第八章

Keras 捲積神經網路(CNN)辨識手寫數字

8.1 CNN捲積神經網路簡介

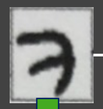
8.1 CNN卷積神經網路簡介

MLP多層感知器 vs. CNN卷積神經網路

- MLP多層感知器 和 CNN卷積神經網路的主要差異是：CNN增加了卷積層1、池化層1、卷積層2、池化層2的處理以提取特徵。

8.1 CNN卷積神經網路簡介

輸入層
input
28X28影像共1層



輸入的數字影像28X28大小，例如：數字7的影像。

第一次卷積

卷積層1
28X28影像
共16層

第1次卷積運算：輸入的數字影像28X28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28X28大小。

第一次縮減取樣

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。

第二次卷積

卷積層2
14X14影像
共36層

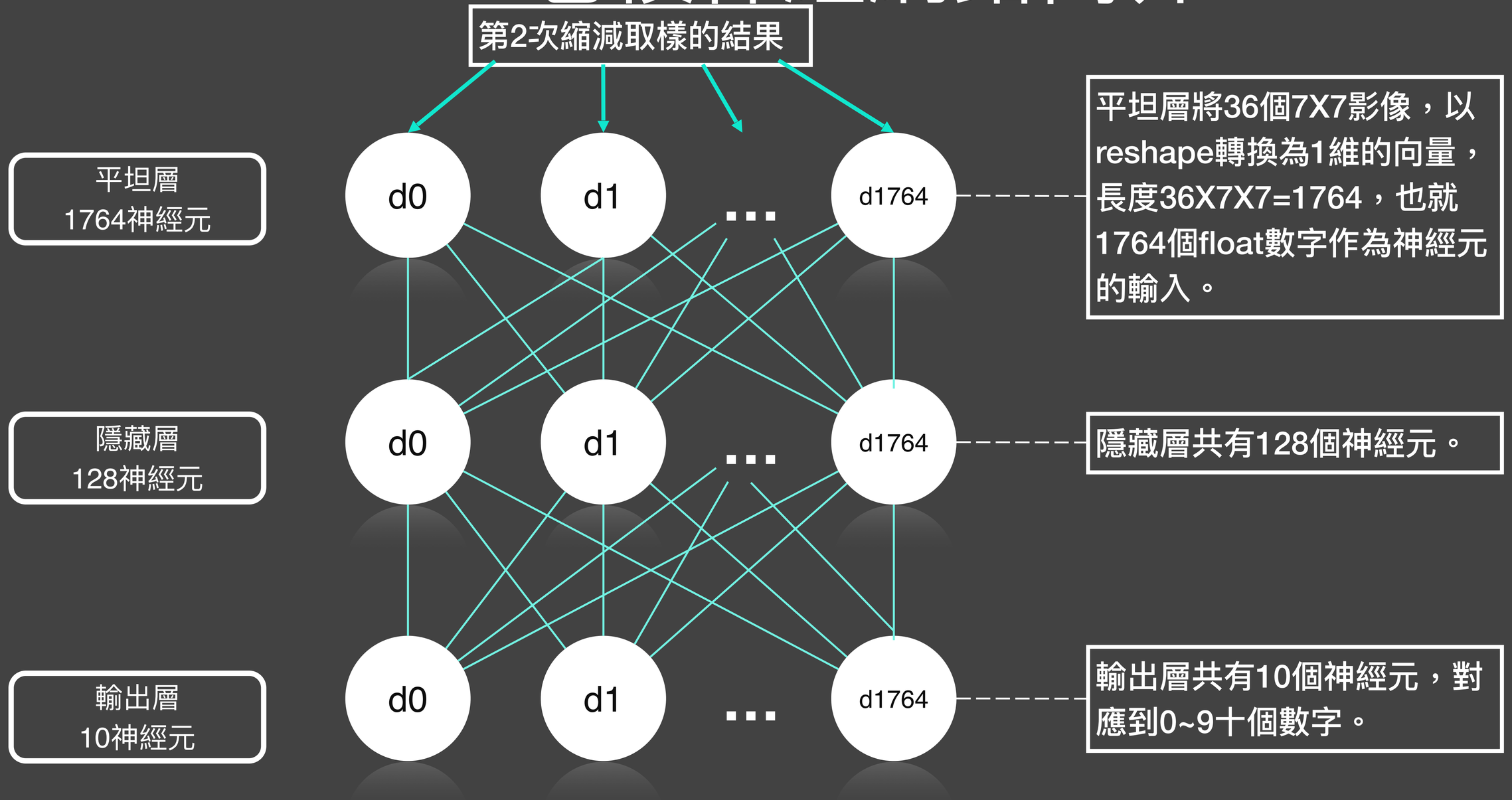
第2次卷積運算：將原本16個的影像，轉換為36個影像，卷積運算不會改變影像大小，所以仍然是大小14X14

第二次縮減取樣

池化層2
7X7影像
共36層

第2次縮減取樣：將36個14X14的影像，縮小為36個7X7的影像

8.1 CNN卷積神經網路簡介



8.1 CNN卷積神經網路簡介

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

- 卷積運算運算方式，如下圖：

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. 透過影像選取的矩陣(3X3)與filter weight (3X3)的乘積，計算出第1列、第1行的數字。

