

# 實戰圖形辨識

## 14.1 圖形辨識原理

請思考一下，圖形影像要如何透過 MLP 類神經做辨識呢？請先想一下之前的鳶尾花範例程式，當時我們使用花瓣花萼的寬度和高度來做特徵值，而花的種類是答案。

若以圖形來看，答案應該是這個圖要表達的意思，例如 0、1、A、B... 等文字的圖形，比較難以理解的是圖形要怎麼轉換成特徵值？一般來說會有幾個選項：

- ✓ 找出圖形的邊緣
- ✓ 依照筆畫，例如中文字的筆劃直橫點豎方向
- ✓ 依照圖的內容

為了讓各位瞭解圖形辨識的原理，這裡先假設要辨識的圖片，只有圈圈和叉叉這二種圖形。請先在紙上畫上許多圈圈和叉叉，並拍照成為圖檔，如下圖所示。



圖 14-1 圈圈圖與叉叉圖

因為只要解析度高，每一個圖形看起來就會不一樣。再來請把圈圈的圖縮小到寬度和高度各有三個點的大小，這裡為了簡化問題，只有黑色和白色二種顏色，忽略掉灰階和彩色的問題。這樣處理後，每一張圈圈和叉叉的圖片就會如下圖所示。

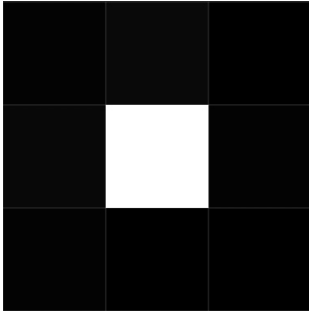


圖 14-2 圈圈圖縮小成 3x3 的大小

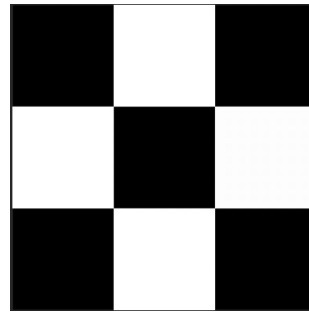


圖 14-3 叉叉圖縮小成 3x3 的大小

請試著想一下，該如何把這 3x3 的資料，用程式的矩陣存放這 3x3 的資料中？方法如下程式所示。

#### ◀ 範例程式：1\_image\_circle\_cross.py

```

1.  ... # 相同，省略
2.  import numpy as np
3.  import matplotlib.pyplot as plt # 繪圖函式庫
4.  circle1=np.array([[1,1,1], # 圈圈圖
5.                   [1,0,1],
6.                   [1,1,1]])
7.  plt.subplot(1,2,1) # 指定繪製在左邊
8.  plt.imshow(circle1) # 繪圖
9.
10. cross1=np.array([[1,0,1], # 叉叉圖
11.                  [0,1,0],
12.                  [1,0,1]])
13. plt.subplot(1,2,2) # 指定繪製在右邊

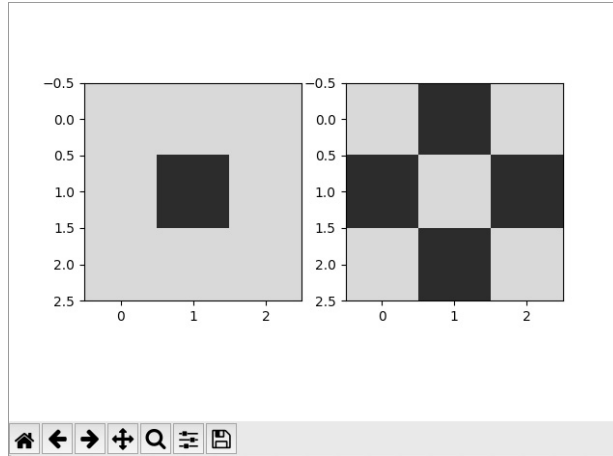
```

```

14. plt.imshow(cross1)          # 繪圖
15. plt.show()                 # 顯示

```

### 執行結果



教學影片 1\_image\_circle\_cross.mp4

## 14.2 將圖片轉換成特徵值

如何將圈圈和叉叉的圖片轉換成 MLP 的特徵值呢？請試著回憶一下，當初是用花瓣和花萼的寬度和高度來做特徵值，也就是每一筆資料有 4 個特徵值，就像是 [4.6, 3.1, 1.5, 0.2] 一維資料。那能否把 3x3 的圈圈圖 [[1,1,1], [1,0,1], [1,1,1]] 這樣的二維資料，轉換成一維？可以透過 `flatten()` 或 `reshape([9])` 函式轉換成一維資料 [1,1,1,1,0,1,1,1,1]，而叉叉圖轉換後就是 [1,0,1,0,1,0,1,0,1]。

