農作物葉片健康或病態識別分類

林呈昱

國立清華大學工業工程與工程管理學系
qaz7416565@gmail.com

摘要

好的農作物收成是農民所樂見的，而對於農作物的健康狀態與否，其農作物上的葉片常常能反映出一些資訊給農民，以讓農民可以察知而及時做出相應措施，但現今由於觀光農場的普及，或許觀光的民眾也會想了解所觀察到的農作物狀況，但由於普遍大眾對農作物不甚了解，所以說如果能藉由圖像辨識並作分類，或許能在參觀之餘透過此技術辨別一些葉片的情況，相信對於參觀民眾會有所收穫，而本次資料集由 kaggle 所提供，其農作物照片共 9 萬多張，包含了 38 種類的健康和病態農作物照片，藉由建構 CNN 模型，並進行多次的訓練及嘗試修改模型以獲致更好的測試準確率，最終的模型在測試集上的準確率來到了 99.03%，相信對於未來由此資料集來源地，基於這些農作物所建構出的觀光農場，能為其觀光客帶來一些幫助。

關鍵字（3 至 5 個字）：Deep Learning、Convolutional Neural Networks、Crop

1　緒論

近年來，農業的轉型進而成立了許多的觀光農場帶動了旅遊產業，成為觀光客嘉日休閒或放鬆的好去處，而本次的研究專案，資料集為從 kaggle 網站獲取農作物的照片共 9 萬多張，其中包含了 38 種類的健康和病態農作物照片，農作物的健康狀態與否，其農作物上的葉片常常能反映出一些資訊給農民，以讓農民可以察知而及時做出相應措施，而假設今日實際將這些資料集之農作物一同栽作在一起，並建構出一個假想的觀光農場，由於普遍大眾對農作物不甚了解，所以說如果能藉由圖像辨識並作分類，或許能在參觀之餘透過此技術辨別一些葉片的情況，相信對於參觀民眾會有所收穫。

而本章分別利用 5W1H 分析此專案及闡述其預期之成果。

1.1　5W1H

- Why

農作物的健康狀態與否，其農作物上的葉片常常能反映出一些資訊給農民，

以讓農民可以察知而及時做出相應措施，但由於普遍大眾對農作物不甚了解，所以觀光客也不易得知其有興趣之農作物之狀態。

- What
  我們想建立一個辨識農作物葉片的系統，讓觀光客能透過這個系統理解他們在觀光時有興趣之農作物的狀態。
- Where
  本專案所假想之觀光農場內，或擁有相同農作物的場合中。
- When
  觀光客對觀光農場內某些農作物有興趣了解時。
- Who
  舉凡對農作物狀態有興趣或需了解其狀態者，可以是農民、此觀光農場之觀光客或管理者等等。
- How
  收集大量農作物之葉片照片，利用卷積神經網路訓練這些照片，以判斷觀光客有興趣之農作物的狀態，得知其是否健康或者屬於何種病態類別的知識。

1.2 預期成果

希望能透過所建構並訓練出的優良 CNN 神經網路模型，幫助在此觀光農場的觀光客，在體驗及觀察農作物的同時，能辨別其所觀察到的農作物健康與否或者是為何種病態，進而獲取一些農作物上的知識。

2 文獻回顧

本章節探討了農作物葉片重要性的相關資料，及本次使用之研究方法 Convolutional Neural Networks , CNN。

2.1 卷積神經網路（Convolutional Neural Networks，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。

一般在進行圖片辨識時，我們不會單純使用 ANN 或 DNN 這種全連接層的神經網路模型架構，因為這樣子做會有幾個問題：

(1) 在一般的圖片辨識問題中，事實上會有一些 pattern 可能會出現在圖片中的某個部位，且這樣的 pattern 可能由許多個鄰近像素構成，如果單純直接操作全連接層的神經網路模型架構會破壞這樣的 pattern 結構。

(2) 全連接層搭配高像素的圖片，會讓整個計算成本大幅增加。

基於上面幾個理由便衍伸出 Convolutional Neural Network (CNN) 卷積神經網路來進行圖像辨識。

整個 CNN 結構主要分成幾個部分 : 卷積層 (Convolution layer)、池化層 (Pooling layer) 以及最後一個全連接層 (Fully Connected layer)。

卷積層主要是由許多不同的 kernel 在輸入圖片上進行卷積運算，而卷積其實就是兩個步驟組成的運算 : 滑動+內積，利用 filter 在輸入圖片上滑動並且持續進行矩陣內積，卷積後得到的圖片我們稱之為 feature map。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。

