智慧化企業整合_Project2

利用深度學習網路進行肺音異常分類

第六組
109034554 蔡丞洲
109034518 蕭詠仁
109034803 危佳容
109034515 江毓翔

目錄

一、 背景介紹

(一)、 背景說明

醫生可透過聽診器聽取肺部聲音(呼吸音)，並進行肺部疾病和異常診斷，是臨床上一種簡單友好、非侵入式且對患者無害的方法，因此被廣泛使用。但因此種方法評估呼吸音，均仰賴醫生專業及經驗，較主觀，且無法進行定性評估。而肺音信號中蘊藏著豐富的肺部病理及生理上的重要資訊，故本研究將究利用深度學習方法進肺音分類，以客觀的方式判斷呼吸音異常與否，研究結果可作為臨床診斷的輔助。

(二)、 5W1H 分析

Why	診器聽取肺部聲音(呼吸音) 評估呼吸音，較主觀，無法進行定性評估。
What	透過呼吸音辨識，進行肺部疾病診斷
Where	醫院或任何需要進行肺部疾病診斷的地點
When	需要判斷呼吸音是否異常時
Who	醫生或個人
How	利用深度學習網路來進行呼吸音辨識

二、 方法介紹

(一)、 Inception v3

Inception 網絡是 CNN 分類器發展史上一個重要的里程碑。在 Inception 出現之前，大部分流行 CNN 僅僅是把卷積層堆疊得越來越多，使網絡越來越深，以此希望能夠得到更好的性能。

例如第一個得到廣泛關注的 AlexNet，它本質上就是擴展 LeNet 的深度，並應用一些 ReLU、Dropout 等技巧。AlexNet 有 5 個卷積層和 3 個最大池化層，它可分為上下兩個完全相同的分支，這兩個分支在第三個卷積層和全連接層上可以相互交換信息。與 Inception 同年提出的優秀網絡還有 VGG-Net，它相比於 AlexNet 有更小的卷積核和更深的層級。

VGG-Net 的泛化性能非常好，常用於圖像特徵的抽取目標檢測候選框生成等。VGG 最大的問題就在於參數數量，VGG-19 基本上是參數量最

多的卷積網絡架構。這一問題也是第一次提出 Inception 結構的 GoogLeNet 所重點關注的，它沒有如同 VGG-Net 那樣大量使用全連接網絡，因此參數量非常小。

GoogLeNet 最大的特點就是使用了 Inception 模塊，它的目的是設計一種具有優良局部拓撲結構的網絡，即對輸入圖像並行地執行多個卷積運算或池化操作，並將所有輸出結果拼接為一個非常深的特徵圖。因為 1*1、3*3 或 5*5 等不同的卷積運算與池化操作可以獲得輸入圖像的不同信息，並行處理這些運算並結合所有結果將獲得更好的圖像表征。

Inception v3

Inception v3 針對前一代 Inception v2 進行一些改進。

1. 避免在網絡的前面使用像 bottleneck 的結構，也就是說 size 需要緩慢的降低，不能一開始降低的太快，這樣會丟失很多信息。
2. 對高維度的表達進行局部處理，會加快網絡的訓練速度。
3. 在較低維度的輸入上進行空間聚合，將不會造成任何表達能力上的損失，因爲 feature map 上，臨近區域上的表達具有很高的相關性，如果對輸出進行空間聚合，那麼將 feature map 的維度降低也不會減少表達的信息。這樣的話，有利於信息的壓縮，並加快了訓練的速度。
4. 平衡網絡的寬度和深度。增加網絡的寬度和深度對網絡的精確度提升都有幫助，但是如果同時增加寬度和深度，計算量會增大，因此在有限的計算資源下，應該平衡網絡的寬度和深度。

Inception V3 一個最重要的改進是卷積分解（Factorization），將 7x7 卷積分解成兩個一維的卷積串聯（1x7 和 7x1），3x3 卷積分解爲兩個一維的卷積串聯（1x3 和 3x1），這樣既可以加速計算，又可使網絡深度進一步增加，增加了網絡的非線性即為每增加一層都要進行 ReLU。另外，網絡輸入從 224x224 變爲 299x299。其他也增加了輔助分類器(BatchNorm)、標籤平滑和 RMSProp 優化器，圖 1 為 Inception v3。