8.1 CNN卷積神經網路簡介

padding='SAME'
會在邊界之外補0

(0X0)+ (0X1)+ (0X0)+
(0X1)+ (1X1)+ (2X1)+
(0X0)+ (4X1)+ (5X0)=7

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

影像(x)

0	1	0
1	1	1
0	1	0

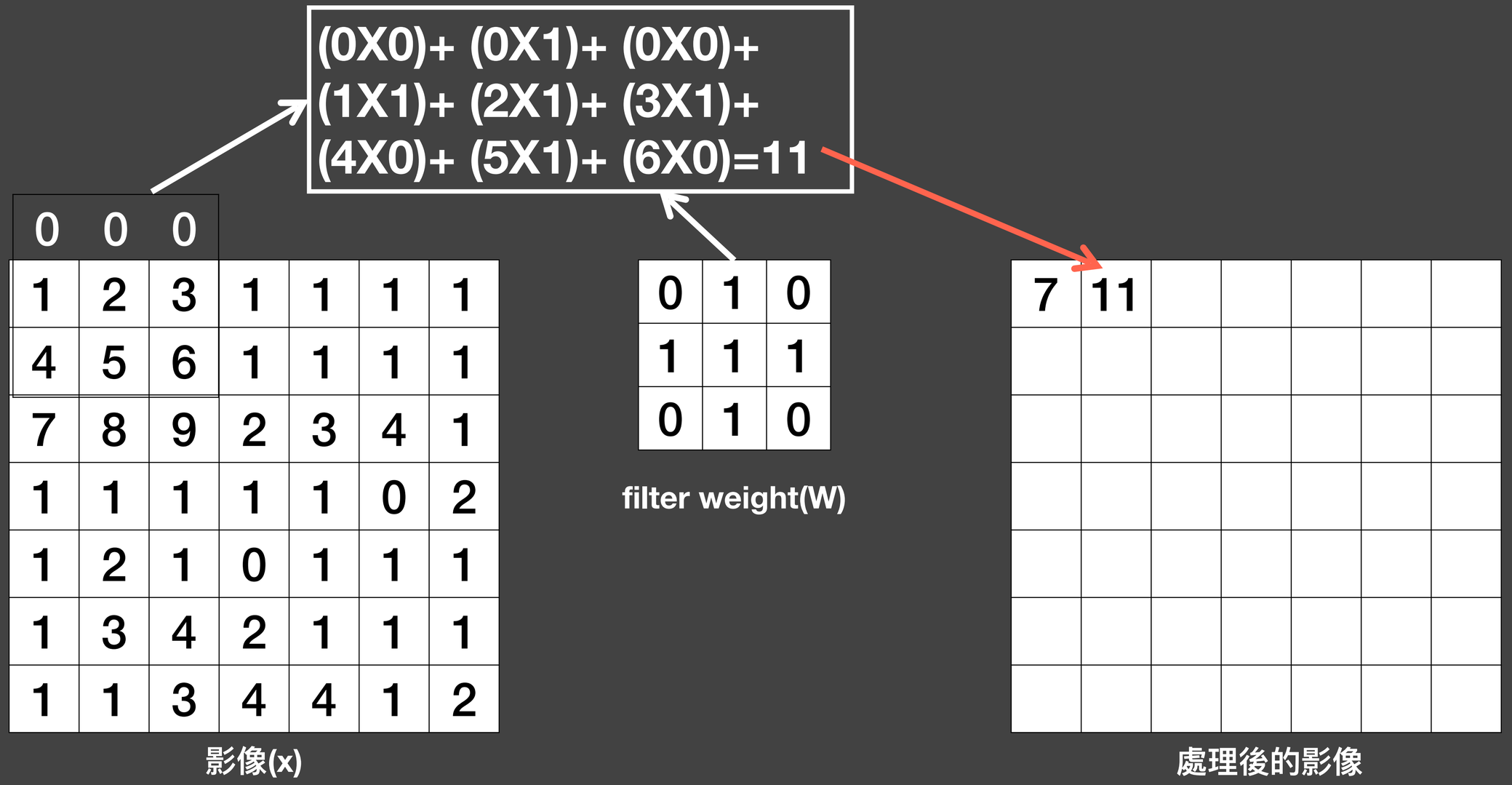
filter weight(W)

7						

處理後的影像

8.1 CNN卷積神經網路簡介

- 再以相同的方式計算第1列、第2行的數字。



8.1 CNN卷積神經網路簡介

- 依照上列相同的方式，依序完成所有運算，就完成影像的處理。

8.1 CNN卷積神經網路簡介

使用單一filter weight卷積運算產生影像

- 如下圖，將數字影響7(大小為28X28)，使用隨機產生的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。可以幫助提取輸入的不同特徵，例如:邊緣、線條和角等。

8.1 CNN卷積神經網路簡介

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 weight，產生16個影像，每個影像提取不同的特徵。

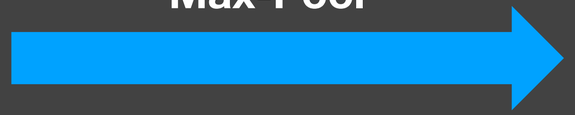
8.1 CNN卷積神經網路簡介

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4X4，經過Max-Pool運算轉換後，影像大小為2X2·