2.2 葉片的重要性

葉是高等植物的營養器官，側邊發育自植物的莖的葉原基。葉內含有葉綠體，是植物進行光合作用的主要場所，同時，植物的蒸散作用是通過葉的氣孔實現的，而葉的形態也是多種多樣的，從非常原始的針狀小型葉發展出各種各樣形態的大型葉。有些葉，已不再行使葉的功能（光合作用和蒸騰作用），而成為花瓣，花刺，葉卷鬚和保護幼葉的牙鱗。

葉片，是作物進行光合作用的主要器官，葉綠體是進行光合作用的細胞器，而葉綠素是葉綠體中的一個色素，它受光照強度的影響。在光照不足的情況下不利於葉綠素的生物合成，作物葉片缺乏葉綠素時，可見黃化現象。所以，在我們栽種農作物密度過大時，遮擋嚴重受光不足，常會引起葉片變黃。這對指導我們種植，和判斷作物的缺素，有很重要的意義。

所以說，隨時觀測葉片，識別葉片的狀態對於農民來說是非常重要的，能否友好的農作物收成，其過程中的葉片健康或病態反映給農民的資訊，即能及時提供給農民做相應的措施，而了解有關於葉片健康或病態的資訊，相信對於觀光農場中觀光客來說，無疑也是一種收穫。

3 研究架構或方法

本次研究其整體執行流程大致可分成以下六個步驟：

(1) 資料收集

(2) 資料前處理

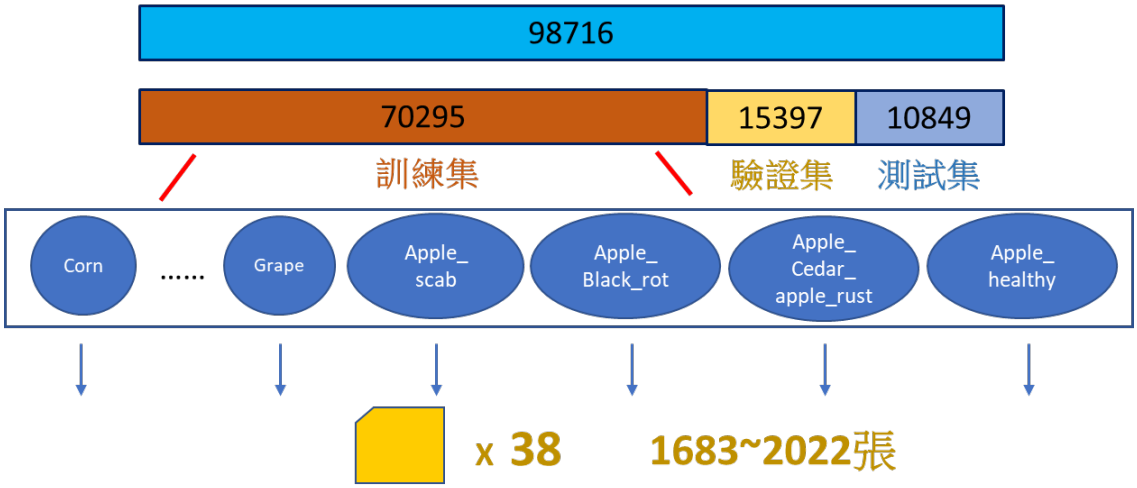
(3) 模型設定
(4) 訓練集訓練
(5) 測試集測試
(6) 輸出結果

其中在第三步會不斷嘗試不同的模型設定，以訓練出一個具有一定程度準確率的模型，以提供具有可信力的系統給觀光客。

3.1 資料前處理(Data-preprocessing)

(1) 資料集分類及劃分

由 Kaggle 公開數據集中取得農作物葉片圖像資料集，
包含了 13 種不同的農作物，其中每種作物的狀態又可分為健康及幾種可能的病態性質，因此每種作物又可分為若干類別，而此 13 種作物共包含了 38 種類別, 依類別分別分成 29 個資料夾。
(總共 98,716 張照片
訓練集 70295 張
驗證集 15397 張
測試集 10849 張)



(2) ImageDataGenerator

使用 ImageDataGenerator 生成更多樣的圖像

(1) rescale=1./255 :

將圖像做歸一化，使其像素值從 [0-255] 縮放到 [0-1]

(2)width_shift_range=0.2

圖片寬度的某個比例，生成更多圖像時圖片水平偏移的幅度

(3)height_shift_range=0.2,

圖片高度的某個比例，生成更多圖像時圖片垂直變換的幅度

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   fill_mode='nearest')
```

(3) flow_from_directory

使用 flow_from_directory 處理每張圖像

(1) target_size=(224, 224)

所有的圖像將被調整到 224*224 的尺寸

(2) color_mode='rgb'

圖像轉換成 1 或 3 個顏色通道

