AIoT：樹莓派應用

補充 B：深度學習 X 影像處理

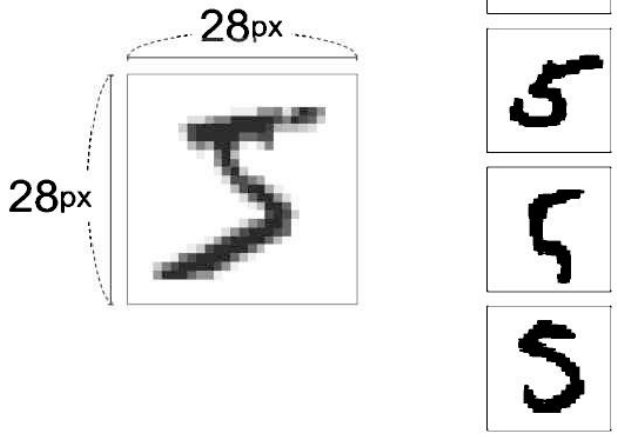
賴秉樑 debugger
學院創辦人

課程網址 https://max543.com/debugger

max 極限翻轉學院
青少年與兒童程式教育

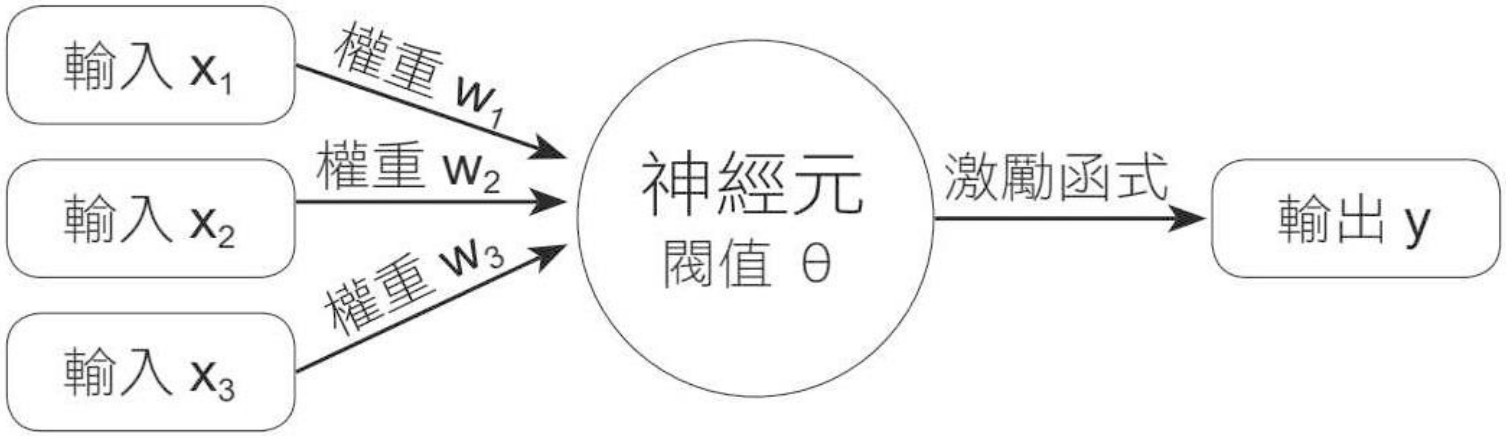
人類的大腦神經

- 人類辨識影像很簡單...，但電腦辨識卻很難...。



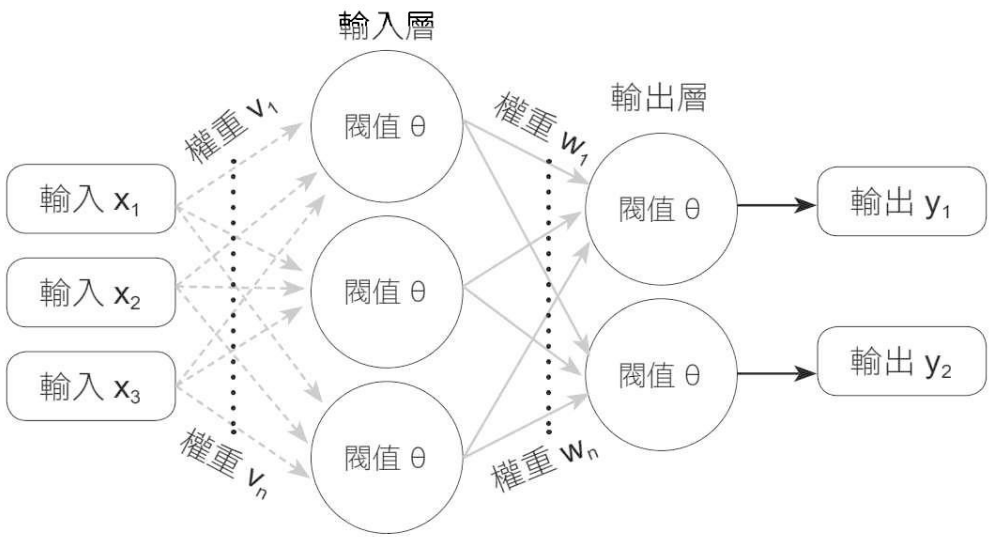
神經網路 (Neural Network)

- 閾值：設下一個門檻，當（輸入 x 權重）與閾值做比較，大於閾值則經過激勵函式轉換，輸出到下一個神經元。

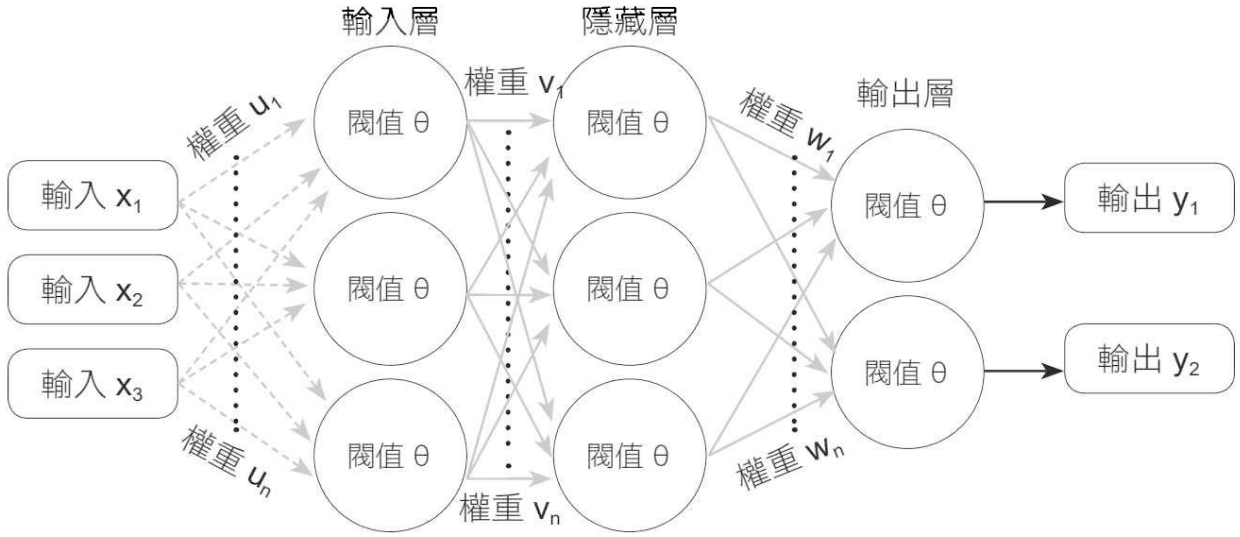


▲ 單一神經元模型：單層感知器

感知器的模型



▲ 2 層感知器



▲ 多層感知器

一個或多個隱藏層稱之為
多層感知器 (MLP, Multilayer Perceptron)

