Group 7

Project 2-花卉品種分類

組員：
109034546 吳欣晏
109034547 馮品叡
109034548 吳仲人
109034549 温芳苓

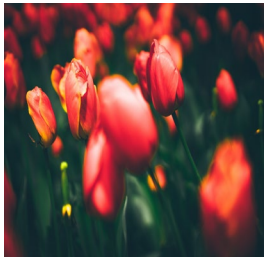
指導教授：邱銘傳 博士

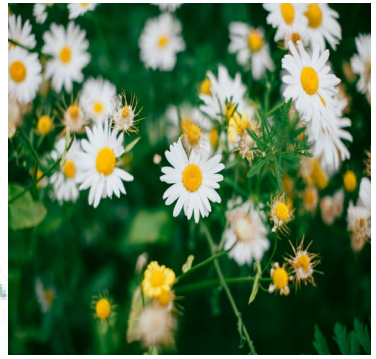
目錄/CONTENT

01

主題說明

主題說明





世界上花的種類舉不勝舉，各種花都有著不同的用途，甚至同一種類的花有許多不同的品種，時常令人混淆造成困擾。

消費者以及廠商時常耗費時間以及勞力做分類

5W1H

購買花卉或是商人販售花卉時，需耗費人力自行辨識，不僅耗時也可能辨識錯誤造成財物損失。

欲購買花卉之消費者、欲利用花卉作為其他用途之廠商。

花店、欲利用花卉作為其他用途(提煉食用油、製作精油)之工廠。

建構花卉品種辨識並自動分類歸檔之技術，節省所浪費之時間以及降低人工辨識的錯誤率。

當人們購買花卉或是商人販售花卉有種類上之疑慮時

各類花卉照片的資料前處理
及 CNN 模型訓練

02

資料前處理過程

資料集簡介

769 筆
圖像資料

雛菊

1052 筆
圖像資料

蒲公英

784 筆
圖像資料

玫瑰

734 筆
圖像資料

向日葵

984 筆
圖像資料

鬱金香

由Kaggle公開數據集中取得花朵的圖像資料集。

Flower Recognition CNN Keras

於colab導入資料集

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

```
Mounted at /content/drive
```

```
from zipfile import ZipFile
file_name = 'drive/MyDrive/flowers.zip'
with ZipFile(file_name, 'r') as zip:
    zip.extractall()
    print('Done')
```

```
Done
```

Flower Recognition CNN Keras

載入會使用到的套件

```
# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns

#configure
# sets matplotlib to inline and displays graphs below the corressponding cell.
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

#model selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import accuracy_score,precision_score,recall_score,confusion_matrix,roc_curve,roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder
```

```
#preprocess.
from keras.preprocessing.image import ImageDataGenerator

#dl libraraies
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from keras.utils import to_categorical

# specifically for cnn
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

import tensorflow as tf
import random as rn

# specifically for manipulating zipped images and getting numpy arrays of pixel values of images.
import cv2
import numpy as np
from tqdm import tqdm
import os
from random import shuffle
from zipfile import ZipFile
from PIL import Image
```

Flower Recognition CNN Keras

定義函式：讀入彩色之圖像，
並調整圖像的大小（尺寸150x150）。

```
def  assign_label(img,flower_type):
        return  flower_type
```

```
def  make_train_data(flower_type,DIR):
        for  img  in  tqdm(os.listdir(DIR)):
                label=assign_label(img,flower_type)
                path  =  os.path.join(DIR,img)
                img  =  cv2.imread(path,cv2.IMREAD_COLOR)  #讀入彩色照片
                img  =  cv2.resize(img,  (IMG_SIZE,IMG_SIZE))  #調整照片尺寸

                X.append(np.array(img))
                Z.append(str(label))
```

