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)
```

```python
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
```

```python
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：

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

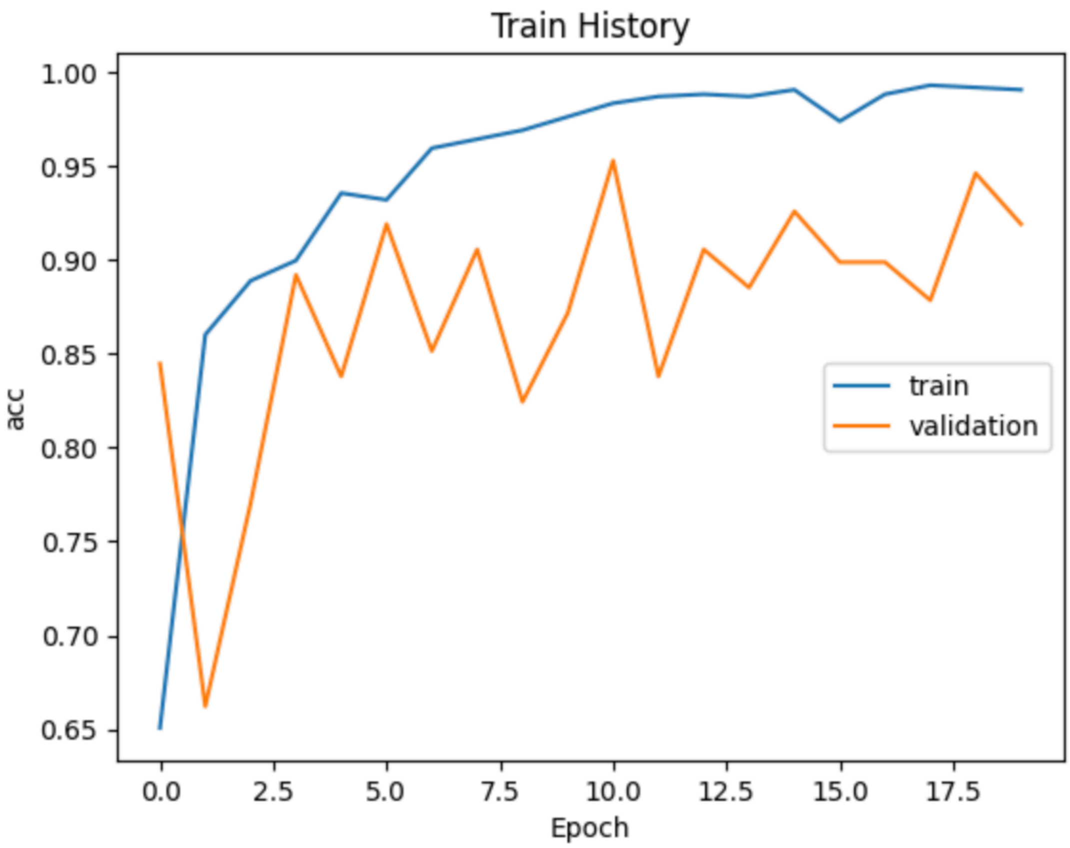
圖一、資料預處理

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 16)      448

 max_pooling2d (MaxPooling2  (None, 128, 128, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 128, 128, 36)      5220

 max_pooling2d_1 (MaxPoolin  (None, 64, 64, 36)        0
 g2D)

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

 flatten (Flatten)           (None, 147456)            0

 dense (Dense)               (None, 128)               18874496

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 18880422 (72.02 MB)
Trainable params: 18880422 (72.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

