

CHAPTER

17

深度學習神經網路 實作案例

- 17-1 啟用 Google Colaboratory 雲端服務
- 17-2 Google Colaboratory 基本使用
- 17-3 深度學習實作案例：
鸚尾花資料集的多元分類
- 17-4 深度學習實作案例：
辨識 MNIST 手寫數字圖片

17-1

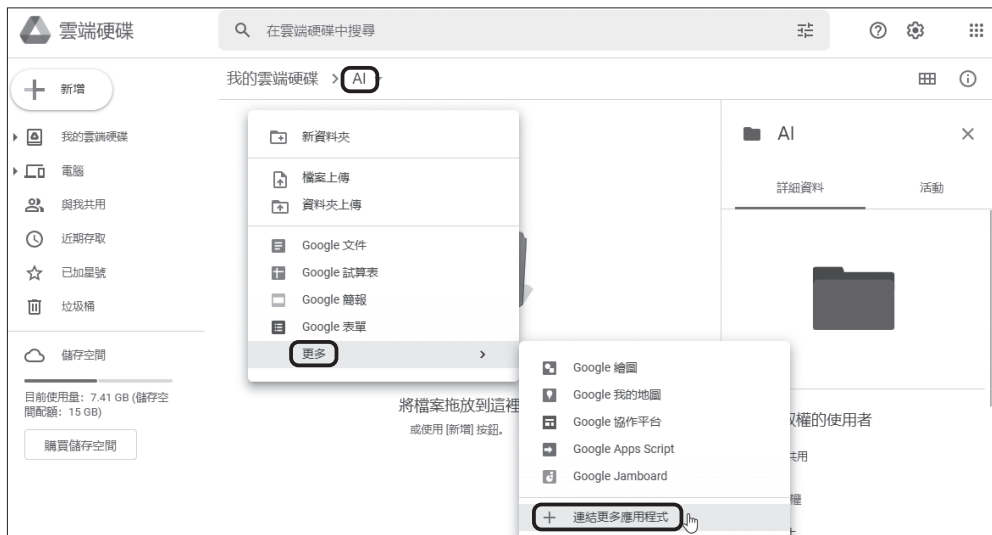
啟用 Google Colaboratory 雲端服務

Google Colaboratory 目前是免費的雲端服務，可以讓我們在 Google 雲端硬碟建立 Colab 筆記本（Notebook），不需安裝資料科學和深度學習的 Python 套件，直接使用 Colaboratory 來開發 Python 深度學習專案。

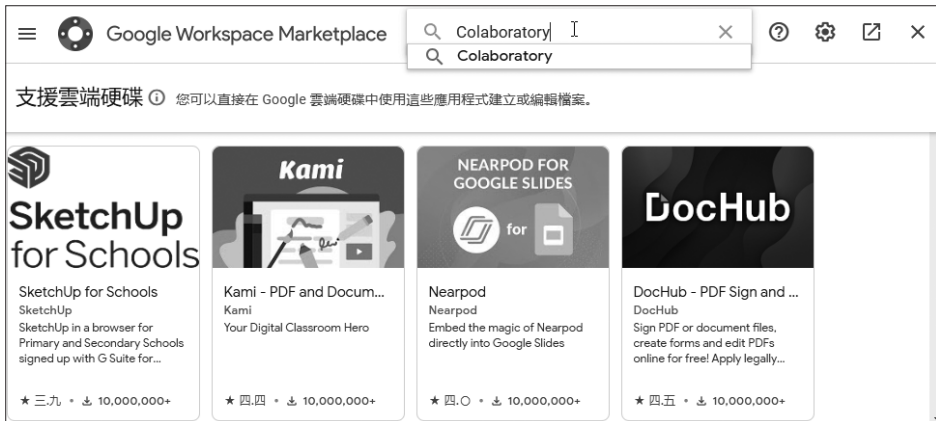
☆ 啟用 Google Colaboratory 雲端服務

在 Colaboratory 雲端服務新增的筆記本是儲存在 Google 雲端硬碟，我們需要先登入 Google 雲端硬碟後，才能使用 Colaboratory 雲端服務來建立 Colab 筆記本，其步驟如下所示：

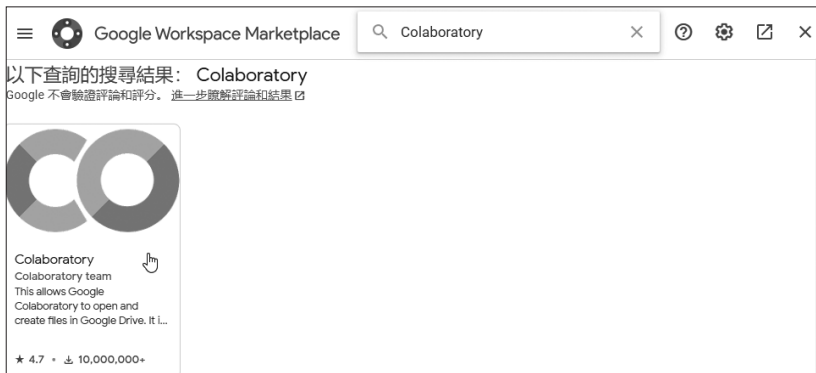
Step 1 請啟動 Chrome 瀏覽器進入 Google Drive 雲端硬碟，如果是第 1 次使用 Colaboratory，請新增和切換至指定資料夾，例如：「AI」資料夾後，在空白處執行右鍵快顯功能表的「更多/連結更多應用程式」命令。



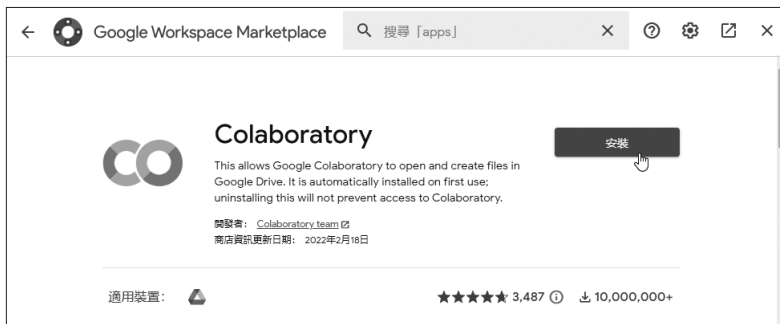
Step 2 在對話方塊上方欄位輸入 Colaboratory，按 **Enter** 鍵搜尋應用程式。



