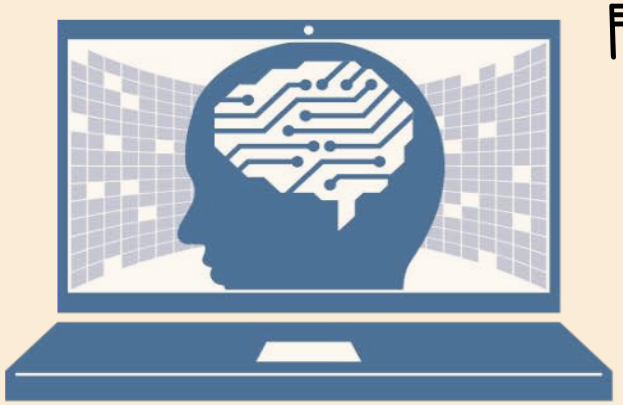
心律不整

-using CNN model to classify

108034553 尤子維

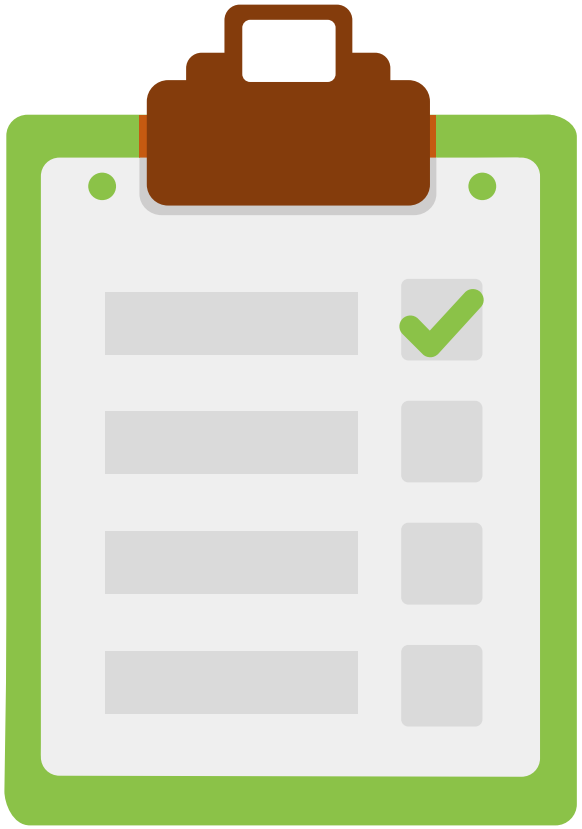
指導教授 邱銘傳教授

目錄



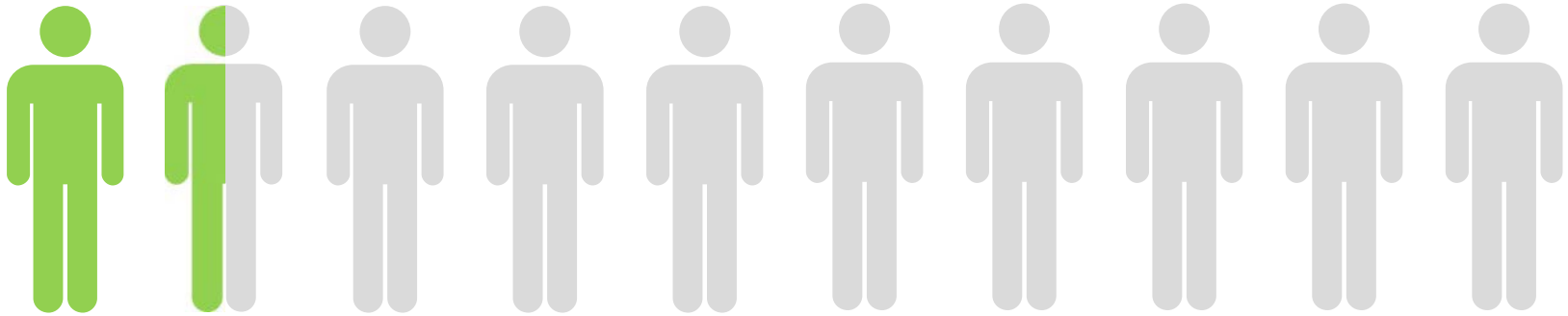
01

背景介紹



心律不整是指心臟的跳動速率過慢、過快或是不規則跳動所引起的疾病，從心跳指令的出發點「節律點」到心肌之間只要出現任何異常狀況，都有可能導致心律不整。全世界因心律不整而導致突發性心跳停止的案例占了所有心血管疾病相關患者的一半死亡原因、以及全球死亡原因的15%。因此藉由神經網路模型分析，由心電描記術(ECG)所紀錄心臟跳動的訊號，及早預測出病人的病狀並進行分類，就可以針對病狀及早治療。

15%
Suffering from skin cancer



02

問題定義

01

WHAT：要解決甚麼問題?

解決單獨由醫生判斷錯誤，可能準確率較低的問題。

02

WHEN： 在甚麼時候要辨別病徵?

當病人至醫院做心電圖檢測時。

03

WHO： 由誰來改善目前確診率不夠高的問題?

醫院的研究部門提出此模型，看診醫生同時採用此模型，增加判斷準確率。

04

WHERE：在哪種地方可以進行問題改善?

醫院的心臟科。

05

WHY：為什麼要改善此問題?

避免錯誤診斷導致病人看診時間延誤、提高醫院心臟科的名聲、給予病人更好的治療。

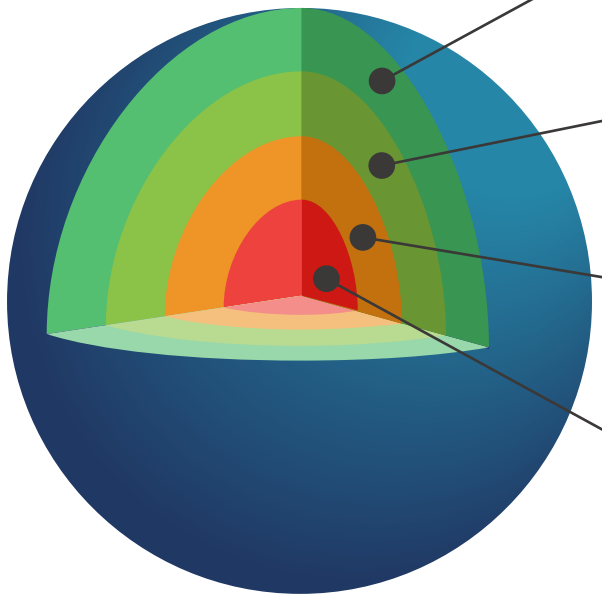
06

HOW：如何解決此問題?

