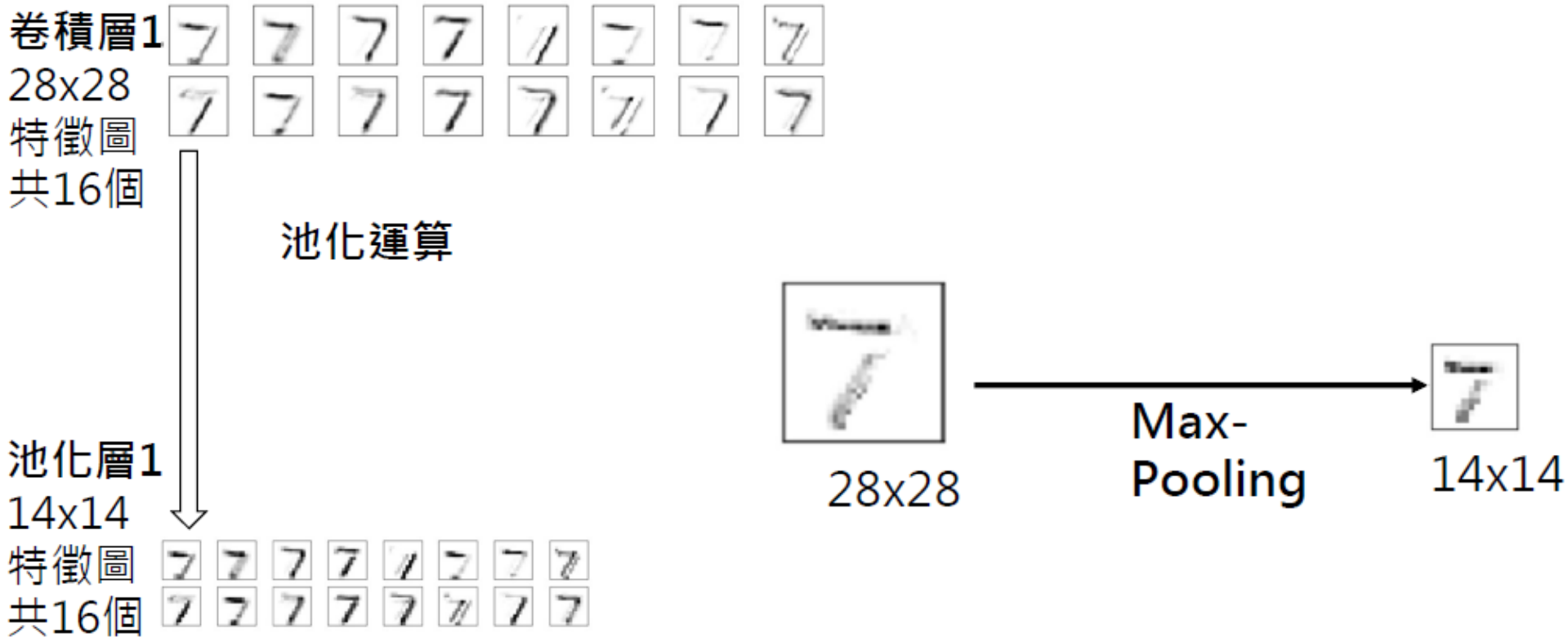
卷積層

偏差：b2
+ -1 bias
(可訓練參數)

ReLU轉換 (激活函數)