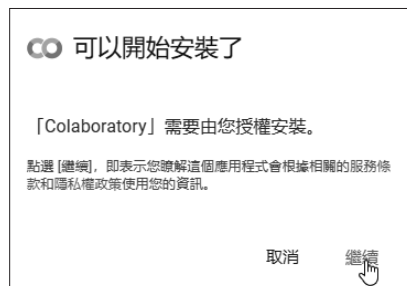
Step 3 在找到 Colaboratory 後，點選此應用程式。



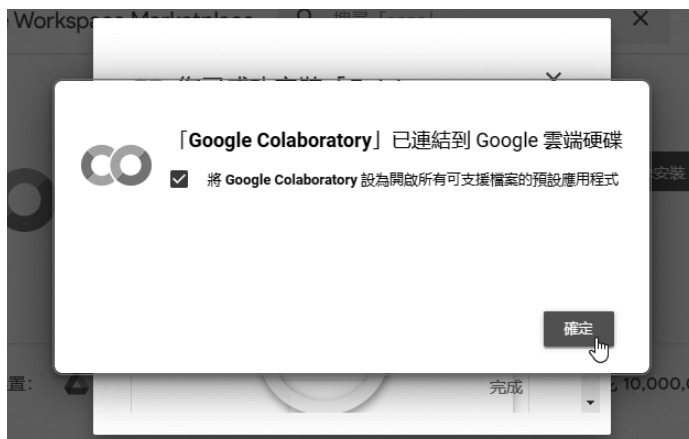
Step 4 然後按安裝鈕進行應用程式安裝。



Step 5 按**繼續**鈕，可以開始安裝了。



Step 6 在選擇 Google 帳戶後，等到成功安裝，可以看到一個訊息視窗已經連接至雲端硬碟，請按**確定**鈕。



Step 7 按**完成**鈕完成 Colaboratory 安裝。



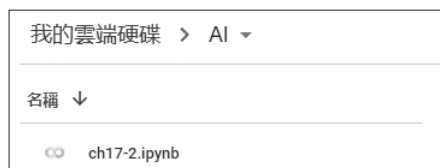
Step 8 請再次開啟「更多」功能表，可以看到新增 Google Colaboratory 命令，如右圖所示：



Step 9 請執行右鍵快顯功能表的「更多/Google Colaboratory」命令，可以建立一份全新的 Colab 筆記本（Notebook），如下圖所示：



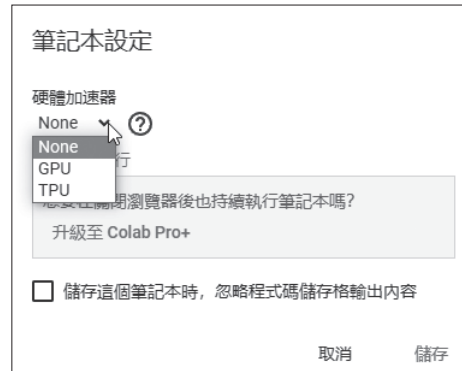
上述圖例上方位在圖示後的是筆記本名稱，Colab 筆記本就是 Jupyter 筆記本，副檔名是 .ipynb，在下方依序是功能表和工具列按鈕，接著是編輯區域，可以看到一個彈出的編輯框，稱為儲存格（Cell），這是 Colab 筆記本的基本編輯單位。我們新增的筆記本檔案是位在執行命令的目錄下，即 Google 雲端硬碟的「AI」資料夾，如下圖所示：



Colab 筆記本的操作方式和 Jupyter 筆記本十分類似，其基本使用方式請參閱第 17-2 節的說明。

☆ 啟用 Colaboratory 的硬體加速器

Colaboratory 雲端服務預設並沒有啟用 GPU/TPU 加速運算，請在 Colab 筆記本執行「執行階段/變更執行階段類型」命令，如右圖所示：



請點選**硬體加速器**欄位下的向下箭頭選單，就可以選擇使用 GPU，或 Google 的 TPU 硬體加速，選好後，按**儲存**鈕儲存設定。

17-2 Google Colaboratory 基本使用

在 Google Colaboratory 新增的 Colab 筆記本有些像是一本現實生活中的筆記本，可以方便我們編輯、呈現和分享探索性資料分析、機器學習或深度學習等資料分析的圖表和訓練結果。

☆ 更改 Colab 筆記本的名稱

在 Colaboratoy 新增筆記本的預設名稱是 `Untitled0.ipynb`，我們可以更改筆記本的名稱，其步驟如下所示：

Step 1 在 Colaboratoy 新增全新的筆記本後，請直接點選預設名稱 `Untitled0.ipynb`，就可以更改名稱，如下圖所示：



Step 2 請輸入新檔名 `ch17-2` 後，按 `Enter` 鍵，即可看到筆記本名稱已經更改成 `ch17-2.ipynb`。



☆ 在 Colab 筆記本編輯和執行 Python 程式碼

當成功新增 Colab 筆記本後，我們就可以在 Colab 筆記本的儲存格編輯和執行 Python 程式碼，其步驟如下所示：

Step 1 請點選儲存格，預設是程式碼的儲存格，當輸入程式碼超過一行時，請按 `Enter` 鍵換行，首先在儲存格輸入運算式：`5+10`，如下圖所示：



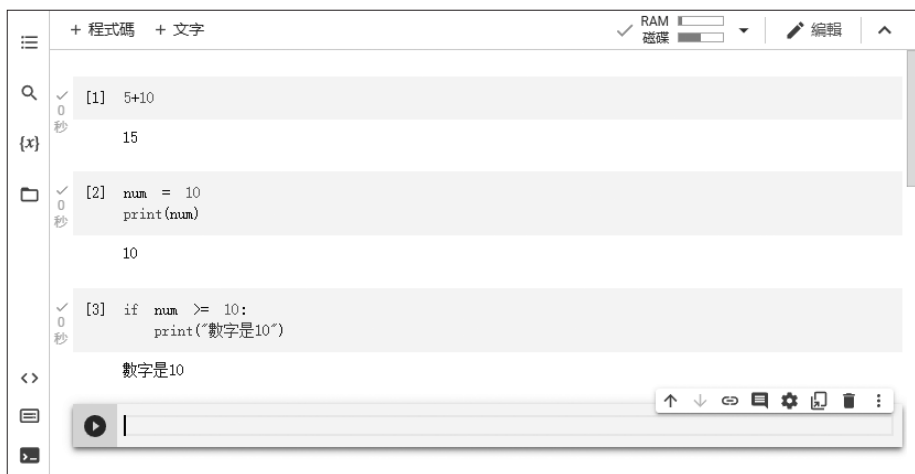
Step 2 按儲存格前方的三角箭頭圖示，可以執行此儲存格的程式碼，看到下方的執行結果 `15`，如下圖所示：



Step 3 點選上方 **+程式碼** 新增程式碼儲存格後，輸入 2 列程式碼，依序定義變數 `num = 10`，和 `print()` 函數顯示變數 `num` 值，按儲存格前方的三角箭頭圖示執行儲存格，可以顯示執行結果 10，如下圖所示：



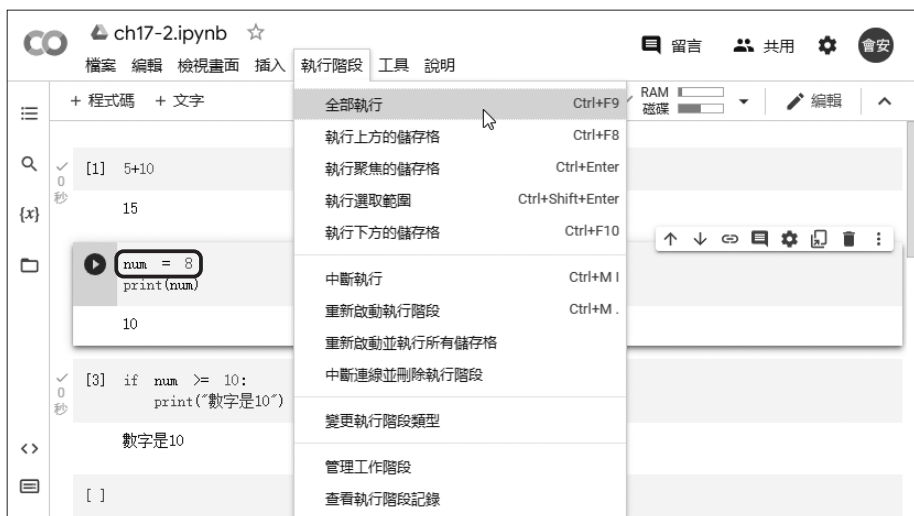
Step 4 再點選上方 **+程式碼** 新增程式碼儲存格，輸入 `if` 條件的程式區塊，在輸入 `if num >= 10:` 後，按 **Enter** 鍵，就會自動縮排 4 個空白字元，然後輸入 `print()` 函數的程式碼，我們也可以按 **Shift** + **Enter** 鍵來執行目前的儲存格，並且在下方自動新增一個儲存格，如下圖所示：