建構神經網路模型，協助醫生進行病人心臟訊號的病徵判斷。

03

模型架構



01 資料輸入

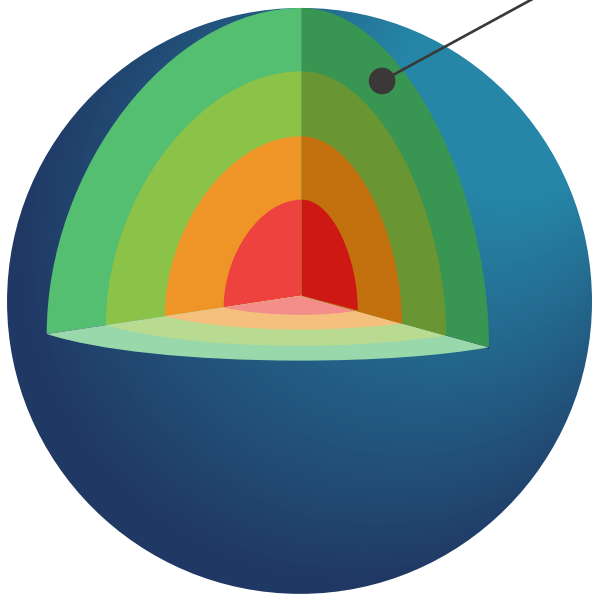
02 資料前處理

03 模型建構

04 結果輸出

03

模型架構



01

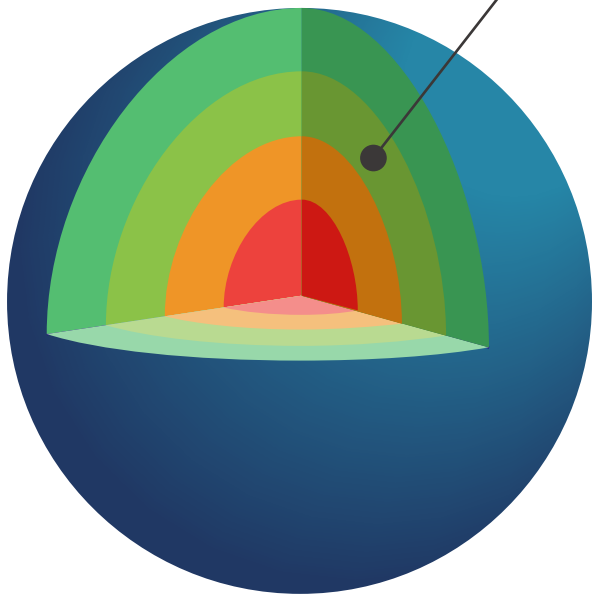
資料輸入

- 本研究透過kaggle線上開放式資料庫所提供之MIT-BIH心律不整數據集，其中樣本數為109446筆(訓練集87554筆、測試集21892筆)，總共分成5個類別

各類別樣本數	訓練集	測試集
N: Non-ecotic beats	72471	18118
S: Supraventricular ectopic beats	2223	556
V: Ventricular ectopic beats	5788	1448
F: Fusion Beats	641	162
Q: Unknown Beats	6431	1608
	87554	21892

03

模型架構



02

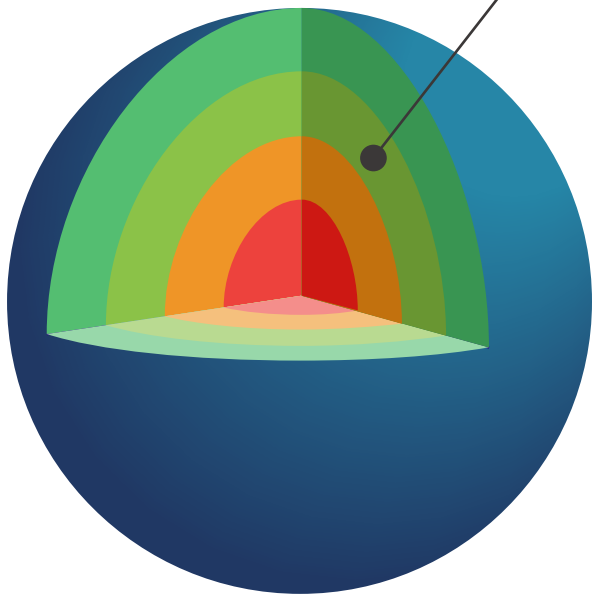
資料前處理

- 由於給定的訓練集資料不平衡，像是類別N的樣本高達72471筆，而類別F僅有641筆，因此為了平衡訓練集樣本，我們採取過採樣及欠採樣的方式，將不足35000筆的類別，重複抽取至35000筆，反之將超過35000筆的類別，以隨機抽取35000筆的方式。

```
#將訓練集採取過採樣及欠採樣
from sklearn.utils import resample
df_1=train_df[train_df[187]==1]
df_2=train_df[train_df[187]==2]
df_3=train_df[train_df[187]==3]
df_4=train_df[train_df[187]==4]
df_0=(train_df[train_df[187]==0]).sample(n=35000,random_state=42)

df_1_upsample=resample(df_1,replace=True,n_samples=35000,random_state=123)
df_2_upsample=resample(df_2,replace=True,n_samples=35000,random_state=124)
df_3_upsample=resample(df_3,replace=True,n_samples=35000,random_state=125)
df_4_upsample=resample(df_4,replace=True,n_samples=35000,random_state=126)

train_df=pd.concat([df_0,df_1_upsample,df_2_upsample,df_3_upsample,df_4_upsample])
```

