智慧化企業整合

花卉辨識
Project 3

指導教授：邱銘傳 教授

110034559 劉兆原

中 華 民 國 一 一 一 年 一 月 七 日

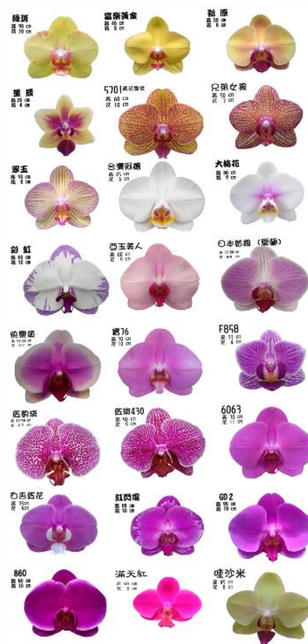
目錄

一、研究背景........1
1.情境說明........1
2.問題描述（5W1H）........2
二、文獻回顧........2
1.卷積神經網路（Convolutional Neural Network，CNN）........2
2.VGG16........3
3.池化層（Pooling Layer）........3
4.全連接層（Fully Connected Layer）........4
三、研究過程........4
1.資料說明........4
2.載入資料........6
3.數據視覺化........6
4.資料前處理........8
四、模型建構_CNN........8
五、模型建構_VGG16........11
六、結論........12

一、研究背景

1.情境說明

世界上花卉之數量屬不勝數，當農民欲選擇有經濟價值之花卉品種或種類時，若遇上外型、顏色相近之花卉，便會出現難以辨識之問題。以蘭花為例，世界上的蘭花約有 870 個屬、28000 個種，除上述品種之外另有約 100,000 種園藝家培養出之交配種與變種。其中，中國有 171 屬 1247 種以及許多亞種、變種和變型。臺灣有 104 屬 478 種。所有的野生蘭科植物均被列為《野生動植物瀕危物種國際貿易公約》(CITES)的保護範圍。它與菊科是開花植物的兩個最大的科，但因成員數量不斷變化而無法確定哪個科更大。蘭花的物種數量幾乎與真骨魚類相等，是鳥類的兩倍多，更是哺乳動物的四倍，大約占了所有種子植物物種總數的約 6-11%。最大的屬是石豆蕙蘭屬（Bulbophyllum）（約 2000 種），然後依序為樹蕙蘭屬（Epidendrum）（約 1500 種），石斛蘭屬（Dendrobium）（約 1400 種）和腋花蕙蘭屬（Pleurothallis）（約 1000 種）。而各品種蘭花之圖片比較以下圖所示：



光是以蘭花舉例，世界上就有數以十萬計之蘭花品種，而這世界上具經濟價值之花卉種類不非只有蘭花，若能發展一套花卉辨識程式，必能解決農民辨識花卉困難之問題。因此本研究發展一套以深度學習為基礎之花卉辨識程式，期以解決上述農民辨識花卉困難之問題。

2.問題描述（5W1H）

為釐清農民辨識花卉困難之問題，本研究以 5W1H 思考，進而深入了解此問題之目的、時間、人員、地點、事件、方法。透過 5W1H 分析，本研究了解到此問題之目的為節省人力辨識之時間與精力；而此問題之時間為農民辨識花卉種類時；而此問題之人員農民；而此問題之地點為花卉種植場地；而此問題之事件為農民辨識花卉種類費時費力；而此問題之方法為運用 CNN 與 VGG16 訓練。本研究將 5W1H 分析內容整理如下表所示：

項目	內容
WHAT	農民辨識花卉種類費時費力
WHERE	花卉種植場地
WHO	農民
WHEN	農民辨識花卉種類時
WHY	節省人力辨識之時間與精力
HOW	以 CNN 與 VGG16 訓練