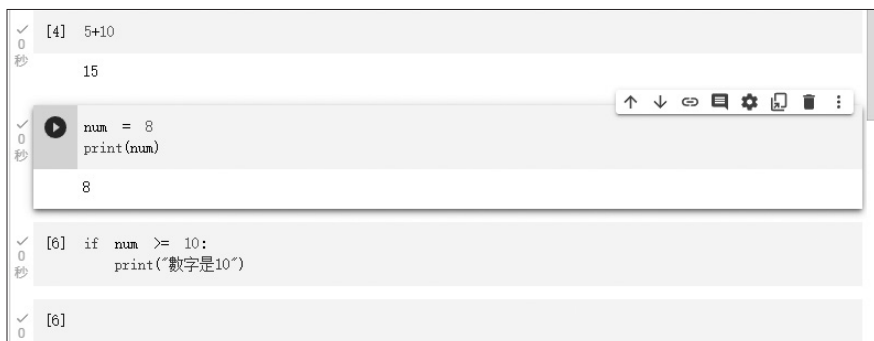
☆ 重新編輯 Colab 筆記本和執行 Python 程式碼

Colab 筆記本可以隨時修改 Python 程式碼來重複執行，其步驟如下所示：

Step 1 請點選第 2 個儲存格成為作用中的程式碼儲存格後，將 `num` 變數的值改為 8，如下圖所示：



Step 2 更改後，在上圖執行「執行階段/全部執行」命令重新執行全部程式碼，即可顯示整份筆記本 Python 程式碼的執行結果，因為變數 `num` 已經從 10 改為 8，現在的 `if` 條件並不成立，所以沒有顯示任何訊息文字，如下圖所示：



Google Colaboratory 預設自動就會定時儲存筆記本，我們也可以自行執行「檔案/儲存」命令來手動儲存筆記本。

☆ Colab 筆記本編輯和命令模式與常用按鍵

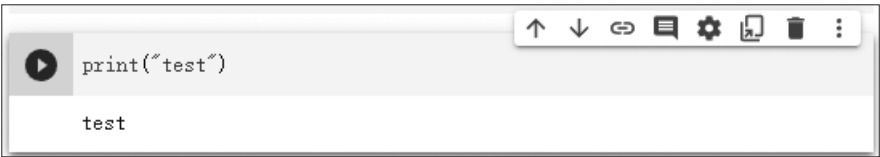
不知你是否注意到！當在作用中的儲存格輸入 Python 程式碼時，儲存格的外框是立體彈出狀態，這是編輯模式（Edit Mode），執行命令時，外框就恢復正常，沒有彈出狀態，此時是命令模式（Command Mode）。在 Colab 筆記本切換模式的按鍵說明，如下所示：

- ◆ 切換成編輯模式：在命令模式點選指定儲存格成為作用中儲存格，即進入此儲存格的編輯模式。
- ◆ 切換成命令模式：在編輯模式按 2 次 **Esc** 鍵，切換成命令模式。




Colab 筆記本一些常用按鍵的說明，如下表所示：

按鍵	說明
Up 和 Down	在編輯模式移至上一個和下一個程式碼與儲存格
A 和 B	在編輯模式可在目前儲存格的上方/下方新增一個作用中的儲存格
Z	在命令模式可以回復剛剛刪除的儲存格
Shift + Enter	在編輯模式執行作用中的儲存格，和移至下一個儲存格
Ctrl + Enter	在編輯模式執行作用中的儲存格
Alt + Enter	在編輯模式執行作用中的儲存格，和在下方新增一個新儲存格，但是並不會移至下一個儲存格
Ctrl + S	手動儲存 Colab 筆記本