### 範例程式：2\_image\_circle\_cross1D.py

```

1. ... # 相同，省略
2. import numpy as np
3. import matplotlib.pyplot as plt # 繪圖函式庫
4. circle1=np.array([[1,1,1], # 圈圈圖
5.                   [1,0,1],
6.                   [1,1,1]])

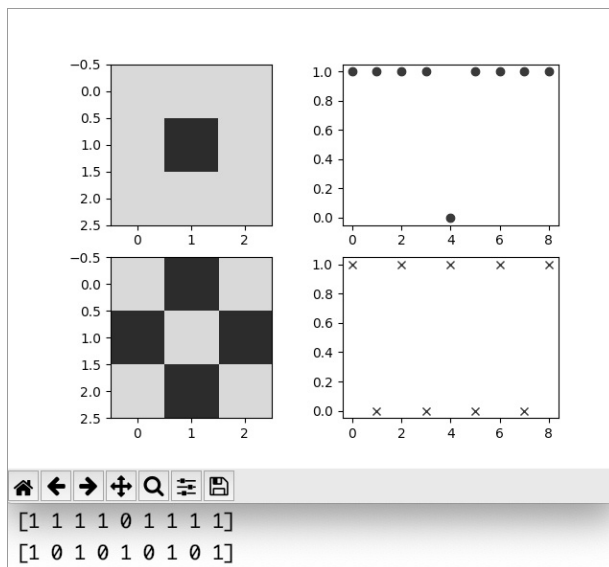
```

```

7. plt.subplot(2,2,1)           # 指定繪製在左上
8. plt.imshow(circle1)         # 繪圖
9.
10. circle2=circle1.flatten()   # 2D 轉 1D 方法 1
11. circle2=circle1.reshape([9]) # 2D 轉 1D 方法 2
12. print(circle2)             # 輸出
13. plt.subplot(2,2,2)         # 指定繪製在左下
14. plt.plot(circle2, 'ob')    # 繪圖
15.
16. cross1=np.array([[1,0,1],   # 叉叉圖
17.                  [0,1,0],
18.                  [1,0,1]])
19. plt.subplot(1,2,2)         # 指定繪製在右上
20. plt.imshow(cross1)        # 繪圖
21.
22. cross2=cross1.reshape([9])  # 2D 轉 1D
23. print(cross2)             # 輸出
24. plt.subplot(2,2,4)         # 指定繪製在右下
25. plt.plot(cross2, 'xb')    # 繪圖
26.
27. plt.show()                 # 顯示

```

### 執行結果



## 14.3 多層感知器 MLP 實戰圖形辨識

既然鳶尾花是用花瓣和花萼的 4 個特徵值來做訓練，那  $3 \times 3$  圖形也把它轉換成 9 個點的特徵值。接下來，請思考一下該如何透過 MLP 類神經做辨識呢？

沒錯，這幾乎和鳶尾花的 MLP 範例程式一模一樣，但其中一個不同點是鳶尾花有四個特徵值，而這裡是九個特徵值，另外還有一個不同點是鳶尾花有三個答案，而這邊只有圈圈和叉叉這兩種答案。

### ◀ 範例程式：3\_image\_circle\_cross\_MLP.py

```
1. ... # import, 省略
2. circle1=np.array([[1,1,1], # 圈圈圖
3. [1,0,1],
4. [1,1,1]])
5. circle2=circle1.flatten() # 2D 轉 1D 方法 1
6.
7. cross1=np.array([[1,0,1], # 叉叉圖
8. [0,1,0],
9. [1,0,1]])
10. cross2=cross1.reshape([9]) # 2D 轉 1D
11. X = np.array([circle2,cross2]) # 定義 X 資料
12. Y = np.array([0,1]) # 定義 Y 資料
13. category=2 # 有 2 種 Label 答案 Y
14. dim=9 # 有 9 個 Feature 特徵 X
15. Y2=tf.keras.utils.to_categorical(Y, num_classes=(category)) # 轉碼
16.
17. model = tf.keras.models.Sequential() # 加入順序層
18. model.add(tf.keras.layers.Dense(units=10, # 加入 10 個神經元
19. activation=tf.nn.relu, # 也能寫成 activation='relu'
20. input_dim= dim )) # 每一筆有四個資料
21. model.add(tf.keras.layers.Dense(units=10, # 加入 10 個神經元
22. activation=tf.nn.relu)) # 也能寫成 activation='relu'
23. model.add(tf.keras.layers.Dense(units= # 3 個答案的神經元輸出的
24. category,activation=tf.nn.softmax )) # 也能寫成 activation='softmax'
24. model.compile(optimizer='adam',
25. loss=tf.keras.losses.categorical_crossentropy,
26. metrics=['accuracy'])
27. model.fit(X, Y2, # 進行訓練
28. epochs=100) # 設定訓練的次數
29. # 測試
30. score = model.evaluate(x_test, y_test, batch_size=128) # 計算正確率
31. print("score:",score) # 輸出
32.
```

```

33. predict = model.predict(x_test)           # 預測
34. print("Ans:",predict)                   # 輸出
35. print("Ans:",np.argmax(predict[0]),np.argmax(predict[1])) # 預測答案
36. predict2 = model.predict_classes(X)     # 取得預測答案
37. print("predict_classes:",predict2)     # 輸出
38. print("y_test",Y)                      # 實際測試的結果

```

### 執行結果

```

Epoch 100/100
2/2 [=====] - 0s 419us/sample - loss: 0.1967 - acc: 1.0000
2/2 [=====] - 0s 10ms/sample - loss: 0.1930 - acc: 1.0000
score: [0.19297751784324646, 1.0]
Ans: 0 1
predict_classes: [0 1]
y_test [0 1]

