is that SANTA？--深度學習實作

資工三 B1043003 陳麒安

深度學習程式碼：

import os

import cv2

import numpy as np

import pandas as pd

import tensorflow as tf

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

from keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Dropout

# 設定訓練和測試資料路徑

Trainpath = '/content/drive/MyDrive/is that santa/train'

Testpath = '/content/drive/MyDrive/is that santa/test'

x\_Train = []

y\_Train = []

x\_Test = []

y\_Test = []

# 定義標籤名稱對應的字典

label\_name = {0: 'not-a-santa', 1: 'santa'}

print("Start data processing . . .")

# 處理訓練資料集

for label, folder in label\_name.items():

path = os.path.join(Trainpath, folder)

for img in os.listdir(path):

imgtrain = cv2.imread(os.path.join(path, img))

height, width = imgtrain.shape[:2]

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))

top = (256 - new\_height) // 2

bottom = 256 - new\_height - top

left = (256 - new\_width) // 2

right = 256 - new\_width - left

imgtrain = cv2.copyMakeBorder(imgtrain, top, bottom, left, right, cv2.BORDER\_CONSTANT, value=(0, 0, 0, 0))

x\_Train.append(imgtrain)

y\_Train.append(label)

print("Train data processing completed!")

# 處理測試資料集

for label, folder in label\_name.items():

path = os.path.join(Testpath, folder)

for img in os.listdir(path):

imgtest = cv2.imread(os.path.join(path, img))

height, width = imgtest.shape[:2]

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))

top = (256 - new\_height) // 2

bottom = 256 - new\_height - top

left = (256 - new\_width) // 2

right = 256 - new\_width - left

imgtest = cv2.copyMakeBorder(imgtest, top, bottom, left, right, cv2.BORDER\_CONSTANT, value=(0, 0, 0, 0))

x\_Test.append(imgtest)

y\_Test.append(label)

print("Test data processing completed!")

# 轉換資料為 NumPy 陣列

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)

# 將資料 reshape 成四維張量並進行正規化

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')

x\_Train4D\_normalize = x\_Train4D / 255

x\_Test4D\_normalize = x\_Test4D / 255

# 將標籤進行 One-Hot 編碼

y\_TrainOneHot = to\_categorical(y\_Train)

y\_TestOneHot = to\_categorical(y\_Test)

# 建立 Sequential 模型

model = Sequential()

# 加入卷積層、池化層、Dropout 和全連接層

model.add(Conv2D(filters=16, kernel\_size=(3, 3), padding='same', input\_shape=(256, 256, 3), activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(filters=36, kernel\_size=(3, 3), padding='same', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

model.summary()

# 編譯模型

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# 訓練模型

train\_history = model.fit(x=x\_Train4D\_normalize, y=y\_TrainOneHot, validation\_split=0.15, epochs=20, batch\_size=32, verbose=1)

# 定義函數顯示訓練歷史

def show\_train\_history(train\_history, train, validation):

plt.plot(train\_history.history[train])

plt.plot(train\_history.history[validation])

plt.title('Train History')

plt.ylabel('acc')

plt.xlabel('Epoch')

plt.legend(['train', 'validation'], loc='center right')

plt.show()

# 顯示訓練準確度和驗證準確度的變化

show\_train\_history(train\_history, 'accuracy', 'val\_accuracy')

# 顯示訓練損失和驗證損失的變化

show\_train\_history(train\_history, 'loss', 'val\_loss')

# 評估模型準確度

loss, accuracy = model.evaluate(x\_Test4D\_normalize, y\_TestOneHot)

print("\nLoss: %.2f, Accuracy: %.2f%%" % (loss, accuracy \* 100))

# 進行預測並顯示混淆矩陣

prediction = np.argmax(model.predict(x\_Test4D\_normalize), axis=1)

conf\_matrix = confusion\_matrix(np.argmax(y\_TestOneHot, axis=1), prediction)

print("Confusion Matrix:")

print(conf\_matrix)

程式說明：

首先觀察資料集的格式，train和test下各分為santa 和not-a-santa，並且圖片的大小並不相同。因此在設定完訓練和測試資料路徑和定義標籤名稱對應的字典後要對資料進行處理。經過測試，在Google Colab中執行此訓練的系統 RAM可接受的圖片大小為256\*256，所以在讀取圖片後在256\*256的空白畫布貼上依照比例縮放的.jpg圖片。