Step6. Pooling 運算

池化層功能介紹



Step6. Pooling 運算

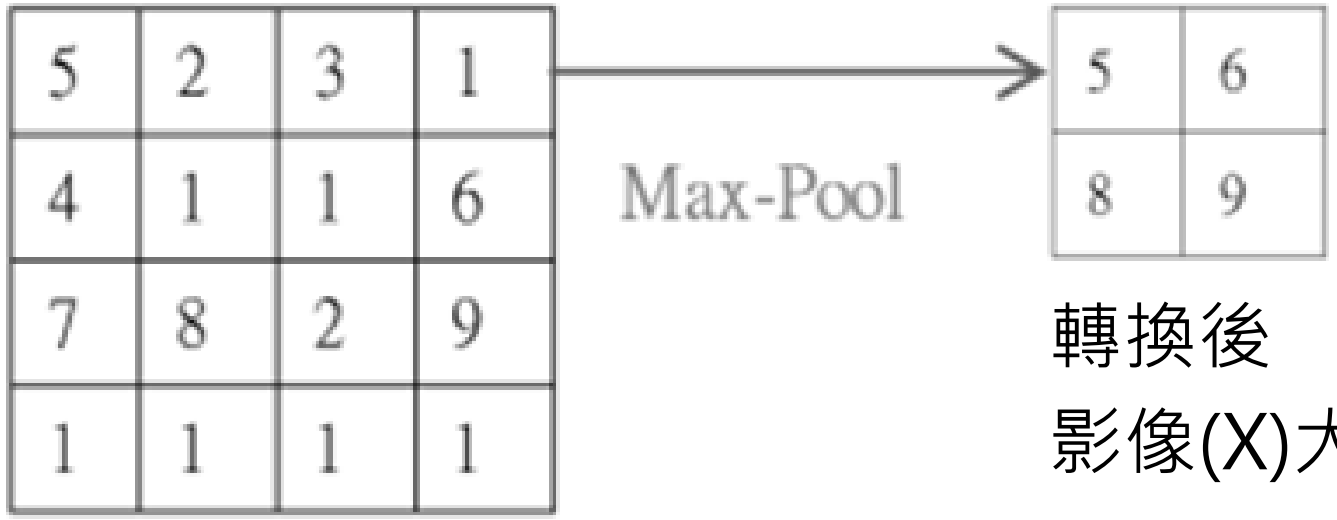
Pooling運算"不"訓練參數，只是單純的運算
但是具有以下功能

- 縮小特徵圖：讓特徵圖更小，更容易快速處理
- 減少訓練參數：特徵圖縮小，減少網路所需訓練參數，加快訓練速度
- 減少 Overfitting：減少訓練參數，可能能夠降低 Overfitting
- Max-Pooling 在 pool size 取最大值：能保留重要特徵，讓特徵更明顯，且讓神經網路對輸入圖像中的小變換、扭曲和平移，對判斷上減少造成影響

Max-pool 運算說明

Max-pool運算可以將影像縮減取樣(downsampling)

如下圖: 原本影像是 4×4，經過 Max-pool 運算轉換後，影像大小變為 2×2



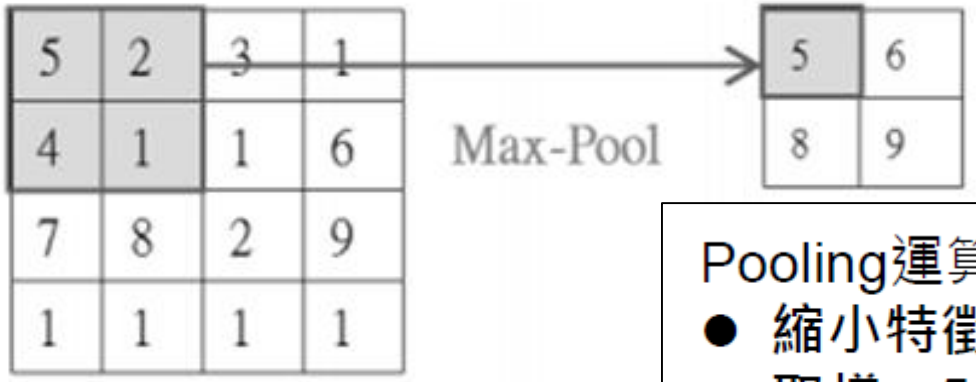
影像(X)大小4X4

轉換後

影像(X)大小2X2

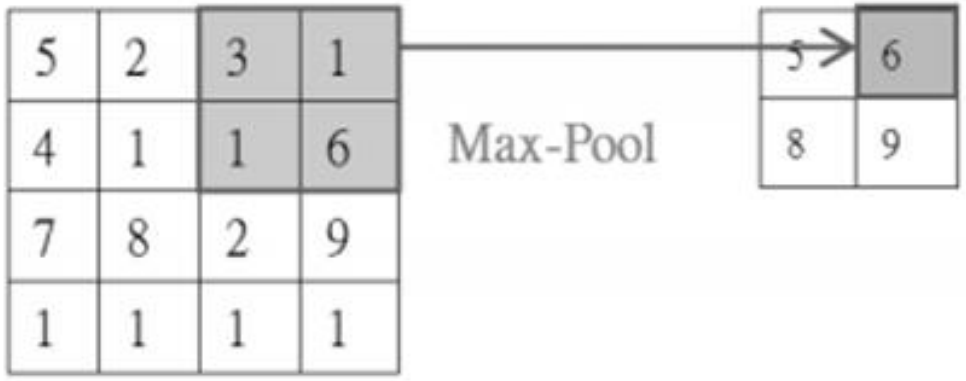
Max-pool 詳細運算說明

- **左上角4個數字**：5、2、4、1最大的是5，所以計算結果是5。



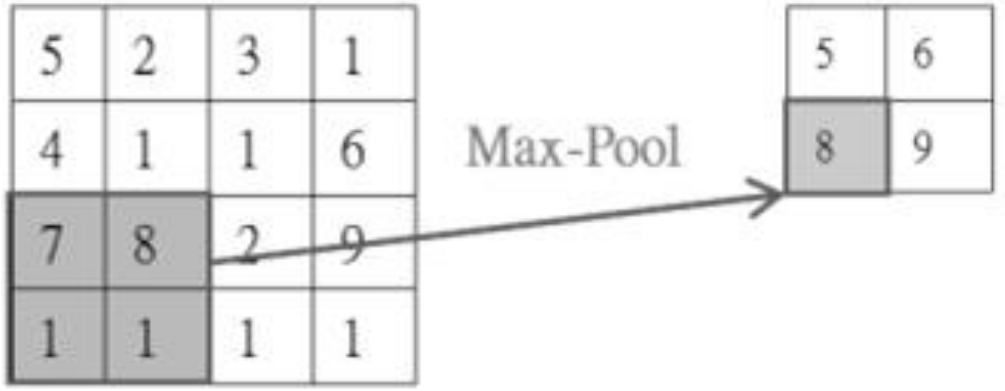
Pooling運算又稱為down-sampling(縮減取樣)因為：
- **縮小特徵圖**：原本4個數字，縮減為1個數字。
- **取樣**：5,2,4,1數字，取樣最大的數字是5。