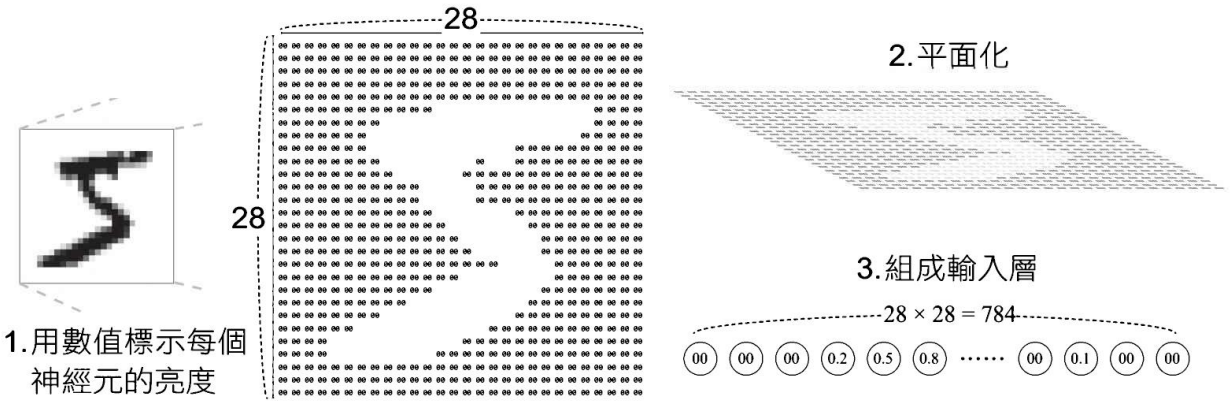
多層感知器的運作

輸入層

多重感知器的流程

輸出層

將影像的一個一個灰階像素（神經元），以 0（最黑）~ 1（最白）之間來儲存。



本質是模仿人類大腦的細胞被激發，引發其他神經細胞的連鎖反應。

理想情況：第二層能辨識不同的筆畫，下一層能合併其他部分，最後在輸出層將代表數字的神經元點亮。

有 10 個神經元，各代表了數字 0 ~ 9。

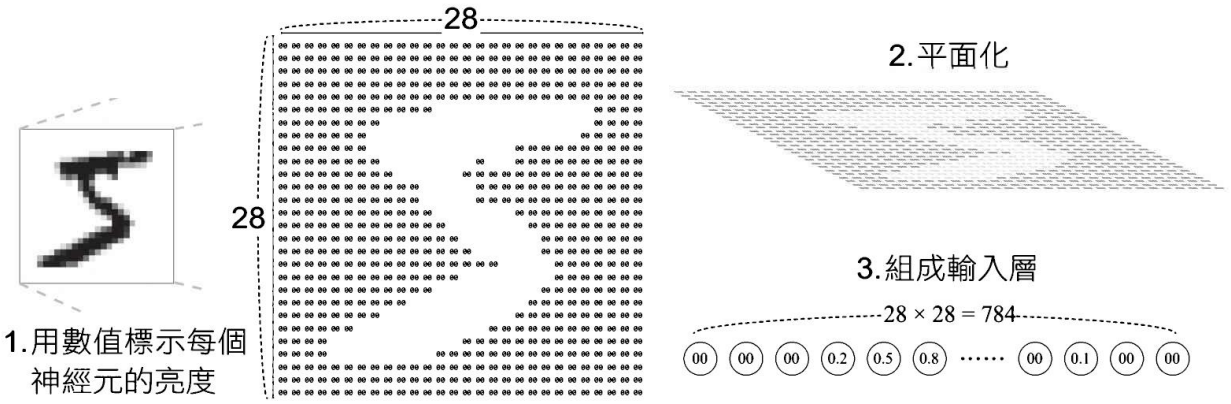
多層感知器的運作

輸入層

多重感知器的流程

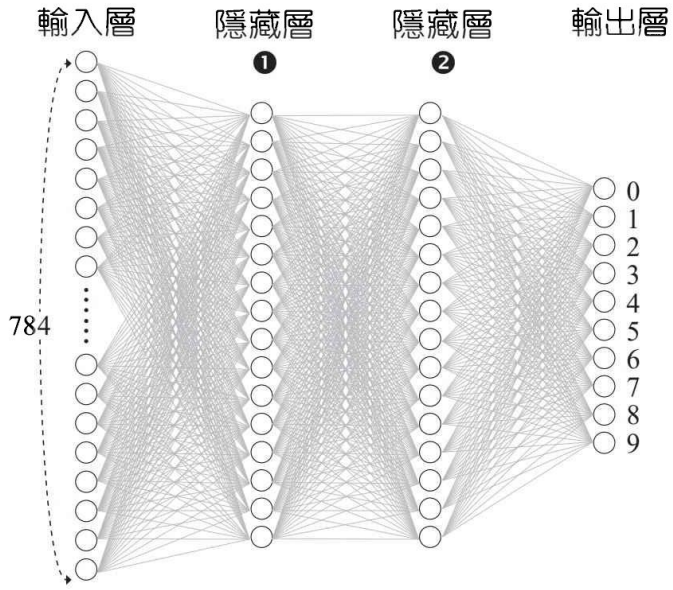
輸出層

將影像的一個一個灰階像素（神經元），以 0（最黑）~ 1（最白）之間來儲存。



本質是模仿人類大腦的細胞被激發，引發其他神經細胞的連鎖反應。

理想情況：第二層能辨識不同的筆畫，下一層能合併其他部分，最後在輸出層將代表數字的神經元點亮。

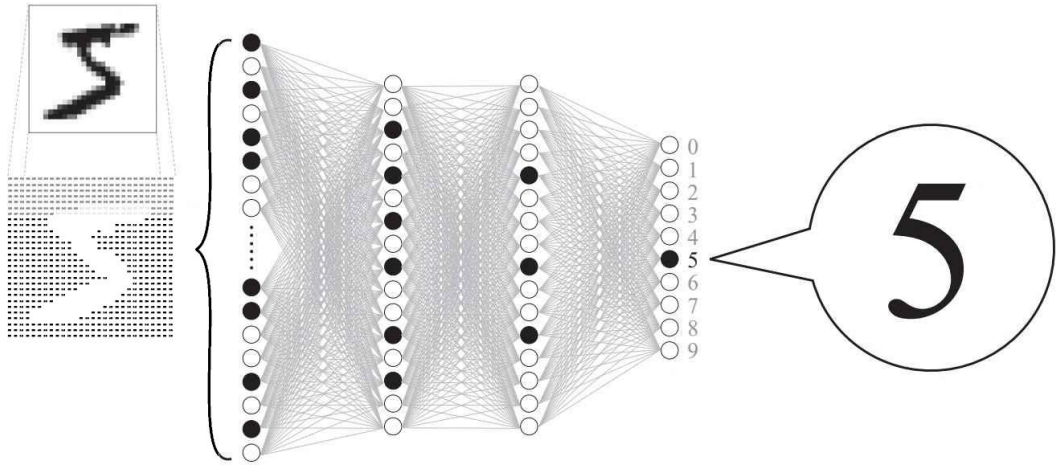


有 10 個神經元，各代表了數字 0 ~ 9。

多層感知器的運作

輸入層