```
103 training_set = train_datagen.flow_from_directory(train_dir,
104                                                  target_size=(224, 224),
105                                                  color_mode='rgb',
106                                                  classes=None,
107                                                  batch_size=batch_size,
108                                                  class_mode='categorical')
109
110 valid_set = valid_datagen.flow_from_directory(valid_dir,
111                                                target_size=(224, 224),
112                                                color_mode='rgb',
113                                                classes=None,
114                                                batch_size=batch_size,
115                                                class_mode='categorical')
```

(4) Labeling & one-hot encoding\

讀取各個資料夾，為同類別資料夾內的圖片進行 labeling，其值為 0-37 共 38 種，並進行 2D one-hot encoding

(1) classes=None

(2) class_mode='categorical'

(3) class_indices

```
103 training_set = train_datagen.flow_from_directory(train_dir,
104                                                  target_size=(224, 224),
105                                                  color_mode='rgb',
106                                                  classes=None,
107                                                  batch_size=batch_size,
108                                                  class_mode='categorical')
109
110 valid_set = valid_datagen.flow_from_directory(valid_dir,
111                                                target_size=(224, 224),
112                                                color_mode='rgb',
113                                                classes=None,
114                                                batch_size=batch_size,
115                                                class_mode='categorical')
```

```
In [27]: class_dict = training_set.class_indices
    ...: print(class_dict)
{'Apple___Apple_scab': 0, 'Apple___Black_rot': 1,
'Apple___Cedar_apple_rust': 2, 'Apple___healthy': 3, 'Blueberry___healthy':
4, 'Cherry_(including_sour)___Powdery_mildew': 5,
'Cherry_(including_sour)___healthy': 6, 'Corn_(maize)___Cercospora_leaf_spot
Gray_leaf_spot': 7, 'Corn_(maize)___Common_rust_': 8,
'Corn_(maize)___Northern_Leaf_Blight': 9, 'Corn_(maize)___healthy': 10,
'Grape___Black_rot': 11, 'Grape___Esca_(Black_Measles)': 12,
'Grape___Leaf_blight_(Isariopsis_Leaf_Spot)': 13, 'Grape___healthy': 14,
'Orange___Haunglongbing_(Citrus_greening)': 15, 'Peach___Bacterial_spot':
16, 'Peach___healthy': 17, 'Pepper,_bell___Bacterial_spot': 18,
'Pepper,_bell___healthy': 19, 'Potato___Early_blight': 20,
'Potato___Late_blight': 21, 'Potato___healthy': 22, 'Raspberry___healthy':
23, 'Soybean___healthy': 24, 'Squash___Powdery_mildew': 25,
'Strawberry___Leaf_scorch': 26, 'Strawberry___healthy': 27,
'Tomato___Bacterial_spot': 28, 'Tomato___Early_blight': 29,
'Tomato___Late_blight': 30, 'Tomato___Leaf_Mold': 31,
'Tomato___Septoria_leaf_spot': 32, 'Tomato___Spider_mites Two-
spotted_spider_mite': 33, 'Tomato___Target_Spot': 34,
'Tomato___Tomato_Yellow_Leaf_Curl_Virus': 35,
'Tomato___Tomato_mosaic_virus': 36, 'Tomato___healthy': 37}
```

3.2 初始模型設定

初始模型(Model 1)設定包含輸入層在那，總共 24 層，卷積與池化層用綠底表示，而全連接層用藍底來表示。

而 Model1 的設置如下:

	Loss	激活函數	最後一層全連接層激活函數	優化器	Epoch
Model 1	categorical	relu	softmax	SGD	10

```python
# Initializing the CNN
classifier = Sequential()

# Convolution Step 1
classifier.add(Convolution2D(96, 11, strides = (4, 4), padding = 'valid', input_shape=(224, 224, 3), activation = 'relu'))
# Max Pooling Step 1
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
classifier.add(BatchNormalization())
# Convolution Step 2
classifier.add(Convolution2D(256, 11, strides = (1, 1), padding='valid', activation = 'relu'))
# Max Pooling Step 2
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding='valid'))
classifier.add(BatchNormalization())
# Convolution Step 3
classifier.add(Convolution2D(384, 3, strides = (1, 1), padding='valid', activation = 'relu'))
classifier.add(BatchNormalization())
# Convolution Step 4
classifier.add(Convolution2D(384, 3, strides = (1, 1), padding='valid', activation = 'relu'))
classifier.add(BatchNormalization())
# Convolution Step 5
classifier.add(Convolution2D(256, 3, strides=(1,1), padding='valid', activation = 'relu'))
# Max Pooling Step 3
classifier.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
classifier.add(BatchNormalization())