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

影像(x)大小4X4



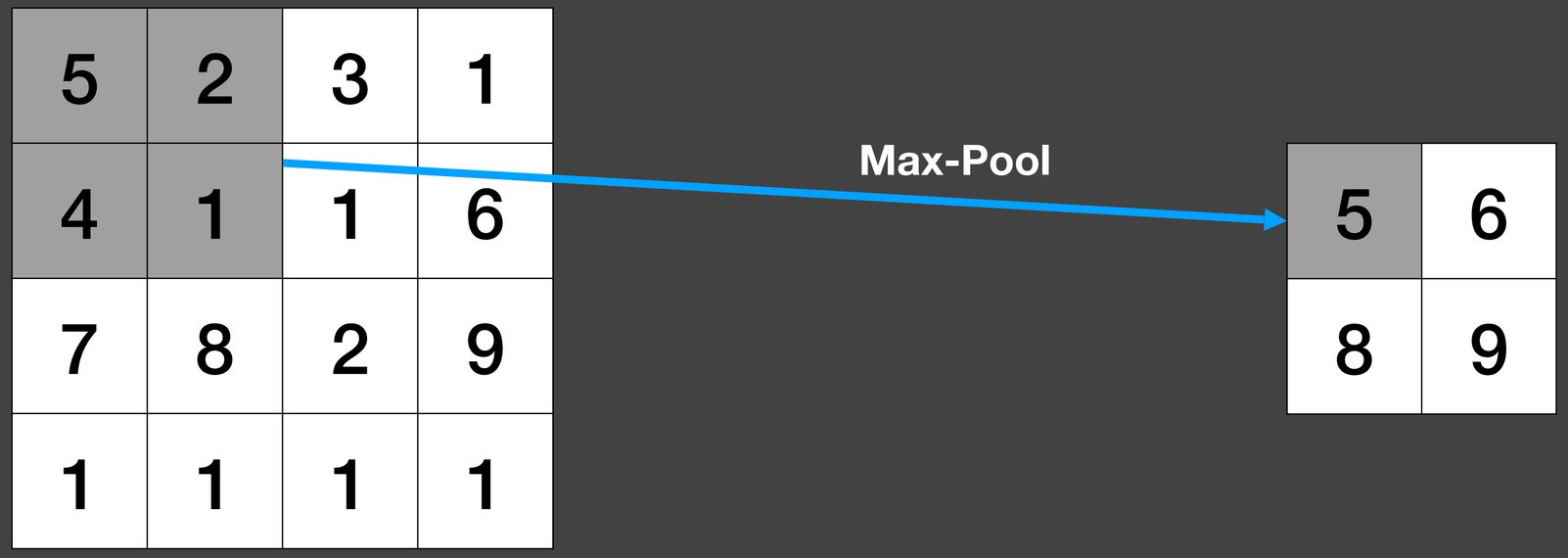
5	6
8	9

轉換後影像(x)大小2X2

8.1 CNN卷積神經網路簡介

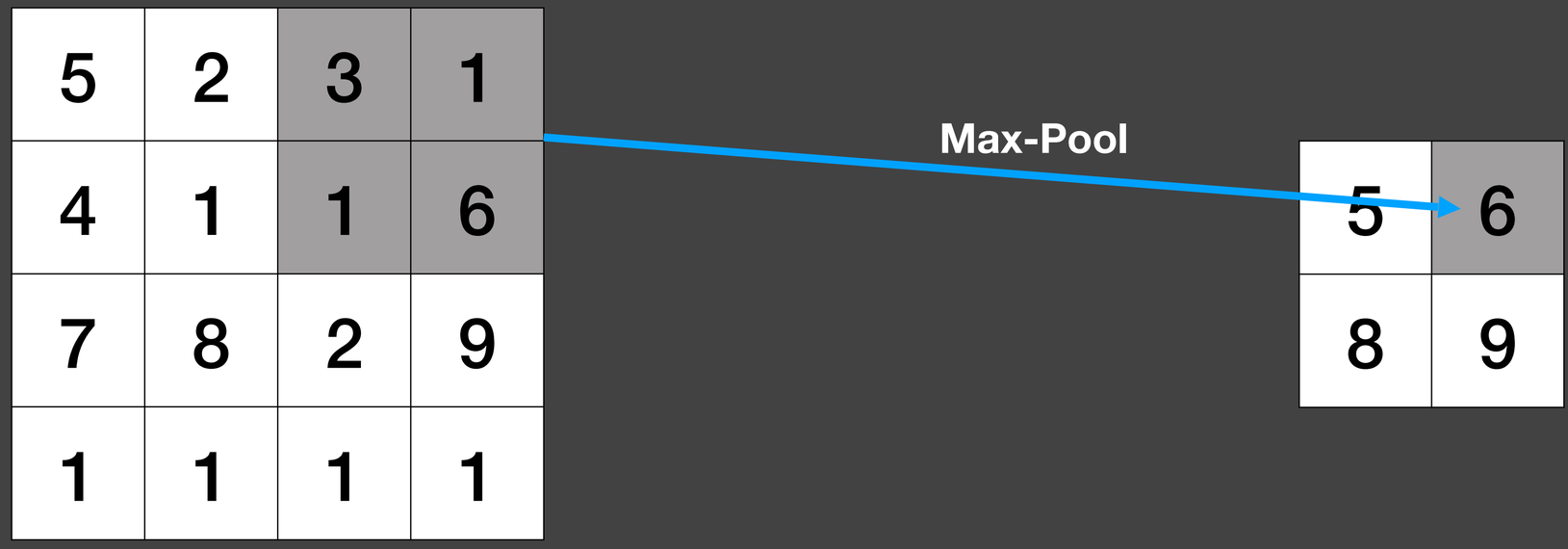
Max-Pool的詳細說明如下：

- 左上角４個數字：取５、２、４、１當中最大的數字，也就是５



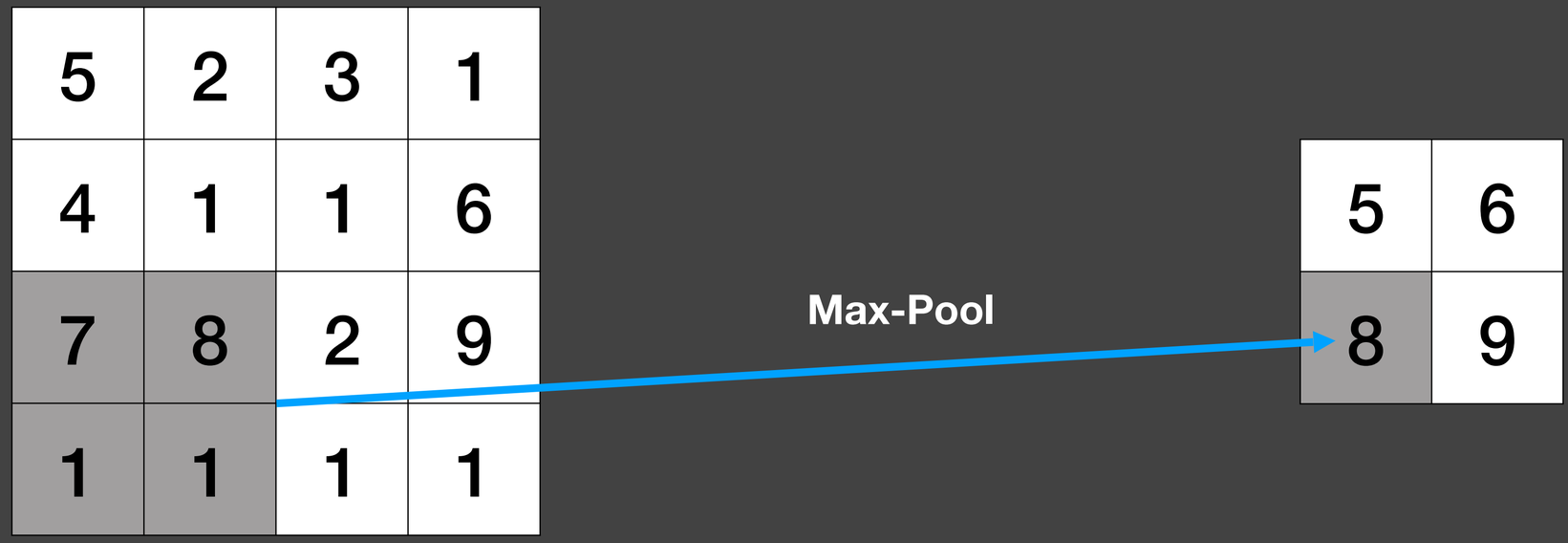
8.1 CNN卷積神經網路簡介

右上角 4 個數字：取 3 、 1 、 1 、 6 當中最大的數字，也就是 6



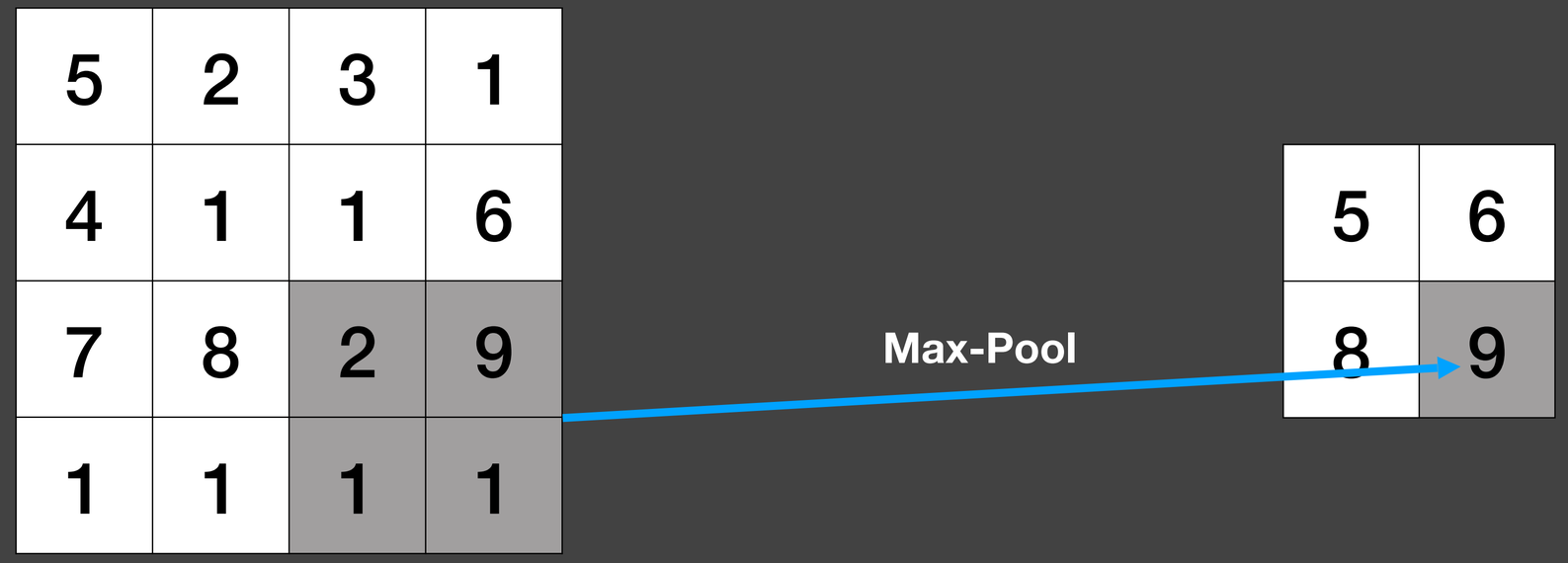
8.1 CNN卷積神經網路簡介

左下角 4 個數字：取 7 、 8 、 1 、 1 當中最大的數字，也就是 8