將影像的一個一個灰階像素（神經元），以 0（最黑）~ 1（最白）之間來儲存。



多重感知器的流程

本質是模仿人類大腦的細胞被激發，引發其他神經細胞的連鎖反應。

理想情況：第二層能辨識不同的筆畫，下一層能合併其他部分，最後在輸出層將代表數字的神經元點亮。

輸出層

有 10 個神經元，各代表了數字 0 ~ 9。

多層感知器的運作

輸入層

多重感知器的流程

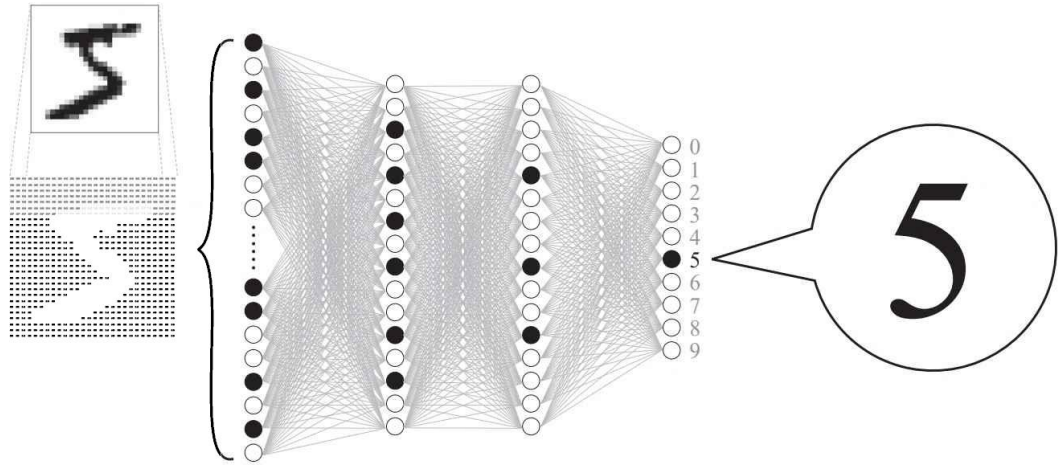
輸出層

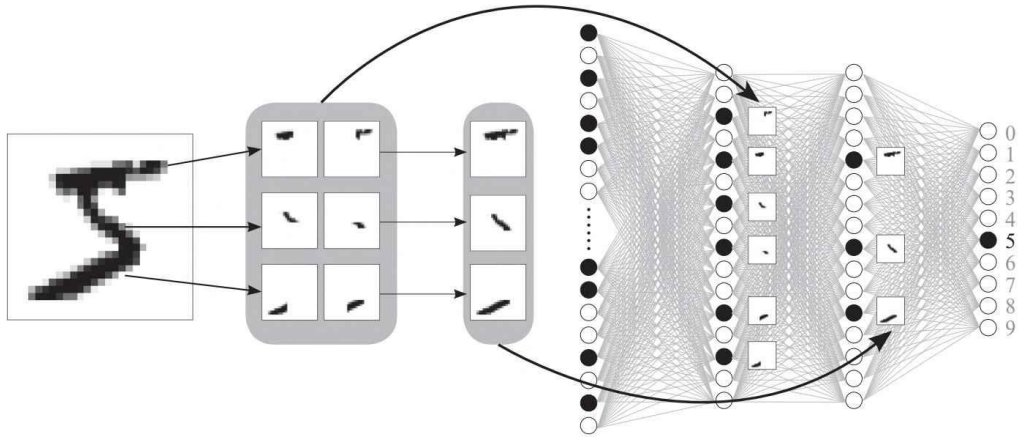
將影像的一個一個灰階像素（神經元），以 0（最黑）~ 1（最白）之間來儲存。

本質是模仿人類大腦的細胞被激發，引發其他神經細胞的連鎖反應。

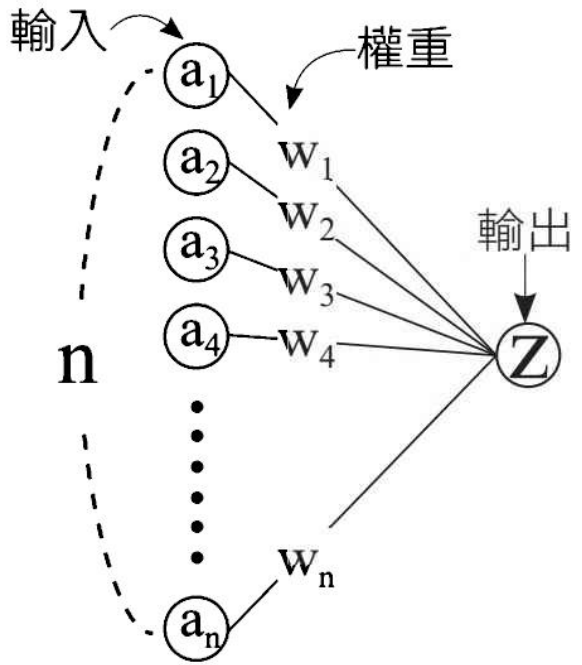
理想情況：第二層能辨識不同的筆畫，下一層能合併其他部分，最後在輸出層將代表數字的神經元點亮。

有 10 個神經元，各代表了數字 0 ~ 9。





各層傳遞的數學模型



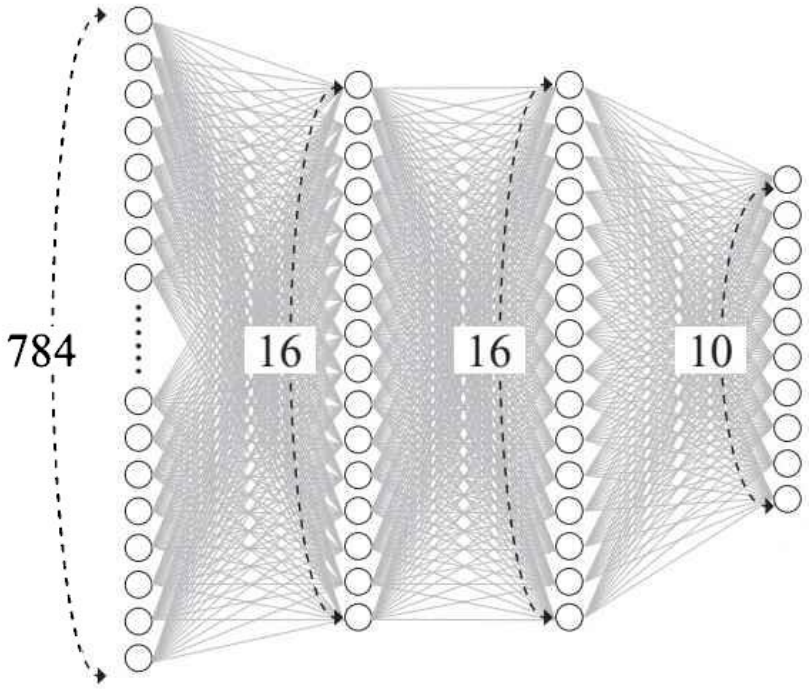
$$f(w_1a_1+w_2a_2+w_3a_3+w_4a_4\ldots+w_na_n-b)$$