- **右上角4個數字**：3、1、1、6最大的是6，所以計算結果是6。

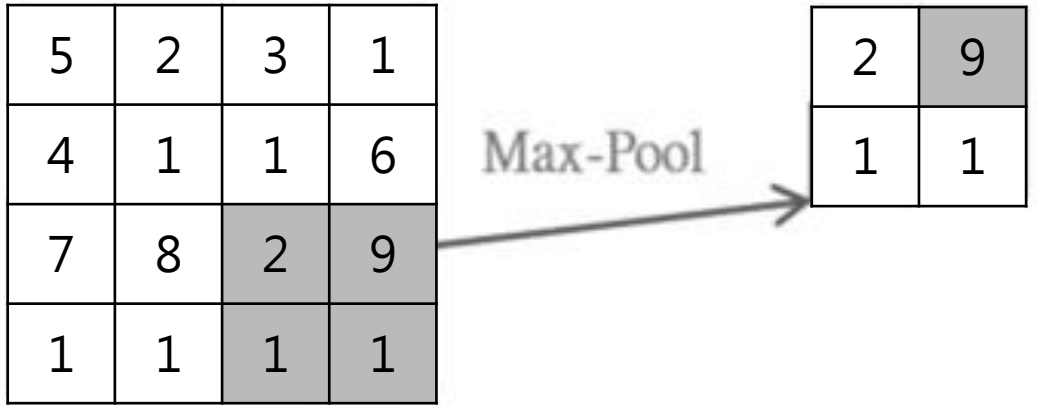


Max-pool 詳細運算說明

- **左下角4個數字**：7、8、1、1最大的是8，所以計算結果是8。

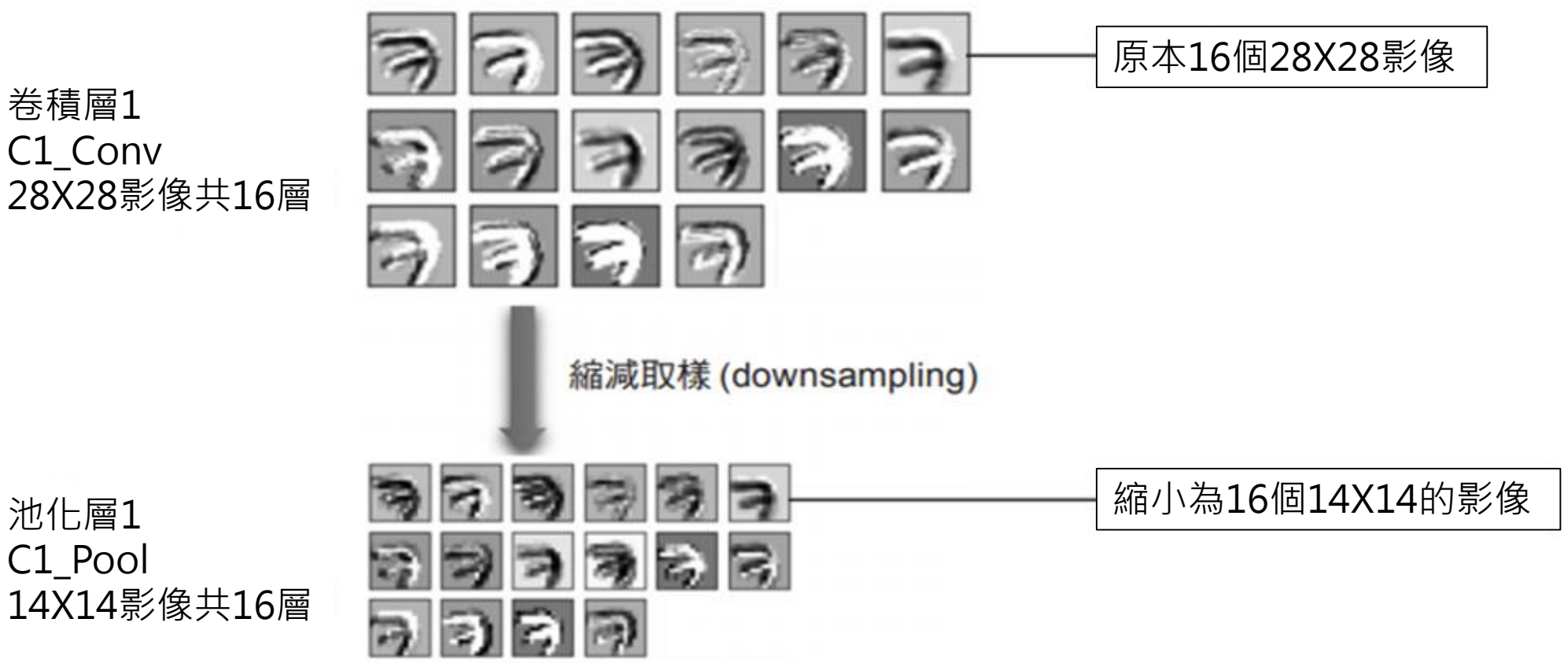


- **右下角4個數字**：2、9、1、1最大的是9，所以計算結果是9。



Step7. 使用Max-Pool將手寫數字影像轉換

使用Max-Pool進行縮減取樣(downsampling)，執行手寫數字影像轉換，將16個28X28影像，縮小爲16個14X14的影像。但是不會改變影像數量仍然是(16)。