8.1 CNN卷積神經網路簡介

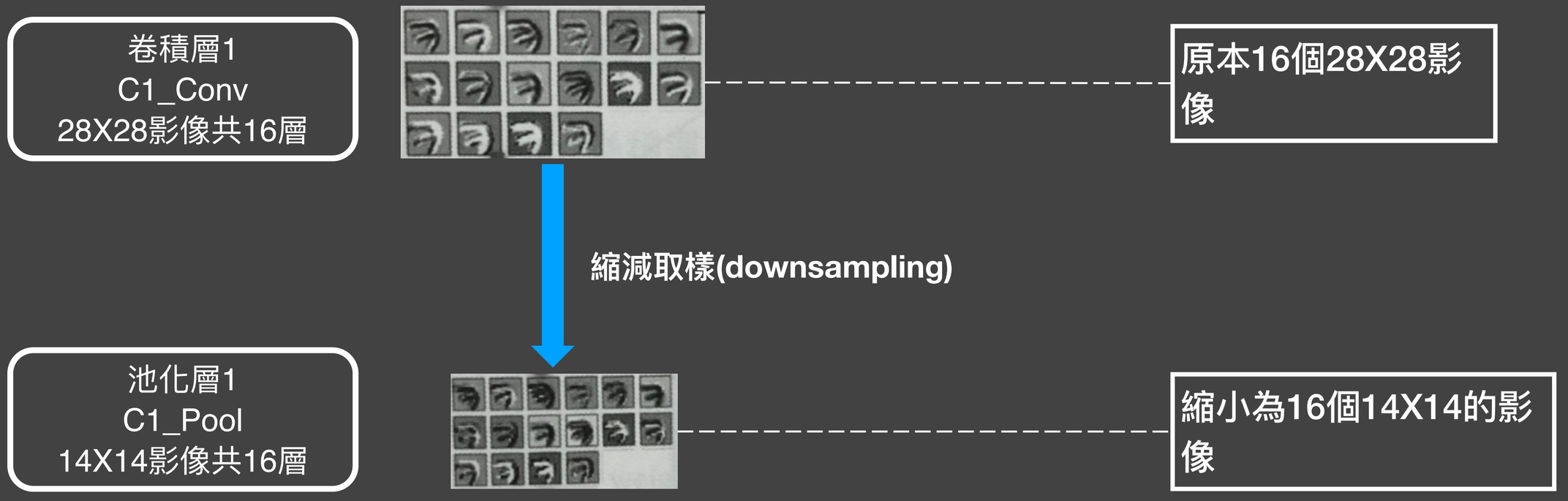
右下角 4 個數字：取、9 、1 、2 當中最大的數字，也就是 9



8.1 CNN卷積神經網路簡介

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，縮小為16個14X14的影像。但是不會改變影像數量(仍然是16)。



8.1 CNN卷積神經網路簡介

- 縮減取樣會縮小影像，有下列的好處：

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：
這在一定程度上也控制了過度擬合(overfitting)

8.2 進行資料預處理(Preprocess)

8.2進行資料預處理

進行資料預處理

- 在命令列輸入下列程式碼，並且按下Shift+Enter執行

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

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

```
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')
```

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

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

8.2進行資料預處理

- 程式碼說明

In [1]:
```
from keras.datasets import mnist
from keras.utils import np_utils
import numpy as np
np.random.seed(10)
```