當儲存格在作用中的編輯模式時，在右上方提供快速工具列的相關功能，如下圖所示：



上述快速工具列圖示的說明，從左至右如下所示：

- ◆ 第 1 個圖示 ：可以將儲存格向上移。
- ◆ 第 2 個圖示 ：可以將儲存格向下移。
- ◆ 第 3 個圖示 ：可以產生儲存格的超連結，如下圖所示：




- ◆ 第 4 個圖示 ：新增留言。
- ◆ 第 5 個圖示 ：開啟編輯器設定，可以設定 Colab 筆記本編輯器，如下圖所示：



◆ 第 6 個圖示 ：在右方分頁產生一個鏡像儲存格。

◆ 第 7 個圖示 ：刪除作用中的儲存格。

◆ 第 8 個圖示 ：顯示更多功能的功能表，如右圖所示：

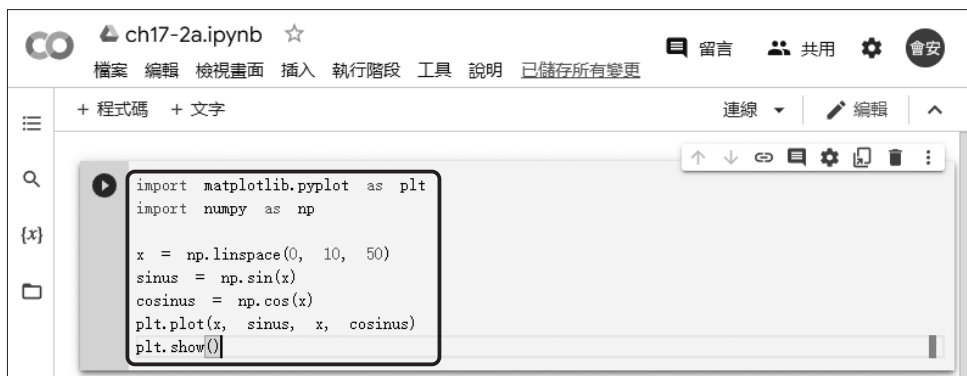


☆ 在 Colab 筆記本顯示 Matplotlib 圖表

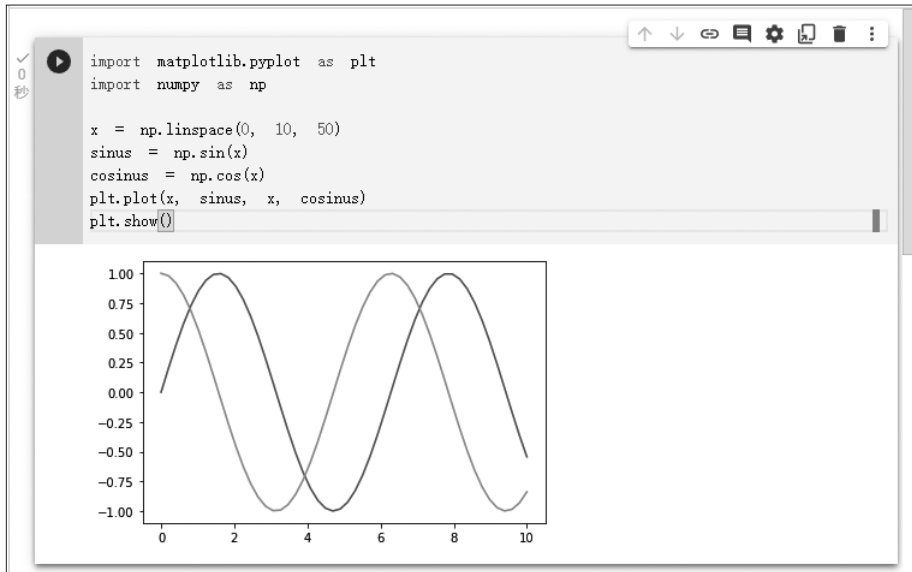
在 Colab 筆記本可以顯示 Matplotlib 圖表，請執行功能表的「檔案/新增筆記本」命令新增一份筆記本，和更名成 ch17-2a.ipynb，此方法新增的筆記本檔案是儲存在「Colab Notebooks」資料夾，如右圖所示：



接著請開啟書附 ch17-2a.py 檔案，複製和貼上 Python 程式碼至目前作用中的程式碼儲存格（請使用 **Ctrl** + **C** 鍵複製和 **Ctrl** + **V** 鍵貼上程式碼），可以看到貼上筆記本的 Python 程式碼，如下圖所示：



請按 **Ctrl** + **Enter** 鍵執行儲存格，可以顯示 Matplotlib 圖表，如下圖所示：



同理，我們只需將本書 Python 程式碼貼入筆記本的儲存格，就可以改用 Colab 筆記本測試執行 Python 程式，或直接執行「檔案/上傳筆記本」命令來上傳 Colab 筆記本的 .ipynb 檔案。

☆ 在 Colab 筆記本掛載 Google 雲端硬碟

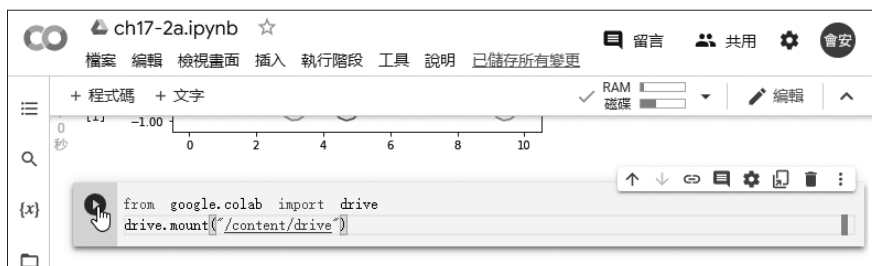
當 Colab 筆記本需要使用儲存在 Google 雲端硬碟的檔案時，例如：使用 Pandas 載入 Google 雲端硬碟的 iris.csv 檔案，我們需要在 Colaboratory 掛載 Google 雲端硬碟，其步驟如下所示：

Step 1 請先上傳 iris.csv 檔案至 Google 雲端硬碟的「Colab Notebooks」資料夾，如右圖所示：



Step 2 請開啟 Colab 筆記本 ch17-2a.ipynb 後，在新增的儲存格輸入下列程式碼來掛載 Google 雲端硬碟，如下所示：

```
from google.colab import drive
drive.mount("/content/drive")
```



Step 3 上述程式碼使用 `drive.mount()` 函數來掛載，請按儲存格前的三角箭頭圖示執行程式碼，可以看到確認對話方塊，請按**連接至 Google 雲端硬碟**鈕來連接 Google 雲端硬碟。

要允許這個筆記本存取你的 Google 雲端硬碟檔案嗎？

這個筆記本要求存取你的 Google 雲端硬碟檔案。獲得 Google 雲端硬碟存取權後，筆記本中執行的程式碼將可修改 Google 雲端硬碟的檔案。請務必在允許這項存取權前，謹慎審查筆記本中的程式碼。

不用了，謝謝

連線至 Google 雲端硬碟

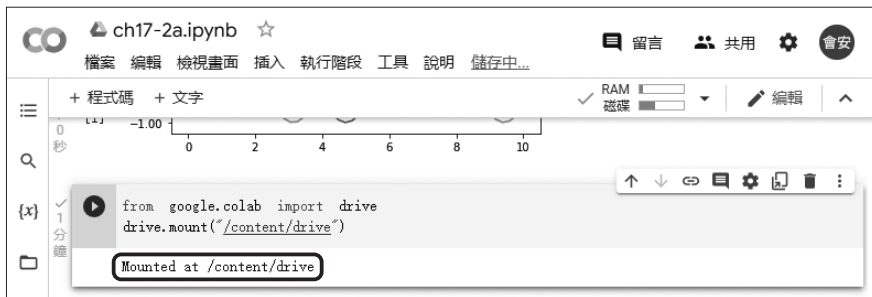
Step 4 請選檔案上傳到哪一個雲端硬碟帳戶，如右圖所示：



Step 5 可以看到授權清單，按允許鈕同意授權，如右圖所示：



Step 6 可以看到已經成功掛載「/content/drive」，如下圖所示：



Step 7 請點選上方 + 程式碼新增儲存格後，輸入程式碼使用 os 模組的 listdir() 函數，可以顯示 Google 雲端硬碟的檔案和資料夾清單，如下所示：

```
import os
os.listdir("/content/drive/My Drive/Colab Notebooks")
```

```
✓ 1 分鐘
[3] from google.colab import drive
    drive.mount("/content/drive")

Mounted at /content/drive

✓ 0 秒
▶ import os
  os.listdir("/content/drive/My Drive/Colab Notebooks")

['iris.csv', 'ch17-2a.ipynb']
```

Step 8 上述執行結果可以看到上傳的 iris.csv 檔案。現在，我們可以再次新增儲存格後，輸入 Python 程式碼，使用 Pandas 載入 iris.csv 和顯示前 5 筆資料，如下所示：

```
import pandas as pd

df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/iris.csv")
df.head()
```

```
✓ 0 秒
[7] import os
    os.listdir("/content/drive/My Drive/Colab Notebooks")

['iris.csv', 'ch17-2a.ipynb']

✓ 0 秒
▶ import pandas as pd
  df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/iris.csv")
  df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

17-3

深度學習實例： 鳶尾花資料集的多元分類

前面已經實作過很多遍鳶尾花資料集了，這是三種鳶尾花的花瓣和花萼資料，這裡改用神經網路訓練預測模型來分類鳶尾花，因為有三種所以是一種多元分類。

17-3-1 認識與探索鳶尾花資料集

鳶尾花資料集是一個 CSV 檔案 iris.csv，請建立 DataFrame 物件 df 來載入此資料集（Colab 筆記本：ch17-3-1.ipynb），如下所示：

```
[2] import pandas as pd

df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/iris.csv")
df.shape

(150, 5)
```

上述程式碼呼叫 read_csv() 函數載入 CSV 檔案 iris.csv 後，使用 shape 屬性顯示資料集形狀，鳶尾花資料集有 5 個欄位共 150 筆資料。

☆ 探索資料

當成功載入資料集後，首先使用 head() 函數看一看前 5 筆資料，如下所示：