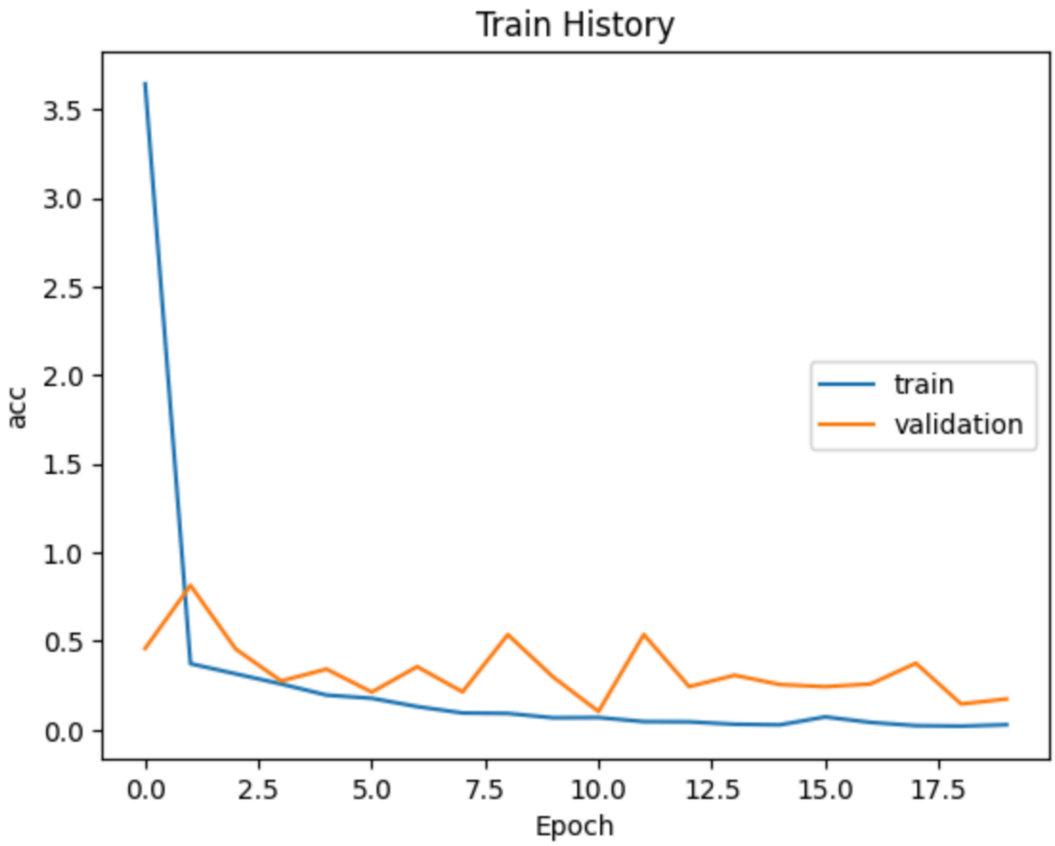
圖二、Model 資訊

```
Epoch 1/20
27/27 [==============================] - 16s 94ms/step - loss: 2.5808 - accuracy: 0.6962 - val_loss: 0.5135 - val_accuracy: 0.8243
Epoch 2/20
27/27 [==============================] - 2s 64ms/step - loss: 0.4204 - accuracy: 0.8337 - val_loss: 0.2488 - val_accuracy: 0.9392
Epoch 3/20
27/27 [==============================] - 2s 64ms/step - loss: 0.2481 - accuracy: 0.8983 - val_loss: 0.2751 - val_accuracy: 0.8851
Epoch 4/20
27/27 [==============================] - 2s 59ms/step - loss: 0.1758 - accuracy: 0.9318 - val_loss: 0.7675 - val_accuracy: 0.6824
Epoch 5/20
27/27 [==============================] - 2s 59ms/step - loss: 0.1294 - accuracy: 0.9510 - val_loss: 0.2670 - val_accuracy: 0.8851
Epoch 6/20
27/27 [==============================] - 2s 57ms/step - loss: 0.0957 - accuracy: 0.9617 - val_loss: 0.2986 - val_accuracy: 0.8649
Epoch 7/20
27/27 [==============================] - 2s 57ms/step - loss: 0.0557 - accuracy: 0.9868 - val_loss: 0.1771 - val_accuracy: 0.9189
Epoch 8/20
27/27 [==============================] - 2s 59ms/step - loss: 0.0353 - accuracy: 0.9916 - val_loss: 0.7602 - val_accuracy: 0.8108
Epoch 9/20
27/27 [==============================] - 2s 57ms/step - loss: 0.0234 - accuracy: 0.9940 - val_loss: 0.2450 - val_accuracy: 0.8986
Epoch 10/20
27/27 [==============================] - 2s 61ms/step - loss: 0.0146 - accuracy: 0.9964 - val_loss: 0.3761 - val_accuracy: 0.8919
Epoch 11/20
27/27 [==============================] - 2s 66ms/step - loss: 0.0135 - accuracy: 0.9964 - val_loss: 0.2326 - val_accuracy: 0.9189
Epoch 12/20
27/27 [==============================] - 2s 68ms/step - loss: 0.0113 - accuracy: 0.9940 - val_loss: 0.7023 - val_accuracy: 0.8514
Epoch 13/20
27/27 [==============================] - 2s 69ms/step - loss: 0.0179 - accuracy: 0.9976 - val_loss: 0.1448 - val_accuracy: 0.9392
Epoch 14/20
27/27 [==============================] - 2s 71ms/step - loss: 0.0090 - accuracy: 0.9976 - val_loss: 0.3811 - val_accuracy: 0.9054
Epoch 15/20
27/27 [==============================] - 2s 69ms/step - loss: 0.0106 - accuracy: 0.9940 - val_loss: 0.4832 - val_accuracy: 0.8581
Epoch 16/20
27/27 [==============================] - 2s 62ms/step - loss: 0.0048 - accuracy: 0.9988 - val_loss: 0.5099 - val_accuracy: 0.8716
```