03

模型架構



02

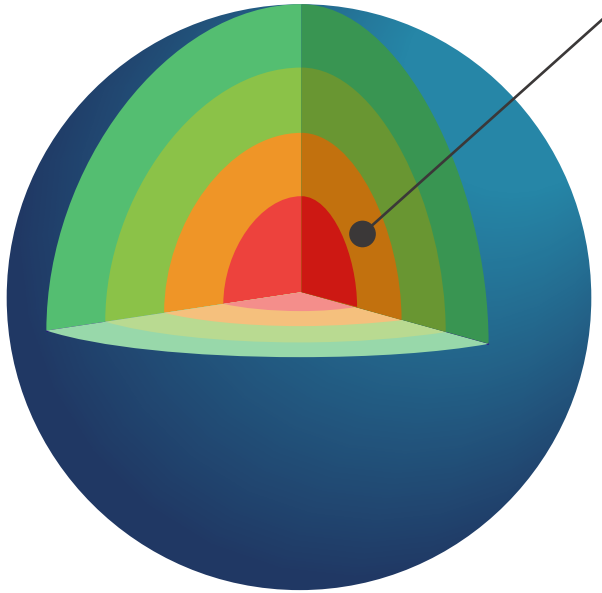
資料前處理

- 將訓練集和驗證集的類別標籤0~4轉為0,1的表達方式
- 將訓練集和驗證集以reshape的方式增加其維度，使其fit CNN模型的輸入

```
115 target_train=train_df[187]
116 target_test=test_df[187]
117
118 y_train=to_categorical(target_train)
119 y_test=to_categorical(target_test)
120 X_train=train_df.iloc[:,:186].values
121 X_test=test_df.iloc[:,:186].values
122
123 X_train = X_train.reshape(len(X_train), X_train.shape[1],1)
124 X_test = X_test.reshape(len(X_test), X_test.shape[1],1)
125
```

03

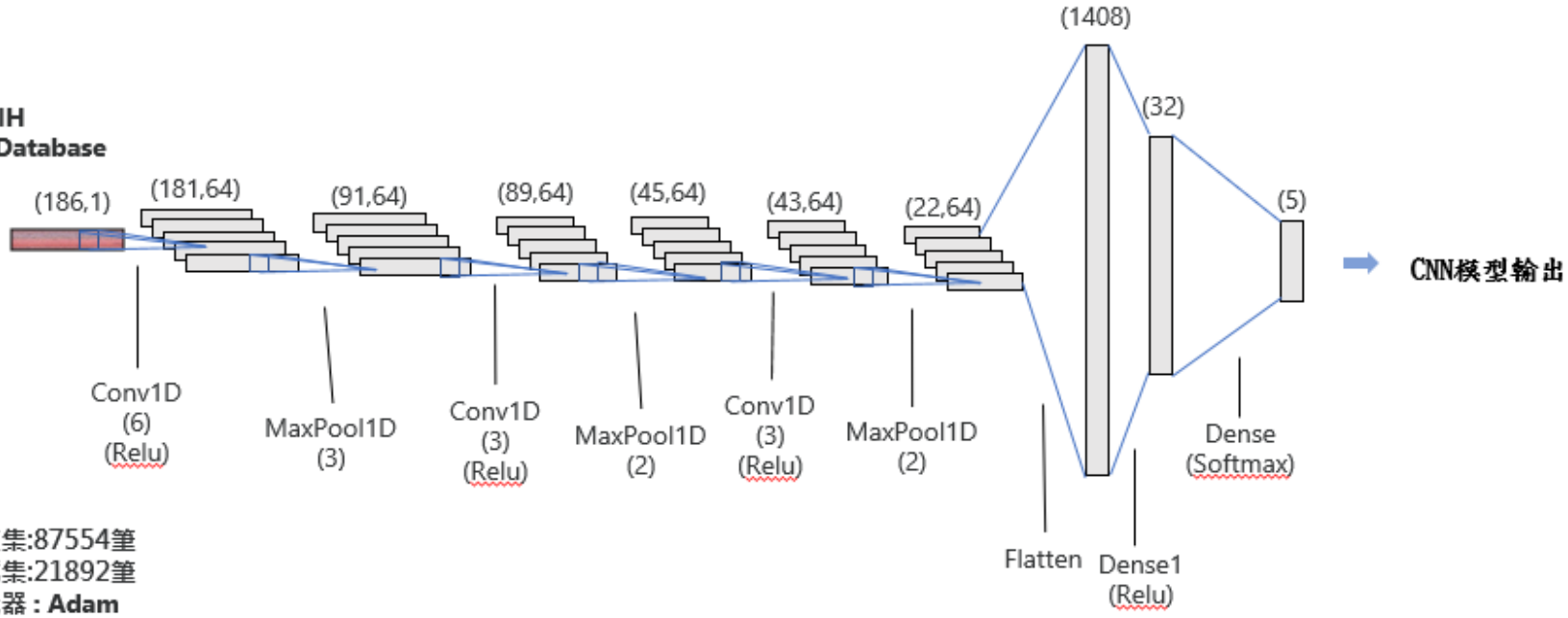
模型架構



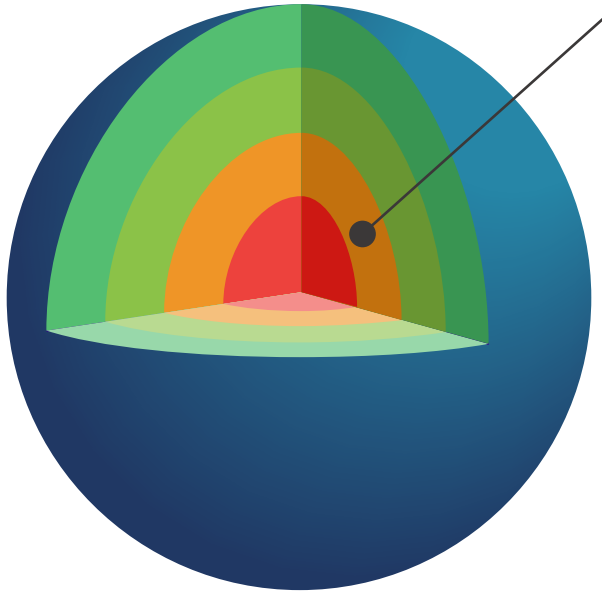
03

模型建構

訓練集數據 → 資料前處理 → CNN模型 → 提取特徵向量 → 訓練結束



訓練集:87554筆
測試集:21892筆
優化器 : **Adam**
損失函數 : **categorical_crossentropy**
Epoch:40
Bacthsiz:32