激勵函式 偏置

$$f\left(\sum_{i=1}^{n} w_ia_i - b\right)$$

機器學習的目的

- 以手寫辨識數字為例：至少超過 13,000 個要調整的參數。
- 用機器學習的運算來達成這浩大的工程。



權重數

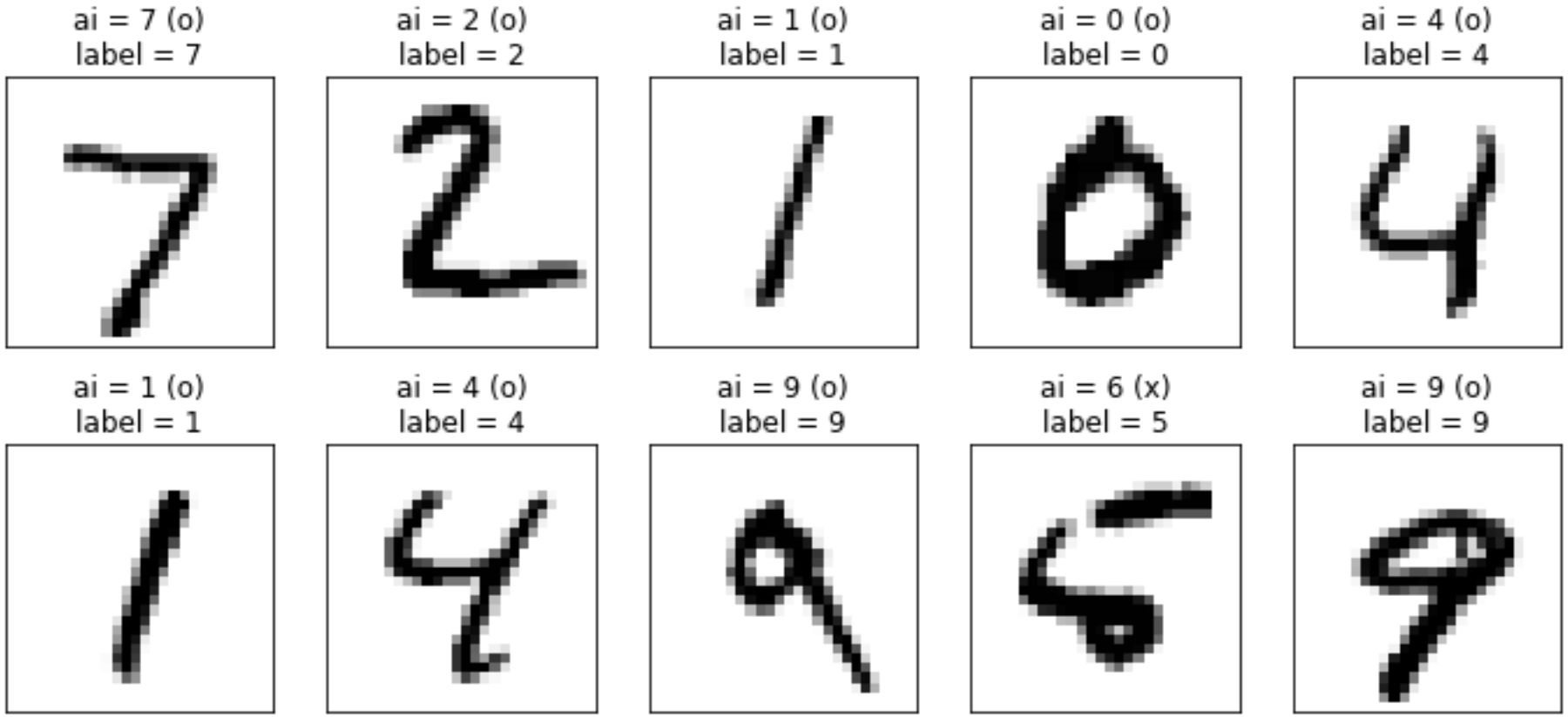
$$784 \times 16 + 16 \times 16 + 16 \times 10 = 12{,}960$$

偏置數

$$16 + 16 + 10 = 42$$

影像處理的 "Hello World"：Mnist 資料集

- 辨識手寫數字 0 ~ 9。

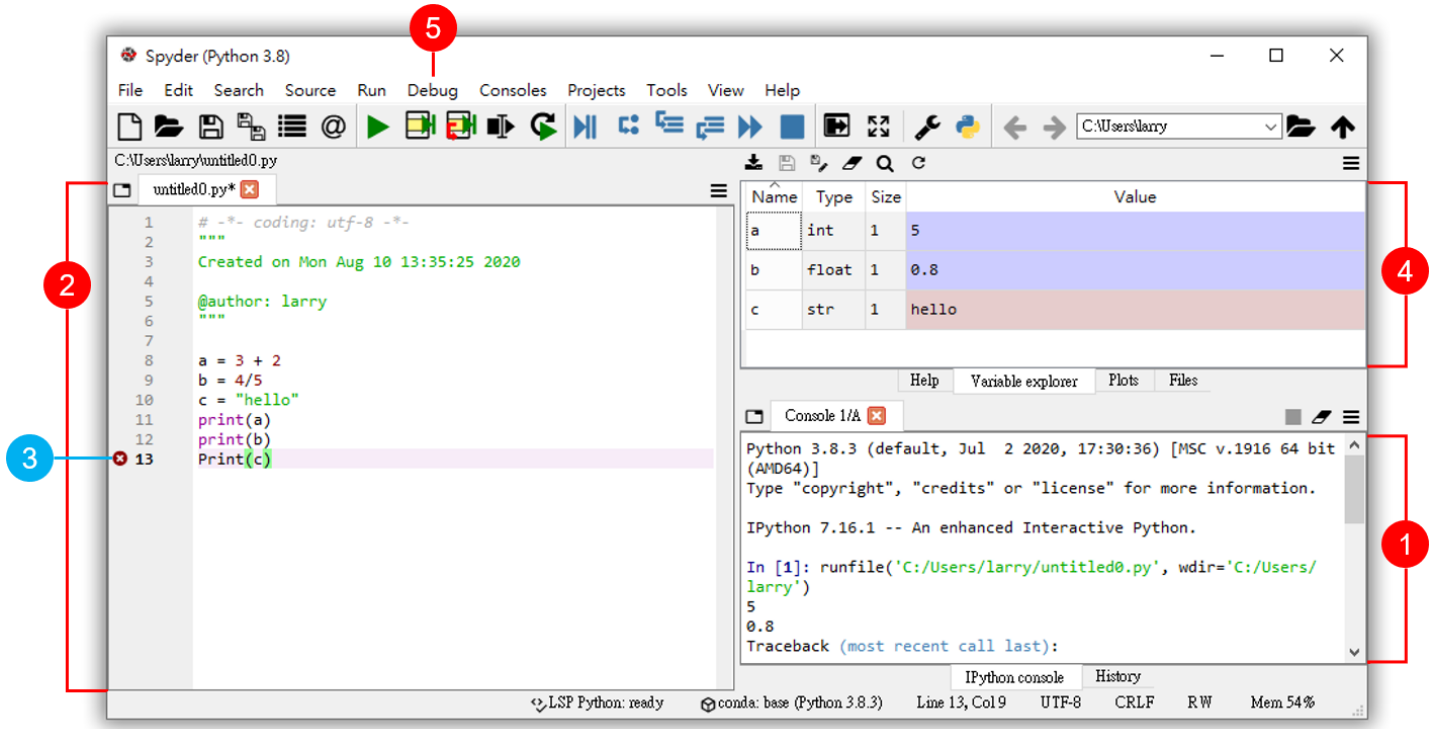


先看機器學習的例子

影像識別：多層神經網路（MLP）

Anaconda

- Anaconda 包含了 Python 直譯器與許多好用的開發工具，而且提供了數百個第三方的函式庫，涵蓋科學、數學、工程、數據分析的 Python 模組。
  - 網址：https://www.anaconda.com/products/individual，下載並安裝適當的版本。
  - 安裝後，執行 Anaconda 的整合開發環境 (IDE)：Spyder。



1. IPython 主控台：用來測次單行程式碼，以及程式的輸出資訊。
2. 程式編輯窗格：編寫程式專案的地方。
3. 語法錯誤提示：在執行程式前，可事先發現語法錯誤的程式碼。
4. 變數窗格：可知道目前程式物件的值為何。
5. Debug (除錯選單)：可依序執行程式碼，用來查找錯誤與追蹤變數值。