```
[3] df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

上表的每一列是一種鳶尾花的花瓣和花萼資料，其各欄位的說明如下所示：

- ◆ `sepal_length`：花萼長度。
- ◆ `sepal_width`：花萼寬度。
- ◆ `petal_length`：花瓣長度。
- ◆ `petal_width`：花瓣寬度。
- ◆ `target`：鳶尾花種類，其值是 `setosa`、`versicolor` 或 `virginica`。

接著，使用 `describe()` 函數顯示資料集描述的統計摘要資訊，如下所示：

```
[4] df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

上述表格顯示 4 個數值欄位的統計摘要資訊，可以看到欄位值的資料量、平均值、標準差、最小和最大等資料描述。

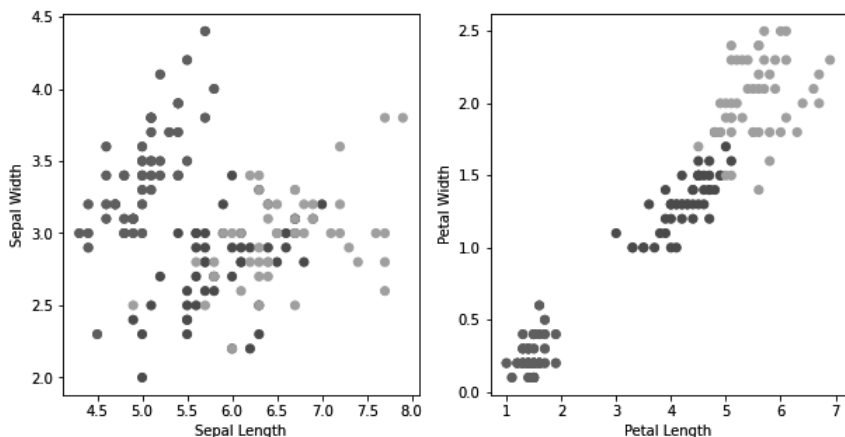
☆ 顯示視覺化圖表

然後使用 `Matplotlib` 視覺化顯示花瓣和花萼長寬的散佈圖，這是套用色彩的 2 個子圖，為了套用色彩，請將 `DataFrame` 物件 `df` 的 `target` 欄位轉換成 0~2 的整數，`colmap` 是色彩對照表，如下所示：

```
[5] import matplotlib.pyplot as plt
import numpy as np

target_mapping = {"setosa": 0,
                  "versicolor": 1,
                  "virginica": 2}
Y = df["target"].map(target_mapping)
colmap = np.array(["r", "g", "y"])
plt.figure(figsize=(10,5))
plt.subplot(1, 2, 1)
plt.subplots_adjust(hspace = .5)
plt.scatter(df["sepal_length"], df["sepal_width"], color=colmap[Y])
plt.xlabel("Sepal Length")
plt.ylabel("Sepal Width")
plt.subplot(1, 2, 2)
plt.scatter(df["petal_length"], df["petal_width"], color=colmap[Y])
plt.xlabel("Petal Length")
plt.ylabel("Petal Width")
plt.show()
```

上述程式碼呼叫 `subplots_adjust()` 函數調整間距，和使用 `scatter()` 函數繪出散佈圖，參數 `color` 是對應 `target` 欄位值來顯示不同色彩，可以分別繪出花萼（Sepal）和花瓣（Petal）的長和寬為座標（x,y）的散佈圖，其執行結果如下圖所示：



上述散佈圖的紅色點是 `setosa`、綠色點是 `versicolor` 和黃色點是 `virginica` 的三種鳶尾花。

17-3-2 鳶尾花資料集的多元分類

Colab 筆記本：ch17-3-2.ipynb 建立鳶尾花資料集的多元分類，首先匯入所需的模組與套件，如下所示：

```
[2] import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn import preprocessing

np.random.seed(5)
```

上述程式碼匯入 NumPy 和 Pandas 套件，並載入神經網路模組 Keras 來匯入 Sequential 模型和 Dense 全連接層，to_categorical 是 One-hot 編碼，Scikit-learn 套件的 preprocessing 模組，然後指定亂數種子是 5（各模組的細節後續一一介紹）。

☆ 資料預處理

在載入鳶尾花資料集後，首先需要進行資料預處理，如下所示：

- ◆ 將 target 欄位的三種分類轉換成整數的 0~2。
- ◆ 分割成特徵資料和標籤資料，並且進行標籤資料的 One-hot 編碼。
- ◆ 執行特徵標準化。
- ◆ 將資料集分割成訓練和測試資料集。

在 Python 程式首先載入鳶尾花資料集，如下所示：

```
[3] df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/iris.csv")
```

上述程式碼呼叫 read_csv() 函數載入資料集後，使用 Scikit-learn 套件的 preprocessing 模組來使用 LabelEncoder 物件進行資料的分類轉換，如下所示：

```
[4] label_encoder = preprocessing.LabelEncoder()
    df["target"] = label_encoder.fit_transform(df["target"])
```

上述程式碼將分類欄位轉換成 0~2 值。然後使用 values 屬性取出 NumPy 陣列，和呼叫 np.random.shuffle() 函數使用亂數來打亂資料，如下所示：

```
[5] dataset = df.values
    np.random.shuffle(dataset)
    X = dataset[:,0:4].astype(float)
    Y = to_categorical(dataset[:,4])
```

上述 Python 分割運算子分割前 4 個欄位的特徵資料後，轉換成 float 型態，第 5 個欄位是標籤資料，需要使用 to_categorical() 函數執行 One-hot 編碼。然後執行標準化（Standardization），如下所示：

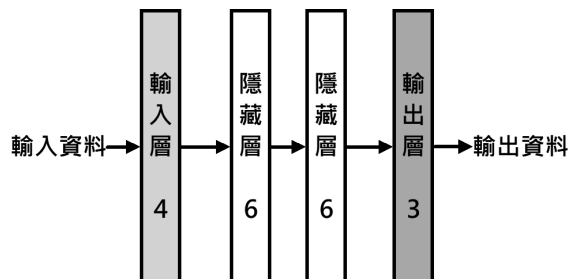
```
[6] scaler = preprocessing.StandardScaler()
    X = scaler.fit_transform(X)
```

最後將資料集的 150 筆資料，使用 Python 分割運算子切割成前 120 筆的訓練資料集；後 30 筆是測試資料集，如下所示：

```
[7] X_train, Y_train = X[:120], Y[:120]
    X_test, Y_test = X[120:], Y[120:]
```

☆ 定義模型

接著定義神經網路模型，規劃的神經網路共有四層，這是一種深度神經網路，如右圖所示：



上述輸入層有 4 個特徵，2 個隱藏層都是 6 個神經元，因為鳶尾花資料集是預測三種分類的鳶尾花，屬於多元分類問題，所以輸出層是 3 類共 3 個神經元。

在 Python 程式定義 Keras 神經網路模型，首先建立 Sequential 物件，如下所示：

```
[8] model = Sequential()
    model.add(Dense(6, input_shape=(4,), activation="relu"))
    model.add(Dense(6, activation="relu"))
    model.add(Dense(3, activation="softmax"))
```

上述程式碼共呼叫 3 次 add() 函數新增 3 層 Dense 層，其說明如下所示：

- ◆ **第 1 層隱藏層**：6 個神經元，使用 input_shape 指定輸入層是 4 個神經元，啟動函數是 ReLU 函數，啟動函數是為了增加神經網路對於非線性規則的能力。
- ◆ **第 2 層隱藏層**：6 個神經元，啟動函數是 ReLU 函數。
- ◆ **輸出層**：3 個神經元，啟動函數是 Softmax 函數。

然後呼叫 summary() 函數顯示模型的摘要資訊，如下所示：