使用 Max-pool 縮減取樣會縮小影像，有下列好處:

1. **減少需處理的資料點:** 減少後續運算所需的時間
2. **讓影像位置差異變小:** 例如手寫數字7，位置上下左右可能不同，但是位置的不同可能會影響辨識。減小影像大小，讓數字的位置差異變小
3. **參數的數量和計算量下降:** 在一定程度上控制過度擬合(overfitting)現象

Step8.建立卷積神經網路辨識Mnist資料集

1. 資料預處理：資料預處理後，會產生Features(數字影像特徵值)與Label(數字的真實的值)
2. 建立模型：我們將建立卷積神經網路CNN (convolutional neural network)。
3. 訓練模型：輸入訓練資料Features(數字影像特徵值)與Label(數字的真實的值)，執行10次訓練週期
4. 評估模型準確率：使用測試資料，評估模型準確率
5. 進行預測：使用已經訓練完成的模型，輸入測試資料進行預測

8.2進行資料預處理(Preprocess)

CNN)與多元感知器(MLP)，進行資料預處理的方式不同:

	reshape	參數說明
MLP	image.reshape(60000,784)	多元感知器因為直接送進神經元處理，所以reshape轉換原始60,000筆資料時，每一筆有784個數字，作為784個神經元的輸入
CNN	image.reshape(60000,28,28,1)	CNN因為必須先進行卷積與池化運算，必須保持影像的維度，所以reshape轉換60,000筆資料時，每一筆有28×28×1的影像，分別是28(寬) ×28(高)×1(單色)

Step1.匯入所需模組

```
from keras.datasets import mnist
from keras.utils import np_utils
import numpy as np
np.random.seed(10)
```

```
Using TensorFlow backend.
```

Step2.讀取mnist資料

```
(x_Train, y_Train), (x_Test, y_Test) = mnist.load_data()
```

Step3.將features(數字影像特徵值)轉換為4維矩陣

將features(數字影像特徵值)以reshape轉換為60000×28×28×1的4維矩陣

```
x_Train4D=x_Train.reshape(x_Train.shape[0],28,28,1).astype('float32')
x_Test4D=x_Test.reshape(x_Test.shape[0],28,28,1).astype('float32')
```

Step4. 將 features (數字影像特徵值)標準化

將features(數字影像特徵值)標準化，可以提高模型預測的準確度，並且更快收斂

```
x_Train4D_normalize = x_Train4D / 255
x_Test4D_normalize = x_Test4D / 255
```

Step5. 將 label (數字的真實的值)以Onehot encoding轉換

使用np_utils.to_categorical，將訓練資料與測試資料的label，進行Onehot encoding轉換

```
y_TrainOneHot = np_utils.to_categorical(y_Train)
y_TestOneHot = np_utils.to_categorical(y_Test)
```

8.3建立模型

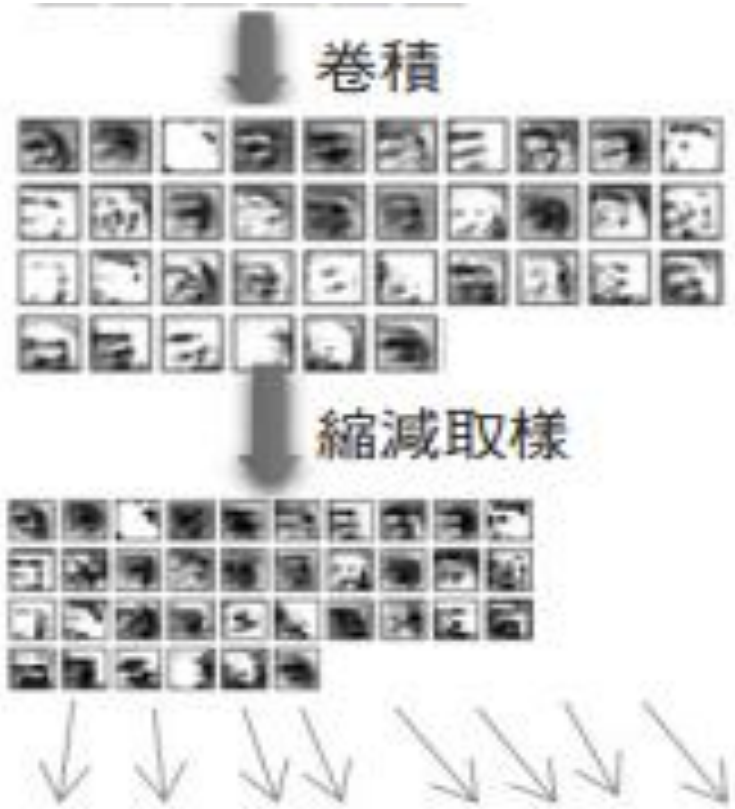
卷積層1 (28X28影像共16層)

```
(Conv2D(filters=16,
        kernel_size=(5,5),
        padding='same',
        input_shape=(28,28,1),
        activation='relu'))
```

池化層1 (14X14影像共16層)

```
(MaxPooling2D(pool_size=(2, 2)))
```

8.3建立模型



卷積層2 (14X14影像共36層)