二、文獻回顧

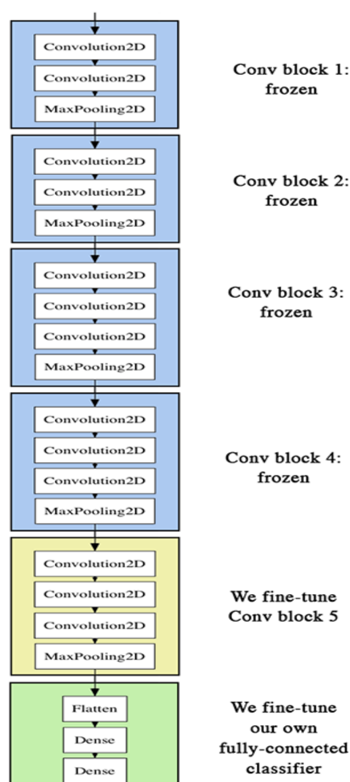
1.卷積神經網路（Convolutional Neural Network，CNN）

卷積神經網路（Convolutional Neural Network, CNN）是一種前饋神經網路，它的人工神經元可以回應一部分覆蓋範圍內的周圍單元，對於大型圖像處理有出色表現。卷積神經網路由一個或多個卷積層和頂端的全連通層（對應經典的神經

網路）組成，同時也包括關聯權重和池化層（pooling layer）。這一結構使得卷積神經網路能夠利用輸入資料的二維結構。與其他深度學習結構相比，卷積神經網路在圖像和語音辨識方面能夠給出更好的結果。這一模型也可以使用反向傳播演算法進行訓練。相比較其他深度、前饋神經網路，卷積神經網路需要考量的參數更少，使之成為一種頗具吸引力的深度學習結構。

2.VGG16

VGG 是英國牛津大學 Visual Geometry Group 的縮寫，主要貢獻是使用更多的隱藏層，大量的圖片訓練，提高準確率至 90%。VGG16/VGG19 分別為 16 層(13 個卷積層及 3 個全連接層)與 19 層(16 個卷積層及 3 個全連接層)，結構圖如下圖所示：



3.池化層（Pooling Layer）

池化（Pooling）是一種非線性形式的降採樣，有多種不同形式的非線性池化

函式，而其中「最大池化（Max pooling）」是最為常見的。它是將輸入的圖像劃分為若干個矩形區域，對每個子區域輸出最大值。

直覺上，這種機制能夠有效地原因在於一個特徵的精確位置遠不及它相對於其他特徵的粗略位置重要。池化層會不斷地減小資料的空間大小，因此參數的數量和計算量也會下降，這在一定程度上也控制了過擬合。一般來說，CNN 網路結構中的卷積層之間都會周期性地插入池化層，而池化操作提供了另一種形式的平移不變性。由於卷積核是一種特徵發現器，我們透過卷積層可以很容易地發現圖像中的各種邊緣。然而，卷積層發現的特徵往往過於精確，我們即使高速連拍拍攝一個物體，相片中的物體的邊緣像素位置也不大可能完全一致，而透過池化層我們可以降低卷積層對邊緣的敏感性。

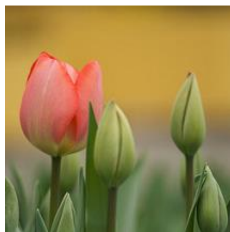
4.全連接層（Fully Connected Layer）

在經過幾個卷積層和最大池化層之後，神經網路中最後的進階推理透過完全連接層來完成。就和常規的非卷積人工神經網路中一樣，完全連接層中的神經元與前一層中的所有啟用都有聯絡。因此，它們的啟用可以作為仿射變換來計算，也就是先乘以一個矩陣然後加上一個偏差（bias）偏移量（向量加上一個固定的或者學習來的偏差量）。

三、研究過程

1.資料說明

本研究從 Kaggle 下載一含五種花卉照片之資料集，包含雛菊、蒲公英、玫瑰、向日葵、鬱金香，其中，雛菊有 764 張、蒲公英有 1052 張、玫瑰有 784 張、向日葵有 733 張、鬱金香有 984 張，而各花卉之範例圖片以下圖所示：