圖三、Training 資訊



圖四、訓練中 train 和 validation 的 accuracy



圖五、訓練中 train 和 validation 的 loss

```
8/8 [==============================] - 1s 84ms/step - loss: 0.5147 - accuracy: 0.8740

Loss: 0.51, Accuracy: 87.40%
```

圖六、評估模型準確率:87.4%

```
8/8 [==============================] - 0s 16ms/step
Confusion Matrix:
[[103  20]
 [ 11 112]]
```

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：

```
Found 838 images belonging to 2 classes.
Found 146 images belonging to 2 classes.
```

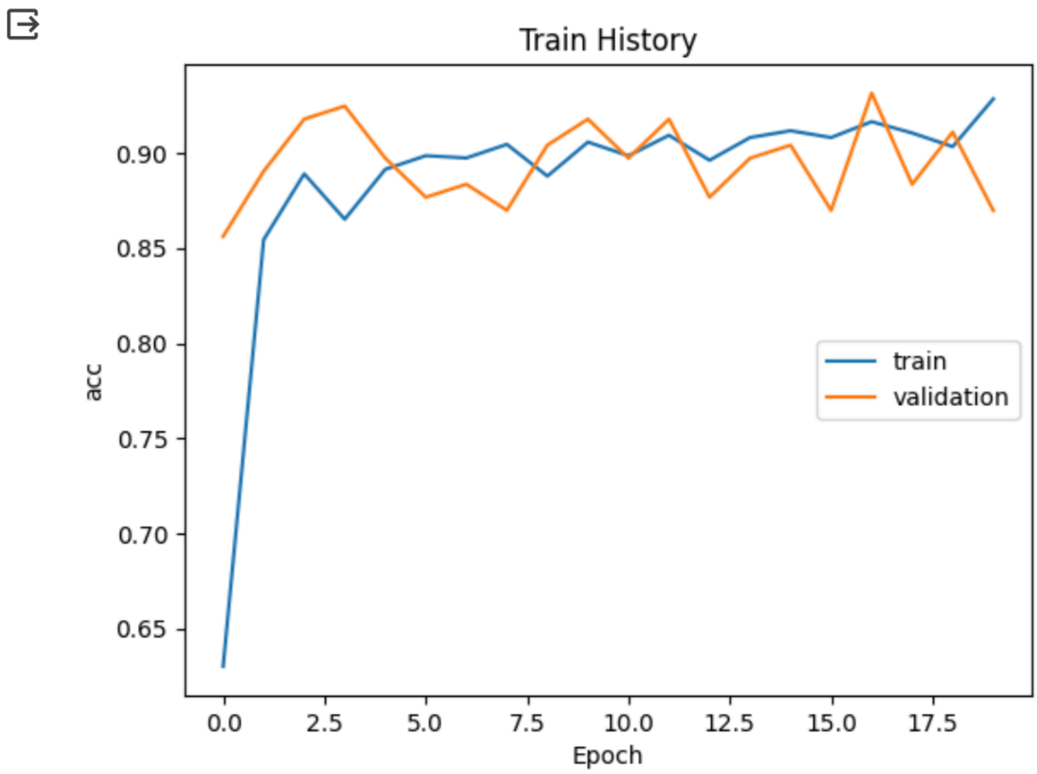
圖八、train_generator 和 validation_generator 資訊

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 256, 256, 16)      448

 activation (Activation)     (None, 256, 256, 16)      0

 max_pooling2d (MaxPooling2  (None, 128, 128, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 128, 128, 36)      5220

 activation_1 (Activation)   (None, 128, 128, 36)      0

 max_pooling2d_1 (MaxPoolin  (None, 64, 64, 36)        0
 g2D)