```
[9] model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 6)	30
dense_1 (Dense)	(None, 6)	42
dense_2 (Dense)	(None, 3)	21

```
Total params: 93
Trainable params: 93
Non-trainable params: 0
```

上述圖例顯示每一層神經層的參數個數，和整個神經網路的參數總數。

☆ 編譯模型

在定義好模型後，需要編譯模型來轉換成低階 TensorFlow 計算圖，如下所示：

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

上述 compile() 函數可以編譯模型，使用 categorical_crossentropy 損失函數，優化器是 adam，評估標準是 accuracy 準確度。

☆ 訓練模型

在編譯模型成 TensorFlow 計算圖後，就可以送入特徵資料來訓練模型，如下所示：

```
[11] model.fit(X_train, Y_train, epochs=100, batch_size=5)
```

上述 fit() 函數是訓練模型，第 1 個參數是訓練資料集 X_train，第 2 個參數是標籤資料集 Y_train，訓練週期是 100 次，批次尺寸是 5，其執行結果只顯示其中最後 6 次訓練週期，如下所示：

```
Epoch 95/100
24/24 [=====] - 0s 2ms/step - loss: 0.1477 - accuracy: 0.9583
Epoch 96/100
24/24 [=====] - 0s 3ms/step - loss: 0.1466 - accuracy: 0.9583
Epoch 97/100
24/24 [=====] - 0s 2ms/step - loss: 0.1443 - accuracy: 0.9583
Epoch 98/100
24/24 [=====] - 0s 2ms/step - loss: 0.1431 - accuracy: 0.9583
Epoch 99/100
24/24 [=====] - 0s 2ms/step - loss: 0.1427 - accuracy: 0.9583
Epoch 100/100
24/24 [=====] - 0s 2ms/step - loss: 0.1407 - accuracy: 0.9583
<keras.callbacks.History at 0x7f5a4675f510>
```

☆ 評估模型

當使用訓練資料集訓練模型後，就可以使用測試資料集來評估模型效能，如下所示：

```
[12] loss, accuracy = model.evaluate(X_test, Y_test)
     print("Accuracy = {:.2f}".format(accuracy))

1/1 [=====] - 0s 140ms/step - loss: 0.1502 - accuracy: 0.9667
Accuracy = 0.97
```

上述程式碼呼叫 `evaluate()` 函數評估模型，參數是 `X_test` 和 `Y_test` 資料集，其執行結果的準確度是 0.97（即 97%）。

☆ 預測鳶尾花的種類

現在，我們可以預測 `X_test` 測試資料集的鳶尾花種類，因為是預測分類資料，首先使用 `model.predict()` 函數預測 `X_test` 測試資料集的可能性分數，然後呼叫 `np.argmax()` 函數取出最大可能性分數的索引，即鳶尾花的種類 0、1 和 2。

因為 `Y_test` 資料集已經執行過 One-hot 編碼，所以改從 `dataset` 陣列再次分割標籤資料且轉換成整數後，即可將預測值和真正的標籤資料進行比較，如下所示：

```
[13] Y_pred = np.argmax(model.predict(X_test), axis=-1)
     print(Y_pred)
     Y_target = dataset[:,4][120:].astype(int)
     print(Y_target)

[2 1 2 0 2 0 2 0 1 0 2 2 1 2 2 2 1 0 1 0 0 1 2 0 2 0 1 2 2 1]
[2 1 2 0 2 0 2 0 1 0 2 2 0 2 2 2 1 0 1 0 0 1 2 0 2 0 1 2 2 1]
```

上述執行結果的上方是預測值，下方是標籤值，2 個陣列只有 1 個錯誤（第 13 個）。

17-4 深度學習實作案例： 辨識 MNIST 手寫數字圖片

MNIST 手寫辨識是一種多元分類，可以將手寫數字圖片分類成 10 類，我們準備使用 CNN 卷積神經網路來打造 MNIST 手寫辨識。

17-4-1 認識與探索 MNIST 手寫數字資料集

MNIST (Mixed National Institute of Standards and Technology) 資料集是 Yann Lecun's 提供的圖片資料庫，包含 60,000 張手寫數字圖片 (Handwritten Digit Image) 的訓練資料，和 10,000 張測試資料。

MNIST 資料集是成對的數字手寫圖片和對應的標籤資料 0~9，其簡單說明如下所示：

◆ 手寫數字圖片：尺寸 28×28 像素的灰階點陣圖。

◆ 標籤：手寫數字圖片對應實際的 0~9 數字。

☆ 載入和探索 MNIST 手寫辨識資料集

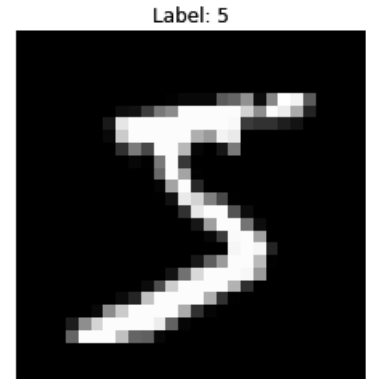
Keras 已經內建 MNIST 手寫辨識資料集，只需匯入 `mnist`，就可以載入 MNIST 資料集 (Colab 筆記本：ch17-4-1.ipynb)，如下所示：

```
[1] 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()
```

上述程式碼匯入 Keras 內建 MNIST 資料集後，呼叫 `load_data()` 函數載入 MNIST 資料集，如果是第 1 次載入，預設會自動下載資料集。

上述程式碼顯示第 1 張圖片，標題文字是對應的真實標籤資料，其執行結果如右圖所示：



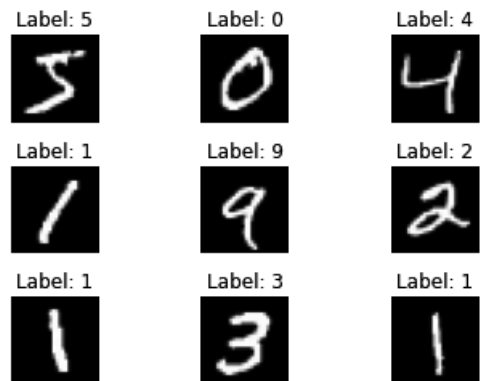
☆ 顯示 MNIST 資料集的前 9 張圖片

然後，我們可以使用 Matplotlib 子圖來同時顯示多張數字圖片，如下所示：

```
[4] 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)
    plt.show()
```

上述程式碼使用 for/in 迴圈顯示資料集的前 9 張圖片，圖表的標題文字是對應的標籤資料，其執行結果如右圖所示：



17-4-2 使用 CNN 打造 MNIST 手寫辨識

Colab 筆記本：ch17-4-2.ipynb 是使用 CNN 卷積神經網路來打造 MNIST 手寫辨識，首先匯入所需的模組與套件，如下所示：

```
[1] 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.layers import Flatten
    from tensorflow.keras.layers import Conv2D
    from tensorflow.keras.layers import MaxPooling2D
    from tensorflow.keras.layers import Dropout
    from tensorflow.keras.utils import to_categorical

    # 指定亂數種子
    np.random.seed(7)
```