Flower Recognition CNN Keras

執行此函式

```
make_train_data('Daisy',FLOWER_DAISY_DIR)
print(len(X))
```

```
100%|██████████| 769/769 [00:01<00:00, 489.09it/s]769
```

```
make_train_data('Sunflower',FLOWER_SUNFLOWER_DIR)
print(len(X))
```

```
100%|██████████| 734/734 [00:01<00:00, 369.73it/s]1503
```

```
make_train_data('Tulip',FLOWER_TULIP_DIR)
print(len(X))
```

```
100%|██████████| 984/984 [00:02<00:00, 418.69it/s]3471
```

```
make_train_data('Dandelion',FLOWER_DANDI_DIR)
print(len(X))
```

```
100%|██████████| 1052/1052 [00:02<00:00, 428.24it/s]452
```

```
make_train_data('Rose',FLOWER_ROSE_DIR)
print(len(X))
```

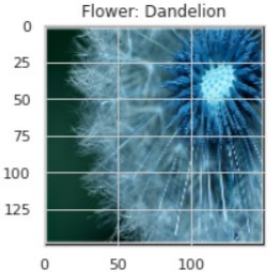
```
100%|██████████| 784/784 [00:01<00:00, 501.24it/s]1553
```

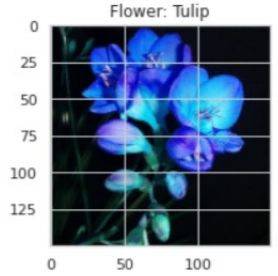
Flower Recognition CNN Keras

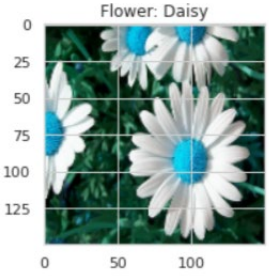
以視覺化的方式呈現

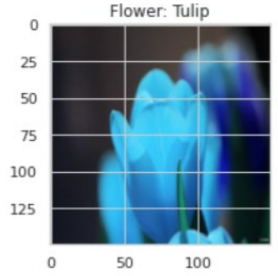
```
fig,ax=plt.subplots(5,2)
fig.set_size_inches(15,15)
for  i  in  range(5):
        for  j  in  range  (2):
                l=rn.randint(0,len(Z))
                ax[i,j].imshow(X[l])
                ax[i,j].set_title('Flower:  '+Z[l])

plt.tight_layout()
```

結果如下所示









Flower Recognition CNN Keras

將 labels 進行 one-hot encoding

```
le=LabelEncoder()
Y=le.fit_transform(Z)
Y=to_categorical(Y,5)  #將  labels  進行  one-hot  encoding
X=np.array(X)
X=X/255  #將每張圖片像素值皆除以255，圖像做歸一化，使其像素值從 [0-255] 縮放到 [0-1]。
```

將資料分成訓練集和測試集

```
x_train, x_test, y_train, y_test=train_test_split(X, Y, test_size=0.25, random_state=42)
```

03 模型建構

發展模型

使用之模型為 Sequential
第一層 Layer 的 filters 為 32
第二層 Layer 的 filters 為 64
第三層 Layer 的 filters為96
第四層Layer 的 filters為96
activation 皆設為 relu。

```
# # modelling starts using a CNN.

model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation ='relu', input_shape = (150,150,3)))
model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))


model.add(Conv2D(filters =96, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation ='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(5, activation = "softmax"))
```

利用LR Annealer

```
[16]  batch_size=128
      epochs=50

      from  keras.callbacks  import  ReduceLROnPlateau
      red_lr=  ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.1)
```

數據擴充以防止過擬合

```
datagen = ImageDataGenerator(
                featurewise_center=False,    # set input mean to 0 over the dataset
                samplewise_center=False,    # set each sample mean to 0
                featurewise_std_normalization=False,    # divide inputs by std of the dataset
                samplewise_std_normalization=False,    # divide each input by its std
                zca_whitening=False,    # apply ZCA whitening
                rotation_range=10,    # randomly rotate images in the range (degrees, 0 to 180)
                zoom_range = 0.1, # Randomly zoom image
                width_shift_range=0.2,    # randomly shift images horizontally (fraction of total width)
                height_shift_range=0.2,    # randomly shift images vertically (fraction of total height)
                horizontal_flip=True,    # randomly flip images
                vertical_flip=False)    # randomly flip images


datagen.fit(x_train)
```