# Flattening Step
classifier.add(Flatten())

# Full Connection Step
classifier.add(Dense(units = 4096, activation = 'relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 4096, activation = 'relu'))
classifier.add(Dropout(0.4))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 1000, activation = 'relu'))
classifier.add(Dropout(0.2))
classifier.add(BatchNormalization())
classifier.add(Dense(units = 38, activation = 'softmax'))

classifier.summary()
```

```python
# Compiling the Model
from keras import optimizers
classifier.compile(optimizer= 'adam',
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

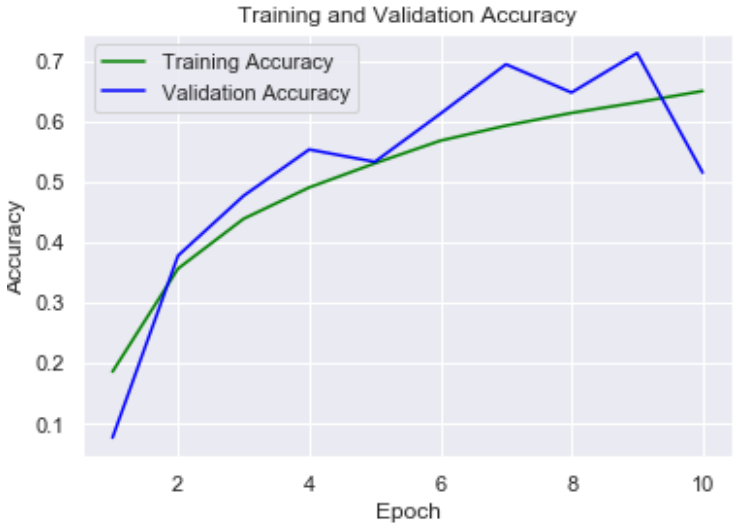
```python
# Compiling the Model
from keras import optimizers
classifier.compile(optimizer=optimizers.SGD(lr=0.001, momentum=0.9, decay=0.005),
                   loss='categorical_crossentropy',
                   metrics=['accuracy'])
```

4 模型結果與調整

4.1 初始模型結果

初始模型的結果，經過了 10 次迭代之後，測試集準確率為 43.91%。由圖可看出訓練集集、驗證集皆保有上升趨勢，因此保存 Model 1 以提供給 Model 2 繼續訓練。

	Loss	激活函數	最後一層全連接層激活函數	優化器	Epoch
Model 1	categorical	relu	softmax	SGD	10




```
361/361 [==============================] - 926s 3s/step
Test_Accuracy: 43.91%
```

```
155 #saving model
156 filepath="model_1"
157 classifier.save(filepath)
```

4.2 模型調整及優化

4.2.1 Model 2

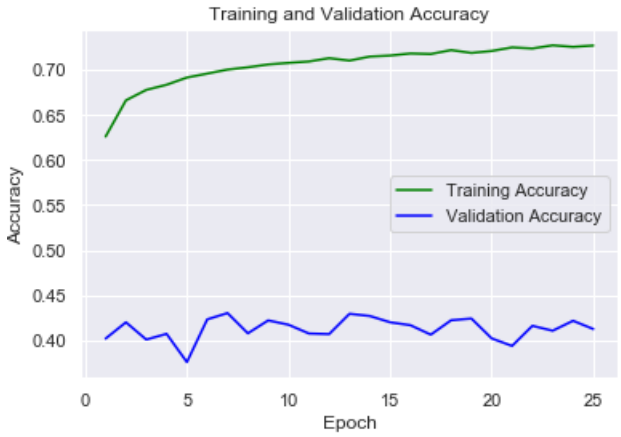
將 model 1 的模型權重讀進去，並繼續訓練，一樣再次保存模型給下次模型使用，最後測試集準確率為 60.10%。

```
62 classifier.load_weights('C:/Users/qaz74/Desktop/project3/model_1')
```