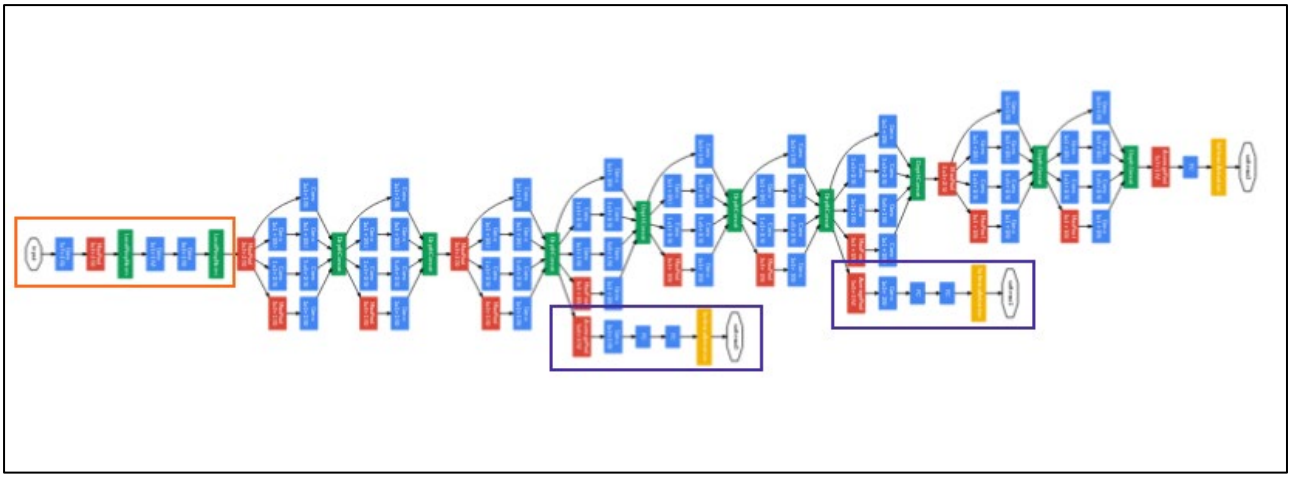


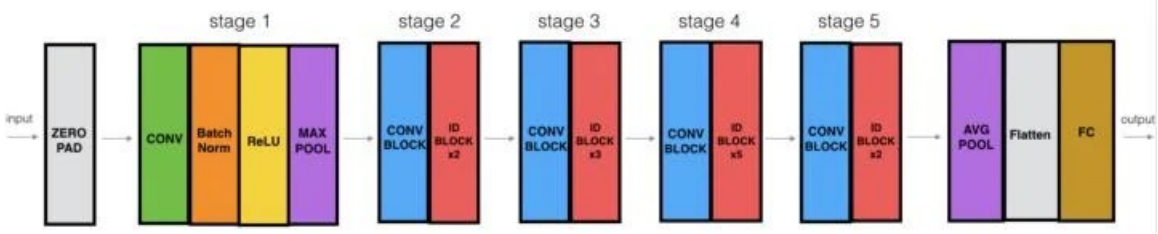
圖 1、Inception v3 (資料來源：https://kknews.cc/code/zko5e5p.html)

(二)、 ResNet-50

ResNet 在 2015 年被提出，在 ImageNet 比賽 classification 任務上獲得第一名，因爲它“簡單與實用”並存，之後很多方法都建立在 ResNet50 或者 ResNet101 的基礎上完成的，檢測，分割，識別等領域都紛紛使用 ResNet，Alpha zero 也使用了 ResNet，所以可見 ResNet 確實很好用。

ResNet-50 網路是參考了 VGG19 網路，在其基礎上進行了修改，並通過短路機制加入了殘差單元。變化主要體現在 ResNet 直接使用 stride=2 的卷積做下采樣，並且用 global average pool 層替換了全連線層。ResNet 的一個重要設計原則是：當 feature map 大小降低一半時，feature map 的數量增加一倍，這保持了網路層的複雜度。

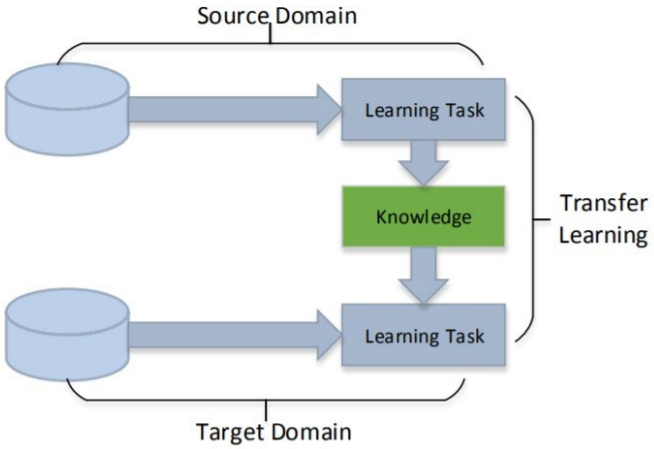
另外考慮當神經網路深度加到一定程度時，會造成模型的不敏感，導致 Loss 值不降反升，因此將殘差網路引進 ResNet 模型中。殘差學習需要學習的內容較少，更容易進行訓練，並且如果在這層網路沒有學習到新的特徵，也會保證網路不會退化，表示模型深度越深，成果也會越好。



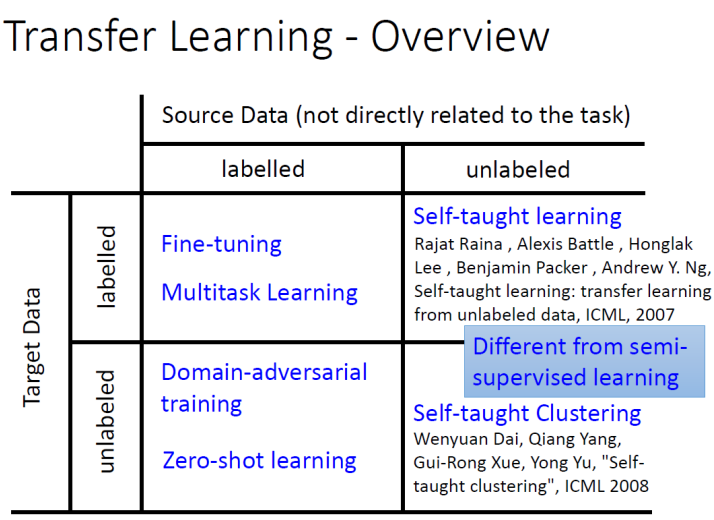
layer name	output size	50-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2 [1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[1×1, 128 3×3, 128 1×1, 512] ×4
conv4_x	14×14	[1×1, 256 3×3, 256 1×1, 1024] ×6
conv5_x	7×7	[1×1, 512 3×3, 512 1×1, 2048] ×3