設定模型參數、模型架構

使用的 optimizer 為 Adam，學習率為0.2%。

```
model.compile(optimizer=Adam(lr=0.002),loss='categorical_crossentropy',metrics=['accuracy'])
```

模型架構

```
model.summary()
```

模型架構之結果

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 96)        55392
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 96)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 96)        83040
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 96)          0
_________________________________________________________________
flatten (Flatten)            (None, 7776)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3981824
_________________________________________________________________
activation (Activation)      (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 2565
=================================================================
Total params: 4,143,749
Trainable params: 4,143,749
Non-trainable params: 0
```

模型擬合訓練數據並驗證測試數據

```
History = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (x_test,y_test),
                              verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)
# model.fit(x_train,y_train,epochs=epochs,batch_size=batch_size,validation_data = (x_test,y_test))
```

擷取最後訓練結果

```
Epoch 39/50
25/25 [==============================] - 13s 529ms/step - loss: 0.5238 - accuracy: 0.7954 - val_loss: 0.7918 - val_accuracy: 0.7447
Epoch 40/50
25/25 [==============================] - 13s 527ms/step - loss: 0.4999 - accuracy: 0.8064 - val_loss: 0.7199 - val_accuracy: 0.7660
Epoch 41/50
25/25 [==============================] - 13s 540ms/step - loss: 0.4892 - accuracy: 0.8141 - val_loss: 0.8682 - val_accuracy: 0.7373
Epoch 42/50
25/25 [==============================] - 13s 526ms/step - loss: 0.4328 - accuracy: 0.8314 - val_loss: 0.7507 - val_accuracy: 0.7475
Epoch 43/50
25/25 [==============================] - 13s 526ms/step - loss: 0.4175 - accuracy: 0.8365 - val_loss: 0.7842 - val_accuracy: 0.7391
Epoch 44/50
25/25 [==============================] - 13s 527ms/step - loss: 0.4457 - accuracy: 0.8292 - val_loss: 0.7749 - val_accuracy: 0.7364
Epoch 45/50
25/25 [==============================] - 13s 529ms/step - loss: 0.4338 - accuracy: 0.8353 - val_loss: 0.7163 - val_accuracy: 0.7650
Epoch 46/50
25/25 [==============================] - 13s 526ms/step - loss: 0.4389 - accuracy: 0.8282 - val_loss: 0.7916 - val_accuracy: 0.7428
Epoch 47/50
25/25 [==============================] - 13s 524ms/step - loss: 0.4144 - accuracy: 0.8465 - val_loss: 0.7579 - val_accuracy: 0.7502
Epoch 48/50
25/25 [==============================] - 13s 525ms/step - loss: 0.4268 - accuracy: 0.8423 - val_loss: 0.7392 - val_accuracy: 0.7595
Epoch 49/50
25/25 [==============================] - 13s 526ms/step - loss: 0.4120 - accuracy: 0.8404 - val_loss: 0.7170 - val_accuracy: 0.7586
Epoch 50/50
25/25 [==============================] - 13s 526ms/step - loss: 0.4082 - accuracy: 0.8475 - val_loss: 0.7017 - val_accuracy: 0.7798
```

模型校度分析

取得測試集準確率約為78%

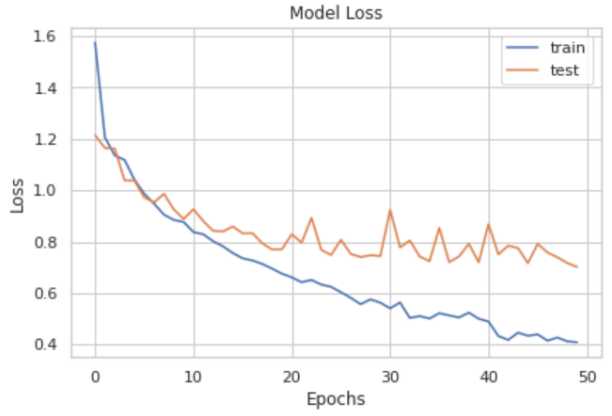
```
train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 25ms/step - loss: 0.2692 - accuracy: 0.8924
9/9 [==============================] - 0s 21ms/step - loss: 0.7017 - accuracy: 0.7798
Train accuracy: 0.8923503756523132
Test accuracy: 0.7798334956169128
```

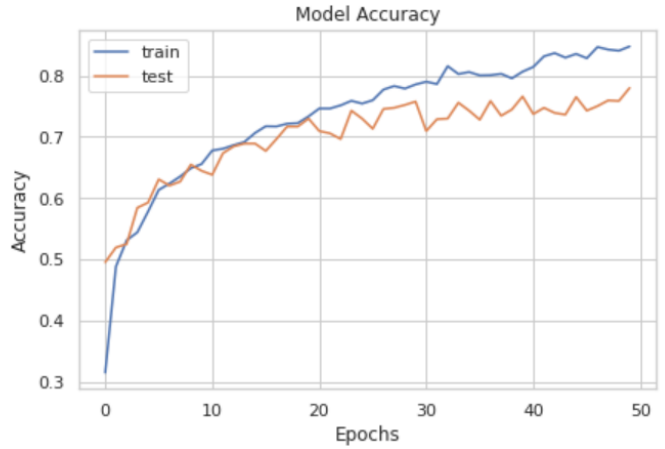
模型校度分析

繪製Loss、Accuracy與Epochs之關係圖