2.載入資料

於程式中，起初本研究於 colab 載入資料集，並載入所有需要之套件，含 CNN、VGG16 之套件，並讓程式讀取 colab 中之資料集。其中程式初始各階段之程式碼以下圖所示：

```
[1] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

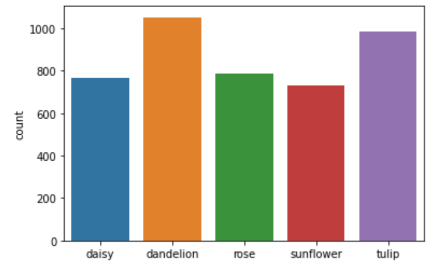
```
[ ] # import all required libraries for reading, analysing and visualizing data
    import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from tensorflow import keras

    import os
    import random
    import cv2
    from tqdm import tqdm

    import matplotlib.pyplot as plt
    import PIL
    import tensorflow as tf
    import numpy as np
    import os

    from tensorflow.keras.models import Model, Sequential
    from tensorflow.keras.layers import Dense, Flatten, Dropout
    from tensorflow.keras.applications import VGG16
    from tensorflow.keras.applications.vgg16 import preprocess_input, decode_predictions
    from tensorflow.keras.preprocessing.image import ImageDataGenerator
    from tensorflow.keras.optimizers import Adam, RMSprop
```

3.數據視覺化

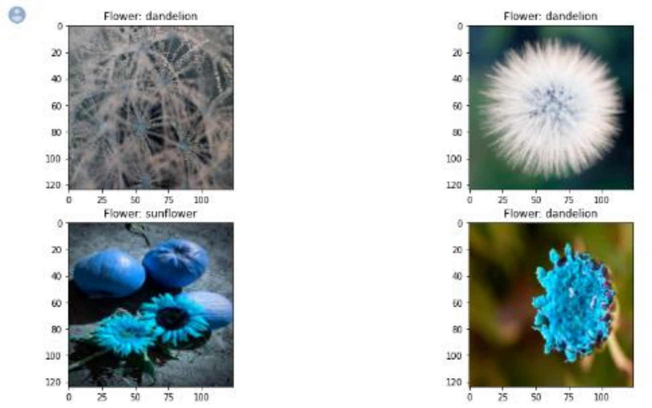
之後，為了解資料集各花卉種類之狀態，設計一程式碼以顯示各花卉品種之資料數量。再來，為確認是否正確讀取指定之花卉資料集，又設計一程式以隨機印出 10 張花卉照片，如下圖所示。

```
l = []
for img in Y:
        l.append(labels[img])
sns.countplot(l);
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43
  FutureWarning
```




```
# any 10 random images
fig, ax = plt.subplots(5, 2)
fig.set_size_inches(15, 20)
for i in range(5):
        for j in range(2):
                l = random.randint(0, len(X))
                ax[i, j].imshow(X[l])
                ax[i, j].set_title('Flower: ' + labels[Y[l]])
```



4.資料前處理

於資料前處理部分，設計程式以標準化圖片，使圖片大小由 0~255 縮小至 0~1；也設計程式做 one-hot encoding，以正規化資料。標準化圖片之程式碼以圖十三所示；而正規化資料之程式碼以下圖所示。

```
[ ] # normalize the data
    X = X / 255 #標準化，[0-255]調整至[0-1]

    # reshape the data
    X = X.reshape(-1, IMG_SIZE, IMG_SIZE, 3)
    Y = Y.reshape(-1, 1)
    Y = keras.utils.to_categorical(Y, 5) #正規化，one-hot encoding
```

其中，將資料集分割為 80%訓練集、20%測試集，以下圖所示：

```
[ ] X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2) #資料分割（訓練0.8，測試0.2)

