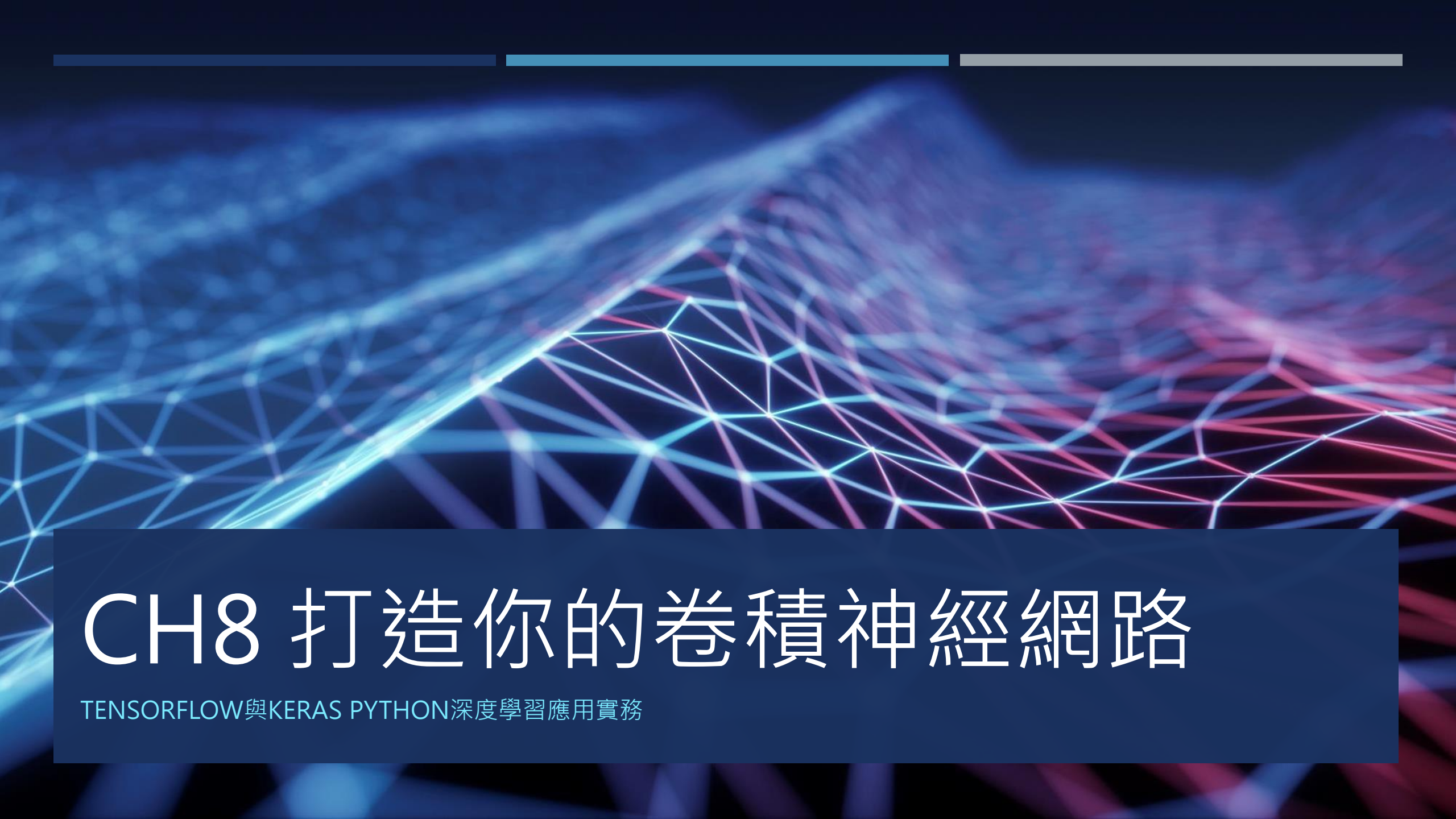


CH8 打造你的卷積神經網路

TENSORFLOW與KERAS PYTHON深度學習應用實務

8-1 認識MNIST手寫辨識資料集

MNIST

- 一種圖片資料庫，擁有6000張手寫數字圖片的訓練資料以及10000張測試資料
- MNIST 資料集是成對的手寫圖片和對應的標籤資料

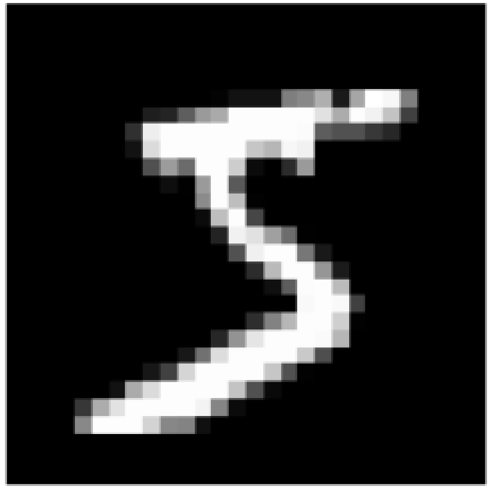
8-1 認識MNIST手寫辨識資料集 — 載入和探索MNIST手寫辨識資料集

```
from  tensorflow.keras.datasets  import  mnist

#  載入  MNIST  資料集,  如果是第一次載入會自行下載資料集
(X_train,  Y_train),  (X_test,  Y_test)  =  mnist.load_data()
#  顯示  Numpy  二維陣列內容
print(X_train[0])
print(Y_train[0])      #  標籤資料

import  matplotlib.pyplot  as  plt

plt.imshow(X_train[0],  cmap="gray")
plt.title("Label:  "  +  str(Y_train[0]))
plt.axis("off")
#  顯示數字圖片
plt.show()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

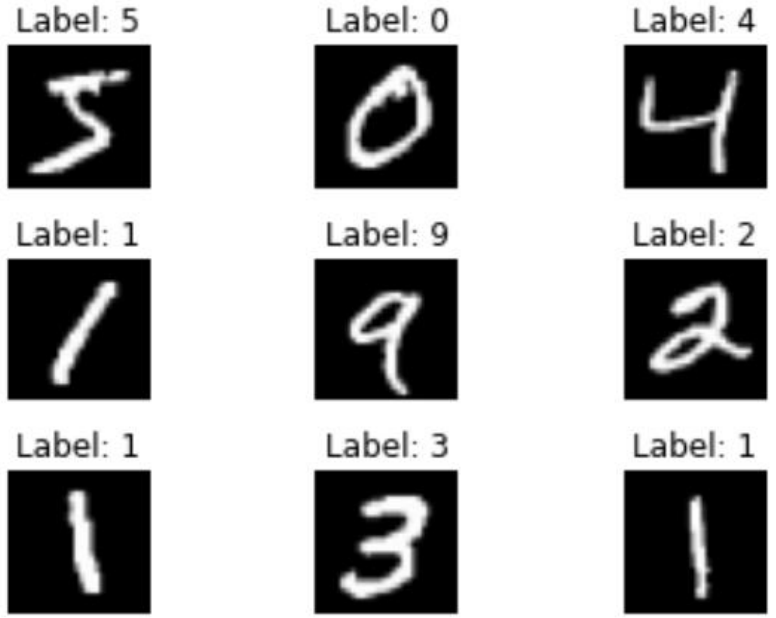
8-1 認識MNIST手寫辨識資料集

```
[[  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   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   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  35 241 225 160 108   1   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   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]]
```



8-1 認識MNIST手寫辨識資料集 — 顯示MNIST資料集的前9張圖片

```
from keras.datasets import mnist
import matplotlib.pyplot as plt

# 載入 MNIST 資料集, 如果需要, 會自行下載
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
# 繪出9個數字圖片
sub_plot= 330
for i in range(0, 9):
        ax = plt.subplot(sub_plot+i+1)
        ax.imshow(X_train[i], cmap="gray")
        ax.set_title("Label: " + str(Y_train[i]))
        ax.axis("off")

plt.subplots_adjust(hspace = .5)
# 顯示數字圖片
plt.show()
```



8-2 使用MLP打造MNIST手寫辨識

MNIST 手寫辨識

- 一種多元分類，可以將手寫數字圖片分類成10類。
- 用MLP打造MNIST手寫辨識，學習的是整張圖片的所有像素，也就是全域樣式。

8-2-1 MLP的資料預處理

將特徵資料(樣本, 28,28)形狀轉成(樣本,784)形狀