```
plt.plot(History.history['loss'])
plt.plot(History.history['val_loss'])
plt.title('Model  Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train',  'test'])
plt.show()
```



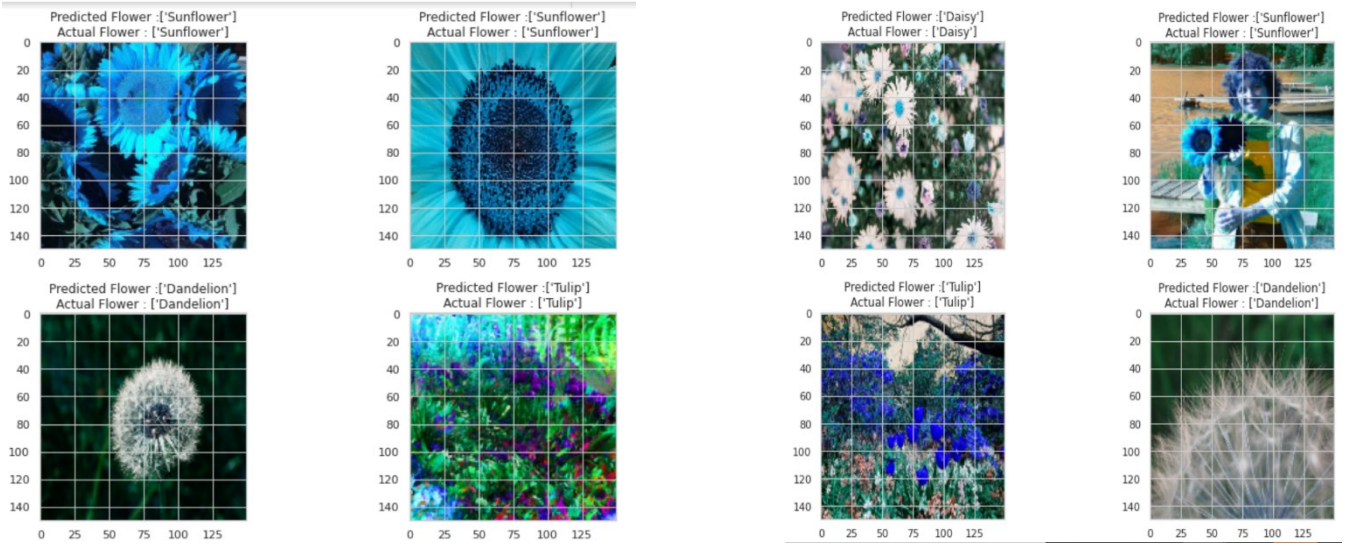