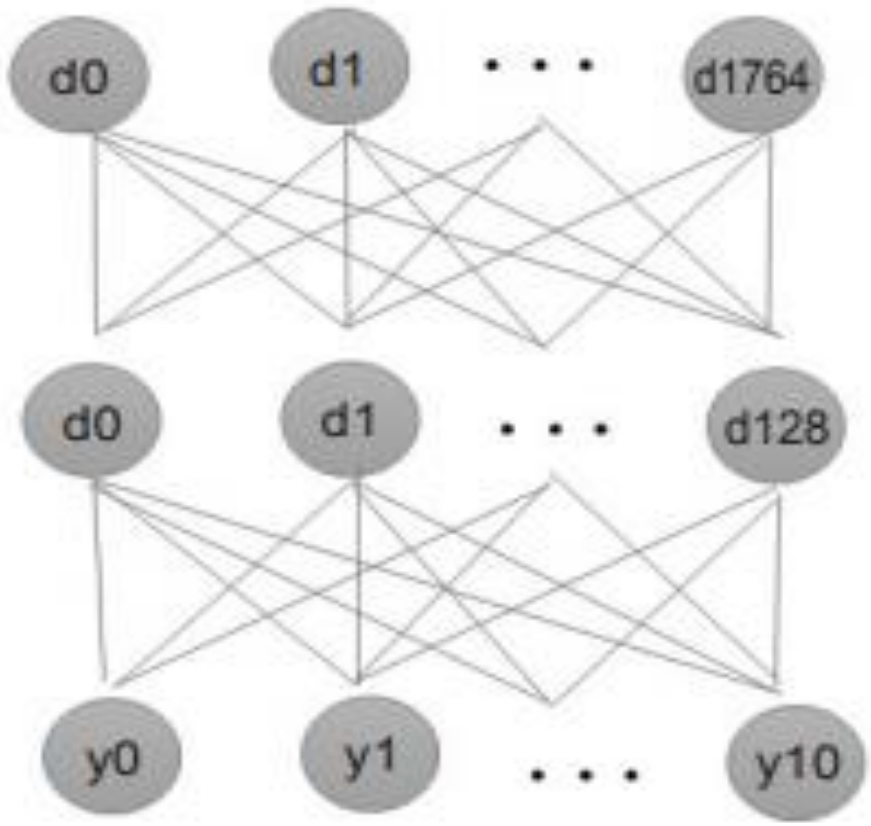
```
(Conv2D(filters=36,
        kernel_size=(5,5),
        padding='same',
        activation='relu'))
```

池化層2 (7X7影像共36層)

```
(MaxPooling2D(pool_size=(2, 2)))
```

```
(Dropout(0.25))
```

8.3建立模型



平坦層　(1764神經元)

```
(Flatten())
```

隱藏層　(128神經元)

```
(Dense(128, activation='relu'))
(Dropout(0.25))
```

輸出層　(10神經元)

```
(Dense(10,activation='softmax'))
```

Step1.匯入所需模組

```
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D
```

程式碼	說明
from keras.models import Sequential	匯入線性堆疊模型
from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D	從 keras 的 layers 模組，匯入後續 CNN 卷積網路要使用的卷積層、池化層、平坦層等

Step2.建立keras的Sequential模型

model = Sequential() #建立一個線性堆疊模型

建立 Sequential 模型，後續只需要使用model.add()方法，將各神經網路層加入模型即可

Step3.建立卷積層1與池化層1

一次完整的卷積運算，包含一個卷積層與一個池化層

- **建立卷積層1**

  輸入的數字影像是28×28，執行第一次卷積運算，使用濾鏡產生16個影像

  卷積運算並不會改變影像大小，所以16個影像仍然是28×28大小

建立卷積層1

使用下列程式碼建立卷積層1，產生16個28×28大小的影像

```
model.add(Conv2D(filters=16,
                 kernel_size=(5,5),
                 padding='same',
                 input_shape=(28,28,1),
                 activation='relu'))
```

以上程式碼加入Conv2D層至模型，輸入下列參數:

程式碼	說明
filters=16,	建立16個濾鏡filter weight
kernel_size=(5,5),	每一個濾鏡5×5大小
padding='same',	此設定讓卷積運算，產生的卷積影像大小不變
input_shape=(28,28,1),	第1,2維度:代表輸入的影像形狀28×28大小，第3個維度:因為是單色灰階影像，所以最後維度是1
activation='relu'	設定ReLU激活函數

建立池化層1

使用以下程式碼建立池化層1

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

輸入參數 pool_size=(2,2)，執行第1次縮減取樣
將16個28×28影像，縮小為14×14的影像

Step4.建立卷積層2與池化層2

- 建立卷積層2

使用下列程式碼建立卷積層2，執行第2次卷積運算:

```
model.add(Conv2D(filters=36,
                 kernel_size=(5,5),
                 padding='same',
                 activation='relu'))
```

將原本16個的影像，轉換為36個影像，影像大小仍然是14×14

以上程式碼加入Conv2D層至模型，輸入下列參數:

程式碼	說明
filters=36,	建立36個濾鏡filter weight
kernel_size=(5,5),	每一個濾鏡filter weight 5×5大小
padding='same',	此設定讓卷積運算並不會改變影像大小
activation='relu'	設定ReLU激活函數

