「那是聖誕老公公嗎？(SANTA)資料集」進行辨識

班級:資工三

學號:B1043001

姓名:許良亦

程序描述:

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

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

一開始，先透過以上程式 from google.colab import drive 把 google 雲端載入

此時的檔案狀態，表示資料集已載入

```
import os    # OpenCV：讀取圖片
import cv2    # OS：走訪圖片
import numpy as np    # Numpy：矩陣運算
import pandas as pd    # pandas ：混淆矩陣視覺呈現
import tensorflow
import matplotlib.pyplot as plt
from keras.models import Sequential
from sklearn.metrics import confusion_matrix
from tensorflow.python.keras.utils.np_utils import to_categorical    # Keras：建立訓練模型
from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout
```

資料載入後，載入相關套件，OS 跟 cv2 用來讀取跟走訪載入的資料集，numpy 用來做矩陣運算，pandas 則是計算混淆矩陣，matplotlib 則是畫出 accuracy 跟 loss 的結果圖，Sequential 則是優化器，然後引入混淆矩陣，最後兩行則是建立模型(卷積層和池化層和 Dropout 和平坦層)

```
[3] Trainpath = '/content/drive/MyDrive/is that santa/train'
    Testpath = '/content/drive/MyDrive/is that santa/test'
    x_Train = []      # 儲存訓練資料集處理後的圖片
    y_Train = []      # 儲存訓練資料集處理後的label{0: 'not-a-santa', 1: 'santa'}
    x_Test = []       # 儲存測試資料集處理後的圖片
    y_Test = []       # 儲存測試資料集處理後的label{0: 'not-a-santa', 1: 'santa'}

    label_name = {0: 'not-a-santa', 1: 'santa'}
    print("Start data processing . . .")
```

接者建立 train 檔路徑跟 test 檔的路徑，x_Train 用來儲存訓練資料集處理後的圖片，y_Train 用來儲存訓練資料集處理後 label 0 為’not-a-santa’，1 為 ’santa’，x_Test 存取測試資料集處理後的圖片，y_Test 存取測試資料集處理後的 label，建立完串列，建立一個 label_name 的字典，印出開始資料處理

```
#  訓練資料集處理
for  label,  folder  in  label_name.items():
        #  創建指向訓練集中當前標籤文件夾的路徑
        path  =  os.path.join(Trainpath,  folder)
        #  讀取並調整圖片大小
        for  img  in  os.listdir(path):
                imgtrain  =  cv2.imread(os.path.join(path,  img))
                height,  width  =  imgtrain.shape[:2]      #讀取圖片的長跟寬
                #  調整尺寸並保持長寬比
                if  height  >  width:
                        new_height  =  256
                        new_width  =  int(width  *  (256  /  height))
                else:
                        new_width  =  256
                        new_height  =  int(height  *  (256  /  width))
                imgtrain  =  cv2.resize(imgtrain,  (new_width,  new_height))
                #  添加邊界使圖片為256x256
                top  =  (256  -  new_height)  //  2
                bottom  =  256  -  new_height  -  top
                left  =  (256  -  new_width)  //  2
                right  =  256  -  new_width  -  left
                #  確保圖片最終的尺寸是  256x256
                #  value  (0,  0,  0,  0)  的前三個  (0,  0,  0)  表示黑色，而最後一個  0  是表示透明)，在這裡設為  0  表示完全不透明
                #  cv2.BORDER_CONSTANT:指定使用常數值填充邊界
                imgtrain  =  cv2.copyMakeBorder(imgtrain,  top,  bottom,  left,  right,  cv2.BORDER_CONSTANT,  value=(0,  0,  0,  0))
                #  將處理後的圖片及其標籤附加到列表中
                x_Train.append(imgtrain)
                y_Train.append(label)
print("Train  data  processing  completed!")
```