```
import  numpy  as  np
from  keras.datasets  import  mnist

#  指定亂數種子
seed  =  7
np.random.seed(seed)
#  載入資料集
(X_train,  Y_train),  (X_test,  Y_test)  =  mnist.load_data()
#  將  28*28  圖片轉換成  784  的向量
X_train  =  X_train.reshape(X_train.shape[0],  28*28).astype("float32")
X_test  =  X_test.reshape(X_test.shape[0],  28*28).astype("float32")
print("X_train  Shape:  ",  X_train.shape)
print("X_test  Shape:  ",  X_test.shape)
```

(樣本數,特徵數)

```
X_train Shape:  (60000, 784)
X_test Shape:  (10000, 784)
```

8-2-1 MLP的資料預處理

執行特徵標準化的正規化
(灰階圖片正規化)

```
import  numpy  as  np

#  因為是固定範圍,  所以執行正規化,  從  0-255  至  0-1
X_train  =  X_train  /  255
X_test  =  X_test  /  255
print(X_train[0][150:175])
```

```
[0.         0.         0.01176471 0.07058824 0.07058824 0.07058824
 0.49411765 0.53333336 0.6862745  0.10196079 0.6509804  1.
 0.96862745 0.49803922 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.        ]
```

8-2-1 MLP的資料預處理

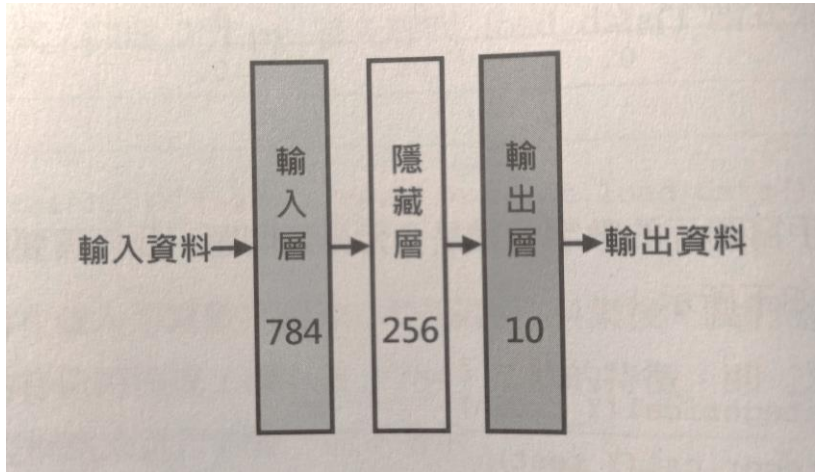
將標籤資料執行One-hot編碼
(因多元分類問題)

```
import  numpy  as  np
from  keras.datasets  import  mnist
from  tensorflow.keras.utils  import  to_categorical

#  One-hot編碼
Y_train  =  to_categorical(Y_train)
Y_test  =  to_categorical(Y_test)
print("Y_train  Shape:  ",  Y_train.shape)
print(Y_train[0])
```

```
Y_train Shape:  (60000, 10)
[0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
```

8-2-2 使用MLP打造MNIST手寫辨識

定義模型




```
import  numpy  as  np
from  tensorflow.keras.datasets  import  mnist
from  tensorflow.keras.models  import  Sequential
from  tensorflow.keras.layers  import  Dense
from  tensorflow.keras.utils  import  to_categorical


#  定義模型
model  =  Sequential()                                          啟動函數
model.add(Dense(256,  input_dim=28*28,  activation="relu"))
model.add(Dense(10,  activation="softmax"))
model.summary()       #  顯示模型摘要資訊
```

```
Model: "sequential_21"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_42 (Dense)             (None, 256)               200960

 dense_43 (Dense)             (None, 10)                2570

=================================================================
Total params: 203,530
Trainable params: 203,530
Non-trainable params: 0
_________________________________________________________________
```

隱藏層

輸出層

8-2-2 使用MLP打造MNIST手寫辨識

編譯模型 & 訓練模型

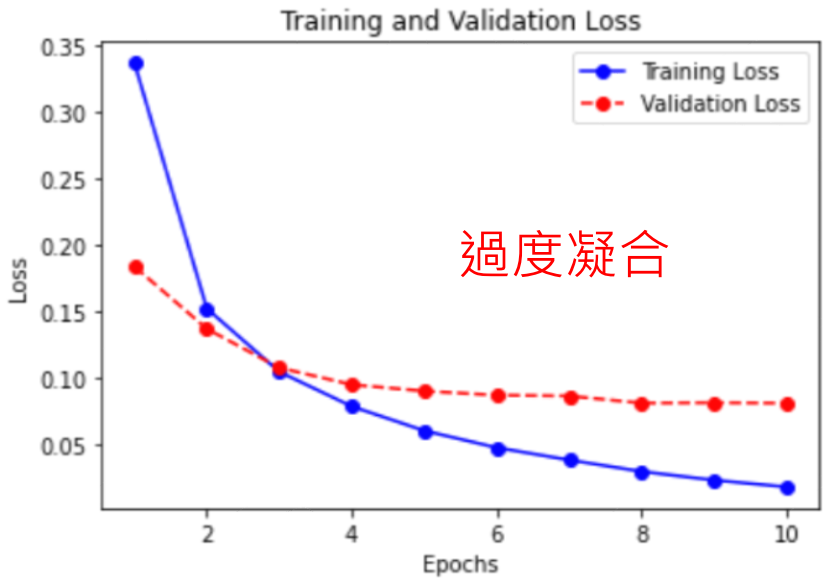
```
Epoch 1/10
375/375 - 4s - loss: 0.3397 - accuracy: 0.9049 - val_loss: 0.1839 - val_accuracy: 0.9506 - 4s/epoch - 11ms/step
Epoch 2/10
375/375 - 3s - loss: 0.1520 - accuracy: 0.9563 - val_loss: 0.1291 - val_accuracy: 0.9617 - 3s/epoch - 7ms/step
Epoch 3/10
375/375 - 2s - loss: 0.1041 - accuracy: 0.9701 - val_loss: 0.1103 - val_accuracy: 0.9682 - 2s/epoch - 7ms/step
Epoch 4/10
375/375 - 3s - loss: 0.0774 - accuracy: 0.9779 - val_loss: 0.0945 - val_accuracy: 0.9722 - 3s/epoch - 8ms/step
Epoch 5/10
375/375 - 2s - loss: 0.0599 - accuracy: 0.9832 - val_loss: 0.0853 - val_accuracy: 0.9737 - 2s/epoch - 4ms/step
Epoch 6/10
375/375 - 2s - loss: 0.0470 - accuracy: 0.9863 - val_loss: 0.0881 - val_accuracy: 0.9728 - 2s/epoch - 4ms/step
Epoch 7/10
375/375 - 2s - loss: 0.0378 - accuracy: 0.9895 - val_loss: 0.0823 - val_accuracy: 0.9741 - 2s/epoch - 4ms/step
Epoch 8/10
375/375 - 2s - loss: 0.0305 - accuracy: 0.9924 - val_loss: 0.0835 - val_accuracy: 0.9753 - 2s/epoch - 4ms/step
Epoch 9/10
375/375 - 2s - loss: 0.0235 - accuracy: 0.9942 - val_loss: 0.0773 - val_accuracy: 0.9772 - 2s/epoch - 4ms/step
Epoch 10/10
375/375 - 2s - loss: 0.0192 - accuracy: 0.9956 - val_loss: 0.0782 - val_accuracy: 0.9778 - 2s/epoch - 4ms/step
```

```
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
傳換成低階TensorFlow計算圖
# 編譯模型                    (損失函數,優化器,評估標準)
model.compile(loss="categorical_crossentropy", optimizer="adam",
                          metrics=["accuracy"])
# 訓練模型 送入特徵資料訓練模型
history = model.fit(X_train, Y_train, validation_split=0.2,
                                  epochs=10, batch_size=128, verbose=2)
```

(訓練資料集,標籤資料集,分割驗證資料,訓練週期,批次尺寸)

8-2-2 使用MLP打造MNIST手寫辨識