    # Explore the dataset
    print("X_train shape:" + str(X_train.shape))
    print("Y_train shape:" + str(Y_train.shape))
    print("X_test shape:" + str(X_test.shape))
    print("Y_test shape:" + str(Y_test.shape))

    X_train shape:(3453, 124, 124, 3)
    Y_train shape:(3453, 5)
    X_test shape:(864, 124, 124, 3)
    Y_test shape:(864, 5)
```

四、模型建構_CNN

CNN 模型建構部分，設計 epochs 為 8 次、優化器為 adam，各段程式碼以下圖所示：

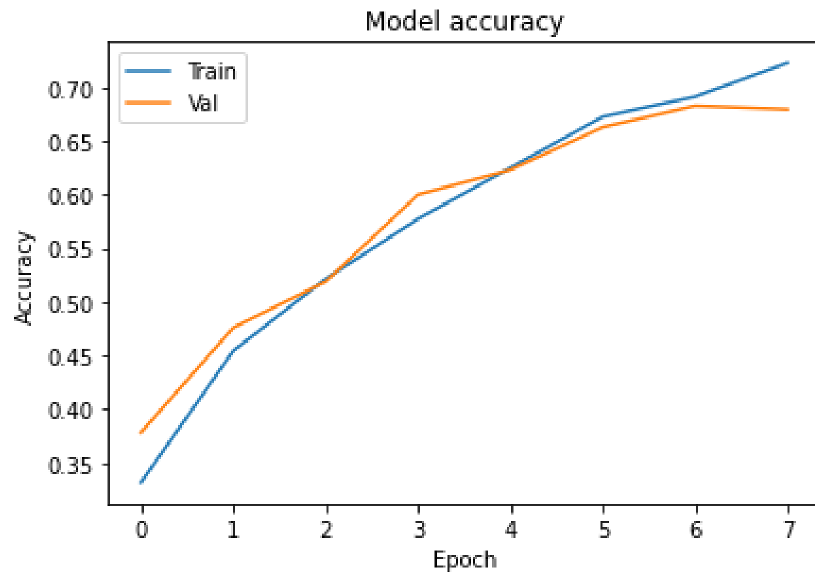
```
[12] from keras.models import Sequential
     from keras.layers import Conv2D, MaxPool2D, Dropout, Flatten
     from keras.layers import Dense
```

```
[13] model = Sequential(
         [
             Conv2D(filters = 32, kernel_size = (3, 3), padding = 'same', activation = 'relu', input_shape = (IMG_SIZE, IMG_SIZE, 3)),
             MaxPool2D(pool_size = (2, 2), strides = (2, 2)),

             Conv2D(filters = 64, kernel_size = (3, 3), padding = 'same', activation = 'relu'),
             MaxPool2D(pool_size = (2, 2), strides = (2, 2)),

             Conv2D(filters = 128, kernel_size = (3, 3), padding = 'same', activation = 'relu'),
             MaxPool2D(pool_size = (2, 2), strides = (2, 2)),

             Conv2D(filters = 256, kernel_size = (3, 3), padding = 'same', activation = 'relu'),
             MaxPool2D(pool_size = (2, 2), strides = (2, 2)),

             Conv2D(filters = 512, kernel_size = (3, 3), padding = 'same', activation = 'relu'),
             MaxPool2D(pool_size = (2, 2), strides = (2, 2)),

             Flatten(),
             Dense(1024, activation = 'relu'),
             Dropout(0.5),
             Dense(5, activation = 'softmax')
         ]
    )
```

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

```
[17] history = model.fit(X_train, Y_train, epochs = 8, validation_split = 0.2)