進入訓練資料集處理階段，首先，透過 for 迴圈遍歷每個標籤及其對應的資料夾。在每個資料夾中，讀取並調整每張圖片的大小，確保最終的尺寸為 256x256。接著，為了確保圖片的最終尺寸，使用 cv2.copyMakeBorder 函數在圖片周圍添加邊界。這裡使用的邊界填充值是黑色。最後，處理過的圖片及其對應的標籤被添加到 x_Train 和 y_Train 這兩個列表中，最後印出訓練資料處理完畢

```
# 測試資料集處理
for label, folder in label_name.items():
        # 創建指向測試集中當前標籤文件夾的路徑
        path = os.path.join(Testpath, folder)
        # 讀取並調整圖片大小
        for img in os.listdir(path):
                imgtest = cv2.imread(os.path.join(path, img))
                height, width = imgtest.shape[:2]    # 讀取圖片的長跟寬
                # 讀取並調整圖片大小
                if height > width:
                        new_height = 256
                        new_width = int(width * (256 / height))
                else:
                        new_width = 256
                        new_height = int(height * (256 / width))
                imgtest = cv2.resize(imgtest, (new_width, new_height))
                # 添加邊界使圖片為256x256
                top = (256 - new_height) // 2
                bottom = 256 - new_height - top
                left = (256 - new_width) // 2
                right = 256 - new_width - left
                # 確保圖片最終的尺寸是 256x256
                # value (0, 0, 0, 0) 的前三個 (0, 0, 0) 表示黑色，而最後一個 0 是表示透明)，在這裡設為 0 表示完全不透明
                # cv2.BORDER_CONSTANT:指定使用常數值填充邊界
                imgtest = cv2.copyMakeBorder(imgtest, top, bottom, left, right, cv2.BORDER_CONSTANT, value=(0, 0, 0, 0))
                # 將處理後的圖片及其標籤附加到列表中
                x_Test.append(imgtest)
                y_Test.append(label)
print("Test data processing completed!")
```

接者進入測試資料集處理階段，採取跟訓練資料資集一樣的方法，把每張圖片的尺寸變為 256，並且放在正中間，使用黑色的邊界填充值。最後，處理過的圖片及其對應的標籤被添加到 x_Test 和 y_Test 這兩個列表中，最後印出測試資料處理完畢

```
Start data processing . . .
Train data processing completed!
Test data processing completed!
```

上圖為目前的執行結果

```
[4]  #  將列表轉為  NumPy  數列
     x_Train_array  =  np.array(x_Train)
     x_Test_array  =  np.array(x_Test)
     y_Train  =  np.array(y_Train)
     y_Test  =  np.array(y_Test)

     #  將影像的特徵值轉換為4維矩陣
     #  影像大小為  256x256，通道數為  3（RGB）
     x_Train4D  =  x_Train_array.reshape(x_Train_array.shape[0],  256,  256,  3).astype('float32')
     x_Test4D  =  x_Test_array.reshape(x_Test_array.shape[0],  256,  256,  3).astype('float32')

     ##  數字影像特徵值標準化
     x_Train4D_normalize  =  x_Train4D  /  255
     x_Test4D_normalize  =  x_Test4D  /  255

     #  label(數字的真實的值)以Onehot  encoding轉換
     y_TrainOneHot  =  to_categorical(y_Train)
     y_TestOneHot  =  to_categorical(y_Test)
```

下一步，將原始的影像資料和標籤轉換成模型可以處理的格式，將列表轉為 NumPy 數列，x_Train_array 和 x_Test_array 分別將訓練和測試集的影像資料轉換為 NumPy 數列，y_Train 和 y_Test 分別將訓練和測試集的標籤轉換為 NumPy 數列，影像的特徵值轉換為 4 維矩陣部分則是 x_Train4D_normalize 和 x_Test4D_normalize 分別將訓練和測試集的影像進行標準化，將像素值縮放到 0 到 1 之間，以便更好地適應模型，最後，標籤 One-Hot Encoding，y_TrainOneHot 和 y_TestOneHot 分別將訓練和測試集的標籤進行 One-Hot Encoding 轉換

```
# 建立 Sequential 模型
model = Sequential()

# 添加卷積層和池化層
'''
filiters 建立濾鏡(濾鏡數量，一個濾鏡會產生一個特徵圖)
kernel_size 濾鏡大小，在此設置成5X5
padding=' same' 卷積運算不改變圖片大小
activation=' relu' 設定激活函數relu
'''
model.add(Conv2D(filters=16, kernel_size=(5, 5), padding='same', input_shape=(256, 256, 3), activation='relu'))
# 設定池化窗口為2X2
model.add(MaxPooling2D(pool_size=(2, 2)))

# 再設一個池化層filters設36
model.add(Conv2D(filters=36, kernel_size=(5, 5), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# 加入Droput可避免過擬合overfitting
model.add(Dropout(0.25))
# 使用Flatten()平坦層將資料壓成1維
model.add(Flatten())
# 建立隱藏層
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
# 建立輸出層
model.add(Dense(2, activation='softmax'))

# 模型做總結
model.summary()

# 使用compile來定義損失函數、優化函數及成效衡量指標
# loss用cross entropy(交叉熵)
# optimizer採用梯度下降法採取最常用的演算法adam
# metrics:模型的評估方式選擇以accuracy來評估
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

模型的部分，建立 Sequential 模型，利用兩個卷積層，一個 filiter 為 16，一個為 36，綠鏡大小都為 5X5，input_shape 則是 (256,256,3)，3 代表 RGB，然後每個卷積層後加一個 2X2 的池化層，最後，依序接上 dropout、flatten、dense、dropout、dense，使用 compile 來定義損失函數、優化函數及成效衡量指標 loss 用 cross entropy(交叉熵)，optimizer 採用梯度下降法採取最常用的演算法 adam，metrics:模型的評估方式選擇以 accuracy 來評估

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 256, 256, 16)      1216

 max_pooling2d_2 (MaxPoolin  (None, 128, 128, 16)      0
 g2D)

 conv2d_3 (Conv2D)           (None, 128, 128, 36)      14436

 max_pooling2d_3 (MaxPoolin  (None, 64, 64, 36)        0
 g2D)

 dropout_2 (Dropout)         (None, 64, 64, 36)        0

 flatten_1 (Flatten)         (None, 147456)            0

 dense_2 (Dense)             (None, 128)               18874496

 dropout_3 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 2)                 258

=================================================================
Total params: 18890406 (72.06 MB)
Trainable params: 18890406 (72.06 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

此時的執行結果