	Loss	激活函數	最後一層全連接層激活函數	優化器	Epoch
Model 2	categorical	relu	softmax	SGD	25

```
Epoch 1/25
274/274 [==============================] - 1852s 7s/step - loss: 1.2402 -
accuracy: 0.6259 - val_loss: 2.9623 - val_accuracy: 0.4018
Epoch 2/25
274/274 [==============================] - 1749s 6s/step - loss: 1.0697 -
accuracy: 0.6661 - val_loss: 3.2971 - val_accuracy: 0.4199
Epoch 3/25
274/274 [==============================] - 1786s 7s/step - loss: 1.0175 -
accuracy: 0.6777 - val_loss: 2.6515 - val_accuracy: 0.4007
Epoch 4/25
274/274 [==============================] - 1826s 7s/step - loss: 0.9956 -
accuracy: 0.6833 - val_loss: 3.1368 - val_accuracy: 0.4071
Epoch 5/25
274/274 [==============================] - 1778s 6s/step - loss: 0.9698 -
accuracy: 0.6914 - val_loss: 3.0172 - val_accuracy: 0.3757
Epoch 6/25
274/274 [==============================] - 1797s 7s/step - loss: 0.9508 -
accuracy: 0.6957 - val_loss: 2.5361 - val_accuracy: 0.4231
Epoch 7/25
274/274 [==============================] - 1803s 7s/step - loss: 0.9333 -
accuracy: 0.7002 - val_loss: 2.7280 - val_accuracy: 0.4301
Epoch 8/25
274/274 [==============================] - 1741s 6s/step - loss: 0.9267 -
accuracy: 0.7027 - val_loss: 2.7332 - val_accuracy: 0.4077

accuracy: 0.7175 - val_loss: 3.1355 - val_accuracy: 0.4062
Epoch 18/25
274/274 [==============================] - 1664s 6s/step - loss: 0.8639 -
accuracy: 0.7218 - val_loss: 2.7810 - val_accuracy: 0.4222
Epoch 19/25
274/274 [==============================] - 1649s 6s/step - loss: 0.8677 -
accuracy: 0.7187 - val_loss: 2.7626 - val_accuracy: 0.4241
Epoch 20/25
274/274 [==============================] - 1623s 6s/step - loss: 0.8618 -
accuracy: 0.7208 - val_loss: 3.0297 - val_accuracy: 0.4020
Epoch 21/25
274/274 [==============================] - 1689s 6s/step - loss: 0.8533 -
accuracy: 0.7248 - val_loss: 3.1380 - val_accuracy: 0.3937
Epoch 22/25
274/274 [==============================] - 1647s 6s/step - loss: 0.8543 -
accuracy: 0.7236 - val_loss: 2.5496 - val_accuracy: 0.4160
Epoch 23/25
274/274 [==============================] - 1641s 6s/step - loss: 0.8393 -
accuracy: 0.7271 - val_loss: 3.3058 - val_accuracy: 0.4105
Epoch 24/25
274/274 [==============================] - 1674s 6s/step - loss: 0.8476 -
accuracy: 0.7254 - val_loss: 2.9179 - val_accuracy: 0.4216
Epoch 25/25
274/274 [==============================] - 1651s 6s/step - loss: 0.8400 -
accuracy: 0.7269 - val_loss: 3.0704 - val_accuracy: 0.4124
```




```
361/361 [==============================] - 926s 3s/step
Test_Accuracy: 69.10%
```

```
155 #saving model
156 filepath="model_2"
157 classifier.save(filepath)
```

4.2.2 Model 3

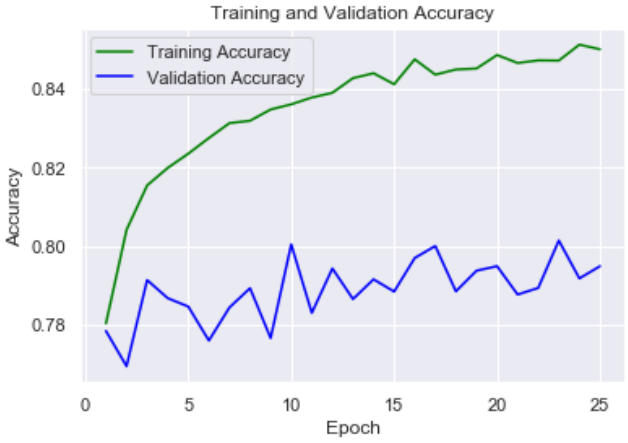
由 Model2 可看出模型開始慢慢在收斂，因此嘗試調整優化器為 adam 尋求更好的解，將 model 2 的模型權重讀進去，並繼續訓練，最後測試集準確率為 80.50%。

```
62 classifier.load_weights('C:/Users/qaz74/Desktop/project3/model_2')
```