```



教學影片 3\_image\_circle\_cross\_MLP.py

## 14.4 實戰手寫圖片 MNIST

手寫圖片將透過 `Tensorflow.keras.datasets` 中的 `MNIST` 函式庫來取得大量圖片，並且實戰做出影像辨識。手寫資料來源為 `MNIST` 機構，其官方網站為 <http://yann.lecun.com/exdb/mnist/>，擁有 6 萬筆數字 0~9 的手寫圖片資料，而測試的圖片也有一萬筆。

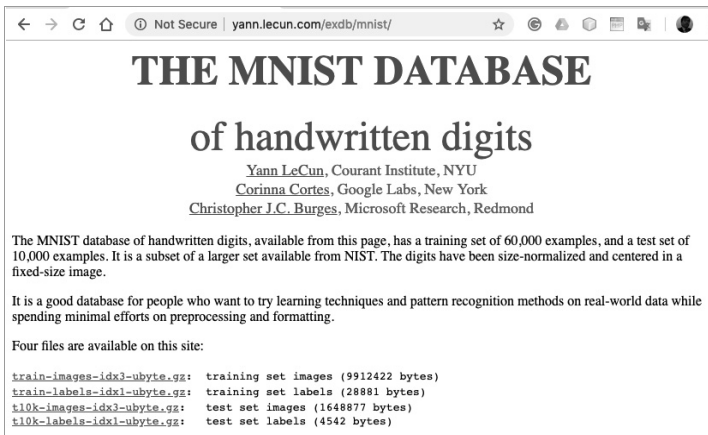


圖 14-4 MNIST 機構

可以透過 `Tensorflow.keras.datasets` 下載和使用 MNIST 手寫資料。

```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

下載的數據集被分成：

- ✓ `x_train`：60000 筆的訓練數據，訓練數據的特徵。
- ✓ `y_train`：60000 筆的訓練數據，訓練數據的標籤答案。
- ✓ `x_test`：10000 筆的測試數據，測試數據的特徵。
- ✓ `y_test`：10000 筆的測試數據，測試數據的標籤答案。

正如前面提到的一樣，每一個 MNIST 數據單元由兩部分組成：一張包含手寫數字的圖片和一個對應的答案標籤。我們把這些訓練的圖片設為「`x_train`」，把這些訓練的標籤設為「`y_train`」。透過以下的程式，瞭解如何將 MNIST 手寫資料下載，並且分開放在不同的變數中。

#### 🔊 範例程式：4\_mnist\_load\_display.py

```
1. import tensorflow as tf # tensorflow 函式庫
2. # 載入資料 (將資料打散，擺入 train 與 test 資料集)
3. (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
4. print('x_train = ' + str(x_train.shape)) # x_train 的陣列大小
5. print('y_train = ' + str(y_train.shape)) # y_train 的陣列大小
```

#### 🔊 執行結果

```
x_train = (60000, 28, 28)
y_train = (60000,)
```

從 MNIST 下載資料 `load_data()`，並放到變數中。此處的 `x_train` 為 60000 筆的訓練數據 X 特徵，而每一筆的大小為(28,28)，意指圖形大小為 28 (寬) x 28 (高)，每一個點只是一個灰階的點，範圍是 0~255 之間，也就是存放每一個點的灰階資料。而測試的部分也是一樣的資料型態，但筆數只有 10000 筆的測試數據。



教學影片 4\_mnist\_load.mp4





在程式的結果有二個重點：

- ✓ 答案 `y_train[0]` 為 5。
- ✓ 通過 `x_train[0]` 的  $28 \times 28$  個點的資料內容。

跟剛剛的圈圈叉叉圖片不一樣的是圖片大小由  $3 \times 3$  換成較大  $28 \times 28$  的點，而顏色資料由 2 種顏色變成 255 種顏色。



教學影片 5\_mnist\_displayNumbers.py

## 14.6 圖形顯示 MNIST 內的資料

為了讓各位更清楚的瞭解資料的內容，同樣使用 `matplotlib.pyplot` 函式將第一筆資料的內容用圖形的方法繪製出來。

### 部分範例程式：6\_mnist\_load\_display.py