    Epoch 1/8
    76/76 [==============================] - 109s 1s/step - loss: 1.4515 - accuracy: 0.3320 - val_loss: 1.3124 - val_accuracy: 0.3785
    Epoch 2/8
    76/76 [==============================] - 107s 1s/step - loss: 1.2146 - accuracy: 0.4545 - val_loss: 1.1697 - val_accuracy: 0.4760
    Epoch 3/8
    76/76 [==============================] - 107s 1s/step - loss: 1.1239 - accuracy: 0.5215 - val_loss: 1.0780 - val_accuracy: 0.5190
    Epoch 4/8
    76/76 [==============================] - 111s 1s/step - loss: 1.0338 - accuracy: 0.5774 - val_loss: 0.9947 - val_accuracy: 0.6000
    Epoch 5/8
    76/76 [==============================] - 110s 1s/step - loss: 0.9540 - accuracy: 0.6250 - val_loss: 0.9459 - val_accuracy: 0.6231
    Epoch 6/8
    76/76 [==============================] - 112s 1s/step - loss: 0.8708 - accuracy: 0.6726 - val_loss: 0.8757 - val_accuracy: 0.6628
    Epoch 7/8
    76/76 [==============================] - 110s 1s/step - loss: 0.8154 - accuracy: 0.6912 - val_loss: 0.8312 - val_accuracy: 0.6826
    Epoch 8/8
    76/76 [==============================] - 110s 1s/step - loss: 0.7352 - accuracy: 0.7227 - val_loss: 0.8044 - val_accuracy: 0.6793
```

其中，訓練、測試、驗證之準確率以下圖所示：




```
[28] # find the accuracy on test set
     test_loss, test_acc = model.evaluate(X_test, Y_test)
     print("Accuracy on test set is %f" %(test_acc * 100) + "%")
```

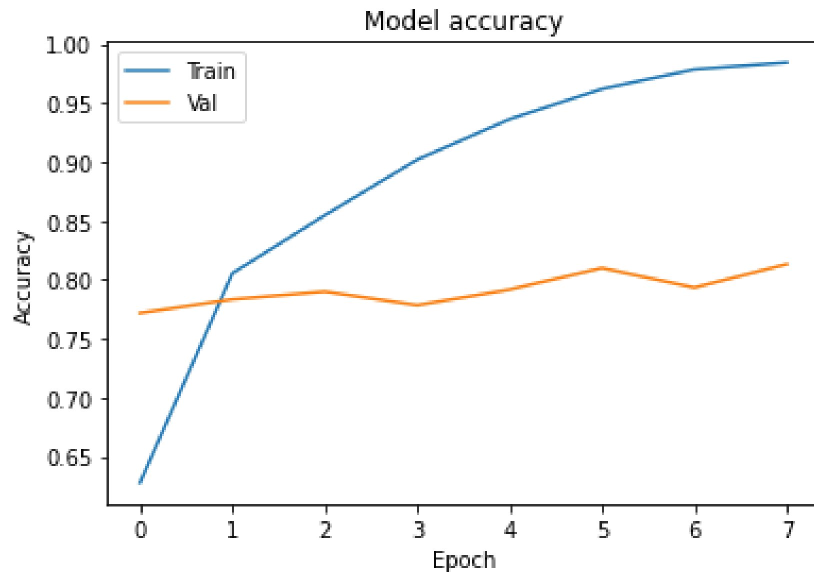
```
41/41 [==============================] - 180s 4s/step - loss: 0.8159 - accuracy: 0.8210
Accuracy on test set is 82.098764%
```

五、模型建構_VGG16

VGG16 模型建構部分，設計皆與 CNN 相同，設計 epochs 為 8 次、優化器為 adam，各段程式碼以下圖所示：

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

[26] history = model.fit(X_train, Y_train, epochs = 8, validation_split = 0.2)

Epoch 1/8
76/76 [==============================] - 459s 6s/step - loss: 0.9684 - accuracy: 0.6279 - val_loss: 0.6630 - val_accuracy: 0.7719
Epoch 2/8
76/76 [==============================] - 461s 6s/step - loss: 0.5328 - accuracy: 0.8055 - val_loss: 0.5755 - val_accuracy: 0.7835
Epoch 3/8
76/76 [==============================] - 457s 6s/step - loss: 0.3877 - accuracy: 0.8547 - val_loss: 0.5896 - val_accuracy: 0.7901
Epoch 4/8
76/76 [==============================] - 461s 6s/step - loss: 0.2590 - accuracy: 0.9019 - val_loss: 0.6776 - val_accuracy: 0.7785
Epoch 5/8
76/76 [==============================] - 472s 6s/step - loss: 0.1870 - accuracy: 0.9363 - val_loss: 0.7518 - val_accuracy: 0.7917
Epoch 6/8
76/76 [==============================] - 458s 6s/step - loss: 0.1180 - accuracy: 0.9619 - val_loss: 0.8933 - val_accuracy: 0.8099
Epoch 7/8
76/76 [==============================] - 458s 6s/step - loss: 0.0724 - accuracy: 0.9785 - val_loss: 0.8523 - val_accuracy: 0.7934
Epoch 8/8
76/76 [==============================] - 465s 6s/step - loss: 0.0588 - accuracy: 0.9843 - val_loss: 0.9178 - val_accuracy: 0.8132
```