	Loss	激活函數	最後一層全連接層激活函數	優化器	Epoch
Model 3	categorical	relu	softmax	adam	25

```
Epoch 1/25
274/274 [==============================] - 2018s 7s/step - loss: 0.7015 - accuracy: 0.7804 -
val_loss: 0.9415 - val_accuracy: 0.7784
Epoch 2/25
274/274 [==============================] - 2017s 7s/step - loss: 0.6058 - accuracy: 0.8042 -
val_loss: 0.7185 - val_accuracy: 0.7695
Epoch 3/25
274/274 [==============================] - 2041s 7s/step - loss: 0.5739 - accuracy: 0.8155 -
val_loss: 0.7353 - val_accuracy: 0.7914
Epoch 4/25
274/274 [==============================] - 2028s 7s/step - loss: 0.5519 - accuracy: 0.8199 -
val_loss: 0.6906 - val_accuracy: 0.7868
Epoch 5/25
274/274 [==============================] - 2019s 7s/step - loss: 0.5374 - accuracy: 0.8236 -
val_loss: 0.6165 - val_accuracy: 0.7847
Epoch 6/25
274/274 [==============================] - 2019s 7s/step - loss: 0.5288 - accuracy: 0.8276 -
val_loss: 0.7519 - val_accuracy: 0.7760
Epoch 7/25
274/274 [==============================] - 2024s 7s/step - loss: 0.5140 - accuracy: 0.8314 -
val_loss: 0.8420 - val_accuracy: 0.7844
Epoch 8/25
274/274 [==============================] - 2060s 8s/step - loss: 0.5111 - accuracy: 0.8320 -
val_loss: 0.8329 - val_accuracy: 0.7893
Epoch 9/25
274/274 [==============================] - 2043s 7s/step - loss: 0.4999 - accuracy: 0.8348 -
val_loss: 0.7863 - val_accuracy: 0.7766
Epoch 10/25
274/274 [==============================] - 2050s 7s/step - loss: 0.4957 - accuracy: 0.8361 -
val_loss: 0.5406 - val_accuracy: 0.8005
Epoch 11/25
274/274 [==============================] - 2046s 7s/step - loss: 0.4948 - accuracy: 0.8379 -
val_loss: 0.6910 - val_accuracy: 0.7830


Epoch 15/25
274/274 [==============================] - 1951s 7s/step - loss: 0.4775 - accuracy: 0.8412 -
val_loss: 0.7685 - val_accuracy: 0.7885
Epoch 16/25
274/274 [==============================] - 1955s 7s/step - loss: 0.4653 - accuracy: 0.8476 -
val_loss: 0.7344 - val_accuracy: 0.7970
Epoch 17/25
274/274 [==============================] - 1985s 7s/step - loss: 0.4686 - accuracy: 0.8437 -
val_loss: 0.6843 - val_accuracy: 0.8001
Epoch 18/25
274/274 [==============================] - 1973s 7s/step - loss: 0.4680 - accuracy: 0.8450 -
val_loss: 0.6433 - val_accuracy: 0.7885
Epoch 19/25
274/274 [==============================] - 1967s 7s/step - loss: 0.4674 - accuracy: 0.8452 -
val_loss: 0.7827 - val_accuracy: 0.7938
Epoch 20/25
274/274 [==============================] - 2036s 7s/step - loss: 0.4565 - accuracy: 0.8487 -
val_loss: 0.8171 - val_accuracy: 0.7949
Epoch 21/25
274/274 [==============================] - 2037s 7s/step - loss: 0.4631 - accuracy: 0.8467 -
val_loss: 0.6373 - val_accuracy: 0.7877
Epoch 22/25
274/274 [==============================] - 1982s 7s/step - loss: 0.4619 - accuracy: 0.8473 -
val_loss: 0.7672 - val_accuracy: 0.7894
Epoch 23/25
274/274 [==============================] - 1966s 7s/step - loss: 0.4510 - accuracy: 0.8473 -
val_loss: 0.5738 - val_accuracy: 0.8015
Epoch 24/25
274/274 [==============================] - 1956s 7s/step - loss: 0.4484 - accuracy: 0.8513 -
val_loss: 0.6403 - val_accuracy: 0.7918
Epoch 25/25
274/274 [==============================] - 1960s 7s/step - loss: 0.4486 - accuracy: 0.8501 -
val_loss: 0.5826 - val_accuracy: 0.7949
```




```
361/361 [==============================] - 1761s 5s/step
Test_Accuracy: 80.50%
```

4.2.3 Model 4

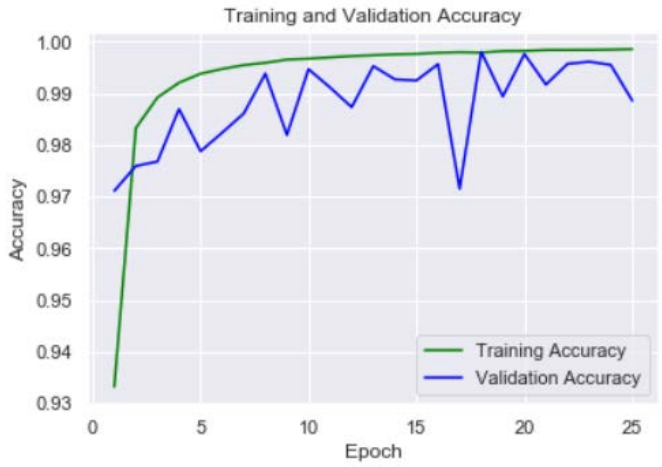
嘗試調整優化器為 adam 尋求更好的解，將 model 2 的模型權重讀進去，並繼續訓練，最後測試集準確率為 99.03%。

```
62 classifier.load_weights('C:/Users/qaz74/Desktop/project3/model_2')
```