```
[6]  #  epochs  代表訓練的週期(將資料訓練過十次
     #  batch_size  每一批訓練256筆資料
     #  validation_split=0.2  代表從訓練資料集中取20%來當作驗證資料集
     #  verbose  訓練日誌顯示模式  1  =進度條

     train_history=model.fit(x=x_Train4D_normalize,  y=y_TrainOneHot,  validation_split=0.2,  epochs=10,  batch_size=256,  verbose=1)
```

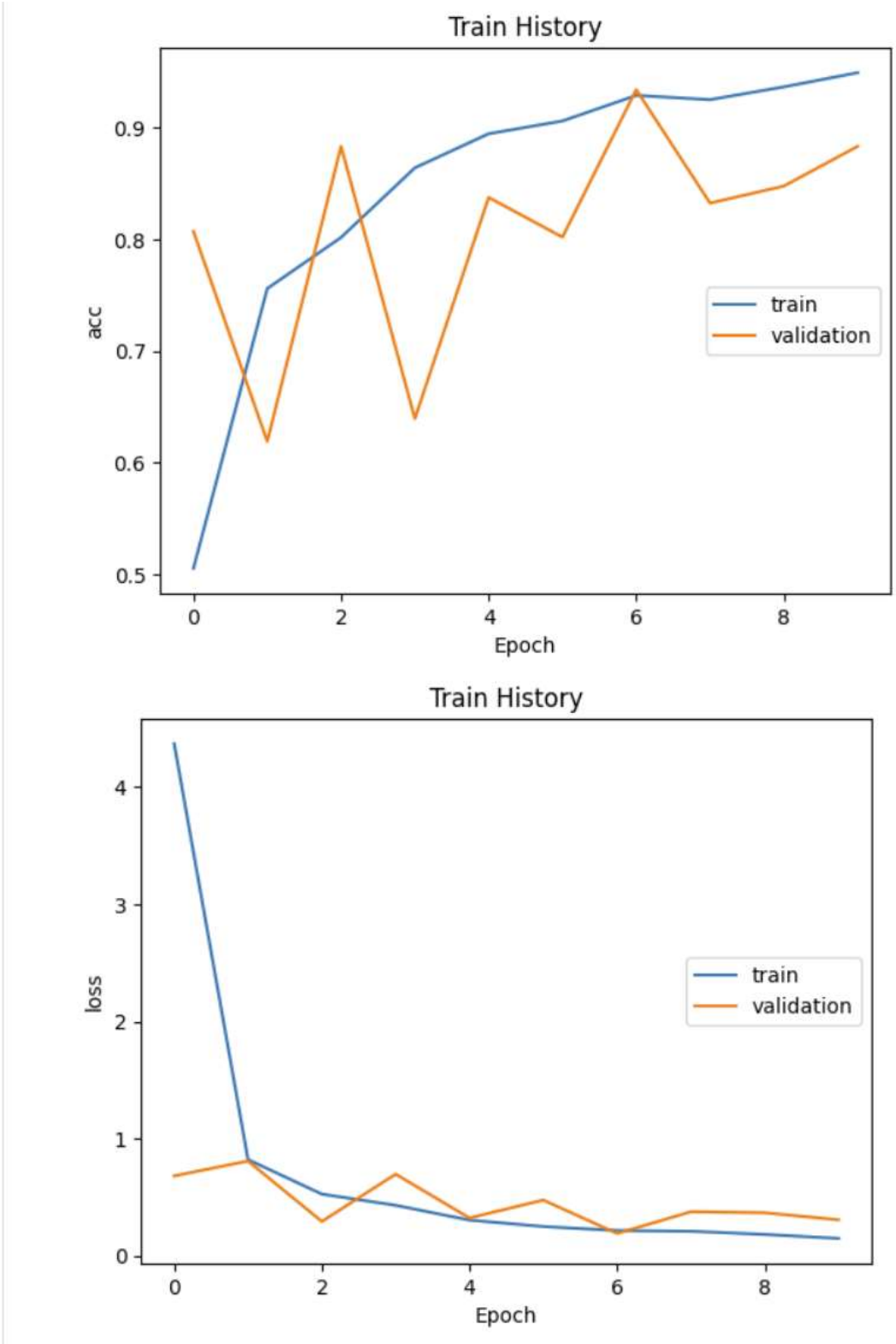
```
Epoch 1/10
4/4 [==============================] - 17s 2s/step - loss: 4.3711 - accuracy: 0.5057 - val_loss: 0.6805 - val_accuracy: 0.8071
Epoch 2/10
4/4 [==============================] - 2s 383ms/step - loss: 0.8201 - accuracy: 0.7560 - val_loss: 0.8077 - val_accuracy: 0.6193
Epoch 3/10
4/4 [==============================] - 2s 376ms/step - loss: 0.5252 - accuracy: 0.8018 - val_loss: 0.2916 - val_accuracy: 0.8832
Epoch 4/10
4/4 [==============================] - 2s 375ms/step - loss: 0.4295 - accuracy: 0.8640 - val_loss: 0.6955 - val_accuracy: 0.6396
Epoch 5/10
4/4 [==============================] - 2s 410ms/step - loss: 0.3032 - accuracy: 0.8945 - val_loss: 0.3216 - val_accuracy: 0.8376
Epoch 6/10
4/4 [==============================] - 2s 534ms/step - loss: 0.2483 - accuracy: 0.9060 - val_loss: 0.4743 - val_accuracy: 0.8020
Epoch 7/10
4/4 [==============================] - 2s 427ms/step - loss: 0.2142 - accuracy: 0.9288 - val_loss: 0.1889 - val_accuracy: 0.9340
Epoch 8/10
4/4 [==============================] - 2s 422ms/step - loss: 0.2080 - accuracy: 0.9250 - val_loss: 0.3745 - val_accuracy: 0.8325
Epoch 9/10
4/4 [==============================] - 2s 406ms/step - loss: 0.1811 - accuracy: 0.9365 - val_loss: 0.3658 - val_accuracy: 0.8477
Epoch 10/10
4/4 [==============================] - 1s 353ms/step - loss: 0.1467 - accuracy: 0.9492 - val_loss: 0.3059 - val_accuracy: 0.8832
```

建立完模型，開始訓練，epochs 設 10，batch_size 設 256 筆資料，validation_split=0.2 從訓練資料集中取 20%來當作驗證資料集，verbose 訓練日誌顯示模式 1 =進度條