若使用 Anacodna，需安裝模型框架

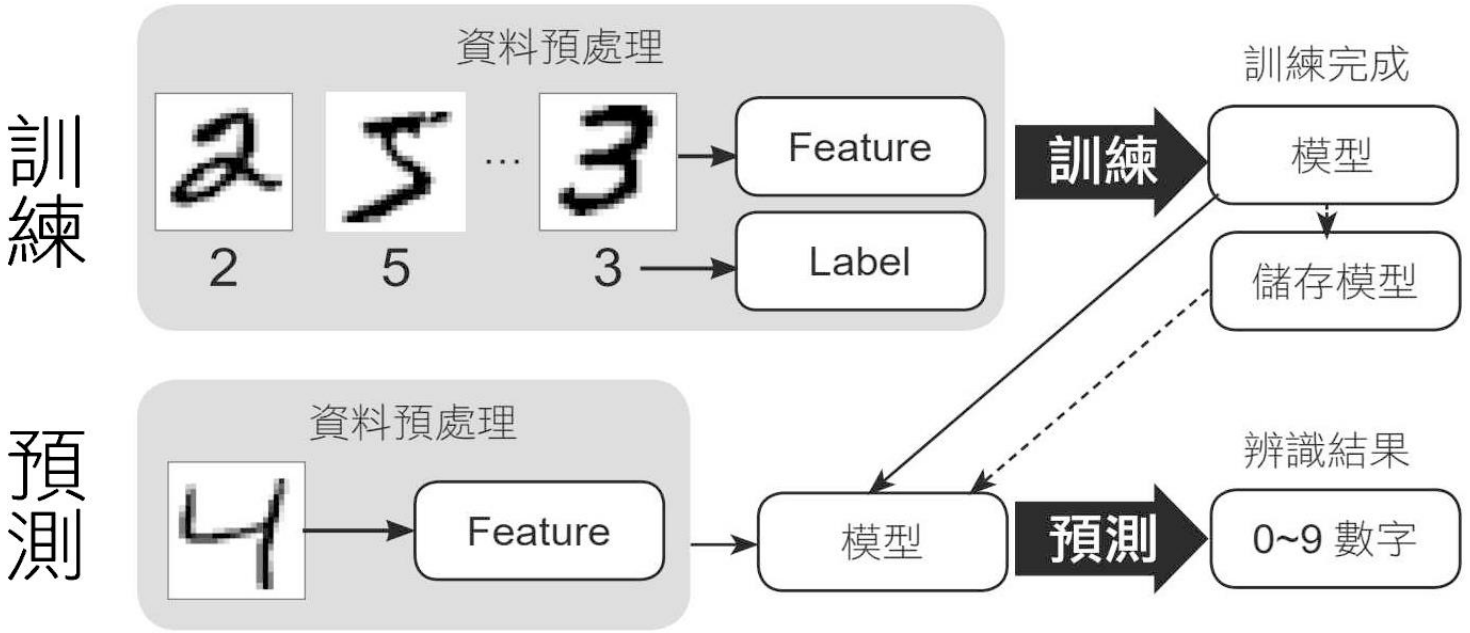
本範例會用到 tensorflow 與 keras：

- 開啟 Anacodna Prompt 視窗，輸入下列指令：

```
>>> pip install tensorflow
>>> pip install keras
```

訓練與預測

- 訓練 (Train)：Mnist 共有 60,000 訓練資料，將 Feature (數字圖片特徵值) 與 Label (數字真實值) 經過預處理，然後進行訓練。
- 預測 (Predict)：將要預測的數字圖片，先經過預處理，送給模型作預測。



Keras_Mnist_MLP.py

將 Feature 特徵值換為 784 個 float 數字標準化，將 Label 轉換為 One-Hot Encoding 編碼。

```
# 匯入 mnist 資料集
from keras.datasets import mnist
from keras.utils import np_utils

# 建立訓練資料和測試資料，包括訓練特徵集、訓練標籤和測試特徵集、測試標籤
(train_feature, train_label), (test_feature, test_label) = mnist.load_data()

# 將 Features 特徵值換為 784 個 float 數字的 1 維向量
train_feature_vector = train_feature.reshape(len(train_feature),
784).astype('float32')
test_feature_vector = test_feature.reshape(len(test_feature),
784).astype('float32')

# Features 特徵值標準化，將 0 ~ 255 的數字，除以 255 得到 0 ~ 1 之間的浮點數
train_feature_normalize = train_feature_vector/255
test_feature_normalize = test_feature_vector/255

# Label 轉換為 One-Hot Encoding 編碼
train_label_onehot = np_utils.to_categorical(train_label)
test_label_onehot = np_utils.to_categorical(test_label)
```

Label 資料預處理

- 0 ~ 9 數字的 One-Hot Encoding (一位有效編碼)，增加模型效率：

真實值	0	1	2	3	4	5	6	7	8	9
0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0
…										
9	0	0	0	0	0	0	0	0	0	1

Keras_Mnist_MLP.py

資料預處理

建立多重感知器

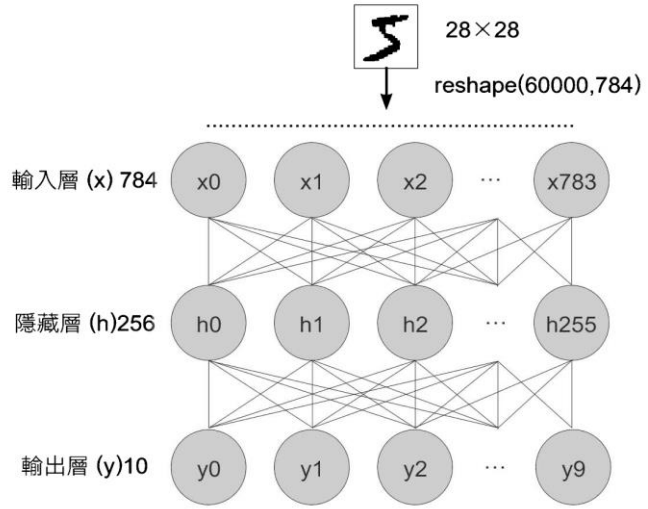
訓練模型

評估準確率

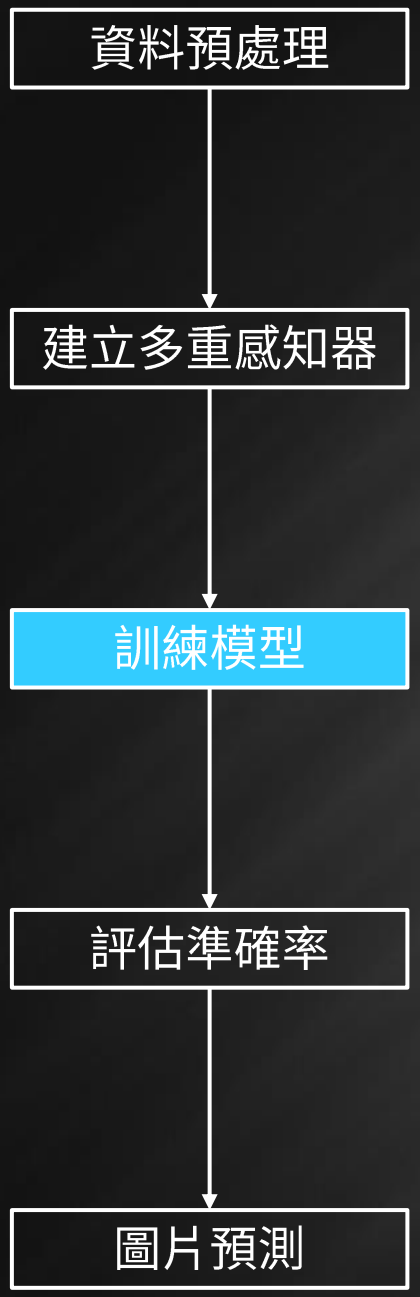
圖片預測

建立 Sequential 模型。

```
# 匯入 keras 的 Sequential 模型
from keras.models import Sequential
from keras.layers import Dense

# 建立模型
model = Sequential()

# 輸入層：784，隱藏層：256，輸出層：10
model.add(Dense(units = 256,
                input_dim = 784,
                kernel_initializer = 'normal',
                activation = 'relu'))
model.add(Dense(units = 10,
                kernel_initializer = 'normal',
                activation = 'softmax'))
```



Keras_Mnist_MLP.py



以 compile() 法定義 Loss 損失函式、Optimizer 最佳化方法，和 metrics 評估準確率的方法。

```
# 定義訓練方式
model.compile(loss = 'categorical_crossentropy',
              optimizer = 'adam', metrics = ['accuracy'])

# 以 (train_feature_normalize, train_label_onehot) 資料訓練，
# 訓練資料保留 20% 作驗證，訓練 10 次、每批次讀取 200 筆資料，顯示簡易訓練過程
train_history = model.fit(x = train_feature_normalize,
                          y = train_label_onehot, validation_split = 0.2,
                          epochs = 10, batch_size = 200, verbose = 2)
```

fit() 可以進行訓練，訓練時必須設定訓練資料和標籤。語法如下：

```
model.fit(x = 特徵值, y = 標籤, validation_split = 驗證資料百分比,
          epochs = 訓練次數, batch_size = 每批次有多少筆, verbose = n)
```