```
[23] plt.plot(History.history['accuracy'])
     plt.plot(History.history['val_accuracy'])
     plt.title('Model  Accuracy')
     plt.ylabel('Accuracy')
     plt.xlabel('Epochs')
     plt.legend(['train',  'test'])
     plt.show()
```



視覺化預測結果與實際結果

預測結果與實際結果相同

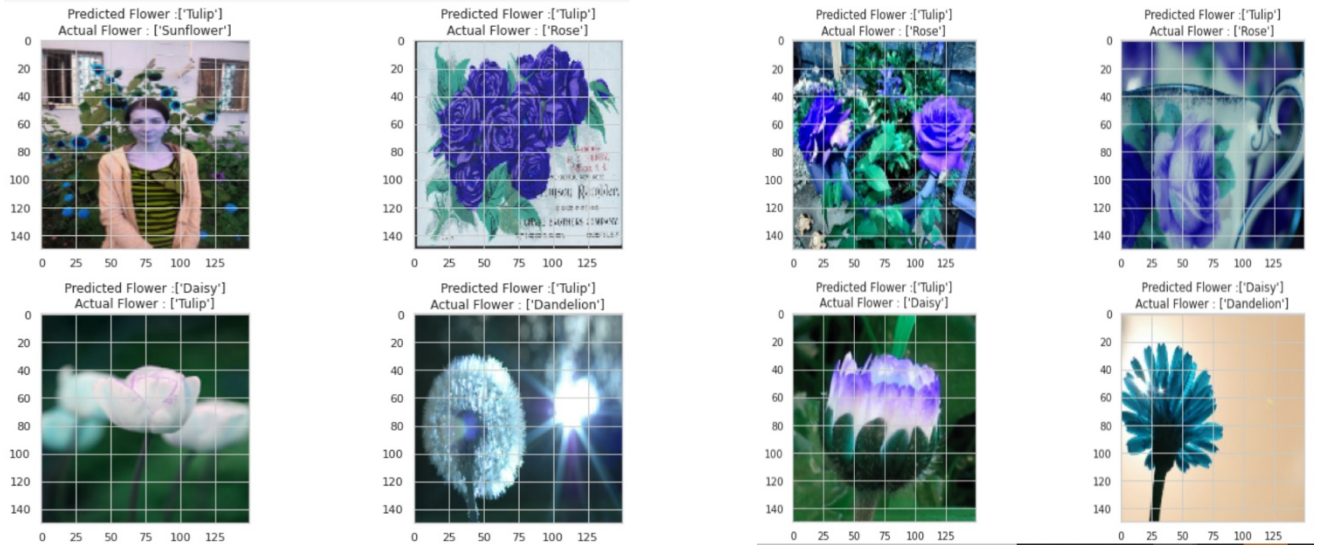
```
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

count=0
fig,ax=plt.subplots(4,2)
fig.set_size_inches(15,15)
for  i  in  range  (4):
        for  j  in  range  (2):
                ax[i,j].imshow(x_test[prop_class[count]])
                ax[i,j].set_title("Predicted  Flower  :"+str(le.inverse_transform([pred_digits[prop_class[count]]]))
                                                        +"\n"+"Actual  Flower  :  "+str(le.inverse_transform([np.argmax(y_test[prop_class[count]])])))
                plt.tight_layout()
                count+=1
```



視覺化預測結果與實際結果

預測結果與實際結果不同

```
[27] warnings.filterwarnings('always')
     warnings.filterwarnings('ignore')

     count=0
     fig,ax=plt.subplots(4,2)
     fig.set_size_inches(15,15)
     for i in range (4):
         for j in range (2):
             ax[i,j].imshow(x_test[mis_class[count]])
             ax[i,j].set_title("Predicted Flower :"+str(le.inverse_transform([pred_digits[mis_class[count]]]))
                               +"\n"+"Actual Flower :"+str(le.inverse_transform([np.argmax(y_test[mis_class[count]])])))
             plt.tight_layout()
             count+=1
```



04

參數優化

超參數優化結果

Activation		Filters		Learning rate	
參數	Test accuracy	參數	Test accuracy	參數	Test accuracy
Sigmoid	21%	8	75%	0.5%	64%
Relu	78%	16	76%	0.2%	78%
Tanh	21%	32	78%	0.05%	75%

Activation Function超參數優化