之後，將縮放後的資料 reshape 成四維張量並進行正規化，並將y\_Train和y\_Test進行 One-Hot 編碼。接著，建立 Sequential 模型並加入卷積層、池化層、Dropout 和全連接層，將kernel\_size改為(3, 3)和model.add(Dense())改為(2, activation='softmax')。

最後，在訓練模型train\_history = model.fit()中，調整以下參數validation\_split=0.15, epochs=20, batch\_size=32, verbose=1。

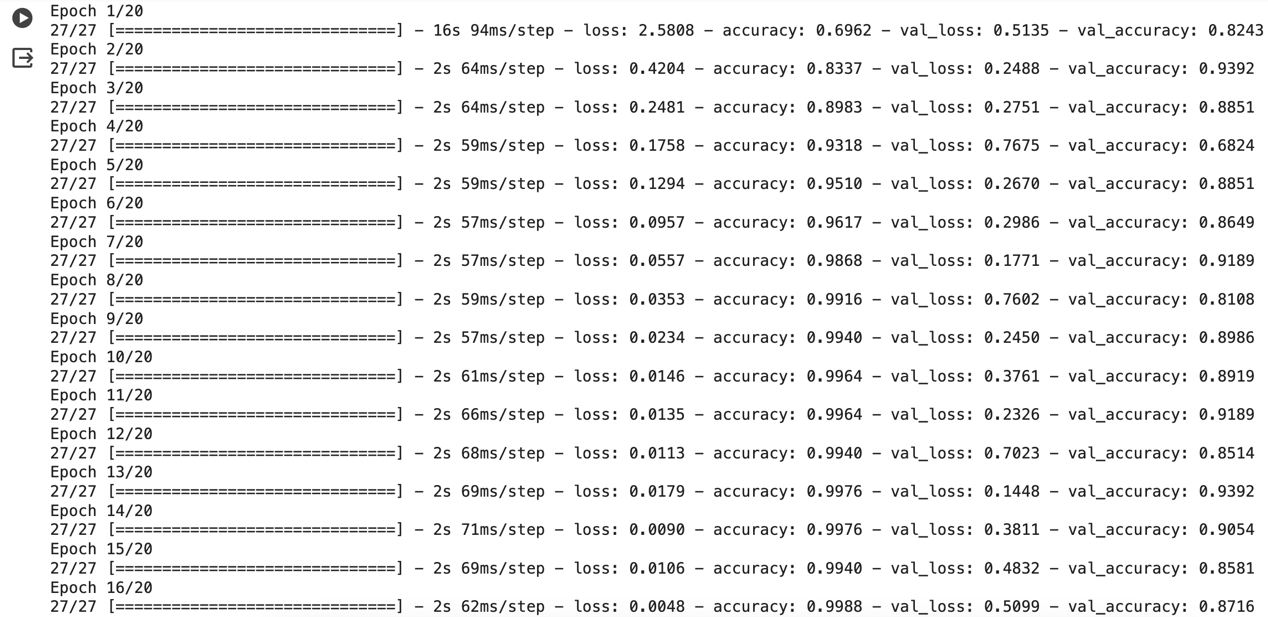
最終得到以下結果：

圖一、資料預處理

一張含有 文字, 螢幕擷取畫面, 字型, 數字 的圖片

自動產生的描述

圖二、Model資訊



圖三、Training資訊

一張含有 文字, 繪圖, 圖表, 行 的圖片

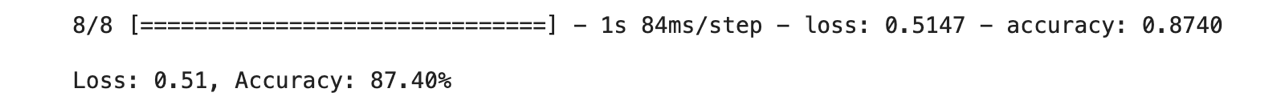
自動產生的描述

圖四、訓練中train和validation的accuracy

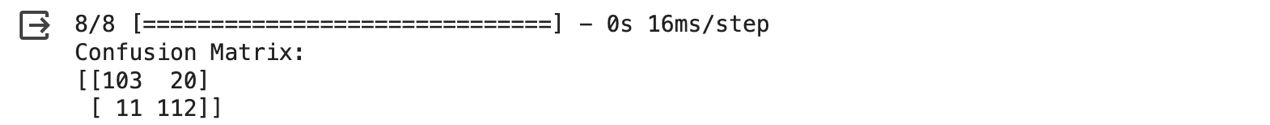
一張含有 文字, 圖表, 繪圖, 螢幕擷取畫面 的圖片

自動產生的描述

圖五、訓練中train和validation的loss



圖六、評估模型準確率:87.4%



圖七、評估模型的 Confusion Matrix

由上面的train和validation的accuracy和loss的折線圖，可看出稍微顯示了過度擬合(overfitting)的特徵。訓練精確度隨著時間線性增長，直到接近100%，然而驗證精確度卻在80 ~ 95%來回跳動，甚至掉到70%以下。驗證損失也是在0.5來回跳動，而訓練損失在線性上保持直到達到接近0。因此，我覺得是因為資料及數量過少的原因導致此現象。我利用在Keras中配置ImageDataGenerator讀取的圖像執行多個隨機變換來完成數據擴充。

數據擴充程式碼(在上面的程式碼內加入)：

# . . .

y\_TestOneHot = to\_categorical(y\_Test)

# 使用ImageDataGenerator進行圖像增強

train\_datagen = ImageDataGenerator(

rescale=1./255,

rotation\_range=40,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

fill\_mode='nearest',

validation\_split=0.15

)

# 資料生成器訓練集和驗證集

train\_generator = train\_datagen.flow\_from\_directory(

directory=Trainpath,

target\_size=(256, 256),

batch\_size=32,

class\_mode='categorical',

subset='training'

)

validation\_generator = train\_datagen.flow\_from\_directory(

directory=Trainpath,

target\_size=(256, 256),

batch\_size=32,

class\_mode='categorical',

subset='validation'

)

train\_generator\_repeated = cycle(train\_generator)

# 設定批次大小和步數

batch\_size = 32

train\_samples = train\_generator.samples