(三)、 遷移學習(Transfer learning)

遷移學習(Transfer learning)是把已訓練好的模型參數遷移到新的模型來幫助新模型訓練。考慮到大部分數據或任務都是存在相關性的，所以通過遷移學習我們可以將已經學到的模型參數通過某種方式來分享給新模型，從而加快並優化模型的學習效率，不用像大多數網絡那樣從零學習，簡單來說就是學習 Source Domain 並應用在 Target Domain 上，來使訓練更加快速。



在遷移學習上分為四種使用方法，主要差別來自於資料是否有標註，而本專題使用的為 Fine-tuning 的方式進行遷移學習。

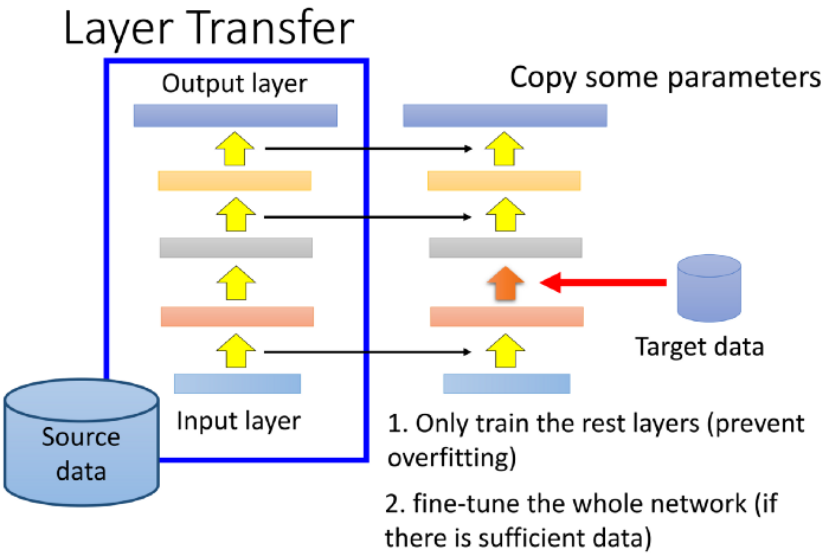


Fine Tuning

Fine Tuning 為目前深度學習(Deep Learning)常見的使用方法，並應用在圖像分類當中。使用方式為使用已經訓練完成的資料集(Source data)作為目標資料(Target data)的 pre-trained model，並且對參數做一些微調。但如果目標資料過少可能會導致 overfitting 的問題，我們使用的方式為 Layer Transfer。

Layer Transfer 一樣是透過使用 Source data 訓練好的模型，但是不把整個模型都放過來，而是萃取其中幾層的參數，其餘沒有萃取的參數再透過訓練 Target data 來取得。通常在語音辨識任務中，會 transfer 模型最後幾層的參數，因為對語音信號來說，每個人使用同樣的發音方式，所得到的聲音結構是不一樣的，原因是人類口腔結構、肺部組織會存在差異，在訓練中，網路前幾層的目的主要就是便是人類的發音方式，後幾層才是去聽人類說了甚麼，且與發音者無關。

在本專題中，Source data 為 ImageNet，Target data 為 TSECC_Tung Sound，使用方式是將已經訓練完的 ImageNet 取代我們模型的全連接層，來使訓練 TSECC_Tung Sound 的資料時，能有更好且更快速的結果。



https://hackmd.io/@allen108108/H1MFrV9WH

三、 個案研究

(一)、 資料介紹

本次專題所使用的 dataset 是現有的資料，其原檔案有 10742 張訓練集圖片及 3403 張測試集圖片檔(.tiff)及尚未對應的 label 名稱檔(.txt)，而現有資料是透過短時距傅立葉轉換將收集到的音頻檔轉換為頻譜圖，其中 nfft=256 表示轉換時，在每一幀進行 512 點快速傅立葉轉換，相鄰幀數交疊大小為 192ms，最後轉出的圖片大小為 129×938。而 label 檔中有異常的資料為呼吸聲中連續的不定聲音(CAS)，檔案中會將兩種情況的有無透過 1 表示有 0 表示沒有列出，表一為檔案統計資料。

表一情緒張數表

情緒	張數
正常(無 CAS)	3221
異常(有 CAS)	7521

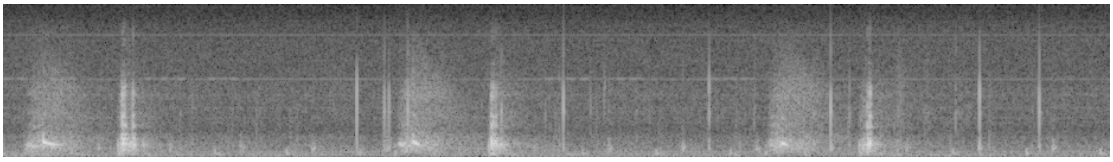


圖 2、可視化頻譜圖

(二)、 資料前處理

Step1:

由於資料原先給予的訓練及測試圖片檔並不是 jpg 檔，所以先個別將訓練及測試的圖片檔案轉換成 jpg 以方便訓練，接著將所有圖片檔案轉為 nparray 的形式存成 npy 檔，最後將所有 label 的.txt 檔整合存成.json 檔以方便標籤。

```python
import os
path=input('請輸入文件路徑：')+'\\'
#獲取該目錄下所有文件，存入列表中
f=os.listdir(path)
n=0
for i in f:
    #設置舊文件名（就是路徑+文件名）
    oldname=path+f[n]
    #設置新文件名
    newname=path+"train_"+str(n+1)+'.JPG'
    #用os模塊中的rename方法對文件改名
    os.rename(oldname,newname)
    print(oldname,'已經改名為：',newname)
    n+=1
```