```
train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 28ms/step - loss: 1.5981 - accuracy: 0.2535
9/9 [==============================] - 0s 33ms/step - loss: 1.6151 - accuracy: 0.2128
Train accuracy: 0.25354719161987305
Test accuracy: 0.21276596188545227
```

Activation設為Sigmoid，
test accuracy約為21%。

```
train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 25ms/step - loss: 0.2692 - accuracy: 0.8924
9/9 [==============================] - 0s 21ms/step - loss: 0.7017 - accuracy: 0.7798
Train accuracy: 0.8923503756523132
Test accuracy: 0.7798334956169128
```

Activation設為Relu，
test accuracy約為78%。

```
train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 30ms/step - loss: 1.7168 - accuracy: 0.2535
9/9 [==============================] - 0s 26ms/step - loss: 1.7725 - accuracy: 0.2128
Train accuracy: 0.25354719161987305
Test accuracy: 0.21276596188545227
```

Activation設為Tanh，

test accuracy約為21%。

Filters超參數優化

```
[21] train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
     test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
     print('Train accuracy:', train_acc)
     print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 22ms/step - loss: 0.2884 - accuracy: 0.8942
9/9 [==============================] - 0s 25ms/step - loss: 0.7857 - accuracy: 0.7549
Train accuracy: 0.8942010998725891
Test accuracy: 0.7548565864562988
```

Filters設為8，
test accuracy約為75%。

```
train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 23ms/step - loss: 0.2531 - accuracy: 0.9078
9/9 [==============================] - 0s 25ms/step - loss: 0.8014 - accuracy: 0.7567
Train accuracy: 0.907772958278656
Test accuracy: 0.7567067742347717
```

Filters設為16，
test accuracy約為76%。

```
train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 25ms/step - loss: 0.2692 - accuracy: 0.8924
9/9 [==============================] - 0s 21ms/step - loss: 0.7017 - accuracy: 0.7798
Train accuracy: 0.8923503756523132
Test accuracy: 0.7798334956169128
```

Filters設為32，
test accuracy約為78%。

Learning Rate超參數優化

```
[21] train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
     test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
     print('Train accuracy:', train_acc)
     print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 25ms/step - loss: 0.7723 - accuracy: 0.7051
9/9 [==============================] - 0s 31ms/step - loss: 0.9539 - accuracy: 0.6420
Train accuracy: 0.705120325088501
Test accuracy: 0.6419981718063354
```

Learning Rate設為0.5%，
test accuracy約為64%。

```
train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 25ms/step - loss: 0.2692 - accuracy: 0.8924
9/9 [==============================] - 0s 21ms/step - loss: 0.7017 - accuracy: 0.7798
Train accuracy: 0.8923503756523132
Test accuracy: 0.7798334956169128
```

Learning Rate設為0.2%，
test accuracy約為78%。

```
train_loss, train_acc = model.evaluate(x_train, y_train, batch_size=batch_size)
test_loss, test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
print('Train accuracy:', train_acc)
print('Test accuracy:', test_acc)
```

```
26/26 [==============================] - 1s 25ms/step - loss: 0.2774 - accuracy: 0.8967
9/9 [==============================] - 0s 22ms/step - loss: 0.7834 - accuracy: 0.7530
Train accuracy: 0.8966687321662903
Test accuracy: 0.7530064582824707
```

Learning Rate設為0.05%，
test accuracy約為75%。

05 結論與未來展望

結論與未來展望

透過Generative Adversarial Network(GA)提升準確率。

AI與生技醫療結合，對社會大眾有所貢獻。

AI與RPA結合可大幅降地成本與時間。

建構花卉品種辨識並自動分類歸檔之技術，節省所浪費之時間以及降低人工辨識的錯誤率。

透過不同超參數的調整可提升準確率。

可合併多種AI MODEL提升準確率。


THANK YOU