validation\_samples = validation\_generator.samples

steps\_per\_epoch = train\_samples / batch\_size

validation\_steps = validation\_samples / batch\_size

# 自訂leaky\_relu函數

def leaky\_relu(x):

return K.relu(x, alpha=0.05) # alpha 為負值部分的斜率，可以調整

# 建立Sequential模型

model = Sequential()

# 添加卷積層、激活層、池化層、Dropout和全連接層

model.add(Conv2D(filters=16, kernel\_size=(3, 3), padding='same', input\_shape=(256, 256, 3)))

model.add(Activation(leaky\_relu))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(filters=36, kernel\_size=(3, 3), padding='same'))

model.add(Activation(leaky\_relu))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128))

model.add(Activation(leaky\_relu))

model.add(Dropout(0.5))

model.add(Dense(2, activation='softmax'))

model.summary()

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

# 使用資料生成器進行模型訓練

train\_history = model.fit(train\_generator, steps\_per\_epoch=steps\_per\_epoch, epochs=20, validation\_data=validation\_generator, validation\_steps=validation\_steps)

# . . .

# 評估模型準確度

loss\_train, accuracy\_train = model.evaluate(train\_generator, steps=steps\_per\_epoch)

print("\n訓練準確度: %.2f%%" % (accuracy\_train \* 100))

loss\_val, accuracy\_val = model.evaluate(validation\_generator, steps=validation\_steps)

print("\n驗證準確度: %.2f%%" % (accuracy\_val \* 100))

loss\_test, accuracy\_test = model.evaluate(x\_Test4D\_normalize, y\_TestOneHot)

print("\n測試準確度: %.2f%%" % (accuracy\_test \* 100))

# 預測並顯示混淆矩陣

prediction = np.argmax(model.predict(x\_Test4D\_normalize), axis=1)

conf\_matrix = confusion\_matrix(np.argmax(y\_TestOneHot, axis=1), prediction)

print("混淆矩陣:")

print(conf\_matrix)