圖 3、tiff 檔轉成.jpg 檔

```python
import numpy as np
import cv2
import os
import json

img = cv2.imread(r'C:\Users\rh\Desktop\VOICE\pre_process\1.JPG')
print(img.shape)
concat = np.reshape(img, (1, 129, 934, 3))
print(concat.shape)


path = r"C:\Users\rh\Desktop\VOICE\test_pic_1"
files = os.listdir(path)
print(files)

for file in files:
 img_path = path +"\\"+ file
 img = cv2.imread(img_path)
 print(img_path)
 img_re = np.reshape(img, (1, 129, 934, 3))
 concat = np.concatenate((concat, img_re), axis = 0)


print(concat.shape)

with open(r"C:\Users\rh\Desktop\test_img.npy", "wb") as fpw:
 np.save(fpw, concat)

with open(r"C:\Users\rh\Desktop\test_img.npy", "rb") as fpr:
    c = np.load(fpr)

print(c.shape)


img = cv2.imread(r'C:\Users\rh\Desktop\VOICE\pre_process\train_1.JPG')
print(img.shape)
concat = np.reshape(img, (1, 129, 934, 3))
print(concat.shape)


path = r"C:\Users\rh\Desktop\VOICE\train_pic_1"
files = os.listdir(path)
```

```
print(files)


for file in files:
 img_path = path +"\\"+ file
 img = cv2.imread(img_path)
 print(img_path)
 img_re = np.reshape(img, (1, 129, 934, 3))
 concat = np.concatenate((concat, img_re), axis = 0)


print(concat.shape)


with open(r"C:\Users\rh\Desktop\train_img.npy", "wb") as fpw:
  np.save(fpw, concat)

with open(r"C:\Users\rh\Desktop\train_img.npy", "rb") as fpr:
    c = np.load(fpr)

print(c.shape)
```

圖 4、將圖片檔轉為 nparray

```
import os
import json

path=r"C:\Users\rh\Desktop\VOICE\train_label"
files = os.listdir(path)

#print(len(files))

s = []

for file in files:
  fname = os.path.splitext(file)[0]
  with open(path+"//"+ file, "r") as test:
    content = test.readline()

    a=content[5]
    s.append(a)

print(s)
print(len(s))
path2=r"C:\Users\rh\Desktop\VOICE\train.json"

with open(path2,"w") as f:
    json.dump(s, f)

#print(content)
```

圖 5、統整標籤檔案

Step2:

接著由於原本的圖片 size(129×938)並不能直接放入神經網路中，所以將圖片利用 resize 來改變圖片的尺寸大小成(64×64)，結果如下圖，最後再將 label 利用 one hot encoder 的方式轉碼使其 shape 能夠與圖片的矩陣對上結果如下圖。

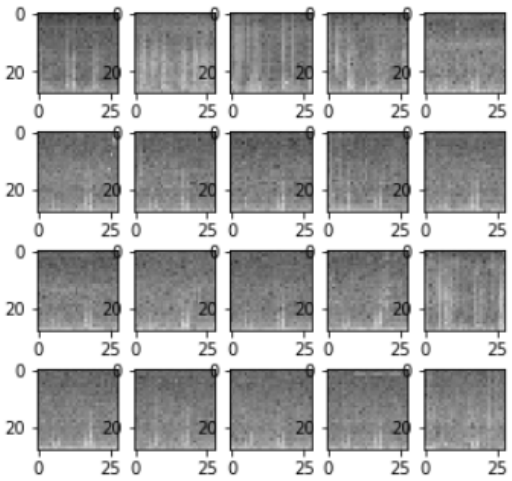


圖 6、可視化頻譜圖

```
#One hot encoder
from keras.utils import np_utils
y_train_label=np_utils.to_categorical(y)
Y_test_label=np_utils.to_categorical(Y_test)
```

```
Using TensorFlow backend.
```

```
x.shape, y_train_label.shape, X_test.shape, Y_test_label.shape
```

```
((10742, 28, 28, 3), (10742, 2), (3403, 28, 28, 3), (3403, 2))
```

圖 7、轉碼程式與輸出圖片及 label 的 shape

Step3:

進行資料的標準化，讓所有的特徵值介於 0 到 1 之間，使梯度運算時能夠更快收斂，降低運算速度，程式碼如下圖，並且為了避免訓練張數太少導致 Overfitting 的情況透過 Data augmentation 的方式增加訓練及驗證張數，其生成的張數會依照不同 batch size 的大小在每一個 epoch 而有變化其生成張數的設定公式為原有 train 的張數乘上所設定的倍數除以 batch size 的商數為 Data augmentation 所生成的張數，其程式碼如下。

```
# Normalize the data
# X_train = np.resize(X_train,(, 100,100,3))
# X_train = np.resize(X_train,(X_train.shape))
X_train = x / 255.0
X_test = X_test / 255.0
print("x_train shape: ",X_train.shape)
print("X_test shape: ",X_test.shape)
```

```
x_train shape:  (10742, 28, 28, 3)
X_test shape:  (3403, 28, 28, 3)
```

圖 8、轉碼程式與輸出圖片及 label 的 shpae

```
#data augmentation
datagen = ImageDataGenerator(
        rotation_range=5,  # randomly rotate images in the range 5 degrees
        zoom_range = 0.1, # Randomly zoom image 10%
        width_shift_range=0.1,  # randomly shift images horizontally 10%
        height_shift_range=0.1,  # randomly shift images vertically 10%
        horizontal_flip=False,  # randomly flip images
        vertical_flip=False)  # randomly flip images
#datagen.fit(X_train)
```