匯入所需模組

```
/home/user/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [2]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

讀取MNIST資料

In [3]:
```
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')
```

使用reshape轉換，轉換成4維矩陣

In [4]:
```
x_train4d_normalize = x_train4d / 255
x_test4d_normalize  = x_test4d / 255
```

標準化

In [5]:
```
y_train_onehot = np_utils.to_categorical(y_train)
y_test_onehot  = np_utils.to_categorical(y_test)
```

One-hot encoding

8.3 建立模型

8.3 建立模型

28 x 28

28 x 28, 16層

14 x 14, 16層

14 x 14, 36層

7 x 7, 36層

1764個神經元 (= 7 x 7 x 36)

128個神經元

10個神經元

8.3 建立模型

查看模型的摘要

- 在命令列輸入下列程式碼，並且按下Shift+Enter依序執行。

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

```
model = Sequential()
```

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

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

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

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

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

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

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

```
model.add(Dropout(0.5))
```

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

8.3 建立模型

- 程式碼說明

```
In [6]: from keras.models import Sequential
        from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
```

匯入所需模組

```
In [7]: model = Sequential()
```

建立Sequential模型

```
In [8]: model.add(
            Conv2D(
                filters=16,
                kernel_size=(5,5),
                padding='same',
                input_shape=(28,28,1),
                activation='relu'
            )
        )
```

建立捲積層1

濾鏡數: 16個
濾鏡大小: 5x5
捲積後影像大小不變(same)
輸入影像維度: 28(高)x28(寬)x1(黑白)
激活函數:ReLU

```
In [9]: model.add(MaxPooling2D(pool_size=(2,2)))
```

建立池化層1
大小: 2x2

8.3 建立模型

- 程式碼說明

```
In [10]: model.add(
             Conv2D(
                 filters=36,
                 kernel_size=(5,5),
                 padding='same',
                 activation='relu'
             )
         )

In [11]: model.add(MaxPooling2D(pool_size=(2,2)))

In [12]: model.add(Dropout(0.25))

In [13]: model.add(Flatten())

In [14]: model.add(Dense(128, activation='relu'))

In [15]: model.add(Dropout(0.5))

In [16]: model.add(Dense(10, activation='softmax'))
```

建立捲積層2

濾鏡數: 36個
濾鏡大小: 5x5
捲積後影像大小不變(same)
激活函數:ReLU

建立池化層2
大小: 2x2

Dropout: 25%

建立平坦層

建立隱藏層，使用ReLU激活函數

Dropout: 50%

建立輸出層，使用softmax激活函數

8.3 建立模型

查看模型的摘要

- 在命令列輸入下列程式碼，並且按下Shift+Enter執行

```
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 16)        416
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 16)        6416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 16)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 36)          14436
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 36)          0
_________________________________________________________________
dropout (Dropout)            (None, 3, 3, 36)          0
_________________________________________________________________
flatten (Flatten)            (None, 324)               0
_________________________________________________________________
dense (Dense)                (None, 128)               41600
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 64,158
Trainable params: 64,158
Non-trainable params: 0
_________________________________________________________________
None
```

卷積層1 & 池化層1

卷積層2 & 池化層2

神經網路層
(平坦、隱藏、輸出層)

8.4 進行訓練

8.4 進行訓練

- **評估模型準確率**

- **在命令列輸入下列程式碼，並且按下Shift+Enter執行**

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

compile方法須輸入下列參數：

- **loss**：設定損失函數(Loss Function)，在深度學習使用cross_entropy交叉熵，訓練的效果比較好。
- **optimizer**：設定訓練時的最優化方法，在深度學習使用adam最優化方法，可以讓訓練更快收斂，並提高準確率。
- **metrics**：設定評估模型的方式是accuracy準確率。

8.4 進行訓練

- **評估模型準確率**
- **在命令列輸入下列程式碼，並**

```
train_history=model.fit(x=x_Train4D_normalize,
                        y=y_TrainOneHot,validation_split=0.2,
                        epochs=10, batch_size=300,verbose=2)
```