上述程式碼匯入 NumPy 和 Pandas 套件，Keras 匯入 Sequential 模型、Dense、Flatten、Conv2D、MaxPooling2D 和 Dropout 層，to_categorical 是 One-hot 編碼，然後指定亂數種子是7。

☆ 資料預處理

在建立 CNN 神經網路前，首先需要執行資料預處理，如下所示：

- ◆ 將特徵資料（樣本數, 28, 28）形狀轉換成 4D 張量（樣本數, 28, 28, 1）形狀，即在最後新增灰階色彩值的通道（Channel）。
- ◆ 執行特徵標準化的正規化（Normalization）。
- ◆ 將標籤資料執行 One-hot 編碼。

在 Python 程式首先呼叫 load_data() 函數來載入資料集，如下所示：

```
# 載入資料集
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

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

上述程式碼在載入手寫數字圖片 MNIST 資料集後，需要將原（樣本數，28，28）形狀的特徵轉換成（樣本數，28，28，1）形狀的 4D 張量，以便送入 CNN 神經網路來進行訓練，如下所示：

```
[3] # 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")
```

上述程式碼呼叫 2 次 reshape() 函數，將特徵的訓練和測試資料集轉換成（樣本數，28，28，1）的 4D 張量，和轉換成浮點數。然後直接除以 255 來執行灰階圖片的正規化（因為值是固定範圍 0~255），如下所示：

```
[4] # 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
```

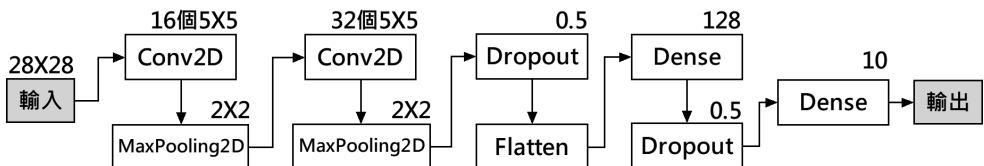
最後，因為是一種多元分類問題，我們需要將標籤資料執行 One-hot 編碼，如下所示：

```
[5] # One-hot編碼
Y_train = to_categorical(Y_train)
Y_test = to_categorical(Y_test)
```

上述程式碼呼叫 2 次 to_categorical() 函數來執行訓練和測試標籤資料的 One-hot 編碼。

☆ 定義模型

在 MNIST 手寫辨識的 CNN 卷積神經網路是使用 2 組卷積和池化層，並且使用 2 個 Dropout 層，如下圖所示：



上述 CNN 的第 1 個卷積層是使用 16 個 (5,5) 過濾器，第 2 個是 32 個 (5,5)，池化層都是最大池化 (2,2)，Dropout 層都是 0.5，Dense 全連接層是 128 個神經元，因為數字有 10 種，最後輸出的 Dense 層是 10 個神經元，如下所示：

```
[6] # 定義模型
model = Sequential()
model.add(Conv2D(16, kernel_size=(5, 5), padding="same",
                  input_shape=(28, 28, 1), activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, kernel_size=(5, 5), padding="same",
                  activation="relu"))
model.add(MaxPooling2D(pool_size=(2, 2)))
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"))
```

上述程式碼共呼叫 9 次 add() 函數來新增各層，其說明如下所示：

- ◆ 定義第 1 組的卷積和池化層：第 1 組 Conv2D 卷積層的過濾器數是 16，過濾器窗格尺寸是 (5,5)，使用 padding 參數 same 補零成相同尺寸，啟動函數是 ReLU 函數，MaxPooling2D 池化層的 (2,2) 元組是各縮小一半。
- ◆ 定義第 2 組卷積和池化層：只有過濾器數改為 32 個。
- ◆ 2 個 Dropout 層：2 個 Dropout 層都是 0.5，即 50%
- ◆ 1 個平坦層：可以轉換成 1 維向量
- ◆ 全連接層：Dense 層的神經元數是 128 個，啟動函數是 ReLU。
- ◆ 輸出層：Dense 層的神經元是 10 個，啟動函數是 Softmax 函數。

然後，我們可以顯示模型摘要資訊，如下所示：

```
[7] model.summary()      # 顯示模型摘要資訊
```

上述函數顯示每一層神經層的參數個數，和整個神經網路的參數總數，其執行結果如下所示：

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, 32)	12832
max_pooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0
dropout (Dropout)	(None, 7, 7, 32)	0
flatten (Flatten)	(None, 1568)	0
dense (Dense)	(None, 128)	200832
dropout_1 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290

Total params: 215,370
 Trainable params: 215,370
 Non-trainable params: 0

☆ 編譯模型

在定義好模型後，我們需要編譯模型來轉換成低階 TensorFlow 計算圖，如下所示：

```
[8] # 編譯模型
    model.compile(loss="categorical_crossentropy", optimizer="adam",
                  metrics=["accuracy"])
```

上述 compile() 函數的損失函數是 categorical_crossentropy，優化器是 adam，metrics 評估標準是 accuracy。

☆ 訓練模型

在編譯模型成 TensorFlow 計算圖後，我們就可以送入特徵資料來訓練模型，如下所示：

```
[9] # 訓練模型
     history = model.fit(X_train, Y_train, validation_split=0.2,
                        epochs=10, batch_size=128, verbose=2)
```

上述 fit() 函數的第 1 個參數是訓練資料集 X_train，第 2 個參數是標籤資料集 Y_train，分割驗證資料 20%，訓練週期是 10 次，批次尺寸是 128，其執行結果如下所示：