圖 9、轉碼程式與輸出圖片及 label 的 shpae

(三)、 模型建立與訓練

接著將建立 DNN 模型，因為再過去沒有人利用此資料集做過相關的訓練或是研究，所以先透過建立淺層 DNN 模型來看看結果如何，再藉由調整參數的方式來改變，避免一開始透過太複雜的模型來訓練，導致神經網路不再進行學習，其架構如下圖，其他參數如下表，訓練則是利用交叉驗證的方式，並將訓練集與測試集比例調成 9:1，其所設定的程式碼如下。

```
model = Sequential(
    [
        Flatten(),
        Dense(16, activation = 'sigmoid'),
        Dropout(0.5),
        Dense(8, activation = 'sigmoid'),
        Dropout(0.5),
        Dense(2, activation='sigmoid')
    ]
)
```

圖 10、DNN 模型

表二、其他超參數設定

參數	數值
Epochs	10
Batch size	16
Learning rate	0.000001
Loss	categorical_crossentropy

```
seed = 7
np.random.seed(seed)
kfold = StratifiedKFold(n_splits = 10, shuffle = True, random_state = seed)
```

```
cvscores = []
i = 0
for train, test in kfold.split(X_train, y_train_label.argmax(1)):
    i += 1
    if i == 5:
        break
  # create model
#    print(1)
#    print(y_train_label[test].shape)
    model=model
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(0.000001), metrics=['accuracy'])
    # Fit the model
    #print(X_train[train].shape)
    #print(y_train_label[train].shape)
    datagen.fit(X_train[train])
    history = model.fit(X_train[train], y_train_label[train],
                        datagen.flow(X_train[train], y_train_label[train], batch_size= batch_size), epochs=10,
                        steps_per_epoch=(X_train[train].shape[0]*2//16), verbose=2)
    # evaluate the model
    scores = model.evaluate(X_train[test], y_train_label[test], verbose=2)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

圖 11、訓練程式碼

(四)、 參數優化

初步訓練結果所設定參數如下表，透過交叉驗證結果可以看出模型還有可訓練的空間，從訓練圖表來看也可以看出模型也有待訓練的空間。

表三、參數設定表_1

可調整參數或函數	初步所選擇參數或函數
圖片 Resize 的大小	128,128
每個 epoch 資料增強張數	9661 張
資料分群	10(訓練 9:驗證 1)
交叉驗證次數	4
Random seed	7
Epoch	10
Batch size	16
Activation function	Relu, Sigmoid
Optimizer	Adam
Loss	Categorical crossentropy
Dense neural	32-16-8-2
Dropout rate	0.4-0.5-0.5
Learning rate	0.000001

```
1208/1208 - 409s - loss: 0.6304 - acc: 0.6952
Epoch 3/10
1208/1208 - 409s - loss: 0.6297 - acc: 0.6956
Epoch 4/10
1208/1208 - 410s - loss: 0.6288 - acc: 0.6960
Epoch 5/10
1208/1208 - 408s - loss: 0.6279 - acc: 0.6964
Epoch 6/10
1208/1208 - 406s - loss: 0.6271 - acc: 0.6967
Epoch 7/10
1208/1208 - 405s - loss: 0.6264 - acc: 0.6970
Epoch 8/10
1208/1208 - 403s - loss: 0.6255 - acc: 0.6973
Epoch 9/10
1208/1208 - 394s - loss: 0.6247 - acc: 0.6976
Epoch 10/10
1208/1208 - 391s - loss: 0.6239 - acc: 0.6978
1074/1074 - 0s - loss: 0.6090 - acc: 0.7002
acc: 70.02%
70.01% (+/- 0.03%)
```

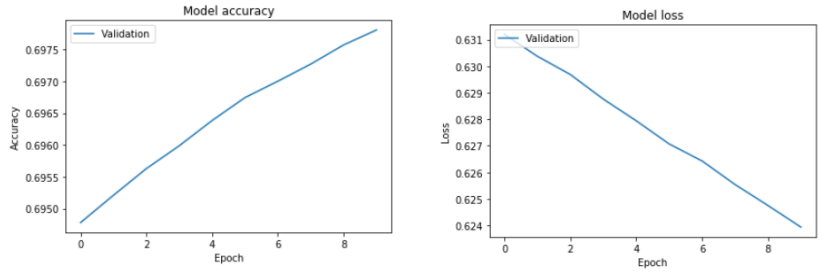


圖 12、初步訓練結果