```
Testing ...
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0287 - accuracy: 0.9926
訓練資料集的準確度 = 0.99
```

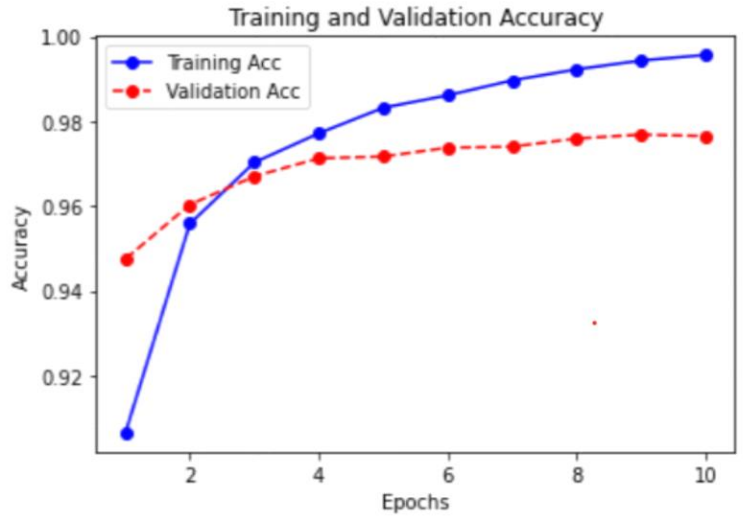


評估模型 — 訓練資料集的準確度

```python
# 評估模型　使用測試資料評估模型的效能
print("\nTesting ...")
loss, accuracy = model.evaluate(X_train, Y_train)
print("訓練資料集的準確度 = {:.2f}".format(accuracy))

# 顯示圖表來分析模型的訓練過程
import matplotlib.pyplot as plt
# 顯示訓練和驗證損失
loss = history.history["loss"]
epochs = range(1, len(loss)+1)
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "bo-", label="Training Loss")
plt.plot(epochs, val_loss, "ro--", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

8-2-2 使用MLP打造MNIST手寫辨識

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0681 - accuracy: 0.9800
測試資料集的準確度 = 0.98
```



評估模型 — 測試資料集的準確度

```python
#  評估模型
loss,  accuracy  =  model.evaluate(X_test,  Y_test)
print("測試資料集的準確度  =  {:.2f}".format(accuracy))
#  顯示訓練和驗證準確度
acc  =  history.history["accuracy"]
epochs  =  range(1,  len(acc)+1)
val_acc  =  history.history["val_accuracy"]
plt.plot(epochs,  acc,  "bo-",  label="Training  Acc")
plt.plot(epochs,  val_acc,  "ro--",  label="Validation  Acc")
plt.title("Training  and  Validation  Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

P.8-11

8-2-3 增加隱藏層的神經元數

定義模型

```
#  定義模型
model  =  Sequential()
model.add(Dense(784,   input_dim=784,   activation="relu"))
model.add(Dense(10,   activation="softmax"))
model.summary()       #  顯示模型摘要資訊
```

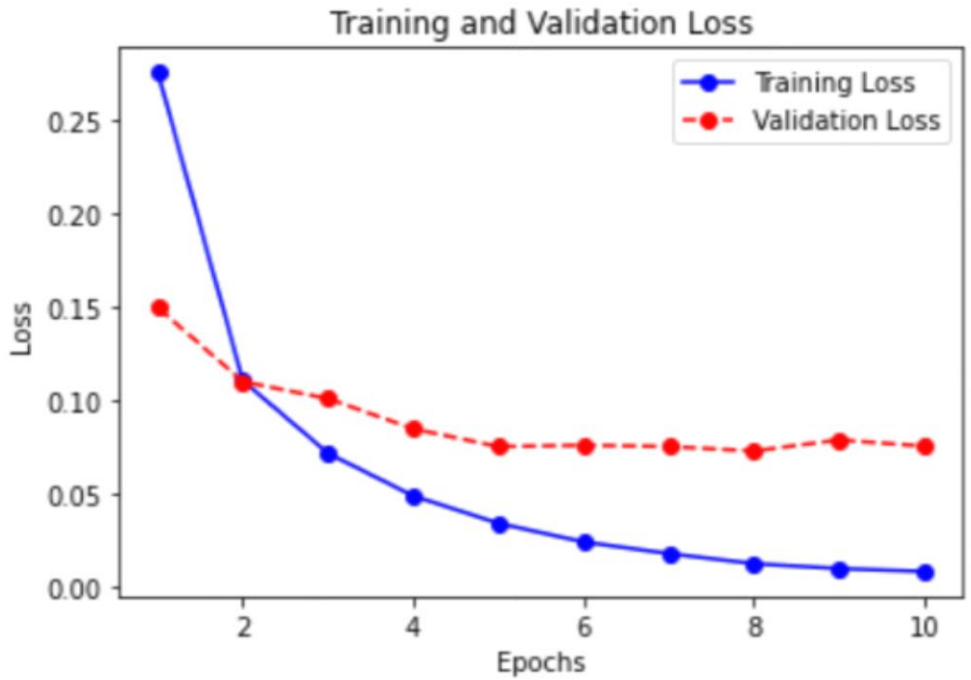
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_2 (Dense)              (None, 784)               615440

 dense_3 (Dense)              (None, 10)                7850

=================================================================
Total params: 623,290
Trainable params: 623,290
Non-trainable params: 0
_________________________________________________________________
```

隱藏層

輸出層

8-2-3 增加隱藏層的神經元數

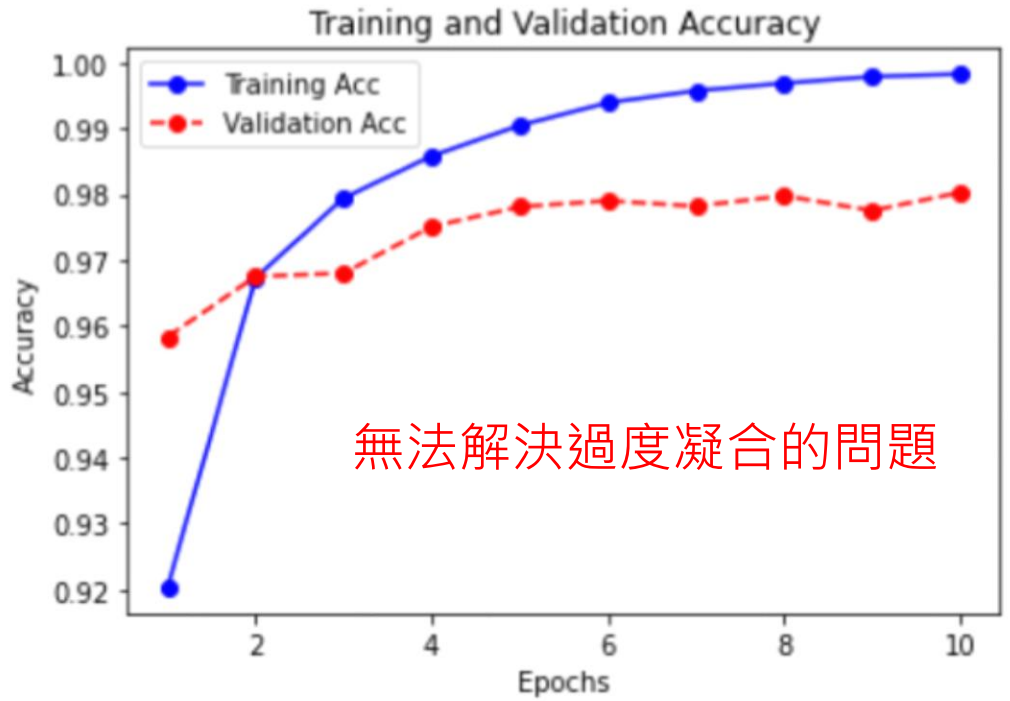
```
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0195 - accuracy: 0.9953
訓練資料集的準確度 = 1.00
```



評估模型 — 訓練資料集的準確度

8-2-3 增加隱藏層的神經元數

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0663 - accuracy: 0.9807
測試資料集的準確度 = 0.98
```