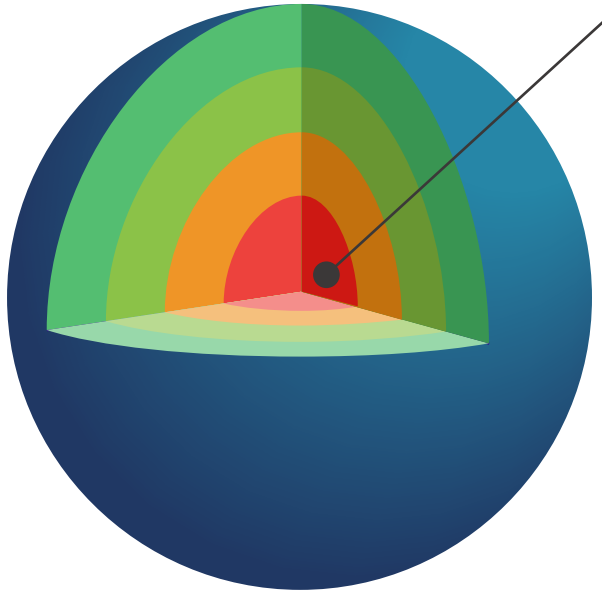
03

模型建構

```
126 im_shape=(X_train.shape[1],1)
127 inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
128 conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
129 #conv1_1=BatchNormalization()(conv1_1)
130 pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
131 conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
132 #conv2_1=BatchNormalization()(conv2_1)
133 pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
134 conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
135 #conv3_1=BatchNormalization()(conv3_1)
136 pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
137 flatten=Flatten()(pool3)
138 #dense_end1 = Dense(64, activation='relu')(flatten)
139 dense_end = Dense(32, activation='relu')(flatten)
140 main_output = Dense(5, activation='softmax', name='main_output')(dense_end)
141 #model_dense_end = Model(inputs= inputs_cnn, outputs=dense_end)
142 model = Model(inputs= inputs_cnn, outputs=main_output)
143 model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])
144 model.summary()
```