```
1. import matplotlib.pyplot as plt          # 圖形顯示的函式庫
2. ...                                     # 同上一個範例，省略
3. num=0                                   # 查看第 num 筆的圖（內容）
4. plt.title('x_train[%d] Label: %d' % (num, y_train[num])) # 第 num 筆和答案
5. plt.imshow(x_train[num], cmap=plt.get_cmap('gray_r')) # 用灰階的方法查看該筆
6. plt.show()                             # 顯示圖片
```

### 執行結果

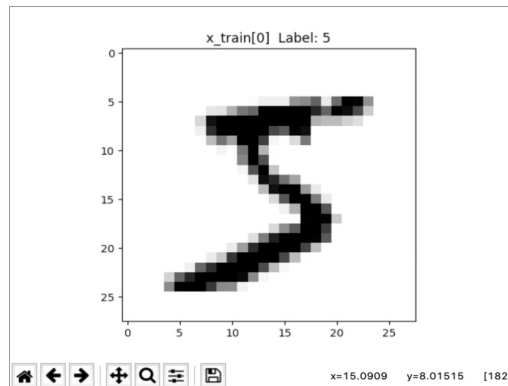


圖 14-6 MNIST 第 0 筆手寫的圖形資料

執行本程式之後，使用滑鼠移動到圖片上面，就可以看到右下角的資料  $x=16$   $y=8$  [182]，顯示該點的位置和其數據資料。請留意一下，這圖形資料因為黑色的部分就靠近 255，邊緣灰色的部分在 100~200 的數字。由此可見 MNIST 的資料不單單是黑色 255 和白色 0 的資料，手寫數字邊緣都是灰階的顏色。



教學影片 6\_mnist\_load\_display.mp4

## 14.7 顯示多張圖片

如果想要多看幾筆的話，可以透過以下程式呈現出前面 16 筆資料。當然，你可以自己稍微調整一下。當看到前面幾筆的時候你會發現，即使是寫出數字 1 的這三次，其圖形的樣子會有些差異，這就是實際上在做資料辨識的時候，為什麼要用這麼多張來訓練的原因，這樣的話就可以把所有的可能性全部都包含在訓練的資料之中。在這個案例中，也會發現訓練的圖就只有單純的數字，不會出現其他的干擾物，例如不同顏色的背景，而且每一張圖就只有要辨識的內容。

### 部分範例程式：7\_mnist\_load\_display\_more.py

```

1. ... # 同上一個範例，省略
2. for num in range(0,16): # 看第 num 筆的圖（內容）
3.     plt.subplot(6,6,num+1) # 畫面切割
4.     plt.title('%d] Label: %d' % (num, y_train[num])) # 第 num 筆和答案
5.     plt.imshow(x_train[num], cmap=plt.get_cmap('gray_r')) # 顯示
6.     plt.show() # 顯示圖片

```

## 執行結果

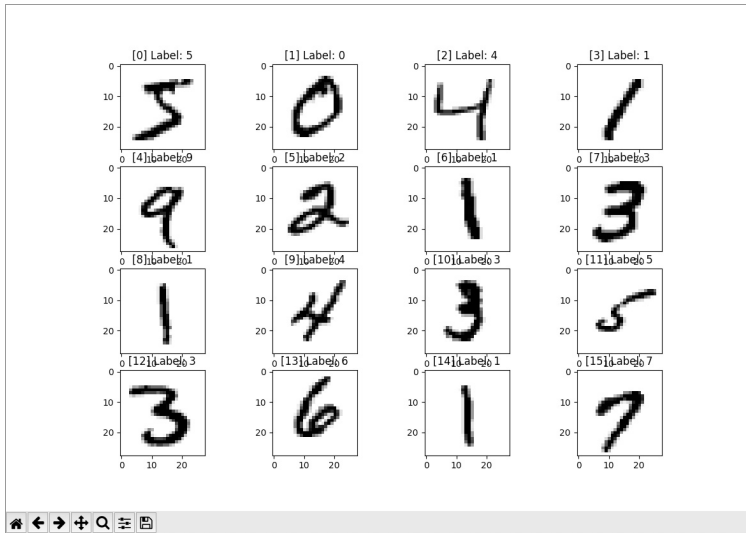


圖 14-7 MNIST 前面 16 筆手寫的圖形資料



教學影片

7\_mnist\_load\_display\_more.mp4

在結果的圖片中會發現很多手寫數字，因為過於潦草，很多連你都無法確定答案，例如圖 14-7 中的 4 和 5。



**TIPS** 統計名言「當採樣資料是錯誤的，所分析出的結果也會是錯的」，以我的經驗，在做人工訓練的時候，其實花最多時間的是在確定訓練資料的正確性。

## 14.8

## 圖形文字的辨識原理

我們曾在之前的章節中介紹過花的辨識，並瞭解如何依照花瓣和花萼之間的關係達到目的，而 MLP 能夠在花的特徵值中，依照答案分類來找出差異性。同樣地，圖形文字也是用一樣的邏輯來進行辨識。

透過以下的範例程式，我們將同一個答案的  $28 \times 28$  的手寫資料圖片，先轉換成 784 個一維資料 ( $28 \times 28 = 784$ )，並把相同的答案以上下平放的方法一起展示在同一張圖片中。你會發現相同的手寫答案，其特徵數據都很類似。

#### 部分範例程式：8\_mnist\_display\_all.py