```
model.evaluate(X_test, Y_test_label)
```

```
3403/3403 [==============================] - 0s 125us/sample - loss: 0.6198 - acc: 0.6873
[0.6198394935060311, 0.6873347]
```

圖 4.2 初步訓練測試集結果

接著進行第一次的參數調整，由於經過多次調整參數與神經網路後利用圖片(128×128)的改善結果都不大，所以將圖片 size 改小成(64× 64)，並增加一層捲機看看結果是否得到改善，調整參數表如下。

表四、參數設定表_2

可調整參數或函數	參數調整所選擇參數或函數
圖片 Resize 的大小	64,64
每個 epoch 資料增強張數	9661 張
資料分群	10(訓練 9:驗證 1)
交叉驗證次數	4
Random seed	42
Epoch	20
Batch size	16

Activation function	Relu, Sigmoid
Optimizer	Adam
Loss	Categorical crossentropy
Dense neural	32-16-8-2
Convolution layer	2,2
Pooling layer	Max pooling(2,2)
Dropout rate	0.4-0.5-0.5
Learning rate	0.000001

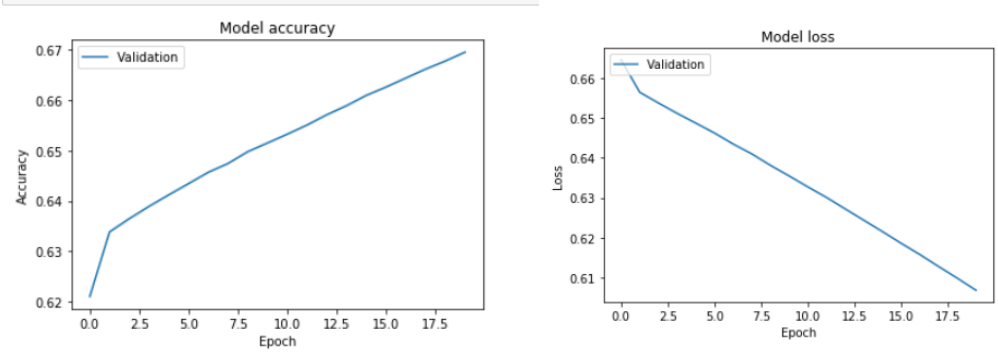
其他使用函數

所使用函數	名稱
各層 kernel 之初始化	he_normal
各層 CNN 之正規化	Batch Normalization
自調整學習率	ReduceLROnPlateau

使用 kernel 初始化是希望 CNN 在進行捲積的過程中，初始參數不是隨機設定，而是希望他能夠服從常態分配，而 Batch Normalization 是為了防止特徵經過激活函數後，會過度激活使特徵會太敏感，讓較不明顯的特徵也被激活，透過 Batch Normalization 也可以讓學習結果收斂得更好，而自調整學習率是希望當神經網路在經過幾個 epoch 後，已經沒辦法再學習，此時需要降低 learning rate，讓梯度能夠繼續下降，使 NN 繼續學習。

```
model = Sequential(
    [
        Conv2D(filters=16, kernel_size=(2, 2), padding='same', activation='sigmoid',kernel_initializer='random_normal'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Flatten(),
        Dense(32, activation = 'relu'),
        Dropout(0.4),
        Dense(16, activation = 'relu'),
        Dropout(0.5),
        Dense(8, activation = 'relu'),
        Dropout(0.5),
        Dense(2, activation='sigmoid')
    ]
)
```

圖 13、初步模型修改



```
model.evaluate(X_train, y_train_label)
10742/10742 [==============================] - 1s 137us/sample - loss: 0.5849 - acc: 0.7101
[0.5849015384363165, 0.71010983]
```

圖 14、初步參數調整後結果

```
model.evaluate(X_test, Y_test_label)
3403/3403 [==============================] - 0s 142us/sample - loss: 0.6158 - acc: 0.6873
[0.615792409500724, 0.6873347]
```

圖 15、初步參數調整後測試集結果

經過第一次的參數調整後，驗證集準確率提高至 0.7023，test 的 loss 也有比初始訓練結果下降了一些，且模型也有在學習的趨勢，尚未有 overfitting 的情況發生，所以接著進行第二次的參數調整及模型改善，調整參數如下表。

表五、參數設定表_3

可調整參數或函數	初步所選擇參數或函數
圖片 Resize 的大小	64,64
每個 epoch 資料增強張數	9661 張
資料分群	10(訓練 9:驗證 1)
交叉驗證次數	4
Random seed	42
Epoch	30
Batch size	16
Activation function	Relu, Sigmoid
Optimizer	Adam
Loss	Categorical crossentropy
Dense neural	16-2
Convolution layer(3)	(3,3)-(2,2)-(2,2)
Pooling layer	Max pooling(2,2)
Dropout rate	0.5
Learning rate	0.000001

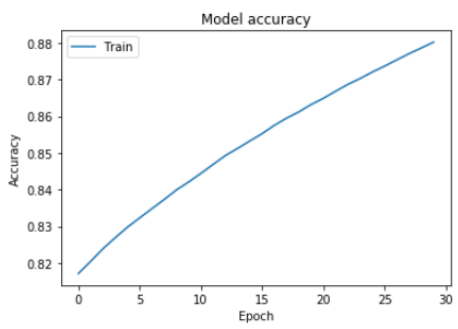
```
model = Sequential(
    [
        Conv2D(filters=16, kernel_size=(3, 3), padding='same', activation='relu',kernel_initializer='random_normal'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Conv2D(filters=16, kernel_size=(2, 2), padding='same', activation='sigmoid',kernel_initializer='random_normal'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        Conv2D(filters=32, kernel_size=(2, 2), padding='same', activation='relu',kernel_initializer='random_normal'),
        MaxPooling2D(pool_size=(2, 2)),
        BatchNormalization(),
        #Dropout(0.5),
        Flatten(),
#         Dense(32, activation = 'relu'),
#         Dropout(0.4),
        Dense(16, activation = 'relu'),
        Dropout(0.5),
#         Dense(8, activation = 'sigmoid'),
#         Dropout(0.5),
        Dense(2, activation='sigmoid')
    ]
)
```

圖 16、第二次模型修改

結果如下圖，可以看出模型出現有點出現 over fitting 的情況，但是情況並沒有很嚴重，validation 與 training 的表現結果沒有差到很多，可是在 testing 上模型的正確率卻跟前幾次差異不大，可是 loss 的表現卻是最好的。