03

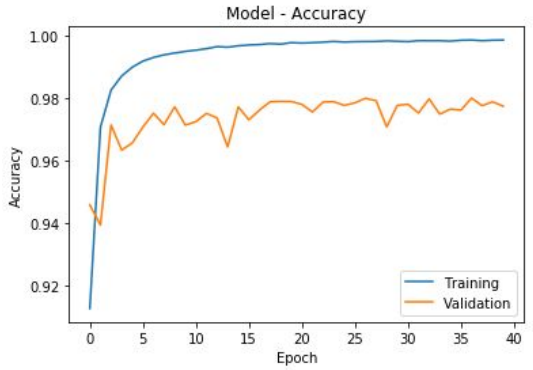
模型架構

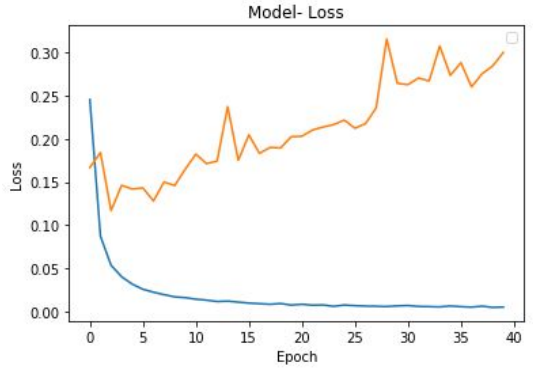


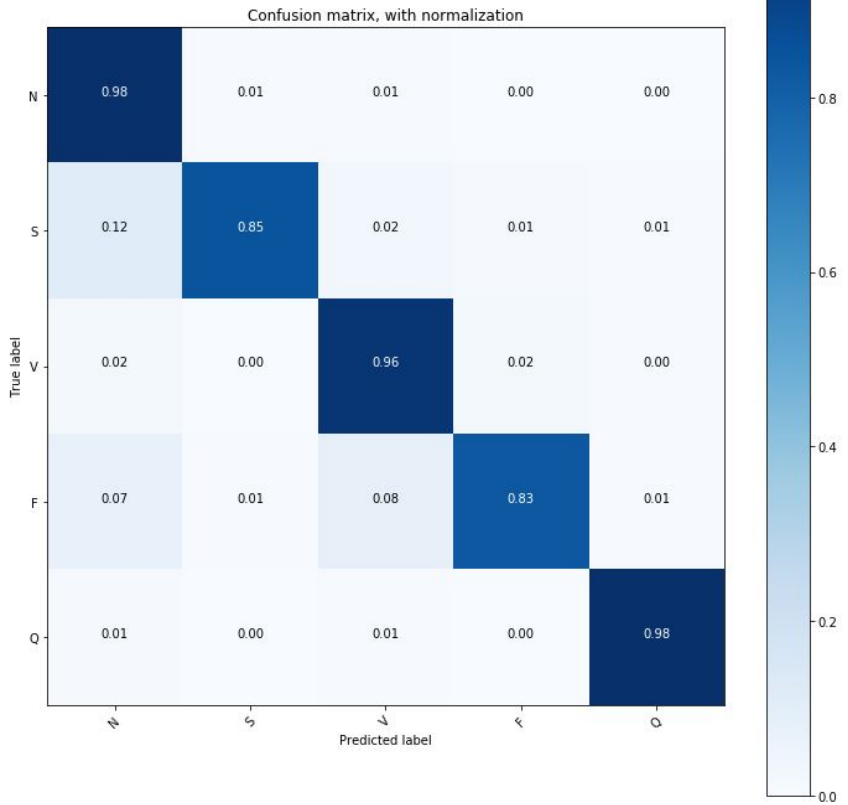
04

結果輸出

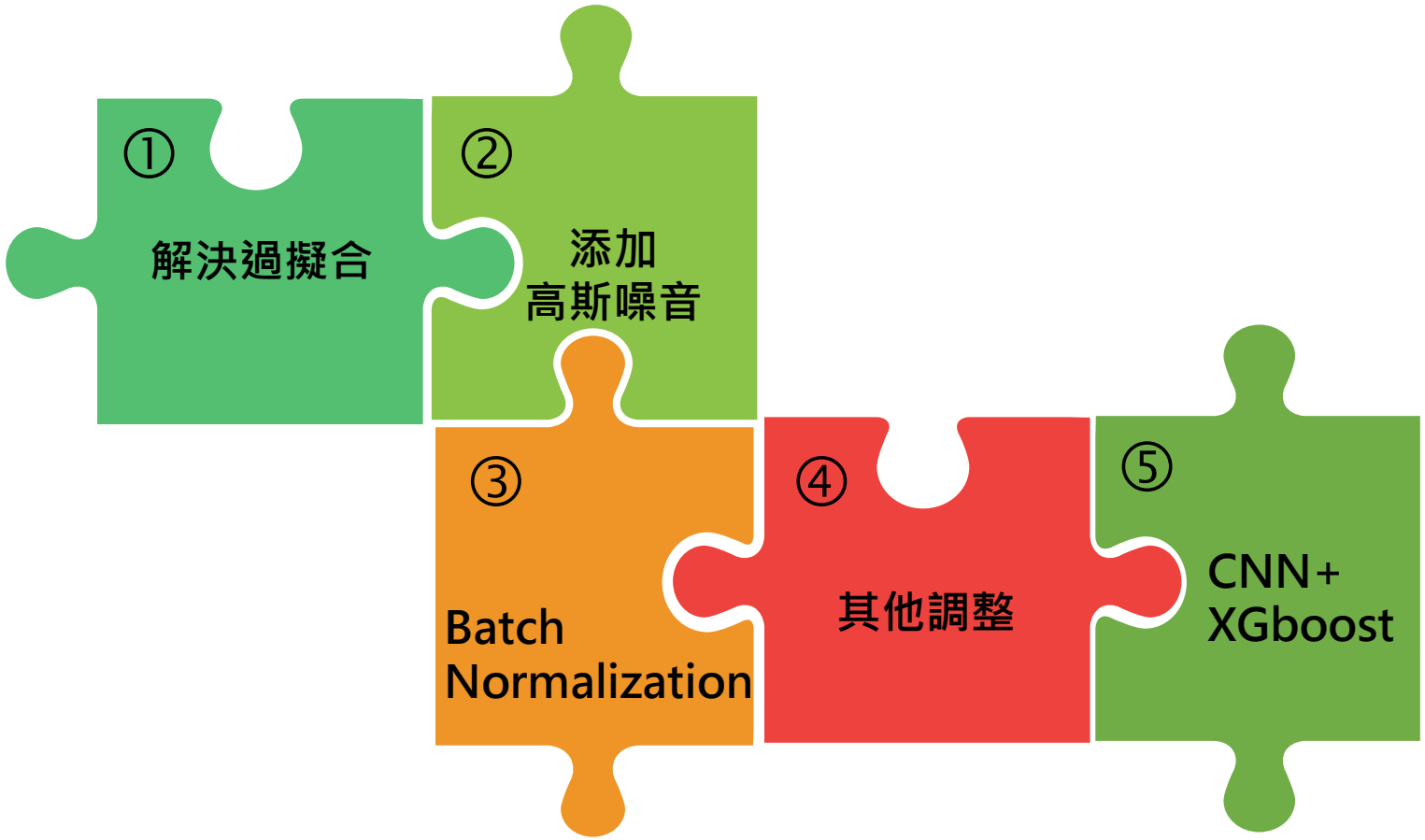
測試集準確率為0.9675







改善方法



04

分析與改善

解決過擬合問題

調整抽取樣本數

有可能是因為數量少的類別重複抽取太多次，導致模型發生過擬合的原因，因此將每一個類別重複抽取從35000筆降至25000筆。

添加提前終止方法

添加了提前終止(Early Stopping)的方法來防止模型發生過擬合的情況，其中Patience設定為5。

```
from sklearn.utils import resample
df_1=train_df[train_df[187]==1]
df_2=train_df[train_df[187]==2]
df_3=train_df[train_df[187]==3]
df_4=train_df[train_df[187]==4]
df_0=(train_df[train_df[187]==0]).sample(n=25000,random_state=42)

df_1_upsample=resample(df_1,replace=True,n_samples=25000,random_state=123)
df_2_upsample=resample(df_2,replace=True,n_samples=25000,random_state=124)
df_3_upsample=resample(df_3,replace=True,n_samples=25000,random_state=125)
df_4_upsample=resample(df_4,replace=True,n_samples=25000,random_state=126)

train_df=pd.concat([df_0,df_1_upsample,df_2_upsample,df_3_upsample,df_4_upsample])

```

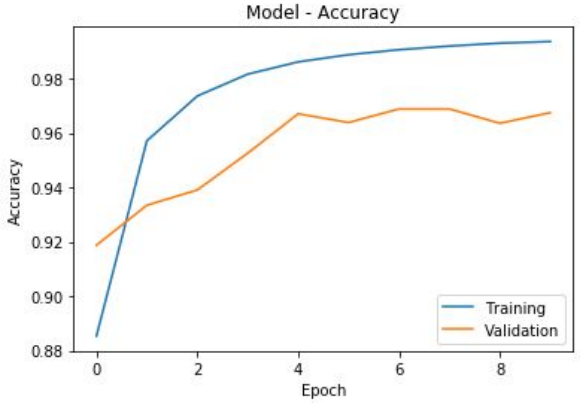
```
callbacks = [EarlyStopping(monitor='val_loss', patience=5),
             ModelCheckpoint(filepath='best_model.h5', monitor='val_loss', save_best_only=True)]
history=model.fit(X_train, y_train,epochs=40, batch_size=32,callbacks=callbacks,validation_data=(X_test,y_test))