```
Epoch 1/10
160/160 - 46s - loss: 0.6692 - accuracy: 0.7839 - val_loss: 0.1254 - val_accuracy: 0.9623
Epoch 2/10
160/160 - 38s - loss: 0.1701 - accuracy: 0.9494 - val_loss: 0.0817 - val_accuracy: 0.9763
Epoch 3/10
160/160 - 38s - loss: 0.1231 - accuracy: 0.9626 - val_loss: 0.0649 - val_accuracy: 0.9804
Epoch 4/10
160/160 - 41s - loss: 0.0998 - accuracy: 0.9700 - val_loss: 0.0539 - val_accuracy: 0.9840
Epoch 5/10
160/160 - 43s - loss: 0.0843 - accuracy: 0.9760 - val_loss: 0.0493 - val_accuracy: 0.9857
Epoch 6/10
160/160 - 40s - loss: 0.0723 - accuracy: 0.9783 - val_loss: 0.0483 - val_accuracy: 0.9862
Epoch 7/10
160/160 - 37s - loss: 0.0697 - accuracy: 0.9791 - val_loss: 0.0414 - val_accuracy: 0.9889
Epoch 8/10
160/160 - 44s - loss: 0.0602 - accuracy: 0.9817 - val_loss: 0.0367 - val_accuracy: 0.9898
Epoch 9/10
160/160 - 44s - loss: 0.0534 - accuracy: 0.9841 - val_loss: 0.0443 - val_accuracy: 0.9870
Epoch 10/10
160/160 - 42s - loss: 0.0516 - accuracy: 0.9839 - val_loss: 0.0373 - val_accuracy: 0.9889
```

使用model.fit訓練，訓練過程會儲存在train_history變數，並且輸入下列參數:

➢輸入訓練資料參數

- x=x_Train4D_normalize(feature數字影像的特徵值)
- y=y_Train_OneHot(label數字影像真實的值)

➢設定訓練與驗證資料比例

- 設定參數validation_split=0.2 ：訓練之前Keras會自動將資料80%分為訓練資料、20%分為驗證資料

➢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

➢設定顯示訓練過程

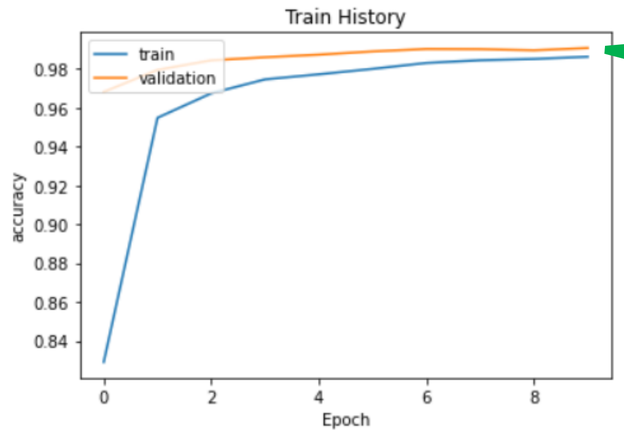
8.4 進行訓練

- 畫出accuracy執行結果

- 貼上第7張的show_train_history函數，然後按下Shift+Enter依序執行下列指令。

```
import matplotlib.pyplot as plt
def show_train_history(train_history,train,validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title('Train History')
    plt.ylabel(train)
    plt.xlabel('Epoch')
    plt.legend(['train', 'validation'], loc='upper left')
    plt.show()
```

```
show_train_history(train_history, 'accuracy', 'val_accuracy')
```



訓練準確率
(藍)
&
測試準確度
(橘)

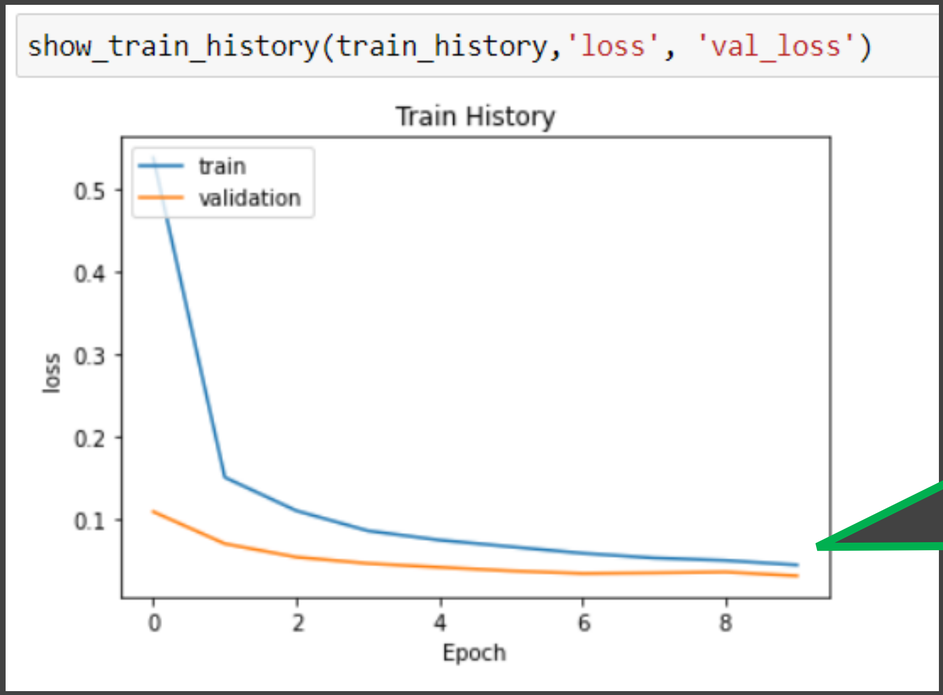
不論訓練還是驗證，準確率都越來越高。

8.4 進行訓練

- 畫出loss誤差執行結果

- 在命令列輸入下列程式碼，並且按下Shift+Enter執行

```
show_train_history(train_history,'loss', 'val_loss')
```



訓練誤差
(藍)
&
測試誤差
(橘)

不論訓練還是驗證，驗證的誤差都越來越低。

8.5 評估模型準確率

8.5 評估模型準確率

- **評估模型準確率**

- **在命令列輸入下列程式碼，並且按下Shift+Enter執行**

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

```
313/313 [==============================] - 5s 14ms/step - loss: 0.0260 - accuracy: 0.9912