```
Epoch 1/10
375/375 - 8s - loss: 0.4006 - accuracy: 0.8765 - val_loss: 0.0809 - val_accuracy: 0.9757 - 8s/epoch - 21ms/step
Epoch 2/10
375/375 - 2s - loss: 0.1353 - accuracy: 0.9593 - val_loss: 0.0622 - val_accuracy: 0.9805 - 2s/epoch - 6ms/step
Epoch 3/10
375/375 - 2s - loss: 0.1030 - accuracy: 0.9688 - val_loss: 0.0508 - val_accuracy: 0.9846 - 2s/epoch - 4ms/step
Epoch 4/10
375/375 - 2s - loss: 0.0856 - accuracy: 0.9736 - val_loss: 0.0441 - val_accuracy: 0.9868 - 2s/epoch - 4ms/step
Epoch 5/10
375/375 - 2s - loss: 0.0742 - accuracy: 0.9771 - val_loss: 0.0383 - val_accuracy: 0.9890 - 2s/epoch - 4ms/step
Epoch 6/10
375/375 - 2s - loss: 0.0676 - accuracy: 0.9789 - val_loss: 0.0368 - val_accuracy: 0.9893 - 2s/epoch - 4ms/step
Epoch 7/10
375/375 - 1s - loss: 0.0637 - accuracy: 0.9807 - val_loss: 0.0402 - val_accuracy: 0.9883 - 1s/epoch - 4ms/step
Epoch 8/10
375/375 - 1s - loss: 0.0574 - accuracy: 0.9828 - val_loss: 0.0350 - val_accuracy: 0.9906 - 1s/epoch - 4ms/step
Epoch 9/10
375/375 - 2s - loss: 0.0516 - accuracy: 0.9840 - val_loss: 0.0328 - val_accuracy: 0.9908 - 2s/epoch - 4ms/step
Epoch 10/10
375/375 - 2s - loss: 0.0518 - accuracy: 0.9840 - val_loss: 0.0347 - val_accuracy: 0.9899 - 2s/epoch - 4ms/step
```

☆ 評估模型

當使用訓練資料集訓練模型後，我們可以分別使用訓練和測試資料集來呼叫 2 次 evaluate() 函數，即可評估模型的效能，如下所示：

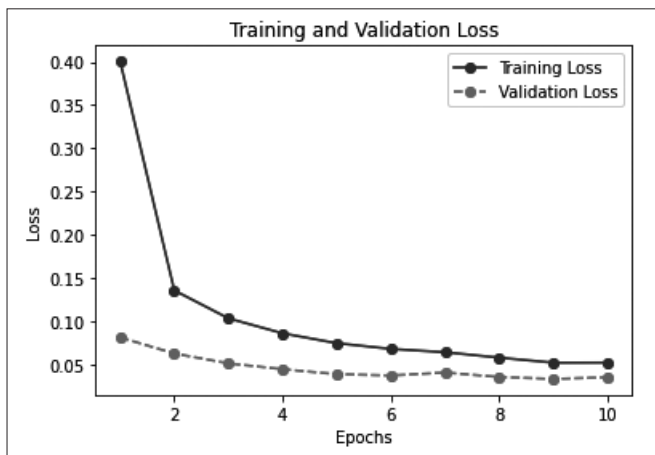
```
[10] # 評估模型
      loss, accuracy = model.evaluate(X_train, Y_train, verbose=0)
      print("訓練資料集的準確度 = {:.2f}".format(accuracy))
      loss, accuracy = model.evaluate(X_test, Y_test, verbose=0)
      print("測試資料集的準確度 = {:.2f}".format(accuracy))
```

```
訓練資料集的準確度 = 0.99
測試資料集的準確度 = 0.99
```


上述訓練資料集和測試資料集的準確度都是 0.99（即 99%）。我們可以繪出訓練和驗證損失的趨勢圖表，幫助我們分析模型效能，如下所示：

```
[11] # 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()
```

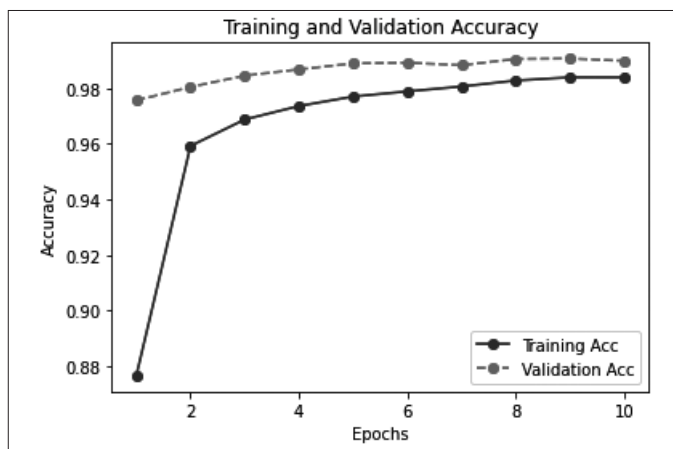
使用 Matplotlib 繪出訓練和驗證損失的圖表，如下圖所示：



從上述圖表可以看出訓練損失和驗證損失都是持續減少。同理，我們可以繪出訓練和驗證準確度的圖表，如下所示：

```
[12] # 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()
```

使用Matplotlib繪出訓練和驗證準確度的圖表，如下圖所示：



從上述圖表可以看出隨著訓練週期的增加，訓練和驗證準確度都是持續的提升。

☆ 預測手寫辨識的數字

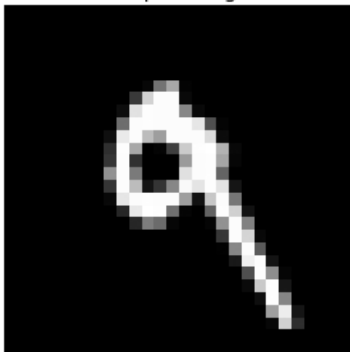
我們準備繪出模型預測手寫數字圖片的預測機率，Python 程式是取出索引值 7 的手寫數字圖片，如下所示：

```
[13] # 載入資料集
      (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
      # 選一個測試的數字圖片
      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
      # 繪出數字圖片
      plt.figure()
      plt.title("Example of Digit:" + str(Y_test[i]))
      plt.imshow(digit, cmap="gray")
      plt.axis("off")
```

上述變數 `digit` 是索引值 7 的手寫數字圖片，然後轉換成 4D 張量 `X_test_digit` 後，執行測試資料正規化，即可繪出索引值 7 的手寫數字圖片 9，如下圖所示：

(-0.5, 27.5, 27.5, -0.5)

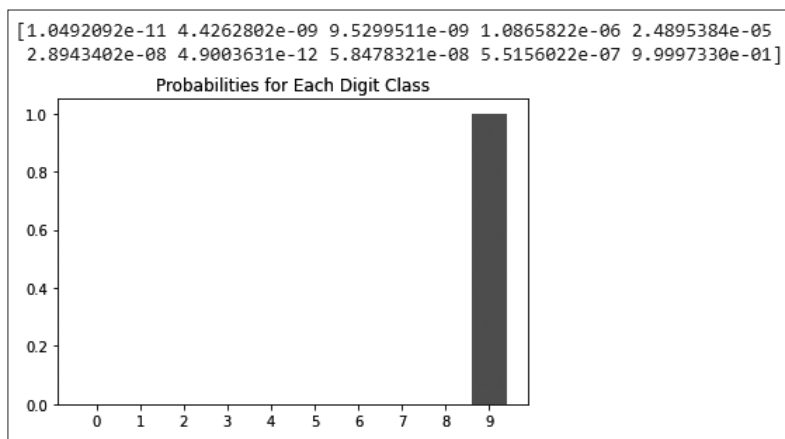
Example of Digit:9



然後呼叫 `predict()` 函數計算 0~9 數字的預測機率，和繪出預測各數字機率的長條圖，如下所示：

```
[14] # 預測結果的機率
probs = model.predict(X_test_digit, batch_size=1)[0]
print(probs)
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()
```

上述程式碼計算出 0~9 數字的機率後，呼叫 `plt.bar()` 函數繪製長條圖，Y 軸是機率 0.0~1.0，X 軸是預測 0~9 數字的機率，因為 `probs` 是二維陣列，所以呼叫 `reshape()` 函數轉換成 10 個元素的一維向量，其執行結果如下圖所示：



上述圖例是預測手寫數字圖片 9 的機率，在機率長條圖可以看出 99% 是預測成數字 9，只有很少機率是預測成其他數字。

★ 學習評量 ★

- ❶ 請簡單說明 Google Colaboratory 雲端服務？
- ❷ 請問如何在 Google Colaboratory 雲端服務使用 GPU/TPU 硬體加速？和存取 Google 雲端硬碟的檔案？
- ❸ 請問在 Google Colaboratory 如何顯示 Matplotlib 圖表？
- ❹ 請使用 anscombe_i.csv 檔案的資料集建立線性迴歸的預測模型，可以使用 x 座標來預測 y 座標。
- ❺ 請問使用 Keras 打造神經網路的基本步驟為何？