```

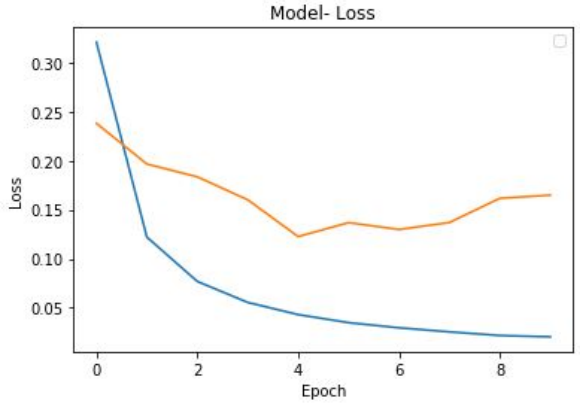
04

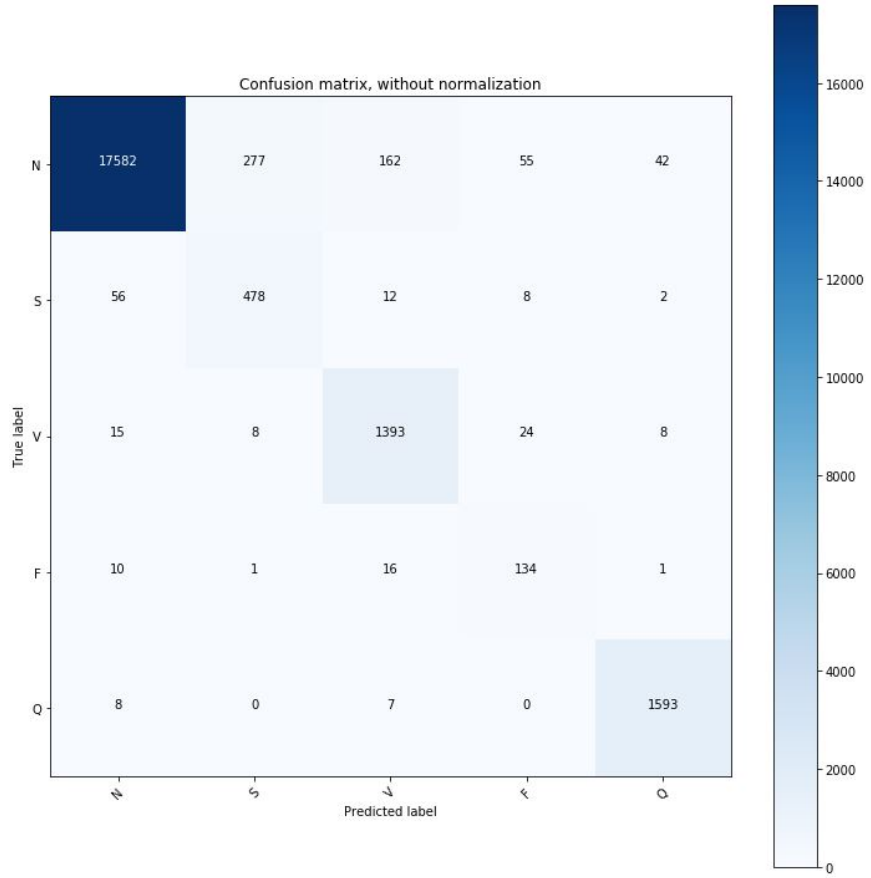
分析與改善

解決過擬合問題

準確率雖然一樣落在96.75%，但是解決了過擬合的問題，並且降低模型運作時間，提升效能







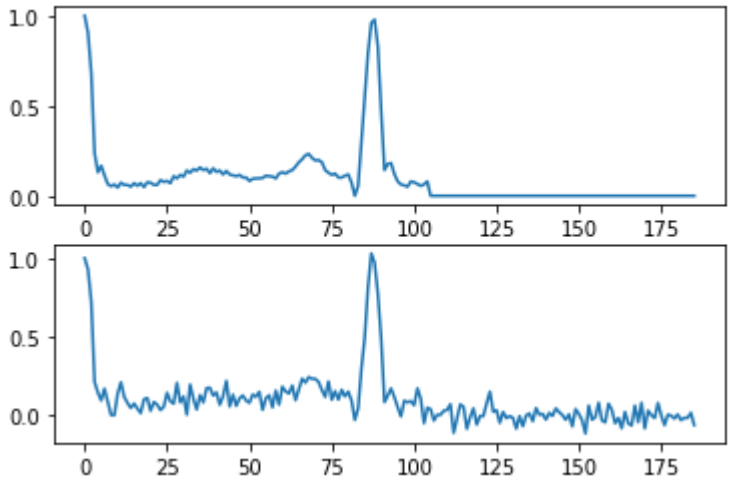
04

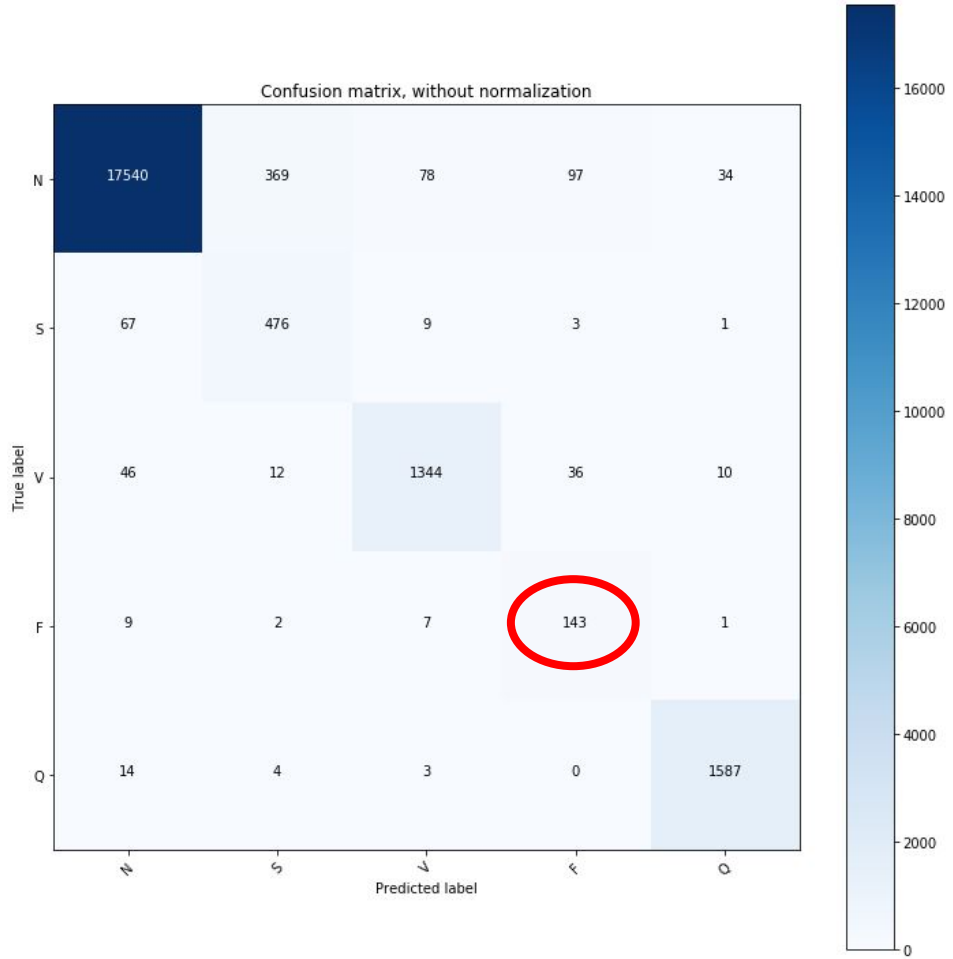
分析與改善

增加高斯噪音

準確率下降至96.34%

```
121
122 def add_gaussian_noise(signal):
123     noise=np.random.normal(0,0.05,186)
124     return (signal+noise)
125 for i in range(len(X_train)):
126     X_train[i,:186]= add_gaussian_noise(X_train[i,:186])
```