```

1.  ... # 同一個範例，省略
2.  def display_mult_flat(start, stop, label):
3.      images = x_train[start].reshape([1, 28*28]) # 將圖片變成一維陣列
4.      for i in range(start+1, stop): # 搜尋
5.          label2 = int(y_train[i]) # 取得該筆的答案
6.          if label2 == label: # 判斷是否是我們要找的標籤
7.              images = np.concatenate((images, x_train[i].reshape([1, 28*28]))) # 加上
8.          plt.imshow(images, cmap=plt.get_cmap('gray_r')) # 用灰階的方法繪製
9.          plt.show() # 顯示圖片
10.
11. display_mult_flat(0, 2000, 7) # 在 0-2000 之間，找出答案是 7
12. display_mult_flat(0, 2000, 1) # 在 0-2000 之間，找出答案是 1

```

#### 執行結果

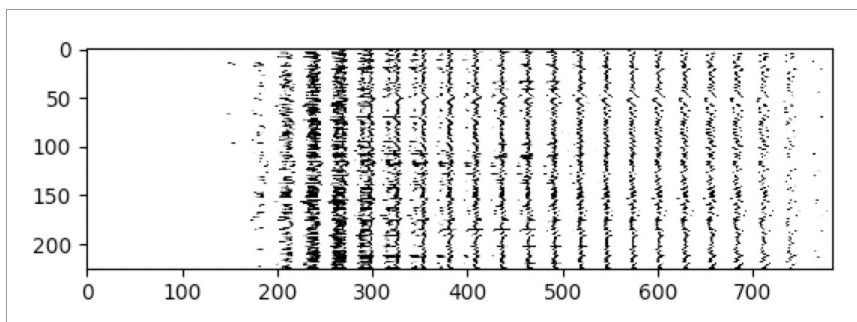


圖 14-8 多筆的手寫 7 呈現的狀態

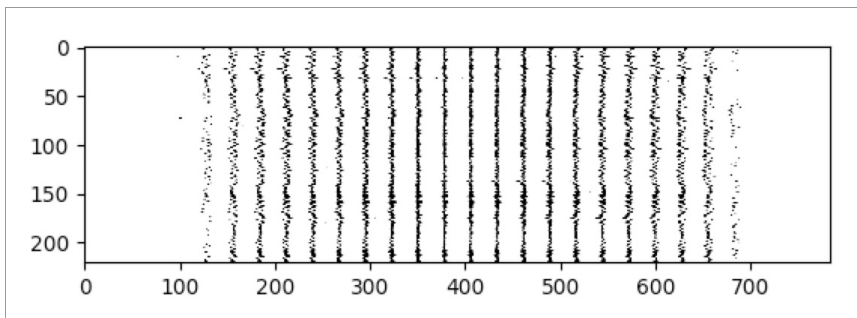



圖 14-9 多筆的手寫 1 呈現的狀態

在結果中你會發現，只要讓各個相同的答案採用此方式展開，就能從相似角度中找出類似的關係，也就能達到圖片或文字辨識目標，這就是以類神經 MLP 方法來辨識圖片的技術原理。

 教學影片 8\_mnist\_display\_all.mp4

## 14.9 圖形資料轉換成 MLP 訓練資料

就如之前 MLP 的分類，MINST 手寫圖片也需要把資料處理一下，計算後的效果會更好。在此會透過三個步驟進行處理：

STEP 1 將圖片轉換成 MLP 的特徵值

STEP 2 特徵值數據標準化

STEP 3 One-hot Encoding 單熱編碼

### 將圖片轉換成 MLP 的特徵值

就如上一節所說，因為要讓類神經 MLP 可以辨識  $28 \times 28$  灰階圖片，所以需要透過 `reshape()` 方法將圖片轉換成 784 個特徵值（ $28 \times 28$  個點）。

#### 部分範例程式：9\_mnist\_onthotending1.py

```
print('x_train before reshape:', x_train.shape)      # 輸出(60000, 28, 28)
dim=img_rows*img_cols*1                             # 784=28x28
x_train = x_train.reshape(x_train.shape[0], dim)     # 改變外型 reshape
x_test = x_test.reshape(x_test.shape[0], dim)
print('x_train after reshape:', x_train.shape)      # 輸出(60000, 784)
```

#### 執行結果

```
x_train before reshape: (60000, 28, 28)
x_train after reshape: (60000, 784)
```

## 🔍 特徵值數據標準化

在類神經的特徵值處理，其實會建議使用「特徵值增強度」這個技巧，它可以有效的增加正確性。什麼是特徵值增強度？就是把原本的數字放到 0 到 1 之間的浮點數，這樣的作法可以在實際應用時避免因為數字不一樣而產生誤差。以測試的圖片範例來說，在圖片判斷時，會因為圖片的明亮度讓資料不一致，而處理 0 到 1 之間的浮點數，就能夠避免這樣的問題產生。而且特徵值數據標準化也可以讓 **Learning rate** 學習效率有更好的表現。使用上，只要在這資料中找出最大值和最小值，並將其分別當成 0 和 1 即可。因為灰階的圖片範圍只有 0~255，所以在此整除 255 就能達成「特徵值增強度」。

### 🔊 部分範例程式：10\_mnist\_onthotending2.py

```
print('x_train before div 255:',x_train[0][180:195]) # 處理前的部分資料
# 標準化匯入資料
x_train = x_train.astype('float32') # 轉換為浮點數
x_test = x_test.astype('float32')
x_train /= 255 # 特徵值增強度
x_test /= 255
print('x_train before div 255 ', x_train[0][180:195]) # 處理後的部分資料
```

### 🔊 執行結果

```
x_train before div 255: [170 253 253 253 253 253 225 172 253 242 195
64 0 0 0 ]
x_train before div 255: [0.666 0.992 0.992 0.992 0.992 0.992 0.882
0.674 0.992 0.949 0.764 0.250 0. 0. 0.]
```

## 🔍 One-hot Encoding 單熱編碼

接下來處理訓練用的答案標籤資料的形狀：

---

```
print('y_train shape:', y_train.shape) # 輸出 (60000, )
```

---

我們來看看前 10 個訓練樣本的標籤答案：

---