```
[7]  #  用matplotlib.pyplot呈現訓練結果
     def  show_train_history(train_history,  train,  validation):
         plt.plot(train_history.history[train])
         plt.plot(train_history.history[validation])
         plt.title('Train  History')
         if  train  ==  'accuracy':
             plt.ylabel('acc')
         else  :
             plt.ylabel('loss')
         plt.xlabel('Epoch')
         plt.legend(['train',  'validation'],  loc='center  right')
         plt.show()

     #  畫出accuracy的執行結果
     show_train_history(train_history,  'accuracy',  'val_accuracy')
     #  畫出loss誤差的執行結果
     show_train_history(train_history,  'loss',  'val_loss')
```

訓練完，畫出 accuracy 跟 loss 的結果



模型訓練結果及測試結果的 accuracy 跟 loss 圖

```
[8]  #  評估模型準確率：將處理過的測試資料放入model.evaluate()中評估模型準確率
     loss,  accuracy  =  model.evaluate(x_Test4D_normalize  ,  y_TestOneHot)
     print(  "\nLoss:  %.2f,  Accuracy:  %.2f%%"  %  (loss,  accuracy*  100  ))
```

```
8/8 [==============================] - 1s 46ms/step - loss: 0.3649 - accuracy: 0.8577

Loss: 0.36, Accuracy: 85.77%
```

計算測試資料的 Loss、 Accuracy 的數值，並印出結果

```
#  顯示混淆矩陣：顯示測試資料預測結果統計與實際結果統計的差異
#  橫向為預測結果  縱向為實際結果
prediction=np.argmax(model.predict(x_Test4D_normalize),  axis=1)
print("Confusion  Matrix:")
pd.crosstab(y_Test,prediction,rownames=['label'],colnames=['predict'])
```

```
8/8 [==============================] - 0s 18ms/step
Confusion Matrix:
```

predict	0	1
label		
0	110	13
1	22	101

最後畫出混淆矩陣