- validation_split：例如：0.2 表示將訓練資料保留 20%，當作驗證資料。
- verbose：是否顯示訓練過程。0：不顯示，1：詳細顯示，2：簡易顯示。

Keras_Mnist_MLP.py

evaluate() 會回傳串列，代表評估模型的損失函式誤差 [0] 和準確率 [1]。

```
# 評估準確率
scores = model.evaluate(test_feature_normalize, test_label_onehot)
print('\n準確率 = ', scores[1])
```

Keras_Mnist_MLP.py

預測測試特徵圖片，並顯示圖像

```
# 預測
prediction = model.predict(test_feature_normalize)
prediction = np.argmax(prediction, axis = 1)

# 顯示圖像、預測值、真實值
show_images_labels_predictions(test_feature, test_label, prediction, 0)
```

模型儲存

影像識別：多層神經網路（MLP）

Keras_Mnist_saveModel.py

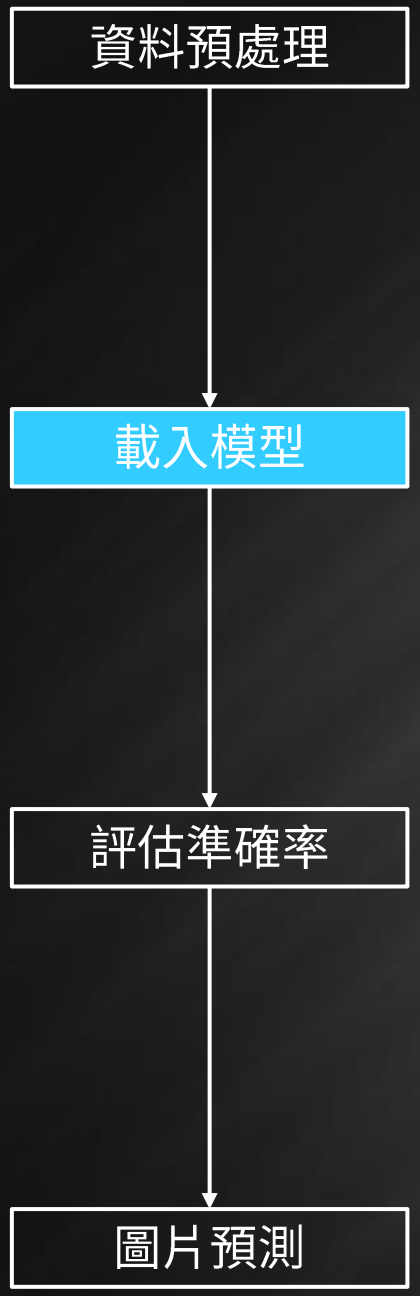
Keras 使用 HDF5 檔案系統 (.h5) 來儲存模型。

```
# 將模型儲存至 HDF5 檔案中
model.save('Mnist_mlp_model.h5')
print("Mnist_mlp_model.h5 模型儲存完畢!")
del model
```

載入模型

影像識別：多層神經網路（MLP）

Keras_Mnist_LoadModel.py



從 HDF5 檔案中載入模型。

```
# 匯入模組的 load_model 方法
from keras.models import load_model

# 從 HDF5 檔案中載入模型
print("載入模型 Mnist_mlp_model.h5")
model = load_model('Mnist_mlp_model.h5')
```

預測自己的圖片

影像識別：多層神經網路（MLP）

本範例會用到 OpenCV：

- 開啟 Anacodna Prompt 視窗，輸入下列指令：

```
>>> pip install opencv-python
```

Keras_Mnist_Predict.py

Keras 使用 HDF5 檔案系統 (.h5) 來儲存模型。

```
# 匯入相關模組
import glob, cv2
# 建立測試特徵集、測試標籤
files = glob.glob("imagedata\*.jpg" )
test_feature = []
test_label = []
for file in files:
    img = cv2.imread(file)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)      # 灰階
    _, img = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)      # 轉為反相黑白
    test_feature.append(img)
    label = file[10:11]            # "imagedata\1.jpg" 第 10 個字元 1 為 Label
    test_label.append(int(label))
test_feature=np.array(test_feature)                    # 串列轉為矩陣
test_label=np.array(test_label)                        # 串列轉為矩陣
# 將 Features 特徵值換為 784 個 float 數字的 1 維向量
test_feature_vector = test_feature.reshape(len(test_feature),
784).astype('float32')
# Features 特徵值標準化
test_feature_normalize = test_feature_vector/25
```

Keras_Mnist_Predict.py

Keras 使用 HDF5 檔案系統 (.h5) 來儲存模型。

```
# 匯入模組的 load_model 方法
from keras.models import load_model

# 從 HDF5 檔案中載入模型
print("載入模型 Mnist_mlp_model.h5")
model = load_model('Mnist_mlp_model.h5')
```