- **建立池化層2，並且加入Dropout避免overfitting**

以下程式碼建立池化層2，輸入參數pool_size=(2,2)，執行第2次縮減取樣，將36個14×14影像，縮小為36個7×7的影像。

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

以下程式碼加入Dropout(0.25)層至模型中。

每次訓練迭代時，隨機地在神經網路中放棄25%的神經元，以避免Overfitting。

```
model.add(Dropout(0.25))
```

Step5.建立神經網路 (平坦層、隱藏層、輸出層)

- 建立平坦層

以下程式碼建立平坦層

```
model.add(Flatten())
```

將根據池化層2的步驟，建立的36個7×7影像，轉換為1維的向量

長度是36×7×7=1764

也就是1764個float數字，正好對應到1764神經元

- 建立隱藏層

以下程式碼建立隱藏層，共有128個神經元。

```
model.add(Dense(128, activation='relu'))
```

並且加入Dropout層至模型中。每次訓練迭代時隨機放棄已訓練好的50%神經元，以避免overfitting。

- 建立輸出層

共有10個神經元，對應到0~9共10個數字

使用softmax激活函數進行轉換， softmax可以將神經元的輸出，轉換為預測每一個數字的機率。

```
model.add(Dense(10,activation='softmax'))
```

Step6.查看模型的摘要

使用下列指令，查看模型的摘要。

```
In [17]: print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 16)        416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 36)        14436
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 36)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 7, 7, 36)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1764)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               225920
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 242,062
Trainable params: 242,062
Non-trainable params: 0
_________________________________________________________________
None
```

卷積層1與池化層1

卷積層2與池化層2

神經網路(平坦層、隱藏層、輸出層)

8.4進行訓練

使用Back Propagation反向傳播算法進行訓練(參考第二章)

Step1.定義訓練方式

在訓練模型之前，必須使用compile方法，對訓練模型進行設定:

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',metrics=['accuracy'])
```

輸入下列參數:

- Loss: 損失函數(Loss Function)，使用cross_entropy交叉熵
- Optimizer: 訓練最優化方法，使用adam最優化方法
- Metrics: 設定評估模型訓練效果的方式，使用accuracy準確率

Step2.開始訓練

執行訓練的程式碼如下:

```
train_history=model.fit(x=x_Train4D_normalize,
                        y=y_TrainOneHot,validation_split=0.2,
                        epochs=20, batch_size=300,verbose=2)
```

使用model.fit進行訓練，訓練過程儲存在train_history變數:

1. **輸入訓練資料參數**

   x=x_Train4D_normalize #(features數字影像真實的值)

   y=y_Train_OneHot #(label數字影像真實的值)

2. **設定訓練與驗證資料比例**

   - 設定參數validation_split=0.2

   訓練前Keras自動將資料分成:80%作為訓練，20%作為驗證

   全部60,000筆，分成48,000訓練，12,000驗證

3. 設定epoch(訓練週期)次數與每一批次筆數
   - Epochs=10: 執行10次訓練週期。
   - Batch_size=300: 每一批次300筆資料。

4. 設定訓練過程
   - Verbose=2: 顯示訓練過程。

   共執行10次epoch(訓練週期)，每一次訓練週期，執行下列功能:
   - 使用48,000筆訓練資料進行訓練，分為每一批次300筆，所以大約分為160批次(48000/300=160)進行訓練。
   - epoch(訓練週期)訓練完成後，會計算此次訓練週期的accuracy(準確率)與loss(誤差)，並且記錄在新增一筆資料記錄在train_history中。

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/10
105s - loss: 0.5142 - acc: 0.8335 - val_loss: 0.1100 - val_acc: 0.9668
Epoch 2/10
98s - loss: 0.1529 - acc: 0.9547 - val_loss: 0.0703 - val_acc: 0.9783
Epoch 3/10
99s - loss: 0.1132 - acc: 0.9656 - val_loss: 0.0623 - val_acc: 0.9798
Epoch 4/10
98s - loss: 0.0898 - acc: 0.9732 - val_loss: 0.0531 - val_acc: 0.9843
Epoch 5/10
110s - loss: 0.0802 - acc: 0.9756 - val_loss: 0.0433 - val_acc: 0.9875
Epoch 6/10
107s - loss: 0.0689 - acc: 0.9797 - val_loss: 0.0415 - val_acc: 0.9875
Epoch 7/10
98s - loss: 0.0600 - acc: 0.9822 - val_loss: 0.0409 - val_acc: 0.9882
Epoch 8/10
123s - loss: 0.0579 - acc: 0.9826 - val_loss: 0.0368 - val_acc: 0.9904
Epoch 9/10
101s - loss: 0.0505 - acc: 0.9841 - val_loss: 0.0346 - val_acc: 0.9901
Epoch 10/10
103s - loss: 0.0489 - acc: 0.9849 - val_loss: 0.0368 - val_acc: 0.9900
```