	Loss	激活函數	最後一層全連接層激活函數	優化器	Epoch
Model 4	binary	relu	sigmoid	adam	25

```
Epoch 1/25
274/274 [==============================] - 893s 3s/step - loss: 0.2028 - accuracy: 0.9332 - val_loss: 0.1096 - val_accuracy:
0.9711
Epoch 2/25
274/274 [==============================] - 831s 3s/step - loss: 0.0497 - accuracy: 0.9832 - val_loss: 0.0892 - val_accuracy:
0.9759
Epoch 3/25
274/274 [==============================] - 654s 2s/step - loss: 0.0309 - accuracy: 0.9891 - val_loss: 0.0876 - val_accuracy:
0.9767
Epoch 4/25
274/274 [==============================] - 608s 2s/step - loss: 0.0226 - accuracy: 0.9920 - val_loss: 0.0391 - val_accuracy:
0.9869
Epoch 5/25
274/274 [==============================] - 605s 2s/step - loss: 0.0176 - accuracy: 0.9937 - val_loss: 0.0808 - val_accuracy:
0.9787
Epoch 6/25
274/274 [==============================] - 603s 2s/step - loss: 0.0151 - accuracy: 0.9947 - val_loss: 0.0817 - val_accuracy:
0.9823
```

```
274/274 [==============================] - 598s 2s/step - loss: 0.0054 - accuracy: 0.9981 - val_loss: 0.0408 - val_accuracy:
0.9893
Epoch 20/25
274/274 [==============================] - 597s 2s/step - loss: 0.0054 - accuracy: 0.9982 - val_loss: 0.0063 - val_accuracy:
0.9976
Epoch 21/25
274/274 [==============================] - 598s 2s/step - loss: 0.0048 - accuracy: 0.9983 - val_loss: 0.0294 - val_accuracy:
0.9916
Epoch 22/25
274/274 [==============================] - 826s 3s/step - loss: 0.0047 - accuracy: 0.9983 - val_loss: 0.0103 - val_accuracy:
0.9957
Epoch 23/25
274/274 [==============================] - 603s 2s/step - loss: 0.0048 - accuracy: 0.9984 - val_loss: 0.0146 - val_accuracy:
0.9961
Epoch 24/25
274/274 [==============================] - 591s 2s/step - loss: 0.0046 - accuracy: 0.9984 - val_loss: 0.0113 - val_accuracy:
0.9955
Epoch 25/25
274/274 [==============================] - 597s 2s/step - loss: 0.0043 - accuracy: 0.9985 - val_loss: 0.0432 - val_accuracy:
0.9885
```




```
361/361 [==============================] - 124s 344ms/step
Test_Accuracy: 99.03%
```