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

 flatten (Flatten)           (None, 147456)            0

 dense (Dense)               (None, 128)               18874496

 activation_2 (Activation)   (None, 128)               0

 dropout_1 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 2)                 258

=================================================================
Total params: 18880422 (72.02 MB)
Trainable params: 18880422 (72.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

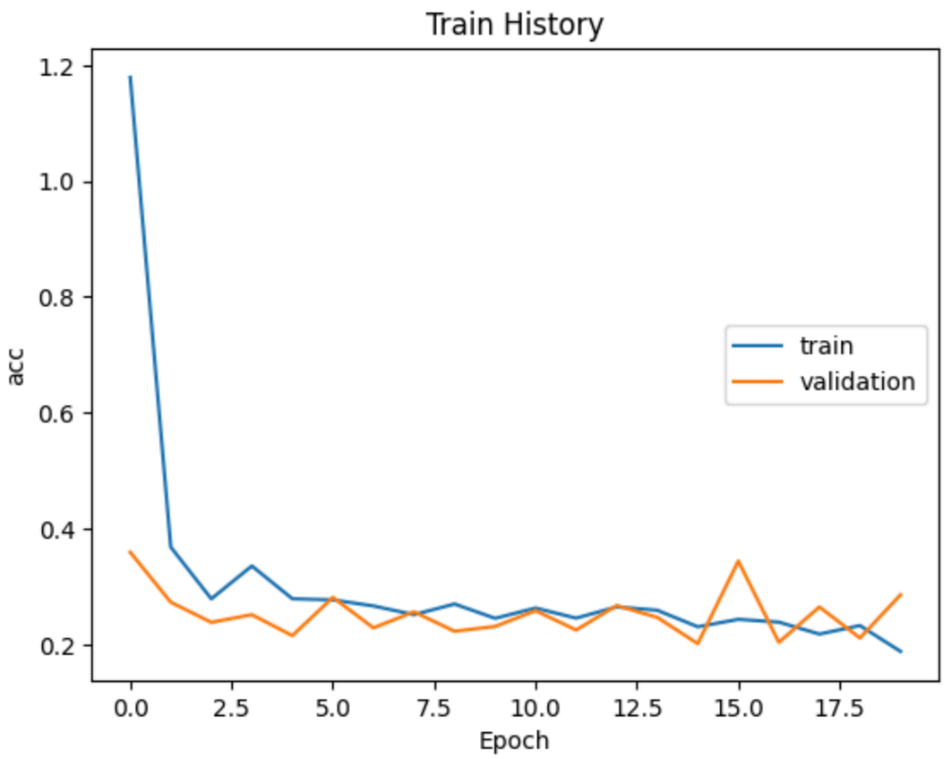
圖九、Model 資訊

```
Epoch 1/20
26/26 [==============================] - 31s 987ms/step - loss: 1.1792 - accuracy: 0.6301 - val_loss: 0.3591 - val_accuracy: 0.8562
Epoch 2/20
26/26 [==============================] - 26s 998ms/step - loss: 0.3681 - accuracy: 0.8544 - val_loss: 0.2729 - val_accuracy: 0.8904
Epoch 3/20
26/26 [==============================] - 27s 1s/step - loss: 0.2787 - accuracy: 0.8890 - val_loss: 0.2381 - val_accuracy: 0.9178
Epoch 4/20
26/26 [==============================] - 24s 912ms/step - loss: 0.3354 - accuracy: 0.8652 - val_loss: 0.2513 - val_accuracy: 0.9247
Epoch 5/20
26/26 [==============================] - 24s 915ms/step - loss: 0.2788 - accuracy: 0.8914 - val_loss: 0.2151 - val_accuracy: 0.8973
Epoch 6/20
26/26 [==============================] - 28s 1s/step - loss: 0.2768 - accuracy: 0.8986 - val_loss: 0.2812 - val_accuracy: 0.8767
Epoch 7/20
26/26 [==============================] - 26s 982ms/step - loss: 0.2664 - accuracy: 0.8974 - val_loss: 0.2285 - val_accuracy: 0.8836
Epoch 8/20
26/26 [==============================] - 24s 947ms/step - loss: 0.2514 - accuracy: 0.9045 - val_loss: 0.2563 - val_accuracy: 0.8699
Epoch 9/20
26/26 [==============================] - 26s 991ms/step - loss: 0.2698 - accuracy: 0.8878 - val_loss: 0.2227 - val_accuracy: 0.9041
Epoch 10/20
26/26 [==============================] - 24s 918ms/step - loss: 0.2450 - accuracy: 0.9057 - val_loss: 0.2310 - val_accuracy: 0.9178
Epoch 11/20
26/26 [==============================] - 25s 958ms/step - loss: 0.2627 - accuracy: 0.8986 - val_loss: 0.2583 - val_accuracy: 0.8973
Epoch 12/20
26/26 [==============================] - 24s 923ms/step - loss: 0.2454 - accuracy: 0.9093 - val_loss: 0.2248 - val_accuracy: 0.9178
Epoch 13/20
26/26 [==============================] - 27s 1s/step - loss: 0.2647 - accuracy: 0.8962 - val_loss: 0.2671 - val_accuracy: 0.8767
Epoch 14/20
26/26 [==============================] - 26s 991ms/step - loss: 0.2590 - accuracy: 0.9081 - val_loss: 0.2467 - val_accuracy: 0.8973
Epoch 15/20
26/26 [==============================] - 24s 911ms/step - loss: 0.2304 - accuracy: 0.9117 - val_loss: 0.2013 - val_accuracy: 0.9041
```