無法解決過度凝合的問題

評估模型 — 測試資料集的準確度

8-2-4 在MLP新增一層隱藏層

定義模型

```
#  定義模型
model  =  Sequential()
model.add(Dense(256,   activation="relu"))
model.add(Dense(10,   activation="softmax"))
model.summary()       #  顯示模型摘要資訊
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_15 (Dense)            (None, 256)               200960
 dense_16 (Dense)            (None, 256)               65792
 dense_17 (Dense)            (None, 10)                2570
=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

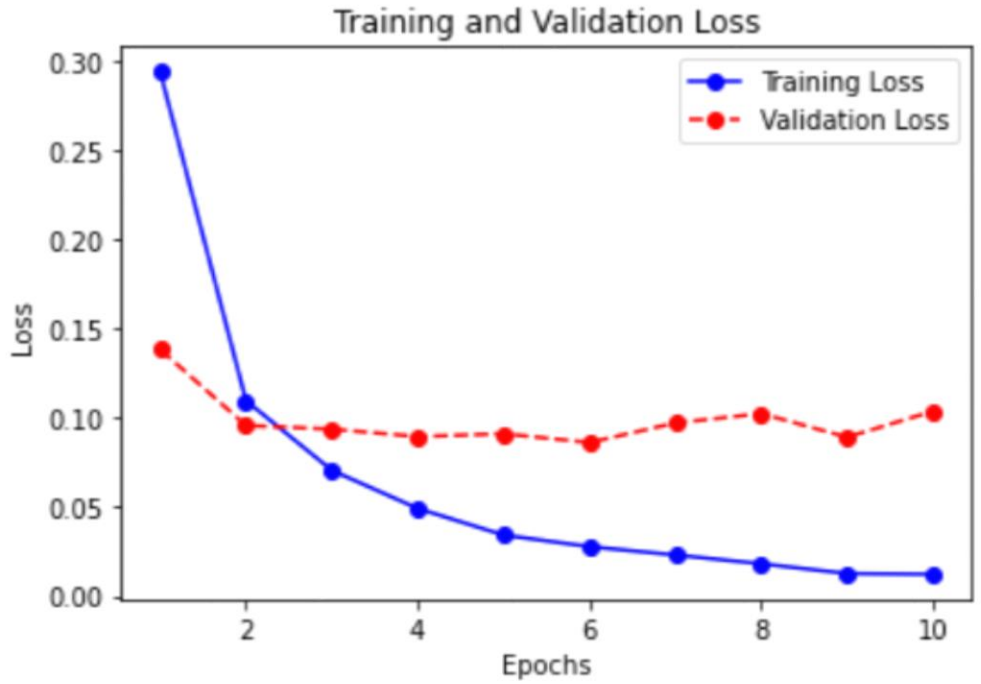
隱藏層

隱藏層

輸出層

8-2-4 在MLP新增一層隱藏層

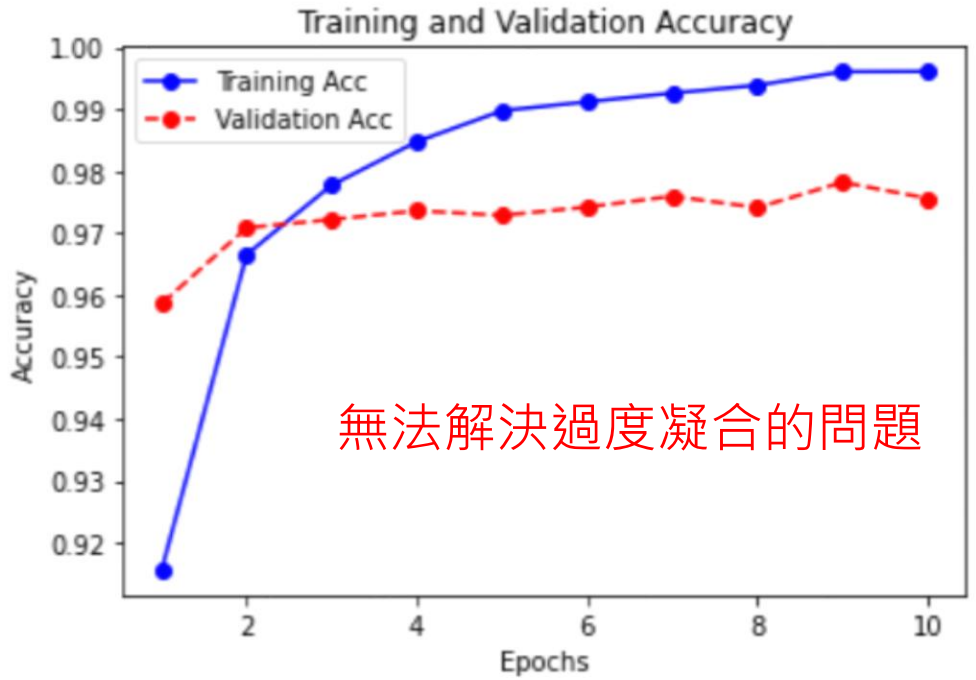
```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0276 - accuracy: 0.9930
訓練資料集的準確度 = 0.99
```



評估模型 — 訓練資料集的準確度

8-2-4 在MLP新增一層隱藏層

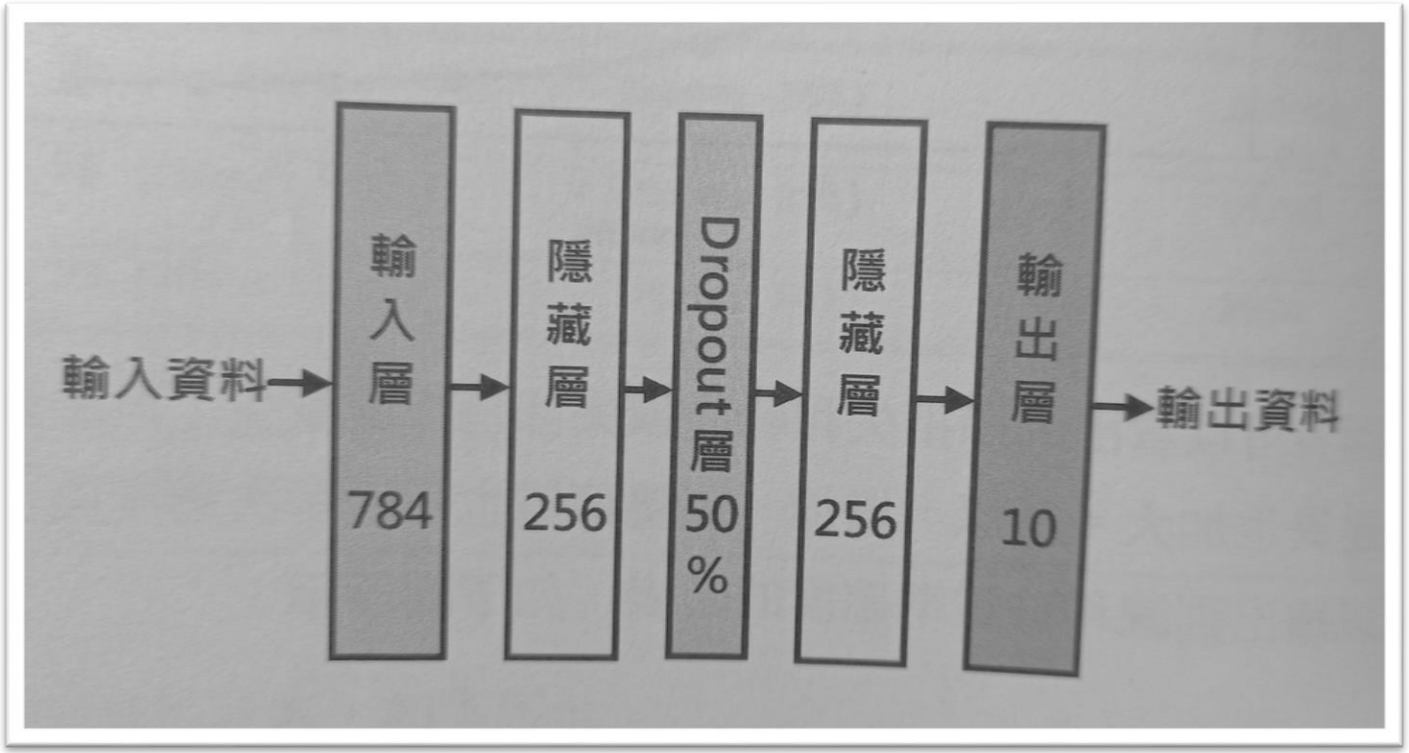
```
313/313 [==============================] - 1s 2ms/step - loss: 0.0771 - accuracy: 0.9796
測試資料集的準確度 = 0.98
```



無法解決過度凝合的問題

評估模型 — 測試資料集的準確度

8-2-5 在MLP使用DROPOUT層



Dropout層可以在不斷增加訓練資料的情況下，幫助我們對抗過度凝合。

8-2-5 在MLP使用DROPOUT層

定義模型