```
cvscores = []
i = 0
for train, test in kfold.split(X_train, y_train_label.argmax(1)):
    i += 1
    if i == 4:
        break
  # create model
#    print(1)
#    print(y_train_label[test].shape)
    model=model
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(0.000001), metrics=['accuracy'])
    # Fit the model
    #print(X_train[train].shape)
    #print(y_train_label[train].shape)
    datagen.fit(X_train[train])
    history = model.fit(X_train[train], y_train_label[train], datagen.flow(X_train[train], y_train_label[train], batch_size= bat
    # evaluate the model
    scores = model.evaluate(X_train[test], y_train_label[test], verbose=2)
    print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    cvscores.append(scores[1] * 100)
print("%.2f%% (+/- %.2f%%)" % (np.mean(cvscores), np.std(cvscores)))
```

```
1208/1208 - 647s - loss: 0.2833 - acc: 0.8668
Epoch 23/30
1208/1208 - 644s - loss: 0.2800 - acc: 0.8687
Epoch 24/30
1208/1208 - 644s - loss: 0.2769 - acc: 0.8703
Epoch 25/30
1208/1208 - 644s - loss: 0.2736 - acc: 0.8721
Epoch 26/30
1208/1208 - 635s - loss: 0.2705 - acc: 0.8737
Epoch 27/30
1208/1208 - 635s - loss: 0.2673 - acc: 0.8754
Epoch 28/30
1208/1208 - 635s - loss: 0.2642 - acc: 0.8771
Epoch 29/30
1208/1208 - 635s - loss: 0.2612 - acc: 0.8786
Epoch 30/30
1208/1208 - 635s - loss: 0.2582 - acc: 0.8802
1074/1074 - 0s - loss: 0.3750 - acc: 0.8212
acc: 82.12%
75.75% (+/- 4.62%)
```



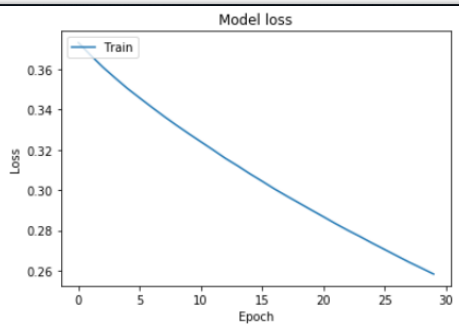


圖 17、最終參數調整後結果

```
model.evaluate(X_test, Y_test_label)
3403/3403 [==============================] - 1s 150us/sample - loss: 0.6115 - acc: 0.6844
[0.6114518211373434, 0.68439615]
```

圖 18、最終參數調整後測試集結果

(五)、 方法比較

由於所使用的模型是自己所建立的，在調整多次後，都無法達到更好的改善，所以試著利用 Transfer learning 的方式來與我們所找的結果進行比較，看看是否能有所改善，其中的為了較公平的比較，Transfer learning 與我們建立 Model 只差在模型不同，其餘剩下的超參數如 epoch、learning rat 等皆為最終參數調整後的值，我們選擇 Resnet50 與 Google Inception V3 為 Transfer learning 的 model，初始權重使用這兩個 Model 在 ImageNet 上所記錄下來的學習結果，其結果如下。

```
Epoch 00026: ReduceLROnPlateau reducing learning rate to 4.782969028838124e-07.
1209/1208 - 106s - loss: 0.6072 - accuracy: 0.7000
Epoch 27/30
1209/1208 - 106s - loss: 0.6083 - accuracy: 0.7005
Epoch 28/30
1209/1208 - 107s - loss: 0.6083 - accuracy: 0.7004
Epoch 29/30

Epoch 00029: ReduceLROnPlateau reducing learning rate to 4.304672074795235e-07.
1209/1208 - 106s - loss: 0.6078 - accuracy: 0.7000
Epoch 30/30
1209/1208 - 105s - loss: 0.6073 - accuracy: 0.7001
34/34 - 10s - loss: 0.5959 - accuracy: 0.7002
accuracy: 70.02%
70.01% (+/- 0.04%)
```

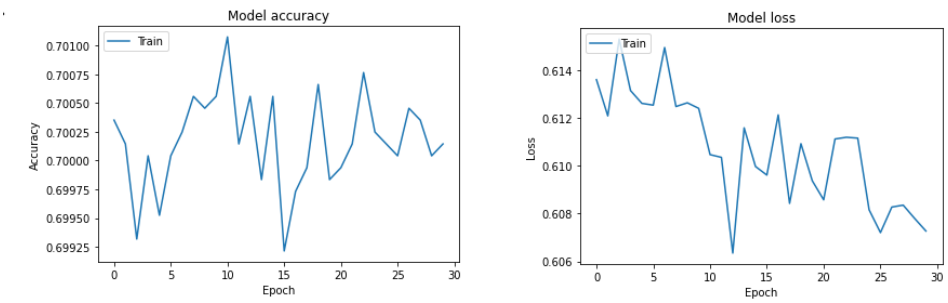


圖 19、 Inception V3 訓練結果

```
model.evaluate(X_test,  Y_test_label)
```

```
107/107 [==============================] - 30s 280ms/step - loss: 0.6170 - accuracy: 0.6873
[0.6170310974121094, 0.6873347163200378]
```

圖 20、 Inception V3 訓練測試集結果