```
print(y_train[:10]). # 輸出 [5 0 4 1 9 2 1 3 1 4]
```

---

接著轉換為 One-hot Encoding 單熱編碼，我們可以透過以下的函式來達成：

```
tf.keras.utils.to_categorical(y, num_classes=None)
```

此函式的用意是將一個類向量（整數）轉換為二進位類矩陣。

#### 部分範例程式：11\_mnist\_onthotending3.py

```
category=10
y_train2 = tf.keras.utils.to_categorical(y_train, category) # 轉換為單熱編碼
y_test2 = tf.keras.utils.to_categorical(y_test, category)
print("y_train2 to_categorical shape=",y_train2.shape) # 輸出 (60000, 10)
print(y_train2[:10]) # 輸出前 10 筆
```

#### 執行結果

```
y_train shape: (60000,)
[5 0 4 1 9 2 1 3 1 4]
y_train2 to_categorical shape= (60000, 10)
[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```



教學影片

9\_mnist\_onthotending.mp4

## 14.10

## 使用 MLP 來辨識圖片和文字

在之前章節中，鳶尾花的範例只有 4 個特徵值，而 MNIST 資料集的圖片是在 784 個特徵值（ $28 \times 28$  個點）中找出相同的類似資料，技術是一樣的，只是特徵值由 4 個變成 784 個。

而我們也已透過圈圈又叉的範例瞭解圖形文字的辨識原理，即剛剛提到圈圈的大小是  $3 \times 3$  也就是 9 個特徵值 2 種答案，而手寫 MNIST 的圖則是  $28 \times 28$  也就是 9 個特徵值 10 種答案，其他顏色的範圍是 0~1 手寫是 0~255 其他都相同。

使用之前所學習到的多層類神經 MLP 模型來完成圖形辨識。繼續剛剛的程式，首先宣告一個宣告順序層（sequential model format）：

---

```
model = tf.keras.models.Sequential() # 宣告順序層
```

---

接下來，定義「匯入層」加入模型，匯入層 `dim=784` 是匯入資料大小。

---

```
model.add(tf.keras.layers.Dense( # 定義匯出「隱藏層」的神經元數量
    units=10, # 加入 10 個神經元
    activation=tf.nn.relu, # 定義激勵函式為 relu
    input_dim=dim)) # 匯入層訓練特徵值 784 個
```

---

中間的「隱藏層」

---

```
model.add(tf.keras.layers.Dense(units=10, # 加入 10 個神經元
    activation=tf.nn.relu)) # 定義激勵函式為 relu
kernel_initializer='normal', # 使用 normal distribution 常態分佈的亂數定義
```

---

「匯出層」加入模型

---

```
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001), # 使用 Adam 移動 0.001
    loss=tf.keras.losses.categorical_crossentropy, # 損失率使用稀疏分類交叉熵
    metrics=['accuracy']) # 模型在培訓和測試期間要評估的度量串列
```

---

請注意，第一層的匯入 `input_dim=784` 為特徵值  $784 = 28 \times 28 \times 1$ ，而最後一層的匯出大小為 10 個神經元，對應於 10 種答案。

可以透過以下程式檢查整個模型系統架構的外觀：

---

```
model.summary()
```

---



## 執行結果

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	7850
dense_1 (Dense)	(None, 10)	110
dense_2 (Dense)	(None, 10)	110
Total params: 8,070		
Trainable params: 8,070		
Non-trainable params: 0		

圖 14-10 model.summary()的匯出

由此可以看出對應程式一共有三層。另外，Param 越大通常代表模型更複雜，需要更多時間計算。Param 參數的計算方法如下：

$$✓ 7850 = (784 \times 10) + 10$$

$$✓ 110 = (10 \times 10) + 10$$

最後的答案可以透過相同的訓練模型並使用測試資料來驗證預測的結果和正確率。

## 部分範例程式：12\_mnist\_MLP.py

```

1. ... # 省略
2. # 訓練模型
3. history=model.fit(x_train, y_train2, # 進行訓練的因和果資料
4.     batch_size=1000, # 設定每次訓練的筆數
5.     epochs=200, # 設定訓練的次數，也就是機器學習的次數
6.     verbose=1) # 訓練時顯示的訊息
7.
8. # 測試
9. score = model.evaluate(x_test, y_test2, batch_size=128) # 計算測試正確率
10. print("score:",score) # 輸出測試正確率
11. predict = model.predict(x_test) # 取得每一個結果的機率
12. print("Ans:",np.argmax(predict[0]),np.argmax(predict[1]),
13.     np.argmax(predict[2]),np.argmax(predict[3])) # 取得預測答案 1
14. predict2 = model.predict_classes(x_test[:10] ) # 取得預測答案 2
15. print("predict_classes:",predict2[:10] ) # 輸出預測答案 2
16. print("y_test",y_test[:]) # 實際測試的結果

```

### 🔊 執行結果

```
10000/1000 [=====] - 0s 17us/sampe - loss: 0.2274 - acc: 0.9390
score: [0.22742516186237335, 0.939]
Ans: 7 2 1 0
predict_classes: [7 2 1 0 4 1 4 4 6 9]
y_test [7 2 1 0 4 1 4 9 5 9]
```