```
import  numpy  as  np
from  keras.datasets  import  mnist
from  keras.models  import  Sequential
from  keras.layers  import  Dense
from  keras.layers  import  Dropout
from  tensorflow.keras.utils  import  to_categorical

#  定義模型
model  =  Sequential()
model.add(Dense(256,  input_dim=28*28,  activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(256,  activation="relu"))
model.add(Dense(10,  activation="softmax"))
model.summary()       #  顯示模型摘要資訊
```

8-2-5 在MLP使用DROPOUT層

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_21 (Dense)             (None, 256)               200960

 dropout (Dropout)            (None, 256)               0

 dense_22 (Dense)             (None, 256)               65792

 dense_23 (Dense)             (None, 10)                2570

=================================================================
Total params: 269,322
Trainable params: 269,322
Non-trainable params: 0
_________________________________________________________________
```

隱藏層1

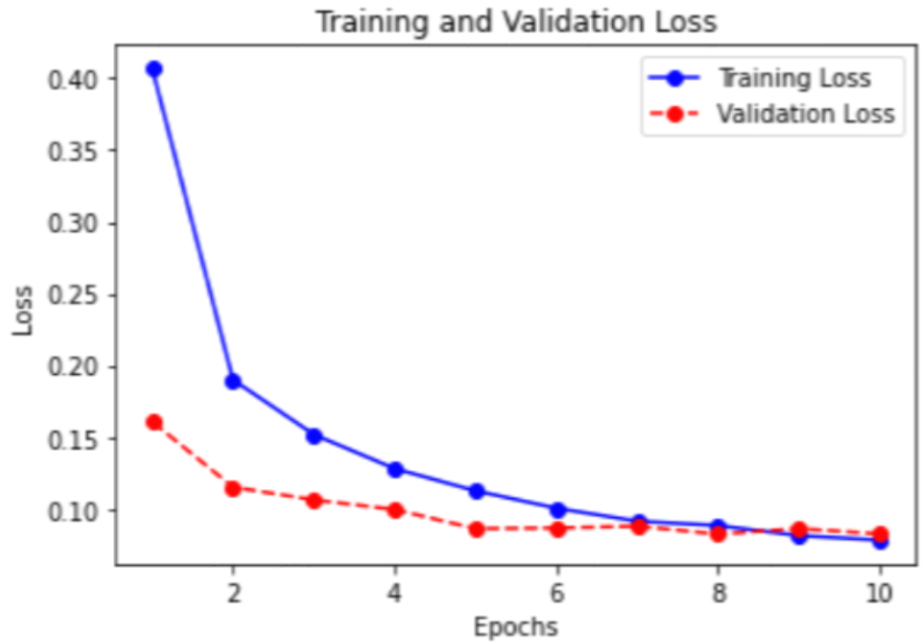
Dropout層(沒有參數)

隱藏層2

輸出層

8-2-5 在MLP使用DROPOUT層

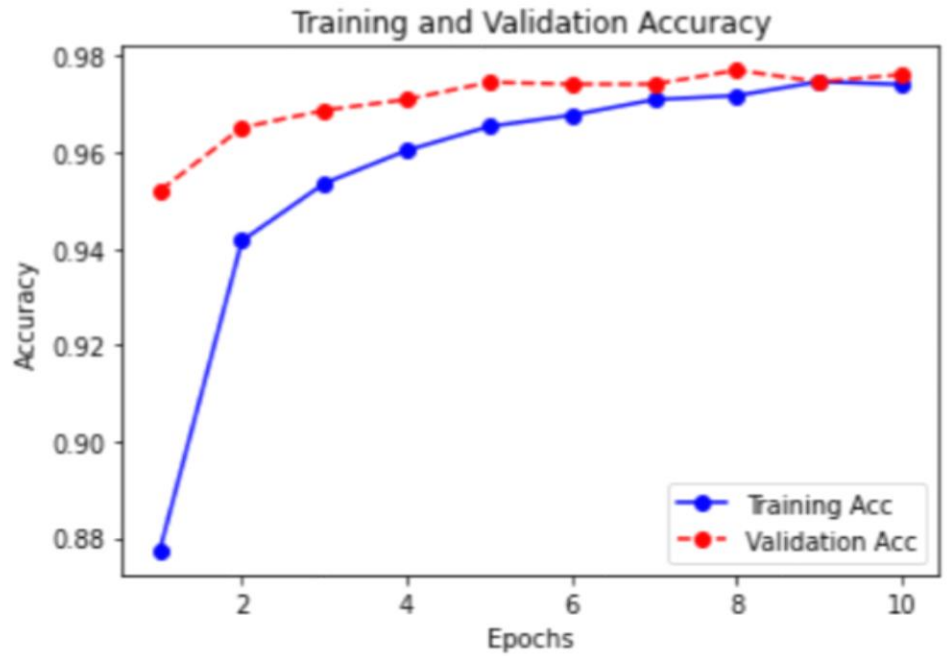
```
Testing ...
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0378 - accuracy: 0.9884
訓練資料集的準確度 = 0.99
```



評估模型 — 訓練資料集的準確度

8-2-5 在MLP使用DROPOUT層

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0743 - accuracy: 0.9788
測試資料集的準確度 = 0.98
```

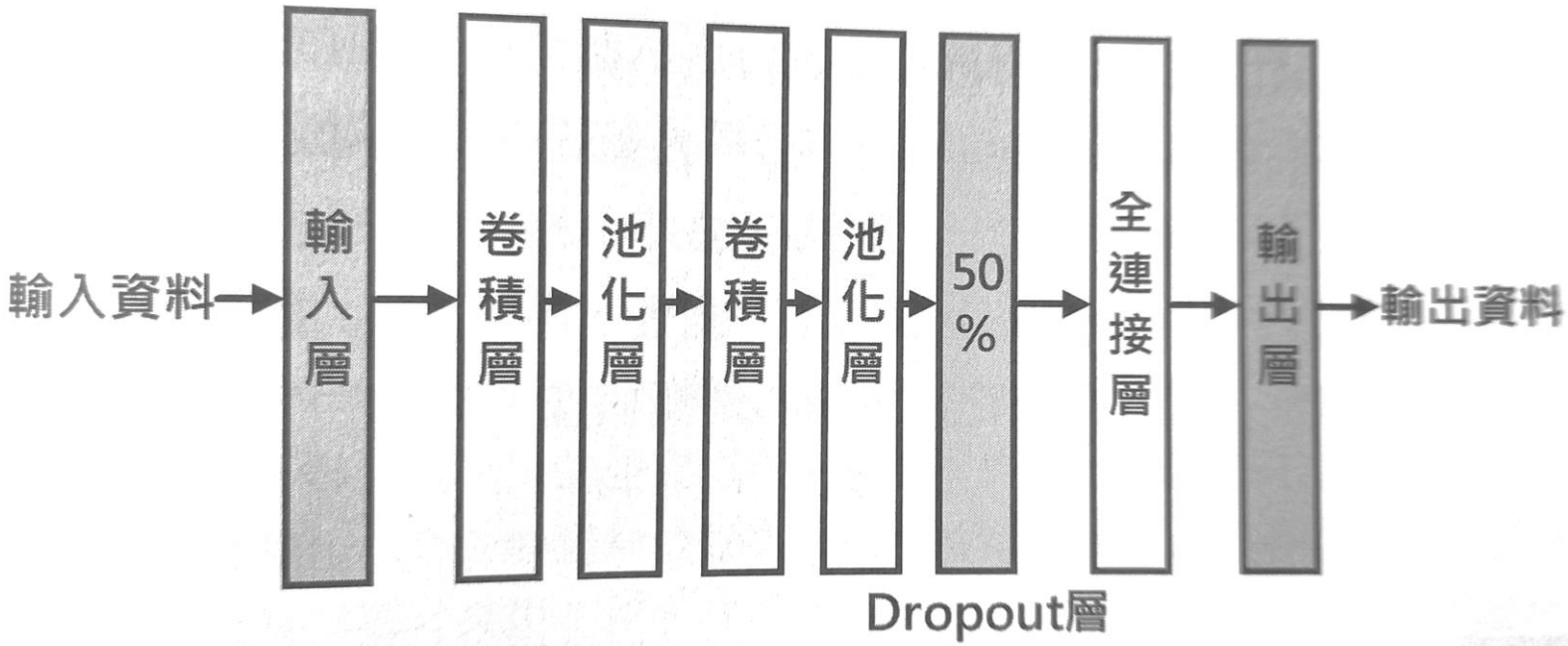


評估模型 — 訓練資料集的準確度

8-3 使用CNN打造MNIST手寫辨識

MLP：全域樣式 vs. CNN：局部樣式

8-3-1 如何使用KERAS打造卷積神經網路(CNN)



8-3-1 如何使用KERAS打造卷積神經網路(CNN)

卷積層(Keras.layers模組)

卷積層	說明
Conv1D	建立 1D 卷積層，可以在時間維度的序列資料上執行卷積運算，例如：語意分析
Conv2D	建立 2D 卷積層，可以在空間維度的二維資料上執行卷積運算，例如：圖片分類與識別
UpSampling1D	建立 1D 輸入的上升取樣層，可以沿著時間軸來將資料重複指定次數
UpSampling2D	建立 2D 輸入的上升取樣層，可以沿著二維空間來將資料重複指定次數

8-3-1 如何使用KERAS打造卷積神經網路(CNN)

池化層(Keras.layers模組)

池化層	說明
MaxPooling1D	建立序列資料的 1D 最大池化
MaxPolling2D	建立空間資料的 2D 最大池化
AveragePooling1D	建立序列資料的 1D 平均池化
AveragePolling2D	建立空間資料的 2D 平均池化

8-3-2 CNN的資料預處理

將特徵資料(樣本數, 28,28)形狀轉成4D張亮(樣本數,28,28,1)形狀 → 在最後新增灰階色彩值的通道