其中，訓練、測試、驗證之準確率以下圖所示：



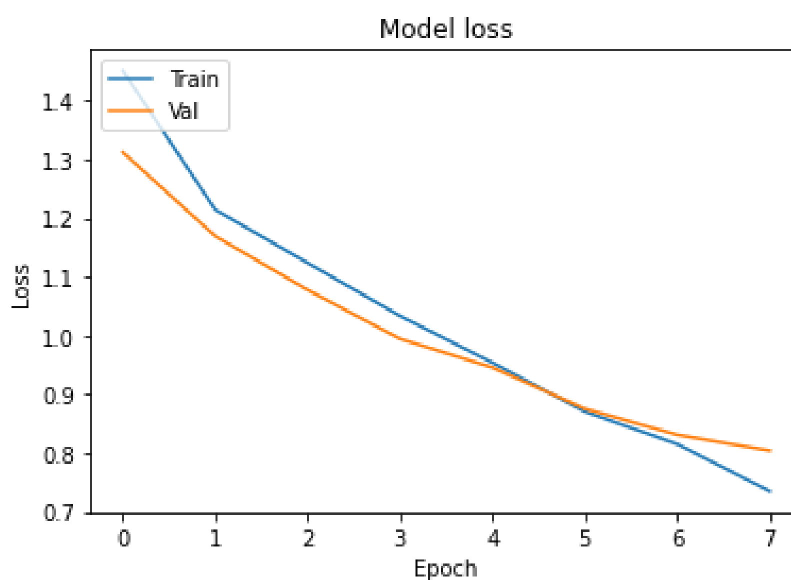
```
[29] # find the accuracy on test set
     test_loss, test_acc = model.evaluate(X_test, Y_test)
     print("Accuracy on test set is %f" %(test_acc * 100) + "%")

     41/41 [==============================] - 180s 4s/step - loss: 0.8159 - accuracy: 0.8210
     Accuracy on test set is 82.098764%
```

六、結論

本研究經兩種模型之訓練後發現，一般 CNN 模型之訓練準確率與 VGG16 模型之訓練準確率皆為 82%，可見於目前相同參數設定下，VGG16 與 CNN 辨識能力無明顯差異。

CNN、VGG16 之 loss 圖以下圖所示，從兩模型之 loss 圖比較中可發現兩模型皆有 overfitting 之狀況，且 VGG16 之 overfitting 之狀況更為嚴重。本研究認為與兩因素有關，一為本研究將兩模型之參數設置相同，或許個別模型置換其他參數便可降低各模型 overfitting 之狀況；二為兩模型之 epochs 之次數皆為 8 次稍嫌不足，若能提升 epochs 次數或許 overfitting 之狀況便會下降。

CNN：



VGG16：