程式碼說明：

首先，在ImageDataGenerator中設定以下參數：

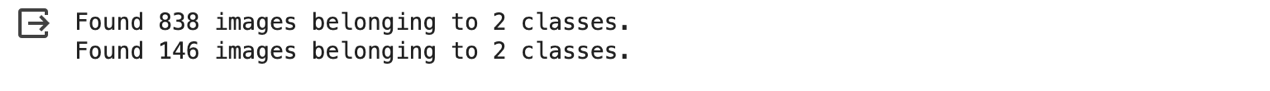
1. rescale = 1./255：將圖像像素值縮放到0和1之間，這有助於模型訓練的穩定性。
2. rotation\_range = 40：隨機旋轉圖像的範圍為40度。
3. width\_shift\_range = 0.2：隨機水平移動圖像的寬度比例為圖像寬度的20%。
4. height\_shift\_range = 0.2：隨機垂直移動圖像的高度比例為圖像高度的20%。
5. shear\_range = 0.2：隨機錯切變換的強度為0.2。
6. zoom\_range = 0.2：隨機縮放圖像的範圍為20%。
7. horizontal\_flip = True：隨機水平翻轉圖像。
8. fill\_mode = 'nearest'：填充新生成像素的方法，使用最接近的像素值。
9. validation\_split = 0.15：將訓練資料集中的一部分（15%）作為驗證資料集。

接著定義資料生成器訓練集和驗證集和定義訓練模型時的參數steps\_per\_epoch和validation\_steps確保訓練的次數有相對應的資料數量。在建立模型部分也有稍作修改，我將原本的relu換成leaky\_relu函數，並且在訓練模型時在train\_history = model.fit()輸入以下參數

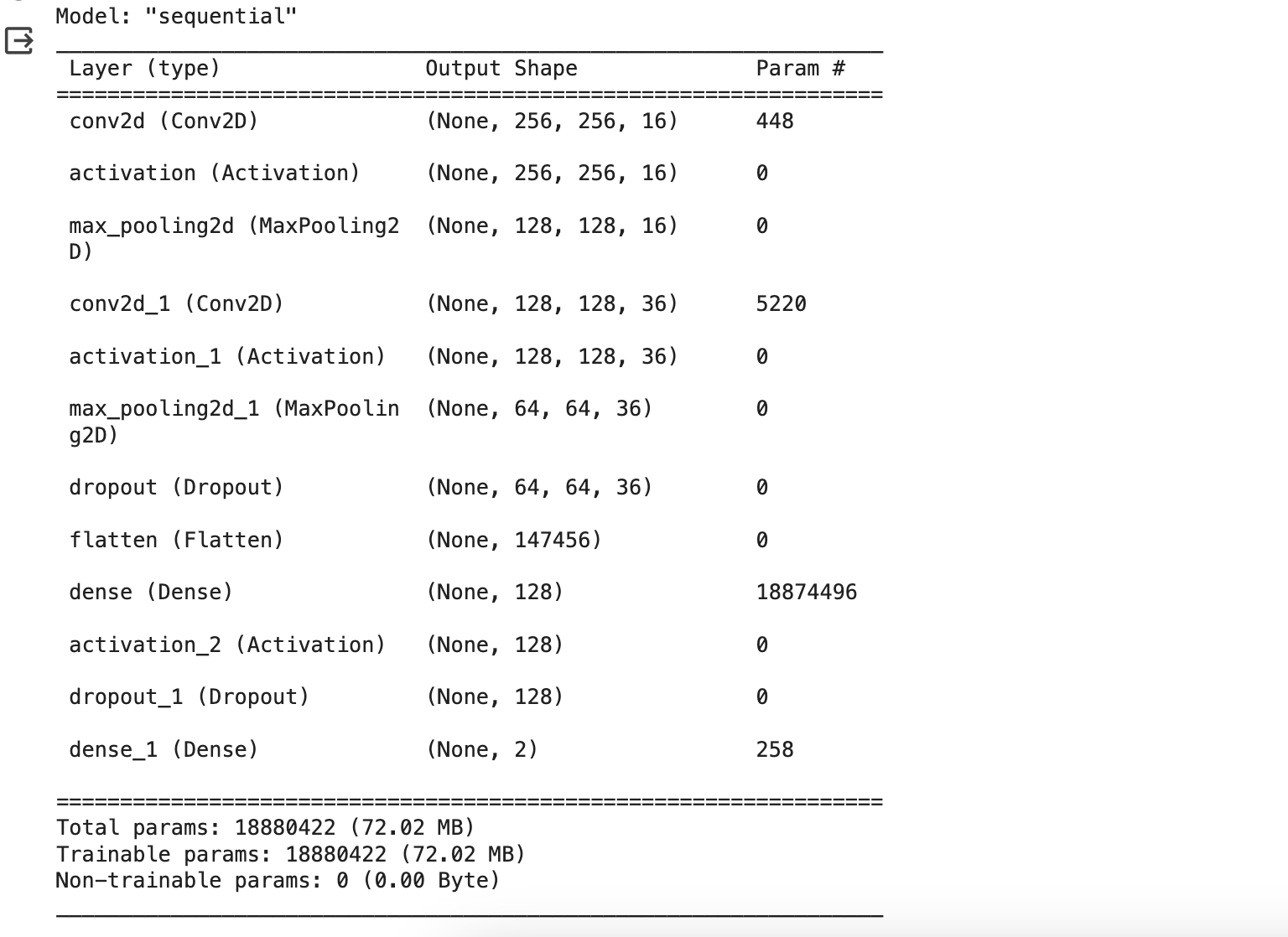
train\_generator, steps\_per\_epoch=steps\_per\_epoch, epochs=20, validation\_data=validation\_generator, validation\_steps=validation\_steps