```
import numpy as np
from keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# 指定亂數種子
seed = 7
np.random.seed(seed)
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
# 將圖片轉換成 4D 張量
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype("float32")
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype("float32")
print("X_train Shape: ", X_train.shape)
print("X_test Shape: ", X_test.shape)
```

```
X_train Shape:  (60000, 28, 28, 1)
X_test Shape:  (10000, 28, 28, 1)
```

8-3-2 CNN的資料預處理

執行特徵標準化的正規化 (灰階圖片正規化)

```
#  因為是固定範圍,  所以執行正規化,  從  0-255  至  0-1
X_train  =  X_train  /  255
X_test  =  X_test  /  255
```

將標籤資料執行One-hot編碼

```
#  One-hot編碼
Y_train  =  to_categorical(Y_train)
Y_test  =  to_categorical(Y_test)
print("Y_train  Shape:  ",  Y_train.shape)
print(Y_train[0])
```

```
Y_train Shape:  (60000, 10)
[0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
```

8-3-2 CNN的資料預處理

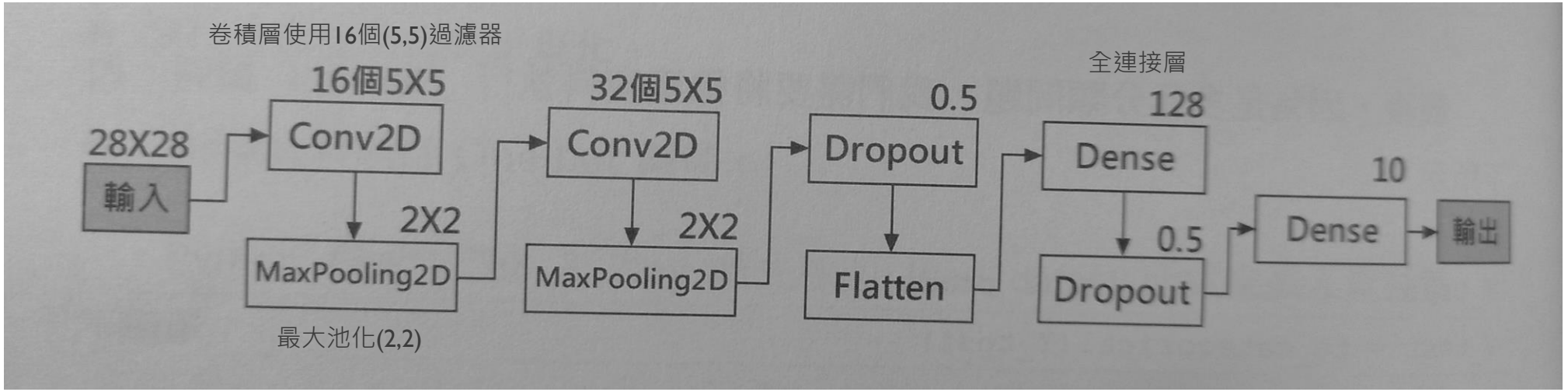
將標籤資料執行One-hot編碼

```
import  numpy  as  np
from  keras.datasets  import  mnist
from  tensorflow.keras.utils  import  to_categorical

#  One-hot編碼
Y_train  =  to_categorical(Y_train)
Y_test  =  to_categorical(Y_test)
print("Y_train  Shape:  ",  Y_train.shape)
print(Y_train[0])
```

```
Y_train Shape:  (60000, 10)
[0.  0.  0.  0.  0.  1.  0.  0.  0.  0.]
```

8-3-3 使用CNN打造MNIST手寫辨識

定義模型




```
model = Sequential()
```

8-3-3 使用CNN打造MNIST手寫辨識

定義第1組的卷積層和池化層

```
model.add(Conv2D(16, kernel_size=(5, 5), padding="same",input_shape=(28, 28, 1),
activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

定義第2組的卷積層和池化層

```
model.add(Conv2D(32, kernel_size=(5, 5), padding="same", activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

8-3-3 使用CNN打造MNIST手寫辨識

Conv2D( )的主要參數

- Filters參數：過濾器數量的整數值，即卷積核數。
- Kernel_size參數：過濾器窗格尺寸的元組，一般是正方形且為奇數，例如：(3,3)。
- Padding參數：補零方式，預設參數值valid不是補零 ; same是補零成相同尺寸。
- Strides參數：指定步幅數的元組，即每次過濾口向右和向下移動的像素數，預設值是(1,1)，即向右和向下各1個像素。

8-3-3 使用CNN打造MNIST手寫辨識

MaxPooling2D的主要參數

- pool_size參數：沿著(垂直,水平)元組紛向的縮小比例，(2,2)元組是各縮小一半。
- Padding參數：同Conv2D
- Strides參數：同Conv2D

8-3-3 使用CNN打造MNIST手寫辨識

定義Droupout、平坦層和全連接層

```
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128,   activation="relu"))
model.add(Dropout(0.5))
model.add(Dense(10,   activation="softmax"))
model.summary()       #  顯示模型摘要資訊
```

```
Model: "sequential_10"
_________________________________________________________________
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, 32)        12832

max_pooling2d_1 (MaxPooling  (None, 7, 7, 32)          0
2D)

dropout_2 (Dropout)          (None, 7, 7, 32)          0

flatten (Flatten)            (None, 1568)              0

dense_27 (Dense)             (None, 128)               200832

dropout_3 (Dropout)          (None, 128)               0

dense_28 (Dense)             (None, 10)                1290

=================================================================
Total params: 215,370
Trainable params: 215,370
Non-trainable params: 0
_________________________________________________________________
```

8-3-3 使用CNN打造MNIST手寫辨識

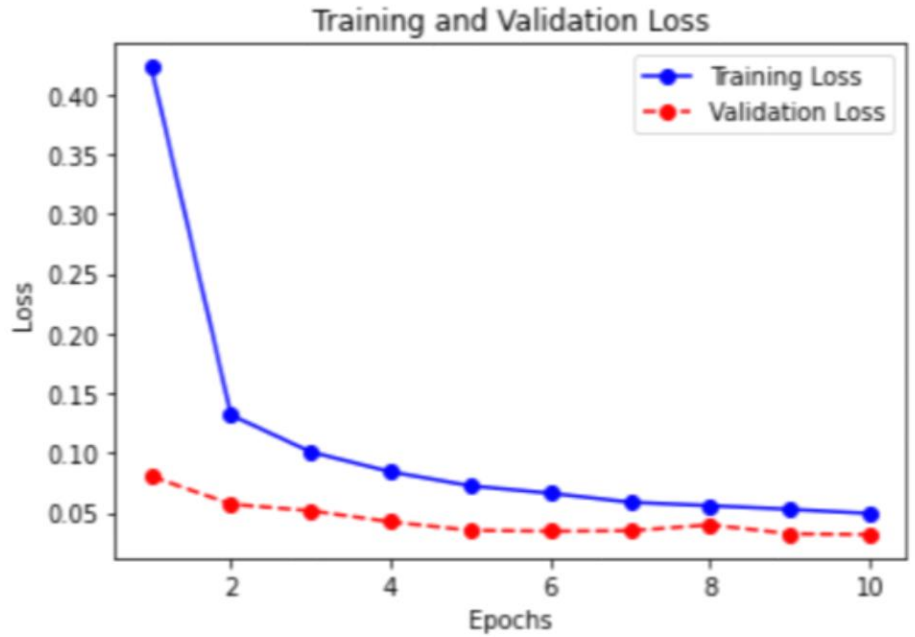
編譯模型 & 訓練模型