4.2.4 Model 5

套用 Model 4 的模型設置在 Model 1 上，發現結果也可以直接達到非常高的準確率，其測試集的準確率為 97.48%

```
62 classifier.load_weights('C:/Users/qaz74/Desktop/project3/model_1')
```

```
Epoch 1/25
274/274 [==============================] - 656s 2s/step - loss: 0.1962 - accuracy: 0.9344 - val_loss: 0.2735 - val_accuracy: 0.
9733
Epoch 2/25
274/274 [==============================] - 844s 3s/step - loss: 0.0519 - accuracy: 0.9831 - val_loss: 0.1168 - val_accuracy: 0.
9708
Epoch 3/25
274/274 [==============================] - 615s 2s/step - loss: 0.0343 - accuracy: 0.9885 - val_loss: 0.0455 - val_accuracy: 0.
9850
Epoch 4/25
274/274 [==============================] - 612s 2s/step - loss: 0.0243 - accuracy: 0.9914 - val_loss: 0.1574 - val_accuracy: 0.
9669
Epoch 5/25
274/274 [==============================] - 610s 2s/step - loss: 0.0193 - accuracy: 0.9932 - val_loss: 0.1807 - val_accuracy: 0.
9676
Epoch 6/25
274/274 [==============================] - 608s 2s/step - loss: 0.0162 - accuracy: 0.9942 - val_loss: 0.1230 - val_accuracy: 0.
9721
Epoch 7/25
274/274 [==============================] - 605s 2s/step - loss: 0.0140 - accuracy: 0.9951 - val_loss: 0.0424 - val_accuracy: 0.
9878
Epoch 8/25
274/274 [==============================] - 604s 2s/step - loss: 0.0124 - accuracy: 0.9956 - val_loss: 0.0312 - val_accuracy: 0.
9888
Epoch 9/25
274/274 [==============================] - 601s 2s/step - loss: 0.0112 - accuracy: 0.9961 - val_loss: 0.0204 - val_accuracy: 0.
9935
Epoch 10/25
274/274 [==============================] - 602s 2s/step - loss: 0.0104 - accuracy: 0.9963 - val_loss: 0.0343 - val_accuracy: 0.
9906
Epoch 11/25
274/274 [==============================] - 598s 2s/step - loss: 0.0092 - accuracy: 0.9968 - val_loss: 0.0742 - val_accuracy: 0.
9816
Epoch 12/25
274/274 [==============================] - 590s 2s/step - loss: 0.0088 - accuracy: 0.9969 - val_loss: 0.1345 - val_accuracy: 0.
9717

Epoch 15/25
274/274 [==============================] - 593s 2s/step - loss: 0.0071 - accuracy: 0.9975 - val_loss: 0.0529 - val_accuracy: 0.
9858
Epoch 16/25
274/274 [==============================] - 593s 2s/step - loss: 0.0067 - accuracy: 0.9977 - val_loss: 0.0261 - val_accuracy: 0.
9906
Epoch 17/25
274/274 [==============================] - 593s 2s/step - loss: 0.0062 - accuracy: 0.9978 - val_loss: 0.0505 - val_accuracy: 0.
9871
Epoch 18/25
274/274 [==============================] - 593s 2s/step - loss: 0.0061 - accuracy: 0.9979 - val_loss: 0.0789 - val_accuracy: 0.
9837
Epoch 19/25
274/274 [==============================] - 594s 2s/step - loss: 0.0060 - accuracy: 0.9979 - val_loss: 0.0215 - val_accuracy: 0.
9930
Epoch 20/25
274/274 [==============================] - 593s 2s/step - loss: 0.0053 - accuracy: 0.9981 - val_loss: 0.0129 - val_accuracy: 0.
9950
Epoch 21/25
274/274 [==============================] - 593s 2s/step - loss: 0.0053 - accuracy: 0.9982 - val_loss: 0.0271 - val_accuracy: 0.
9917
Epoch 22/25
274/274 [==============================] - 593s 2s/step - loss: 0.0053 - accuracy: 0.9982 - val_loss: 0.1791 - val_accuracy: 0.
9711
Epoch 23/25
274/274 [==============================] - 593s 2s/step - loss: 0.0051 - accuracy: 0.9982 - val_loss: 0.0280 - val_accuracy: 0.
9936
Epoch 24/25
274/274 [==============================] - 597s 2s/step - loss: 0.0047 - accuracy: 0.9983 - val_loss: 0.0936 - val_accuracy: 0.
9791
Epoch 25/25
274/274 [==============================] - 594s 2s/step - loss: 0.0048 - accuracy: 0.9983 - val_loss: 0.0329 - val_accuracy: 0.
9910
```

```
361/361 [==============================] - 1728s 5s/step
Test_Accuracy: 97.48%
```

5 結論與未來展望

5.1 結論

1) 準確率大幅提升

	Loss	激活函數	最後一層全連接層激活函數	優化器	Epoch	Test Accuracy
Model 1	categorical	relu	softmax	SGD	10	43.91 %
Model 2	categorical	relu	softmax	SGD	25	69.10 %
Model 3	categorical	relu	softmax	adam	25	80.05 %
Model 4	binary	relu	sigmoid	adam	25	99.03 %

2) 為觀光客在此觀光農場提供可信的辨識系統
3) 為後續系統建立奠定基礎

5.2 未來展望

1) 加入更多圖像
2) 加入更多物種
   目前只有 13 種的農作物，加入更多的物種以因應更大規模的觀光農場。
3) 開發不同適用範圍的系統
   農作物的種植類型，可能會因國家、氣候…..等等因素，而有不同區域性分布的種植類型，可針對此開發專屬某區域的系統 。

誌謝

感謝全體 IIE 師生人員

參考資料

https://hackmd.io/@allen108108/rkn-oVGA4
https://zh.wikipedia.org/wiki/%E5%8D%B7%E7%A7%AF%E7%A5%9E%E7%BB%8F%E7%BD%91%E7%BB%9C
https://zh.wikipedia.org/wiki/%E8%91%89
https://kknews.cc/agriculture/vyqvjxq.html