Keras_Mnist_Predict.py

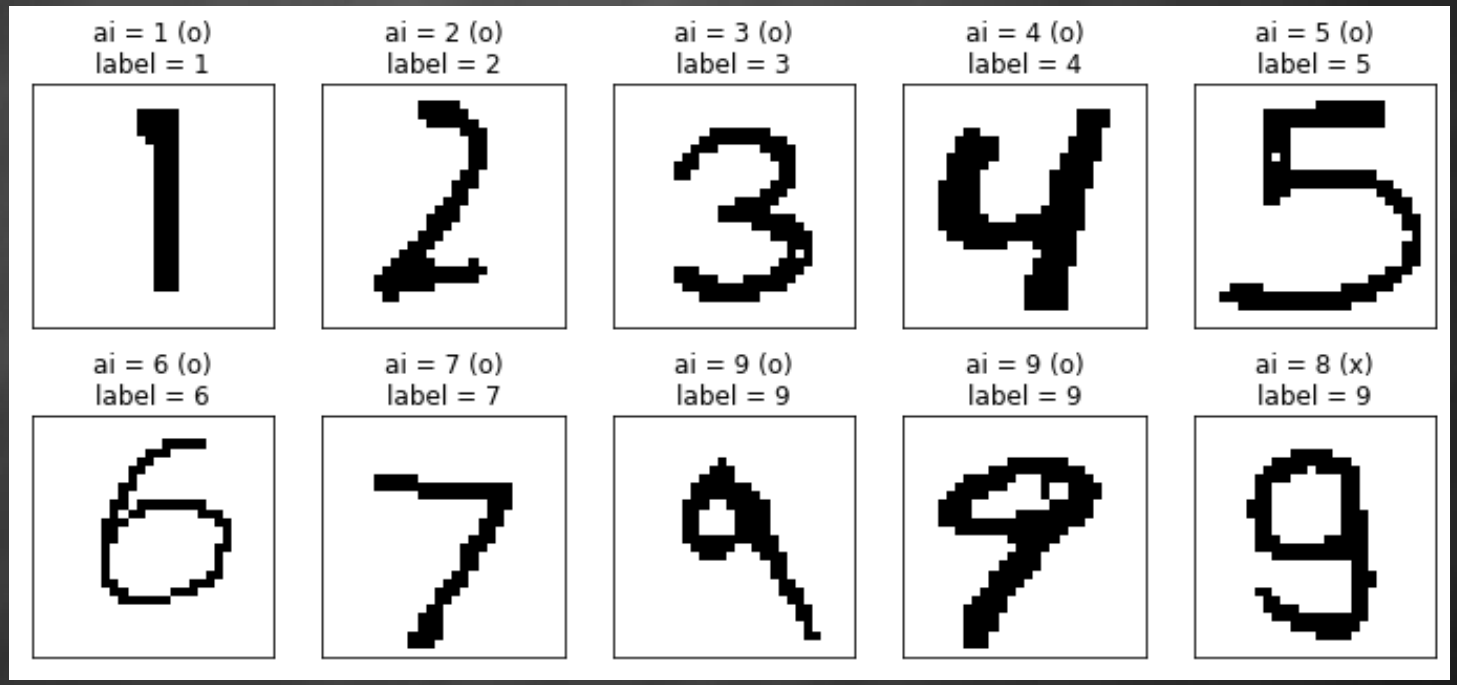
資料預處理

載入模型

圖片預測

預測測試特徵圖片，並顯示圖像

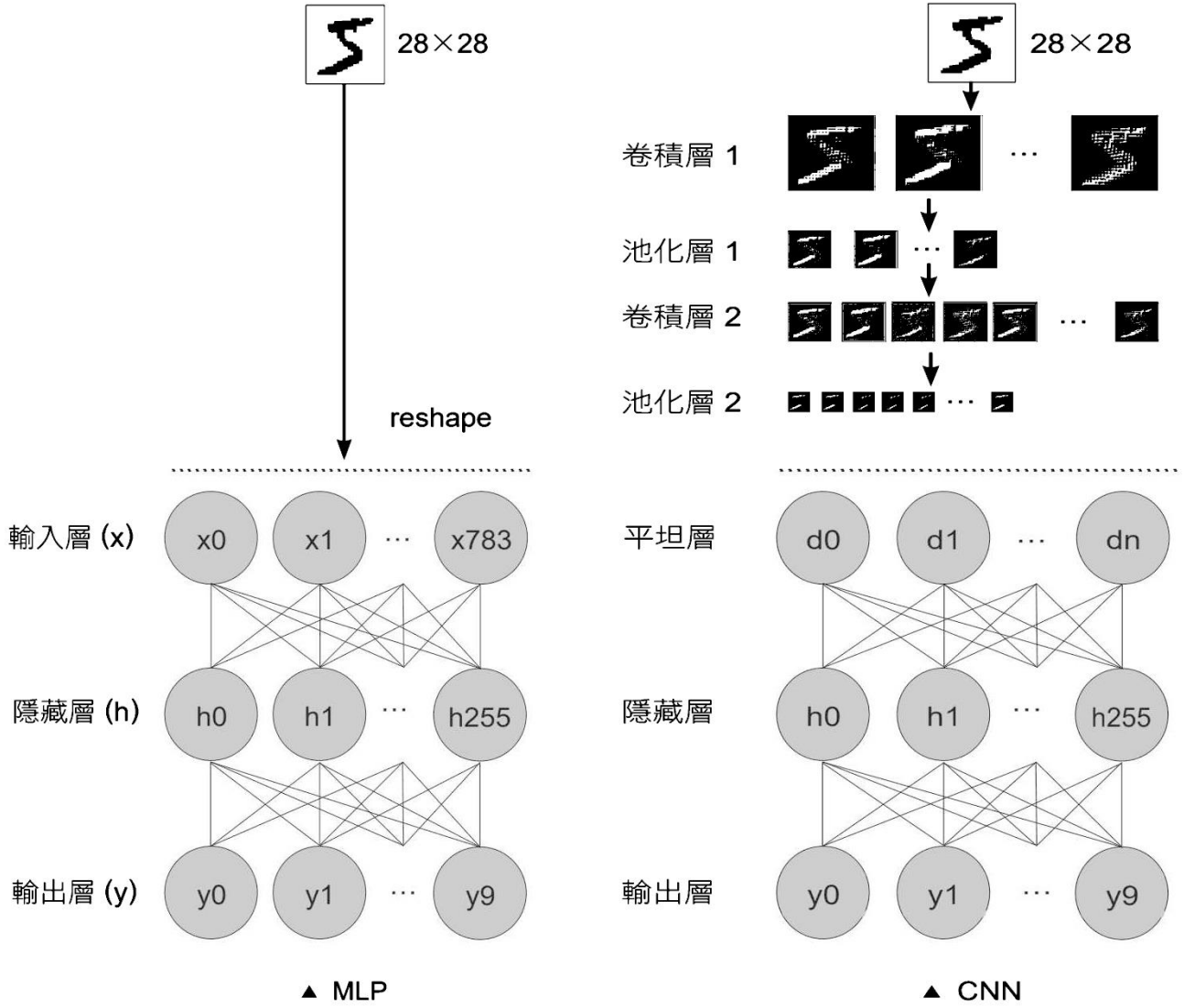
```
# 預測
prediction = model.predict(test_feature_normalize)
prediction = np.argmax(prediction, axis = 1)

# 顯示圖像、預測值、真實值
show_images_labels_predictions(test_feature, test_label, prediction, 0,
len(test_feature))
```



再看深度學習的例子

影像識別：卷積神經網路（CNN）

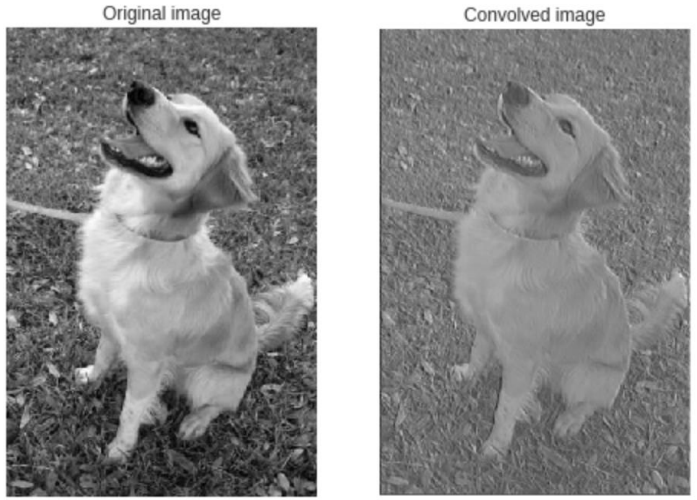
MLP vs CNN 結構



▲ MLP　　▲ CNN

卷積層 (Convolution Layer)

- 卷積層是將原始圖片與特定的濾鏡 (Feature Detector) 進行卷積運算。




```
# 建立卷積層 1
model.add(Conv2D(filters = 10,                  # filters：每一個 filter 都會產生一個濾鏡效果。
                 kernel_size = (3, 3),          # kernel_size：設定濾鏡大小，一般為 5*5 或 3*3。
                 padding = 'same',              # padding：設定卷積運算圖片大小，'same' 代表與原圖相同。
                 input_shape = (28, 28, 1), # input_shape：設定原圖大小，(28, 28, 1) 代表 28*28/張。
                 activation = 'relu'))          # activation：激勵函式，relu 函式會將小於 0 的資訊設為 0。
```