圖十、Training 資訊



圖十一、訓練中 train 和 validation 的 accuracy



圖十二、訓練中 train 和 validation 的 loss

```
26/26 [==============================] - 22s 856ms/step - loss: 0.1822 - accuracy: 0.9224

Training Accuracy: 92.24%
4/4 [==============================] - 4s 787ms/step - loss: 0.2553 - accuracy: 0.8904

Validation Accuracy: 89.04%
8/8 [==============================] - 0s 55ms/step - loss: 0.5505 - accuracy: 0.7480

Test Accuracy: 74.80%
8/8 [==============================] - 0s 20ms/step
Confusion Matrix:
[[89 34]
 [28 95]]
```

圖十三、評估模型準確率:74.8%和 Confusion Matrix

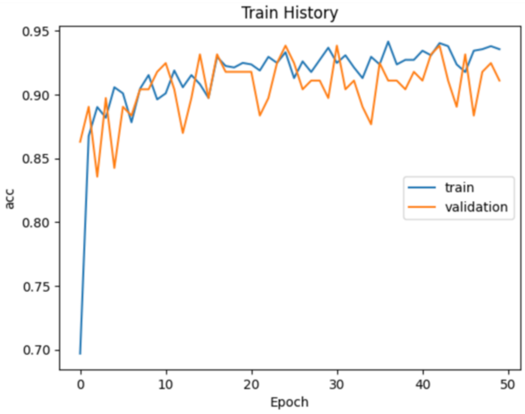
可以看出，train 和 validation 的 accuracy 和 loss 和沒有經過數據擴充的折線圖相比，有以下特點：

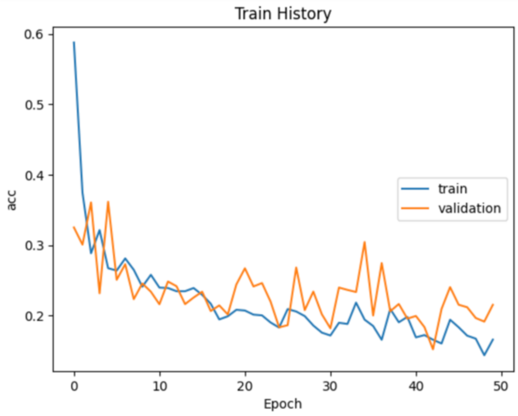
1. Accuracy 降低： 因為數據擴充增加了訓練集的多樣性，這可能使得模型更難在訓練集上取得完美擬合。因此，雖然訓練集的精度下降，但這也可能代表模型更好地泛化到未見過的數據。

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)：






```
26/26 [==============================] - 21s 794ms/step - loss: 0.1409 - accuracy: 0.9439

Training Accuracy: 94.39%
4/4 [==============================] - 5s 1s/step - loss: 0.2244 - accuracy: 0.9178

Validation Accuracy: 91.78%
8/8 [==============================] - 1s 89ms/step - loss: 0.8171 - accuracy: 0.7358

Test Accuracy: 73.58%
8/8 [==============================] - 0s 33ms/step
Confusion Matrix:
[[117   6]
 [ 59  64]]
```