教學影片 10\_mnist\_MLP.mp4

## 14.11 實戰－服飾的圖形辨識 Fashion-MNIST

在 `Tensorflow.keras.datasets` 的數據庫中還有一個和 MNIST 數據庫很類似的圖片數據 Fashion-MNIST，它是 Zalando 公司的圖像數據集，其中有包含 60,000 張訓練圖片和 10,000 張訓練圖片。每個圖片都是一個  $28 \times 28$  灰階圖像，其數字範圍是 0 到 255，而圖片分成 10 個類別的標籤答案，分別是：

- 0 T-shirt / top：T 卹/上衣
- 1 Trouser：褲子
- 2 Pullover：套頭衫
- 3 Dress：禮服
- 4 Coat：外套
- 5 Sandal：涼鞋
- 6 Shirt：襯衫
- 7 Sneaker：運動鞋
- 8 Bag：包包袋子
- 9 Ankle boot：長靴

接下來，將透過 `Tensorflow.keras.datasets` 的數據庫來取得大量圖片，並且實戰做出影像辨識。手寫資料來源為 MNIST 機構，其官方網站為 <https://github.com/zalando-research/fashion-mnist>。

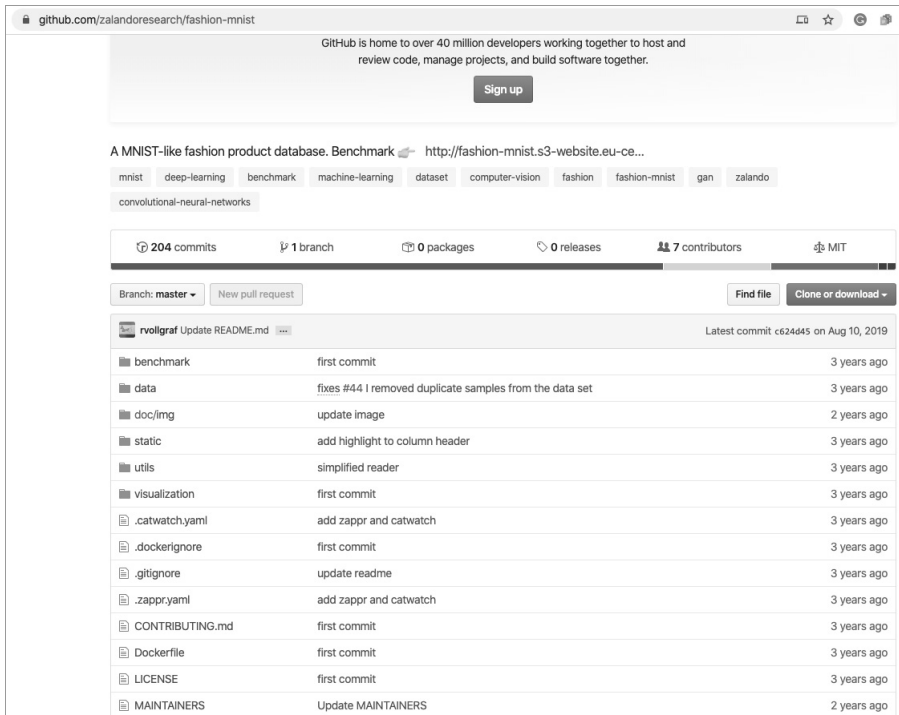


圖 14-11 Fashion-MNIST 資料來源

透過 `Tensorflow.keras.datasets` 下載和使用 Fashion-MNIST 手寫資料。

```
import tensorflow as tf
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
```

下載的數據集被分成：

- ✓ `x_train`：60000 筆的訓練數據，訓練數據的特徵。
- ✓ `y_train`：60000 筆的訓練數據，訓練數據的標籤答案。
- ✓ `x_test`：10000 筆的測試數據，測試數據的特徵。
- ✓ `y_test`：10000 筆的測試數據，測試數據的標籤答案。

#### 🔊 範例程式：13\_fashion\_mnist\_load\_display.py

1. `import tensorflow as tf` # tensorflow 函式庫
2. # 載入服飾資料（將資料打散，擺入 `train` 與 `test` 資料集）

```

3. (x_train, y_train), (x_test, y_test) = tf.keras.datasets.fashion_mnist.load_data()
4. print('x_train = ' + str(x_train.shape)) # x_train 的陣列大小
5. print('y_train = ' + str(y_train.shape)) # y_train 的陣列大小
6. print('x_test = ' + str(x_test.shape)) # x_test 的陣列大小
7. print('y_test = ' + str(y_test.shape)) # y_test 的陣列大小

```

### 🔊 執行結果

```

x_train = (60000, 28, 28)
y_train = (60000,)
x_test = (10000, 28, 28)
y_test = (10000,)

```

由服飾資料 Fashion\_MNIST 下載資料 `load_data()`，並放到變數中。此處的 `x_train` 為 60000 筆的訓練數據 X 特徵，而每一筆的大小為 (28,28)，意指圖形大小為 28 (寬) × 28 (高)，每一個點只是一個灰階的點，範圍是 0~255 之間，也就是存放每一個點的灰階資料。而測試的部分也是一樣的資料型態，但筆數只有 10000 筆的測試數據。服飾資料 `fashion_mnist` 和手寫資料 `mnist` 幾乎一樣。