例如：上面有 10 個濾鏡，會產生 10 張 28 * 28 的卷積運算圖片。

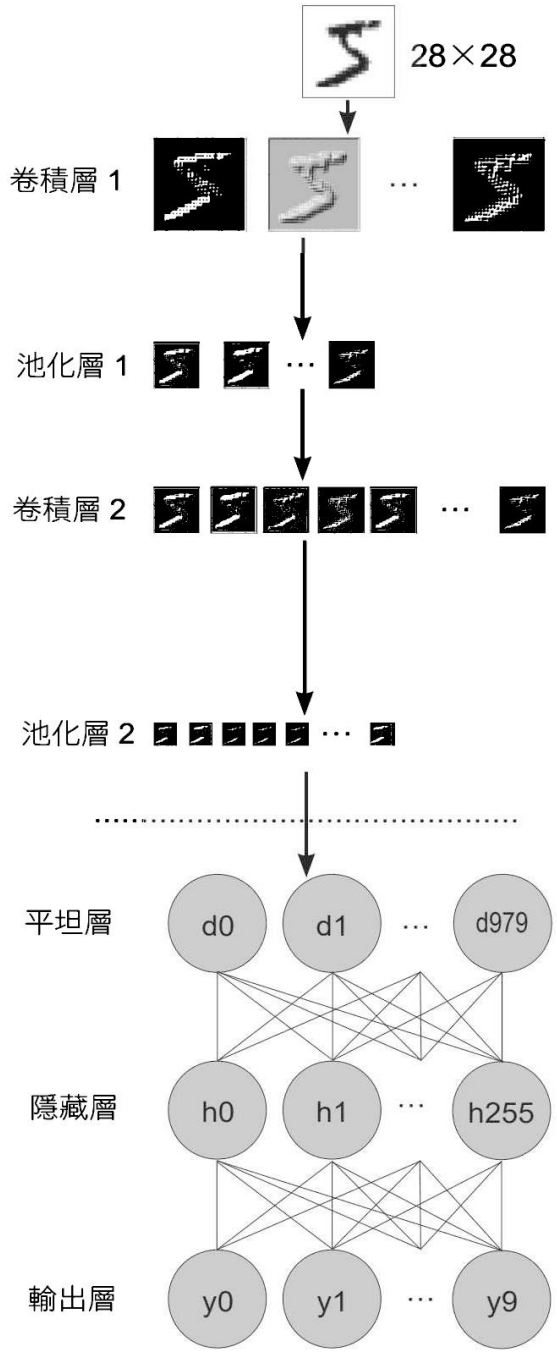
池化層（Pooling Layer）

- 池化層是採用 Max Pooling，指挑出矩陣中的最大值，相當於只挑出圖片局部最明顯的特徵，可縮減卷積層產生的卷積運算圖片數量。

```
# 建立池化層 1
model.add(MaxPooling2D(pool_size = (2, 2)))
```

例如：10 張 28*28 的卷積運算圖片。

- pool_size：pool_size = (2, 2))，就會得到 10 張 14*14。
- pool_size：pool_size = (4, 4))，就會得到 10 張 7*7。




```
# 卷積層 1：10 個 28*28 卷積圖片
Conv2D(filters=10,
       kernel_size=(3,3),
       padding='same',
       input_shape=(28,28,1),
       activation='relu')

# 池化層 1：10 個 14*14 圖片
MaxPooling2D(pool_size=(2, 2))

# 卷積層 2：20 個 14*14 卷積圖片
Conv2D(filters=20,
       kernel_size=(3,3),
       padding='same',
       activation='relu')

# 池化層 2：20 個 7*7 圖片
MaxPooling2D(pool_size=(2, 2))

# 建立平坦層：20*7*7=980 個神經元
Flatten()

# 建立隱藏層：256 個神經元
Dense(256, activation='relu')

# 建立輸出層：10 個神經元
Dense(10,activation='softmax')
```

Keras_Mnist_CNN.py

將 Feature 特徵值換為 (60000, 28, 28, 1) 的四維向量，再以 float 數字標準化，將 Label 轉換為 One-Hot Encoding 編碼。

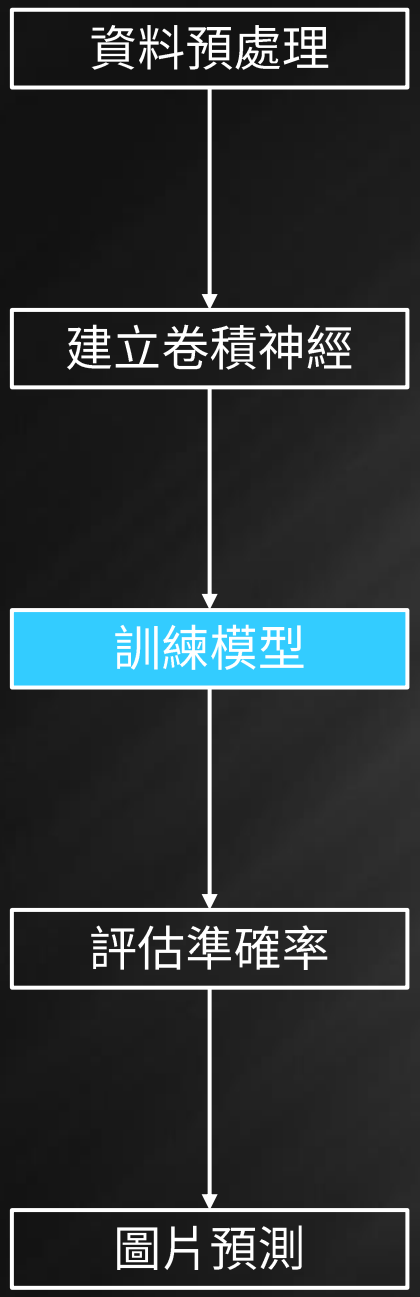
```
# 匯入 mnist 資料集
from keras.datasets import mnist
from keras.utils import np_utils

# 建立訓練資料和測試資料，包括訓練特徵集、訓練標籤和測試特徵集、測試標籤
(train_feature, train_label), (test_feature, test_label) = mnist.load_data()

# 將 Features 特徵值換為 60000*28*28*1 的 4 維矩陣
train_feature_vector =train_feature.reshape(len(train_feature),
28, 28, 1).astype('float32')
test_feature_vector = test_feature.reshape(len(test_feature),
28, 28, 1).astype('float32')

# Features 特徵值標準化
train_feature_normalize = train_feature_vector/255
test_feature_normalize = test_feature_vector/255

# Label 轉換為 One-Hot Encoding 編碼
train_label_onehot = np_utils.to_categorical(train_label)
test_label_onehot = np_utils.to_categorical(test_label)
```

Keras_Mnist_CNN.py

建立含有卷積層 1、池化層 1、卷積層 2、池化層 2、平坦層、隱藏層、輸出層。

```
# 匯入 mnist 資料集
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense

# 建立模型
model = Sequential()
#建立卷積層 1
model.add(Conv2D(filters = 10,
                 kernel_size = (3, 3),
                 padding = 'same',
                 input_shape = (28, 28, 1),
                 activation = 'relu'))

# 建立池化層 1
model.add(MaxPooling2D(pool_size=(2, 2)))   # (10, 14, 14)：10 個 14*14 圖片

# 建立卷積層 2
model.add(Conv2D(filters = 20,
                 kernel_size = (3, 3),
                 padding = 'same',
                 activation = 'relu'))

# 建立池化層 2
model.add(MaxPooling2D(pool_size=(2, 2)))   # (20, 7, 7)：20 個 7*7 圖片
# Dropout 層防止過度擬合，斷開比例:0.2
model.add(Dropout(0.2))
# 建立平坦層：20*7*7 = 980 個神經元
model.add(Flatten())
# 建立隱藏層
model.add(Dense(units = 256, activation = 'relu'))
# 建立輸出層
model.add(Dense(units = 10, activation = 'softmax'))
```

Keras_Mnist_CNN.py



以 compile() 法定義 Loss 損失函式、Optimizer 最佳化方法，和 metrics 評估準確率的方法。

```
# 定義訓練方式
model.compile(loss = 'categorical_crossentropy',
              optimizer = 'adam', metrics = ['accuracy'])

# 以 (train_feature_normalize, train_label_onehot) 資料訓練
# 訓練資料保留 20% 作驗證，訓練 10 次、每批次讀取 200 筆資料，顯示簡易訓練過程
train_history = model.fit(x = train_feature_normalize,
                          y = train_label_onehot, validation_split = 0.2,
                          epochs = 10, batch_size = 200, verbose = 2)
```

Keras_Mnist_CNN.py

evaluate() 會回傳串列，代表評估模型的損失函式誤差 [0] 和準確率 [1]。

```
# 評估準確率
scores = model.evaluate(test_feature_normalize, test_label_onehot)
print('\n準確率 = ', scores[1])
```

Keras_Mnist_CNN.py

資料預處理

建立卷積神經

訓練模型

評估準確率

圖片預測

predict() 進行預測，下面是以特徵值標準化後的 test_feature_normalize 作預測。

```
# 預測
prediction = model.predict(test_feature_normalize)
prediction = np.argmax(prediction, axis = 1)
```

顯示 Mnist 資料集的前 10 筆預測結果。

```
# 顯示圖像、預測值、真實值
show_images_labels_predictions(test_feature, test_label, prediction, 0)
```