```
Epoch 00028: ReduceLROnPlateau reducing learning rate to 4.304672074795235e-07.
1209/1208 - 27s - loss: 0.6153 - accuracy: 0.7001
Epoch 29/30
1209/1208 - 27s - loss: 0.6184 - accuracy: 0.7002
Epoch 30/30
1209/1208 - 26s - loss: 0.6191 - accuracy: 0.7001
34/34 - 1s - loss: 0.6109 - accuracy: 0.7002
accuracy: 70.02%
70.01% (+/- 0.04%)
```

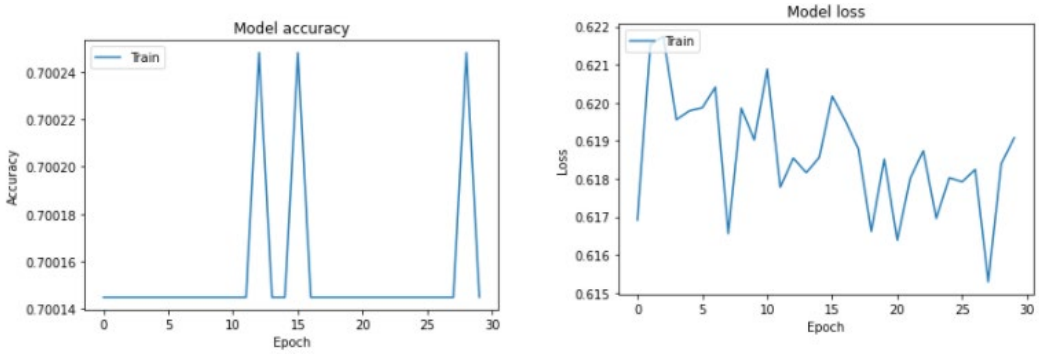


圖 21、 ResNet50 訓練結果

```
model.evaluate(X_test,  Y_test_label)
```

```
107/107 [==============================] - 4s 39ms/step - loss: 0.6213 - accuracy: 0.6873
[0.6213282346725464, 0.6873347163200378]
```

圖 22、 ResNet50 訓練測試集結果

透過訓練結果可以看出利用 Transfer Learning 的結果沒有比想像中的好，表現結果甚至不如我們自己所建立的 model，但是其中有可能是因為其他超參數的設定並不是這兩種 model 最合適的設定，所以導致這兩個模型在學習的時候準確率沒有很好的改善，雖然 loss 有下降的趨勢，但是幅度並不明顯，下表為三個不同 model 的訓練比較，雖然測試集的準確率出現了大致相同的情況，但是從 loss 的值來看也可以看出我們所訓練的 model 在相同情況下利用同組 data 的表現最佳。

Model	Training	Validation	Test
	Accuracy/Loss	Accuracy/Loss	Accuracy/Loss

Proposed method	0.8882/0.2582	0.8212/0.3750	0.6844/0.6115
Inception V3	0.7001/0.5959	0.7002/0.6073	0.6873/0.6170
ResNet50	0.7001/0.6109	0.7002/0.6205	0.6873/0.6213

四、 結論

(一)、 貢獻

本次專題的結果而言，我們針對資料集做出的模型，相較於其他現成模型的表現都較佳，雖然在測試集上面沒有顯著的差異，但是訓練過程中的表現很好，對於肺音異常分類在這項結果當中，雖然測試集的表現結果沒有很理想，我們提出的建置模型方法可以針對不同的資料及進行設計，透過參數調整使模型更加擬合資料，判別得更加準確，為了避免過度擬合的情況發生，利用 Dropout 等等的手法保持模型在訓練當中的彈性，使其在分類時保持適當的空間判別測試集的資料。

(二)、 局限性

由於這次的模型架構是針對該組資料及建置，所以導入其他不同儀器蒐集的肺音異常資料可能會使辨識率更低，這個架構固然可行，但可能對於當中的參數要自己進行更改，如若要用這個模型在現實中實行，需要參考原始資料集蒐集的規格，利用相同的設備進行才可以重現本次專題的實驗結果。

(三)、 適用性

對於本次實驗結果，我們認為可以朝向臨床實施，如若醫生判斷是否有肺音異常時不太肯定，可以將資料丟入模型當中提供建議；在醫生確定患者是否肺音異常時，可以協同別的醫生進行判斷，將該名患者的資料進行標註，餵入模型訓練當中，使模型更加完善。

(四)、 未來改善

我們認為測試集的準確率可以透過增加資料集來提升，或者透過患者定期進行肺音資料的蒐集（例如一年一次肺音檢查），將本次資料與前次資料（已由醫生正確判別完成）做比對，透過兩筆資料之間的相關係數、殘差與最新資料的肺音異常辨識等等的結果，來作為判斷這名患者此次是否有肺音異常的預測（例如將這幾筆資料作為迴歸分析的自變項），透過深度學習並聯的方式來降低錯誤率。

五、 參考文獻

[1] Chamberlain, D., Kodgule, R., Ganelin, D., Miglani, V., & Fletcher, R. R. (2016, August). Application of semi-supervised deep learning to lung sound analysis. In *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (pp. 804-807). IEEE.

[2] Li, L., Xu, W., Hong, Q., Tong, F., & Wu, J. (2016, October). Classification between normal and adventitious lung sounds using deep neural network. In *2016 10th International Symposium on Chinese Spoken Language Processing (ISCSLP)* (pp. 1-5). IEEE.

[3] https://jkjahuang.pixnet.net/blog/post/9028106-%E3%80%90chest%E3%80%91%E5%91%BC%E5%90%B8%E9%9F%B3%E8%81%BD%E8%A8%BA

[4] https://www.itread01.com/content/1550265688.html

[5] https://blog.gtwang.org/programming/keras-resnet-50-pre-trained-model-build-dogs-cats-image-classification-system/

[6] https://hackmd.io/@allen108108/H1MFrV9WH