```
Epoch 1/10
375/375 - 45s - loss: 0.4230 - accuracy: 0.8658 - val_loss: 0.0804 - val_accuracy: 0.9755 - 45s/epoch - 120ms/step
Epoch 2/10
375/375 - 53s - loss: 0.1317 - accuracy: 0.9608 - val_loss: 0.0571 - val_accuracy: 0.9827 - 53s/epoch - 141ms/step
Epoch 3/10
375/375 - 55s - loss: 0.1007 - accuracy: 0.9693 - val_loss: 0.0515 - val_accuracy: 0.9855 - 55s/epoch - 148ms/step
Epoch 4/10
375/375 - 44s - loss: 0.0841 - accuracy: 0.9738 - val_loss: 0.0423 - val_accuracy: 0.9872 - 44s/epoch - 116ms/step
Epoch 5/10
375/375 - 49s - loss: 0.0724 - accuracy: 0.9779 - val_loss: 0.0351 - val_accuracy: 0.9898 - 49s/epoch - 130ms/step
Epoch 6/10
375/375 - 43s - loss: 0.0662 - accuracy: 0.9803 - val_loss: 0.0344 - val_accuracy: 0.9899 - 43s/epoch - 115ms/step
Epoch 7/10
375/375 - 43s - loss: 0.0587 - accuracy: 0.9817 - val_loss: 0.0348 - val_accuracy: 0.9891 - 43s/epoch - 115ms/step
Epoch 8/10
375/375 - 44s - loss: 0.0558 - accuracy: 0.9832 - val_loss: 0.0397 - val_accuracy: 0.9885 - 44s/epoch - 117ms/step
Epoch 9/10
375/375 - 43s - loss: 0.0528 - accuracy: 0.9837 - val_loss: 0.0322 - val_accuracy: 0.9912 - 43s/epoch - 114ms/step
Epoch 10/10
375/375 - 43s - loss: 0.0493 - accuracy: 0.9849 - val_loss: 0.0315 - val_accuracy: 0.9916 - 43s/epoch - 114ms/step
```

```
#  編譯模型  (損失函數,優化器,評估標準)
model.compile(loss="categorical_crossentropy",  optimizer="adam",
                              metrics=["accuracy"])
#  訓練模型
history  =  model.fit(X_train,  Y_train,  validation_split=0.2,
(訓練資料集,標籤資料集,分割驗證資料,訓練週期,批次尺寸) epochs=10,  batch_size=128,  verbose=2)
```

8-3-3 使用CNN打造MNIST手寫辨識

```
1875/1875 [==============================] - 19s 10ms/step - loss: 0.0183 - accuracy: 0.9945
訓練資料集的準確度 = 0.99
Saving Model: mnist_tf.h5 ...
```

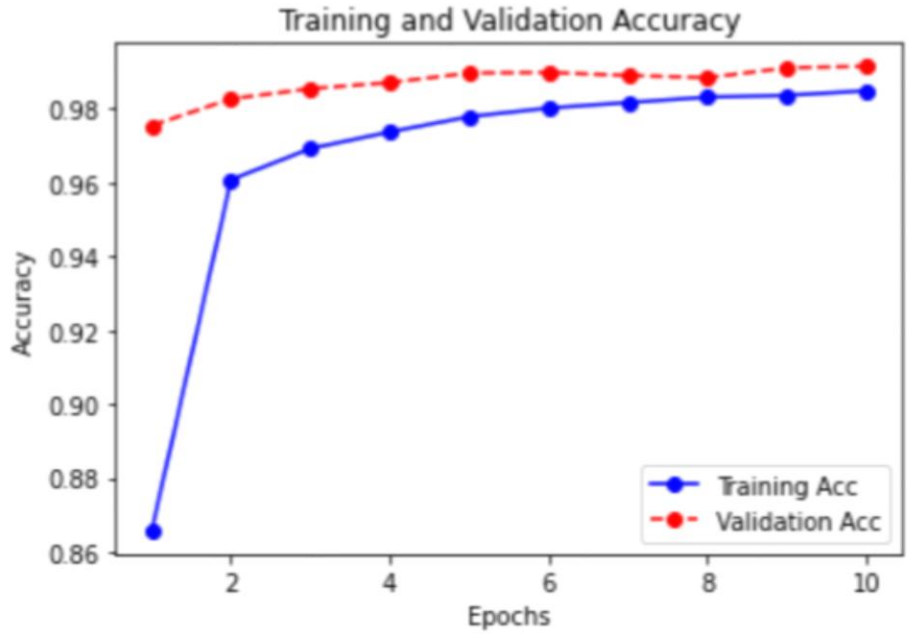


評估模型 — 訓練資料集的準確度

```python
# 評估模型
loss, accuracy = model.evaluate(X_train, Y_train)
print("訓練資料集的準確度 = {:.2f}".format(accuracy))
# 儲存Keras模型
print("Saving Model: mnist_tf.h5 ...")
model.save("mnist_tf.h5")
# 顯示圖表來分析模型的訓練過程
import matplotlib.pyplot as plt
# 顯示訓練和驗證損失
loss = history.history["loss"]
epochs = range(1, len(loss)+1)
val_loss = history.history["val_loss"]
plt.plot(epochs, loss, "bo-", label="Training Loss")
plt.plot(epochs, val_loss, "ro--", label="Validation Loss")
plt.title("Training and Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

8-3-3 使用CNN打造MNIST手寫辨識

```
313/313 [==============================] - 3s 10ms/step - loss: 0.0220 - accuracy: 0.9923
測試資料集的準確度 = 0.99
```



評估模型 — 測試資料集的準確度

```python
loss, accuracy = model.evaluate(X_test, Y_test)
print("測試資料集的準確度 = {:.2f}".format(accuracy))
#  顯示訓練和驗證準確度
acc = history.history["accuracy"]
epochs = range(1, len(acc)+1)
val_acc = history.history["val_accuracy"]
plt.plot(epochs, acc, "bo-", label="Training Acc")
plt.plot(epochs, val_acc, "ro--", label="Validation Acc")
plt.title("Training and Validation Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

8-4 手寫辨識的預測結果 — 使用混淆矩陣分析預測結果

```
import  numpy  as  np
import  pandas  as  pd
from  tensorflow.keras.datasets  import  mnist
from  tensorflow.keras.models  import  Sequential
from  tensorflow.keras.models  import  load_model
from  tensorflow.keras.utils  import  to_categorical

Y_test_bk  =  Y_test.copy()      #  備份  Y_test  資料集
Y_test  =  to_categorical(Y_test)
#  建立Keras的Sequential模型
model  =  Sequential()
model  =  load_model("mnist.h5")
#  編譯模型
model.compile(loss="categorical_crossentropy",  optimizer="adam",
                          metrics=["accuracy"])
#  評估模型
print("Testing  ...")
loss,  accuracy  =  model.evaluate(X_test,  Y_test,  verbose=0)
print("測試資料集的準確度  =  {:.2f}".format(accuracy))
#  計算分類的預測值
print("\n[redicting  ...")
Y_pred  =  np.argmax(model.predict(X_test),axis=1)
#  顯示混淆矩陣
tb  =  pd.crosstab(Y_test_bk.astype(int),  Y_pred.astype(int),
                                rownames=["label"],  colnames=["predict"])
print(tb)
tb.to_html("Ch8_4.html")
```

Y_pred = model.predict_classes(X_test)

(真實標籤值,預測值,列名稱,欄名稱)

8-4 手寫辨識的預測結果 — 使用混淆矩陣分析預測結果