80%作為訓練資料，
20%作為驗證資料

使用訓練資料，計算accuracy與loss

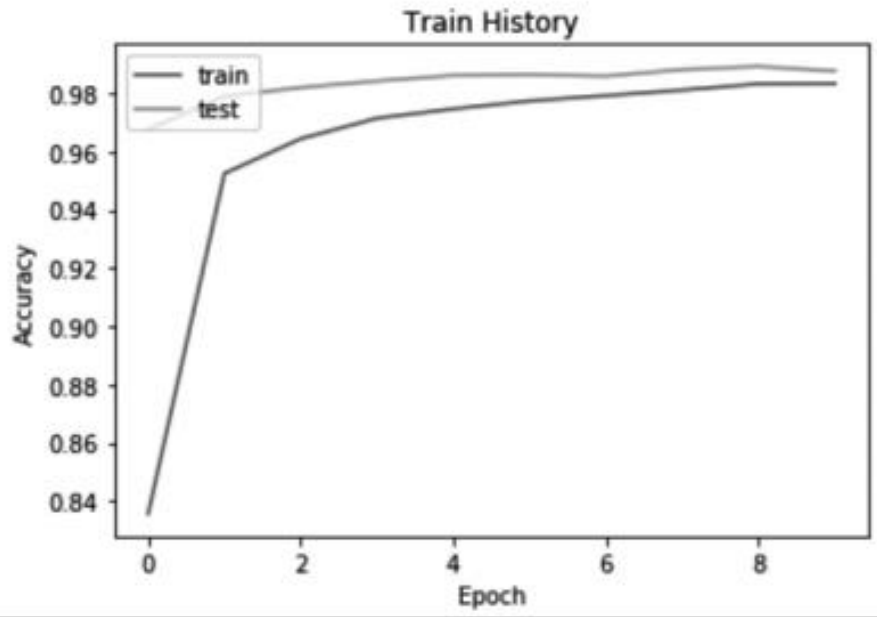
使用驗證資料，計算accuracy與loss

以上執行畫面，你可以看到共執行10個Epoch訓練執行時期，你可以發現loss(誤差)越來越小，accuracy(準確率)越來越高。

Step3. 畫出accuracy執行結果

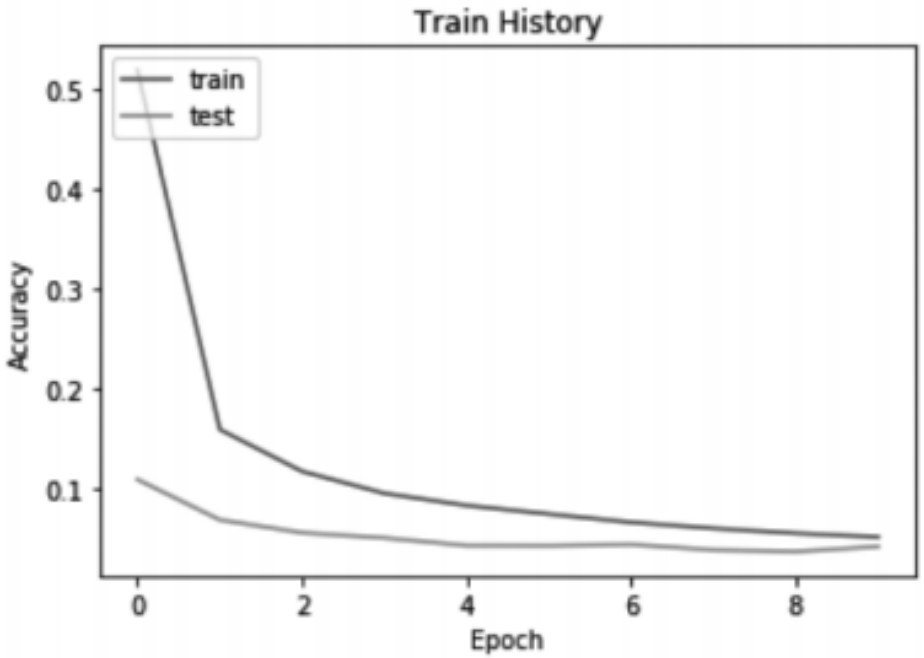
之前訓練步驟，會將每一個訓練週期的accuracy(準確率)與loss(誤差) ，紀錄在train_history 變數。
我們可以使用下列程式碼，讀取train_history，畫出accuracy準確率執行結果。
show_train_history請參考第7章的說明：

```
In [21]: show_train_history('acc','val_acc')
```



Step4. 畫出loss誤差執行結果

```
In [22]: show_train_history('loss','val_loss')
```



以上執行畫面，「loss訓練的誤差」是藍色，「val_loss驗證的誤差」是黃色，總共執行10個Epoch訓練週期，你可以發現，不論訓練與驗證，驗證的誤差越來越低。

8.5評估模型準確率

以下列程式碼，評估模型準確率

```
scores = model.evaluate(x_Test4D_normalize , y_TestOneHot)
scores[1]
```

```
10000/10000 [==============================] - 3s

0.9935000000000005
```

以上執行結果準確率是0.993。程式碼說明如下:

程式碼	說明
scores=model.evaluate()	使用model.evaluate進行評估模型準確率，評估後的準確率存在scores
x_Test4D_normalize ,	測試資料的features(已標準化測試資料的數字影像)
y_TestOneHot	測試資料的label(數字影像真實的值)

8.6進行預測

Step1.執行預測

```
prediction=model.predict_classes(x_Test4D_normalize)
```

```
 9888/10000 [============================>.] - ETA: 0s
```