最後，使用測試資料評估模型準確度並顯示混淆矩陣。

最終得到以下結果：



圖八、train\_generator和validation\_generator資訊



圖九、Model資訊

一張含有 文字, 螢幕擷取畫面, 文件, 黑與白 的圖片

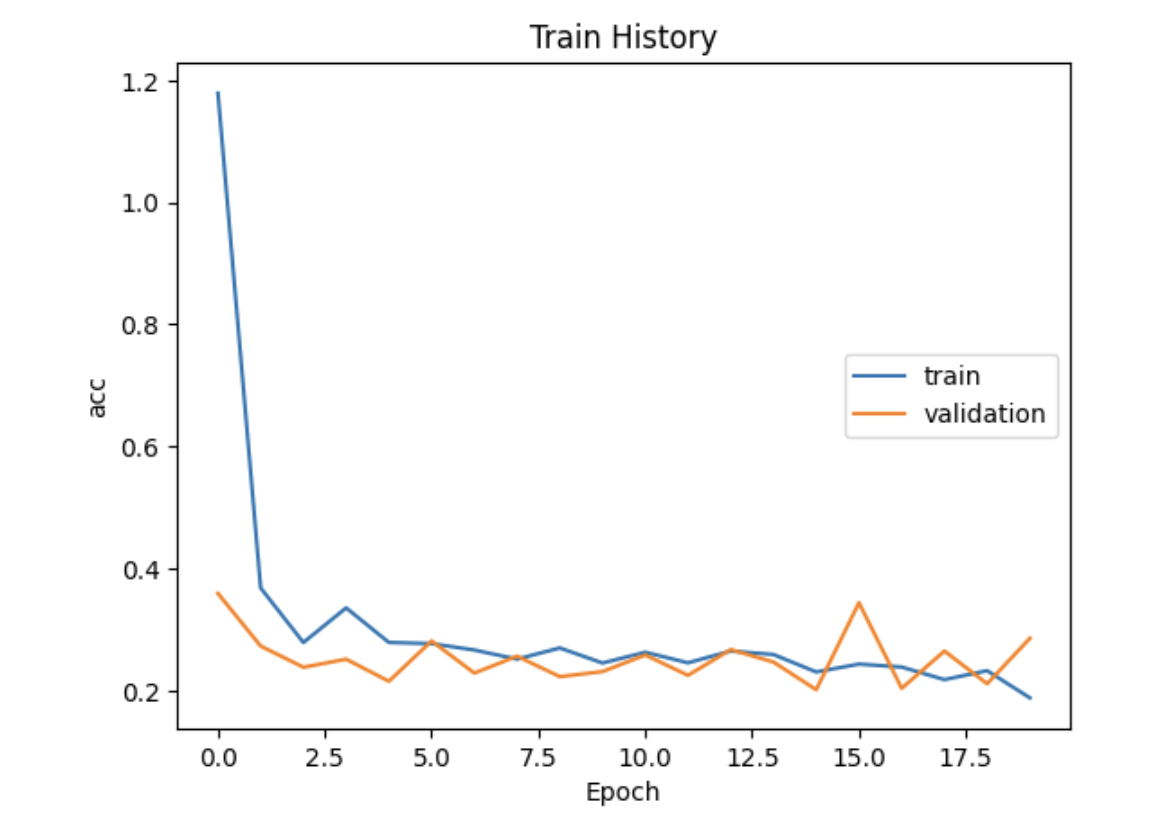
自動產生的描述

圖十、Training資訊

一張含有 文字, 圖表, 繪圖, 行 的圖片

自動產生的描述

圖十一、訓練中train和validation的accuracy



圖十二、訓練中train和validation的loss

一張含有 文字, 收據, 字型, 白色 的圖片

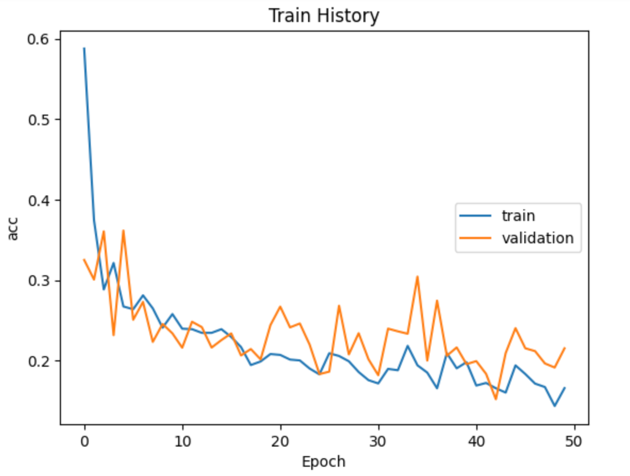
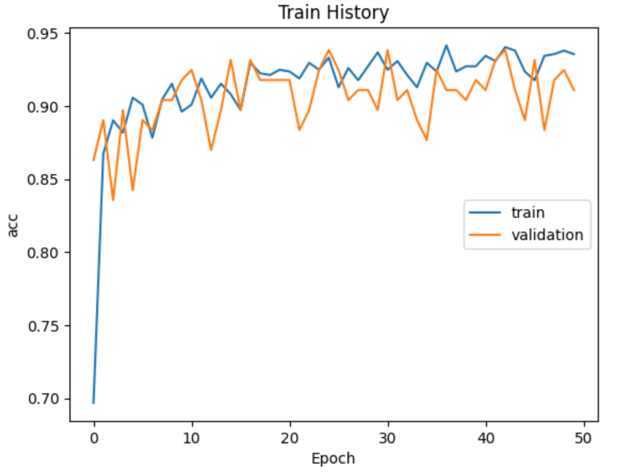
自動產生的描述

圖十三、評估模型準確率:74.8%和Confusion Matrix

可以看出，train和validation的accuracy和loss和沒有經過數據擴充的折線圖相比，有以下特點：

1. Accuracy降低： 因為數據擴充增加了訓練集的多樣性，這可能使得模型更難在訓練集上取得完美擬合。因此，雖然訓練集的精度下降，但這也可能代表模型更好地泛化到未見過的數據。
2. Overfitting減少： 數據擴充引入了更多的變化和多樣性，這可以防止模型對訓練集中的某些特定特徵過度擬合。當模型在訓練時看到更多變化時，其對於特定樣本的過度依賴性會減少，從而減少了在未見過數據上的過擬合風險。
3. 數值擺動幅度減小： 數據擴充能夠平滑化模型在訓練集上的預測結果。當模型在訓練過程中能夠觀察到更多類似但不完全相同的圖像時，其預測結果可能會更加一致，減少了隨機性和不確定性。

補充 (擴充資料後epoch=50)：

一張含有 文字, 收據, 字型, 白色 的圖片

自動產生的描述