```
Testing ...
測試資料集的準確度 = 0.99

Predicting ...
predict    0     1     2    3    4    5    6     7    8    9
label
0        976     0     0    0    0    0    3     1    0    0
1          0  1131     1    0    0    1    0     2    0    0
2          1     0  1030    0    0    0    0     1    0    0
3          0     0     0  999    0    6    0     2    3    0
4          0     0     0    0  978    0    1     0    0    3
5          1     0     0    1    0  888    1     0    0    1
6          7     2     0    0    1    0  947     0    1    0
7          0     1     4    0    0    0    0  1021    1    1
8          1     0     2    1    0    0    0     1  967    2
9          3     1     0    0    7    4    0     0    1  993
```

8-4 手寫辨識的預測結果 — 繪出0~9數字的預測機率

```
#  選一個測試的數字圖片
#  i  =  np.random.randint(0,  len(X_test))
i  =  7
digit  =  X_test[i].reshape(28,  28)
#  將圖片轉換成  4D  張量
X_test_digit  =  X_test[i].reshape(1,  28,  28,  1).astype("float32")
#  因為是固定範圍,  所以執行正規化,  從  0-255  至  0-1
X_test_digit  =  X_test_digit  /  255
```

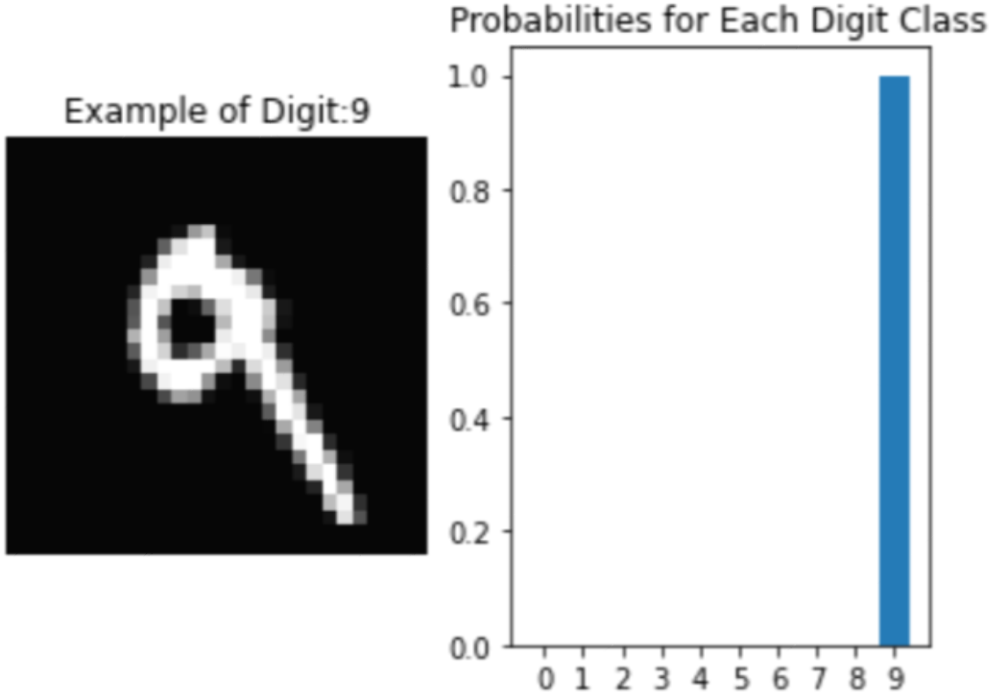
8-4 手寫辨識的預測結果 — 繪出0~9數字的預測機率

```
#  繪出圖表的預測結果
plt.figure()
plt.subplot(1,2,1)
plt.title("Example  of  Digit:"  +  str(Y_test[i]))
plt.imshow(digit,  cmap="gray")
plt.axis("off")
#  預測結果的機率
print("Predicting ...")
probs  =  model.predict(X_test_digit,  batch_size=1)
print(probs)
plt.subplot(1,2,2)
plt.title("Probabilities  for  Each  Digit  Class")
plt.bar(np.arange(10),  probs.reshape(10),  align="center")
plt.xticks(np.arange(10),np.arange(10).astype(str))
plt.show()
```

```
Y_probs = model.predict_proba(X_test)
```

8-4 手寫辨識的預測結果 — 繪出0~9數字的預測機率

```
Predicting ...
[[5.8450844e-08 9.5135618e-07 5.8919454e-06 1.7431778e-06 9.2469231e-04
  9.4193922e-07 4.5052944e-09 4.1075841e-06 2.3364915e-05 9.9903834e-01]]
```



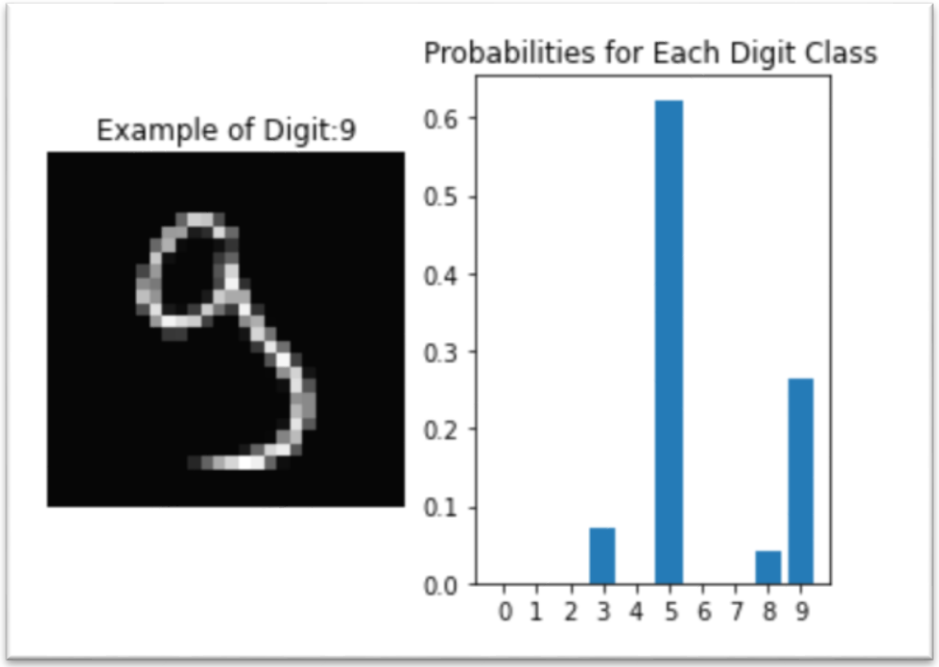
8-4 手寫辨識的預測結果 — 篩選分類錯誤和繪出各預測錯誤的機率

```
#  將圖片轉換成  4D  張量
X_test_bk  =  X_test.copy()       #  備份  X_test  測試資料集
X_test  =  X_test.reshape(X_test.shape[0],  28,  28,  1).astype("float32")
#  因為是固定範圍,  所以執行正規化,  從  0-255  至  0-1
X_test  =  X_test  /  255
#  建立Keras的Sequential模型
model  =  Sequential()
model  =  load_model("mnist.h5")
#  編譯模型
model.compile(loss="categorical_crossentropy",  optimizer="adam",
                              metrics=["accuracy"])
#  測試資料集的分類和機率的預測值
print("Predicting  ...")
Y_pred  =  np.argmax(model.predict(X_test),axis=1)     #  分類
Y_probs  =  model.predict(X_test)       #  機率
#  建立分類錯誤的  DataFrame  物件
df  =  pd.DataFrame({"label":Y_test,  "predict":Y_pred})
df  =  df[Y_test!=Y_pred]     #  篩選出分類錯誤的資料
print(df.head())
df.head().to_html("Ch8_4b.html")
```

```
Predicting ...
     label  predict
445      6        0
449      3        5
582      8        2
740      4        9
938      3        5
Index:  1709
```

8-4 手寫辨識的預測結果 — 篩選分類錯誤和繪出各預測錯誤的機率

```
#  隨機選  1  個錯誤分類的數字索引
i  =  df.sample(n=1).index.values.astype(int)[0]
print("Index:  ",  i)
digit  =  X_test_bk[i].reshape(28,  28)
#  繪出圖表的預測結果
plt.figure()
plt.subplot(1,2,1)
plt.title("Example  of  Digit:"  +  str(Y_test[i]))
plt.imshow(digit,  cmap="gray")
plt.axis("off")
plt.subplot(1,2,2)
plt.title("Probabilities  for  Each  Digit  Class")
plt.bar(np.arange(10),  Y_probs[i].reshape(10),  align="center")
plt.xticks(np.arange(10),np.arange(10).astype(str))
plt.show()
```



感謝您