使用model.predict_classes，輸入參數x_Test4D_normalize(已標準化測試資料的數字影像)進行預測。

Step2. 預測結果

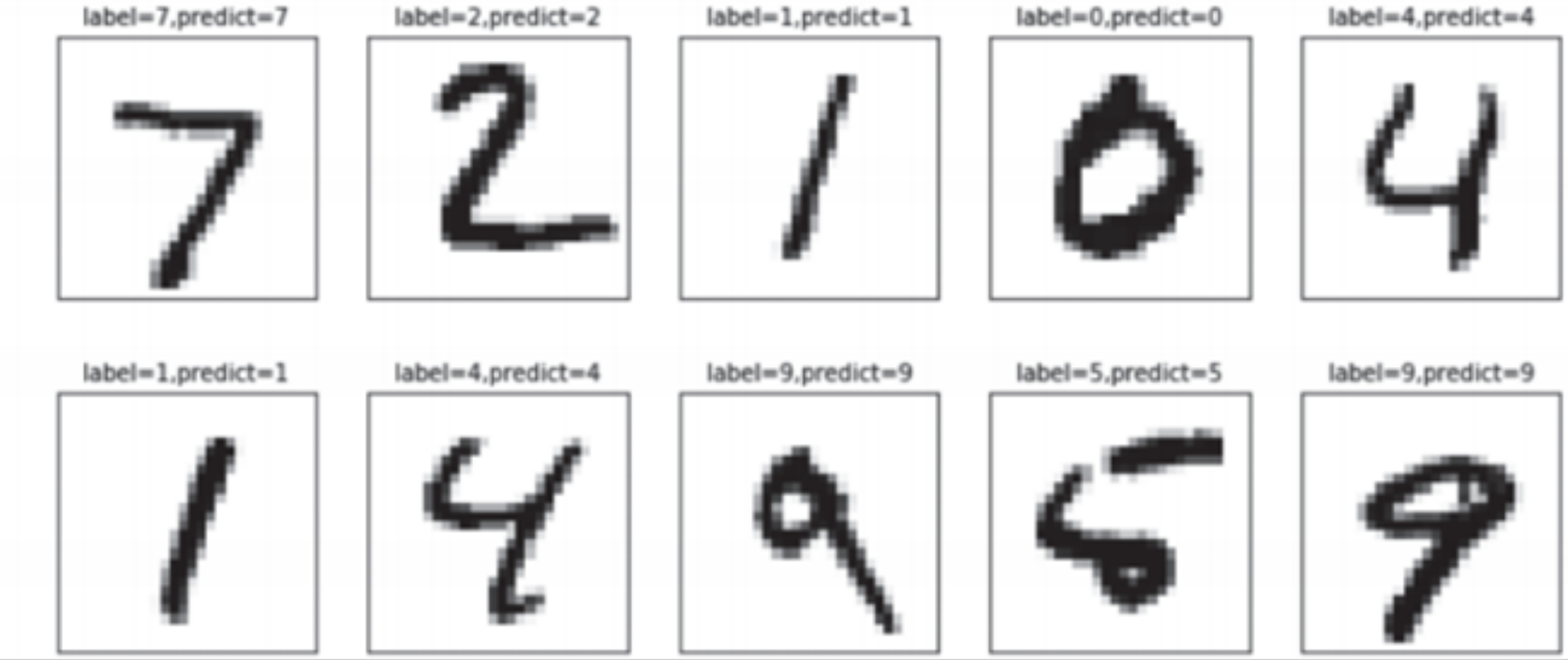
你可以下列指令查看預測結果的前10筆資料：

```
In [25]: prediction[:10]
Out[25]: array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9])
```

Step3. 顯示前10筆預測結果

使用第6章建立的show_images_labels_prediction函數，顯示前10筆預測結果，傳入測試資料影像、label(眞實值)及predict(預測結果)。

```
In [27]: plot_images_labels_prediction(x_Test,y_Test,prediction,idx=0)
```



8.7顯示混淆矩陣(confusion matrix)

Step1.使用pandas crosstab建立混淆矩陣(confusion matrix)

```
import pandas as pd
pd.crosstab(y_Test,prediction,
            rownames=['label'],colnames=['predict'])
```

程式碼	參數說明
import pandas as pd	匯入pandas模組，後續以pd使用
pd.crosstab(	使用pd.crosstab建立混淆矩陣，輸入下列參數:
y_Test,	測試資料數字影像的真實值
prediction,	測試資料數字影像的預測結果
rownames=['label'],	設定行的名稱是label
colnames=['predict'])	設定列的名稱是predict

predict	0	1	2	3	4	5	6	7	8	9
label										
0	978	0	0	0	0	0	1	1	0	0
1	0	1132	1	0	0	1	0	1	0	0
2	3	1	1020	0	0	0	0	7	1	0
3	0	0	2	1004	0	2	0	2	0	0
4	0	1	0	0	975	0	1	0	1	4
5	1	0	0	6	0	881	2	1	0	1
6	5	3	0	1	1	2	946	0	0	0
7	0	1	1	1	0	0	0	1022	1	2
8	7	2	3	3	1	2	0	3	947	6
9	4	4	0	2	4	2	0	6	0	987

對角線是預測正確的數字

真實值是8，但是預測是3

結論

本章使用卷積神經網路CNN(Convolutional Neural Networks)，辨識Mnist資料集中的手寫數字，分類精度接近為99%

不過這只是單色手寫數字辨識，相對來說比較簡單

下一章將介紹更具挑戰性的任務

使用卷積神經網路，辨識CIFAR-10彩色影像資料集

影像共10個分類:飛機、汽車、鳥、貓、鹿、狗、青蛙、船、卡車