### 🔍 服飾資料 fashion\_mnist 每一筆的 Image 資料內容

同樣地，將透過以下的程式來查看服飾資料 `fashion_mnist` 的第 0 筆資料，直接印出每一個點的灰階數字。

### 🔊 部分範例程式：14\_fashion\_mnist\_displayNumer.py

```

1. ... # 同上一個範例，省略
2. def printMatrixE(a): # 查看單一圖片的每一點資料
3.     rows = a.shape[0] # 取得圖片高度
4.     cols = a.shape[1] # 取得圖片寬度
5.     for i in range(0,rows): # 處理每一列的每一點資料
6.         str1=""
7.         for j in range(0,cols): # 處理同一行的每點資料
8.             str1=str1+"%.30f " % a[i, j]) # 取得單點資料
9.         print(str1) # 輸出整排的資料
10.    print("")
11.
12. printMatrixE(x_train[0]) # 查看第 0 筆的圖 (內容)
13. print('y_train[0] = ' + str(y_train[0])) # 查看第 0 筆的答案

```

## 執行結果

```

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 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 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 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 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 0 0 0 0 0 0 0 0 0 1 0 0 13 73 0 0 1 4 0 0 0 0 0 0 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 36 136 127 62 54 0 0 0 1 3 4 0 0 0 3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 102 204 176 134 144 123 23 0 0 0 0 0 12 10 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 155 236 207 178 107 156 161 109 64 23 77 130 72 15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 69 207 223 218 216 216 163 127 121 122 146 141 88 172 66
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 200 232 232 233 229 223 223 215 213 164 127 123 196 229 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 183 225 216 223 228 235 227 224 222 224 221 223 245 173 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 193 228 218 213 198 180 212 210 211 213 223 220 243 202 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 3 0 12 219 220 212 218 192 169 227 208 218 224 212 226 197 209 52
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 6 0 99 244 222 220 218 203 198 221 215 213 222 220 245 119 167 56
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 55 236 228 230 228 240 232 213 218 223 234 217 217 209 92 0
0 0 0 1 4 6 7 2 0 0 0 0 0 0 0 0 0 0 0 0 0 237 226 217 223 222 219 222 221 216 223 229 215 218 255 77 0
0 3 0 0 0 0 0 0 0 0 0 62 145 204 228 207 213 221 218 208 211 218 224 223 219 215 224 244 159 0
0 0 0 0 18 44 82 107 189 228 220 222 217 226 200 205 211 230 224 234 176 188 250 248 233 238 215 0
0 57 187 208 224 221 224 208 204 214 208 209 200 159 245 193 206 223 255 255 221 234 221 211 220 232 246 0
3 202 228 224 221 211 214 205 205 205 220 240 80 150 255 229 221 188 154 191 210 204 209 222 228 225 0
98 233 198 210 222 229 229 234 249 220 194 215 217 241 65 73 106 117 168 219 221 215 217 223 223 224 229 29
75 204 212 204 193 205 211 225 216 185 197 206 198 213 240 195 227 245 239 223 218 212 209 222 220 221 230 67
48 203 183 194 213 197 185 190 194 192 202 214 219 221 220 236 225 216 199 206 186 181 177 172 181 205 206 115
0 122 219 193 179 171 183 196 204 210 213 207 211 210 200 196 194 191 195 191 198 192 176 156 167 177 210 92
0 0 74 189 212 191 175 172 175 181 185 188 189 188 193 198 204 209 210 210 211 188 188 194 192 216 170 0
2 0 0 0 66 200 222 237 239 242 246 243 244 221 220 193 191 179 182 182 181 176 166 168 99 58 0 0
0 0 0 0 0 0 0 0 40 61 44 72 41 35 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 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 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 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 0 0 0 0 0 0 0

```

`y_train[0] = 9`

圖 14-12 Fashion\_MNIST 第 0 筆的資料內容和答案為 9



教學影片

11\_fashion\_mnist\_load.mp4

## 14.12

## 圖形化顯示 Fashion\_MNIST 服飾資料

為了讓各位更清楚的瞭解資料的內容，同樣使用 `matplotlib.pyplot` 函式將 Fashion\_MNIST 服飾資料的內容用圖形的方法繪製出來。透過以下程式，呈現出前面 36 筆的資料。

### 部分範例程式：15\_fashion\_mnist\_display\_images.py

```

1. ... # 同一個範例，省略
2. for num in range(0,36): # 看前 36 筆的圖
3.     plt.subplot(6,6,num+1) # 畫面切割
4.     plt.title('%d Label: %d' % (num, y_train[num])) # 第 num 筆和答案
5.     plt.imshow(x_train[num], cmap=plt.get_cmap('gray_r')) # 顯示
6.     plt.show() # 顯示圖片

```

### 執行結果



圖 14-13 Fashion\_MNIST 服飾資料前面 36 筆的圖形

執行本程式之後，將滑鼠移動到圖片上面，就可以看到右下角的資料  $x=16$   $y=8$  [182]，會顯示該點的位置和其數據資料。請留意一下，這圖形資料因為黑色的部分就靠近 255，然後文間的邊緣有很多是 1 數字在 100~200 灰階顏色。



教學影片 12\_fashion\_mnist\_display\_images.mp4