0.9911999702453613
```

99.12%

- **scores = model.evaluate(**：
使用model.evaluate進行評估模型準確率，評估後的準確率會儲存在scores

- **x=x_Test4D_normalize**：
測試資料的features(已標準化測試資料的數字影像)

- y=y_TestOneHot)：
測試資料的label(數字影像的真實值)

8.6 進行預測

8.6 進行預測

- 執行預測

- 按下Shift+Enter依序執行下列指令。

執行預測

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

```
C:\Users\nan\anaconda3\lib\site-packages\tensorflow\python\keras\engine\sequential.py:450: UserWarning: `model.predict_classes()` is deprecated and will be removed after 2021-01-01. Please use instead:* `np.argmax(model.predict(x), axis=-1)`,   if your model does multi-class classification   (e.g. if it uses a `softmax` last-layer activation).* `(model.predict(x) > 0.5).astype("int32")`,   if your model does binary classification   (e.g. if it uses a `sigmoid` last-layer activation).
  warnings.warn('`model.predict_classes()` is deprecated and '
```

```
prediction[:10]
```

```
array([7, 2, 1, 0, 4, 1, 4, 9, 5, 9], dtype=int64)
```

查看預測結果的前10筆資料

8.6 進行預測

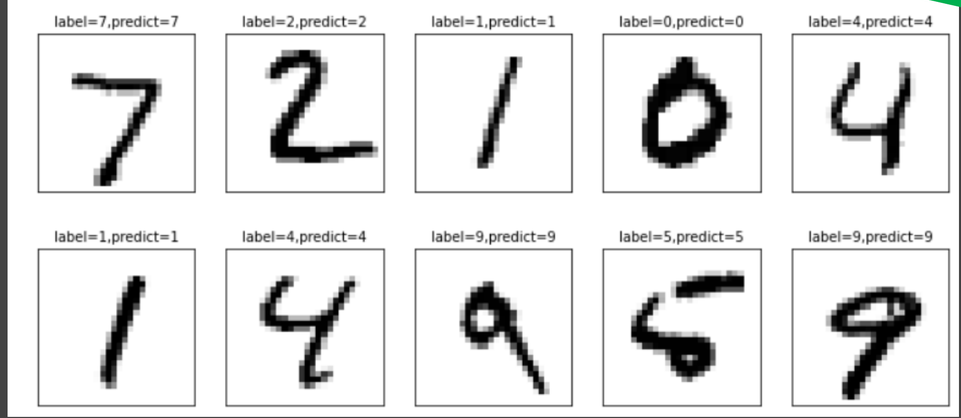
- 使用show_images_labels_prediction函數顯示前10筆預測結果

- 在命令列輸入下列程式碼，並且按下Shift+Enter依序執行。

```
import matplotlib.pyplot as plt
def plot_images_labels_prediction(images,labels,prediction, idx,num=10):
    fig = plt.gcf()
    fig.set_size_inches(12, 14)
    if num>25: num=25
    for i in range(0, num):
        ax=plt.subplot(5,5, 1+i) #建立subgraph子圖形為5行5列
        ax.imshow(images [idx], cmap='binary') #畫出subgraph子圖形
        title= "label=" +str(labels[idx]) #設定子圖形title，顯示標籤欄位
        if len(prediction)>0: #如果有傳入預測結果
            title+=",predict="+str(prediction[idx]) #標題title加入預測結果
        ax.set_title(title,fontsize=10) #設定子圖形的標題title與大小
        ax.set_xticks([]);ax.set_yticks([]) #設定不顯示刻度
        idx+=1 #讀取下一筆
    plt.show()
```

貼上第六章建立的
show_images_labels_prediction函數

```
plot_images_labels_prediction(x_Test,y_Test,prediction,idx=0)
```



顯示10筆預測結果

8.7

顯示混淆矩陣
(Confusion Matrix)

8.7 顯示混淆矩陣

- 使用pandas crosstab建立混淆矩陣

- 在命令列輸入下列程式碼，並且按下Shift+Enter執行

匯入pandas模組，後續會以pd引用

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

predict	0	1	2	3	4	5	6	7	8	9
label										
0	977	0	0	0	0	0	1	1	1	0
1	0	1133	2	0	0	0	0	0	0	0
2	1	1	1028	0	0	0	0	1	1	0
3	0	0	1	1000	0	6	0	1	2	0
4	0	0	1	0	970	0	1	0	0	10
5	2	0	0	1	0	887	1	0	0	1
6	8	2	0	0	1	2	944	0	1	0
7	0	2	3	0	0	0	0	1019	1	3
8	7	1	2	1	0	0	0	1	959	3
9	0	3	1	0	3	3	0	3	1	995

實際上是8，但被辨識成0

正確的預測

使用pd.crosstab建立混淆矩陣，輸入下列參數:

- 測試資料數字影像的真實值
- 測試資料數字影像的預測結果
- 設定行的名稱是label
- 設定列的名稱是predict

8.8 結論

8.8 結論

- 使用卷積神經網路CNN(convolutional neural network)，辨識Mnist料集中的手寫數字，其分類精度接近為99%。
- 這只是單色手寫數字辨識，相對來說比較單。
- 下一章我們將介紹更具挑戰性，使用卷積神經網路，辨識CIFAR-10資料集，辨識彩色影像共10個分類：飛機、汽車、鳥、貓、鹿、狗、青蛙、船、卡車。