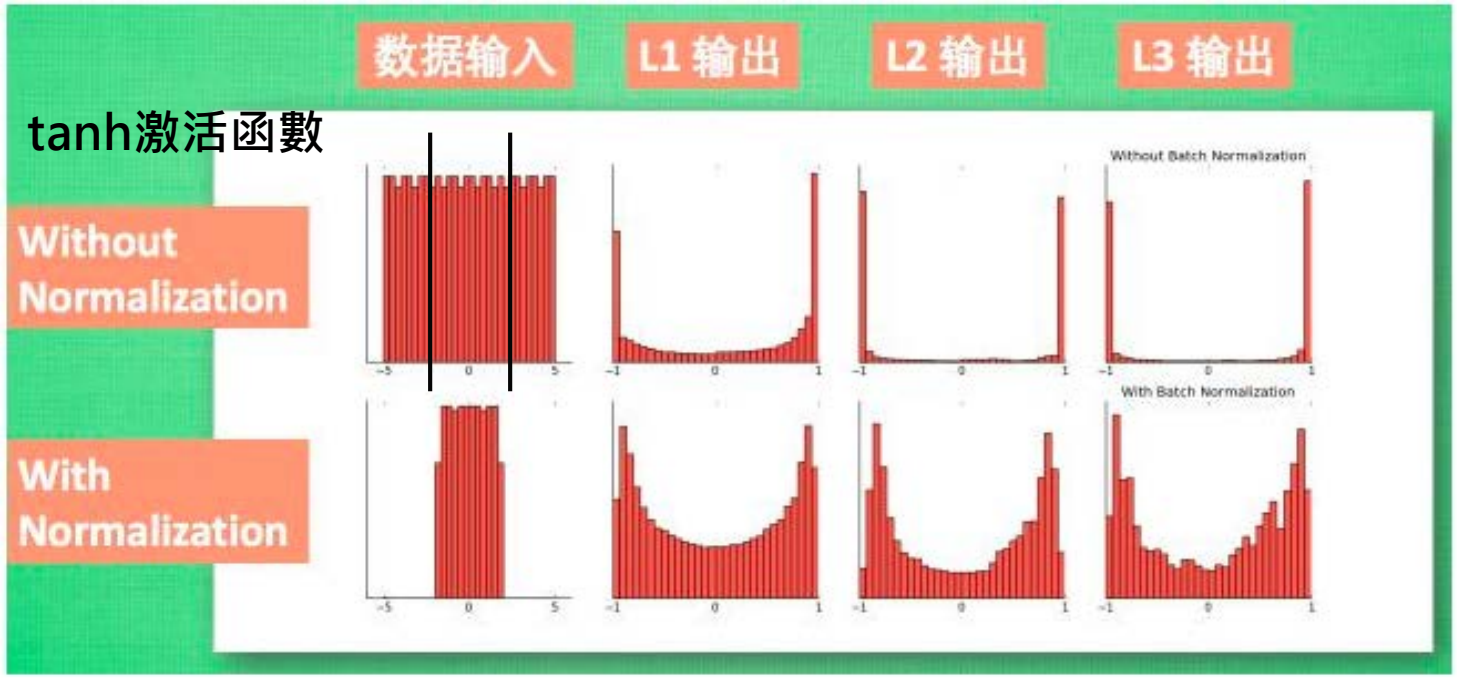




分析與改善

Batch Normalization

- Batch Normalization 視為對神經網絡中的每一層進行預處理的過程。它的最大好處就是讓每一層的值在有效的範圍內傳遞下去。



優點在於：

- 快速學習（能增加學習率)
- 不會過度依賴預設值（不會對預設值產生過度反應）
- 控制過度學習（減少 Dropout 等必要性）

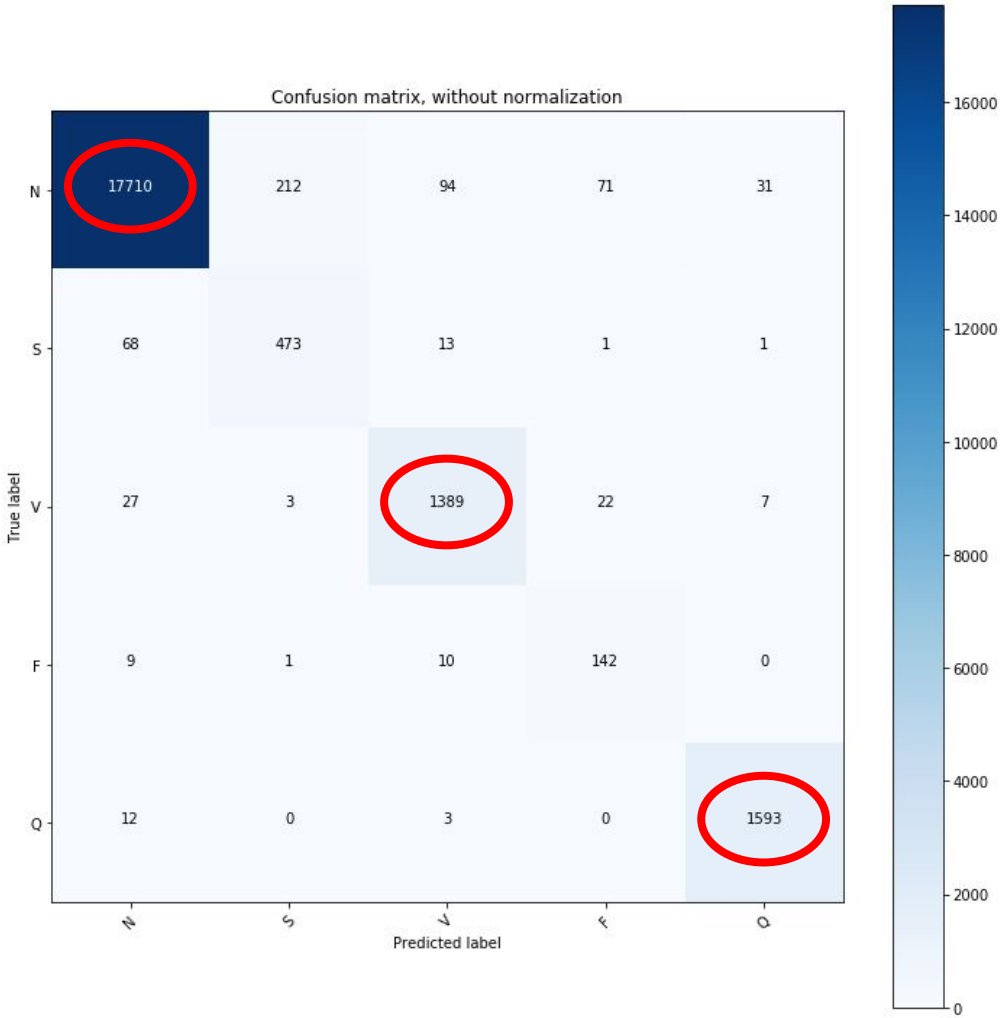
04

分析與改善

Batch Normalization

準確率提升至97.89%

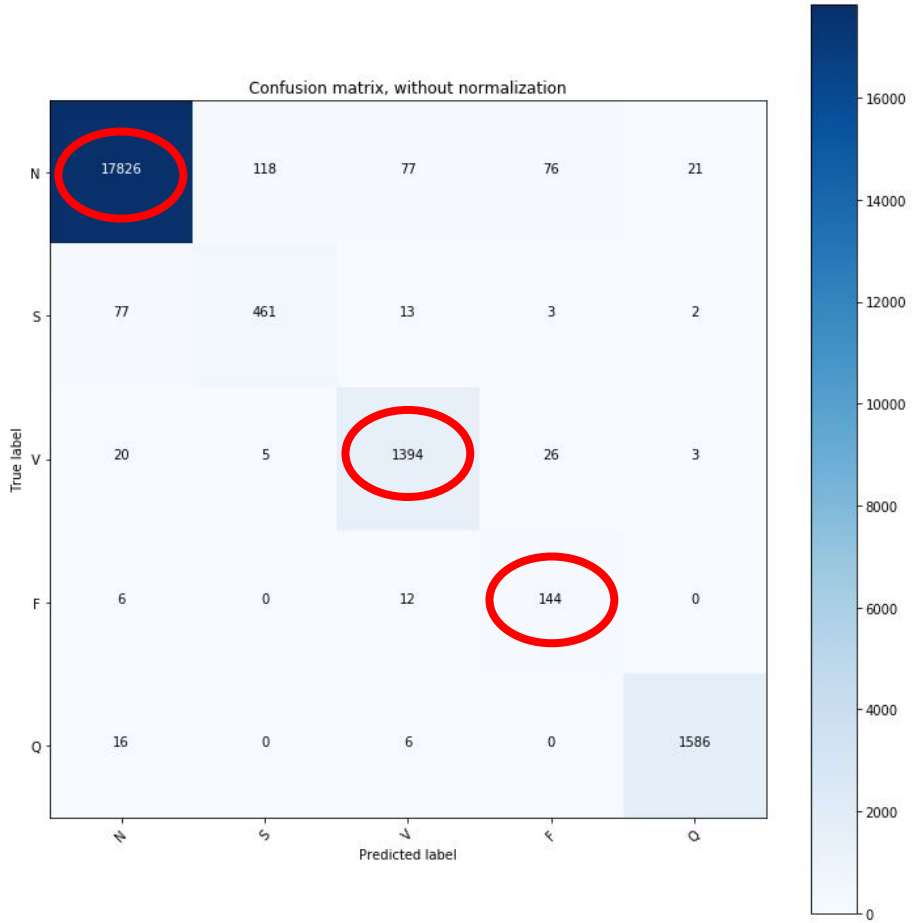
```
132 im_shape=(X_train.shape[1],1)
133 inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
134 conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
135 conv1_1=BatchNormalization()(conv1_1)
136 pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
137 conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
138 conv2_1=BatchNormalization()(conv2_1)
139 pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
140 conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
141 conv3_1=BatchNormalization()(conv3_1)
142 pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
143 flatten=Flatten()(pool3)
144 #dense_end1 = Dense(64, activation='relu')(flatten)
145 dense_end = Dense(32, activation='relu')(flatten)
146 main_output = Dense(5, activation='softmax', name='main_output')(dense_end)
147 #model_dense_end = Model(inputs= inputs_cnn, outputs=dense_end)
148 model = Model(inputs= inputs_cnn, outputs=main_output)
149 model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])
150 model.summary()
151
```



其他調整

增加dense層，以及調整batch size的參數，最終獲得最佳的模型準確率達到99.2%

```
im_shape=(X_train.shape[1],1)
inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
conv1_1=BatchNormalization()(conv1_1)
pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
conv2_1=BatchNormalization()(conv2_1)
pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
conv3_1=BatchNormalization()(conv3_1)
pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
flatten=Flatten()(pool3)
dense_end1 = Dense(64, activation='relu')(flatten)
dense_end = Dense(32, activation='relu')(flatten)
main_output = Dense(5, activation='softmax', name='main_output')(dense_end)
#model_dense_end = Model(inputs= inputs_cnn, outputs=dense_end)
model = Model(inputs= inputs_cnn, outputs=main_output)
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])
model.summary()

callbacks = [EarlyStopping(monitor='val_loss', patience=5),
             ModelCheckpoint(filepath='best_model.h5', monitor='val_loss', save_best_only=True)]
history=model.fit(X_train, y_train,epochs=40, batch_size=50,callbacks=callbacks,validation_data=(X_test,y_test))
```

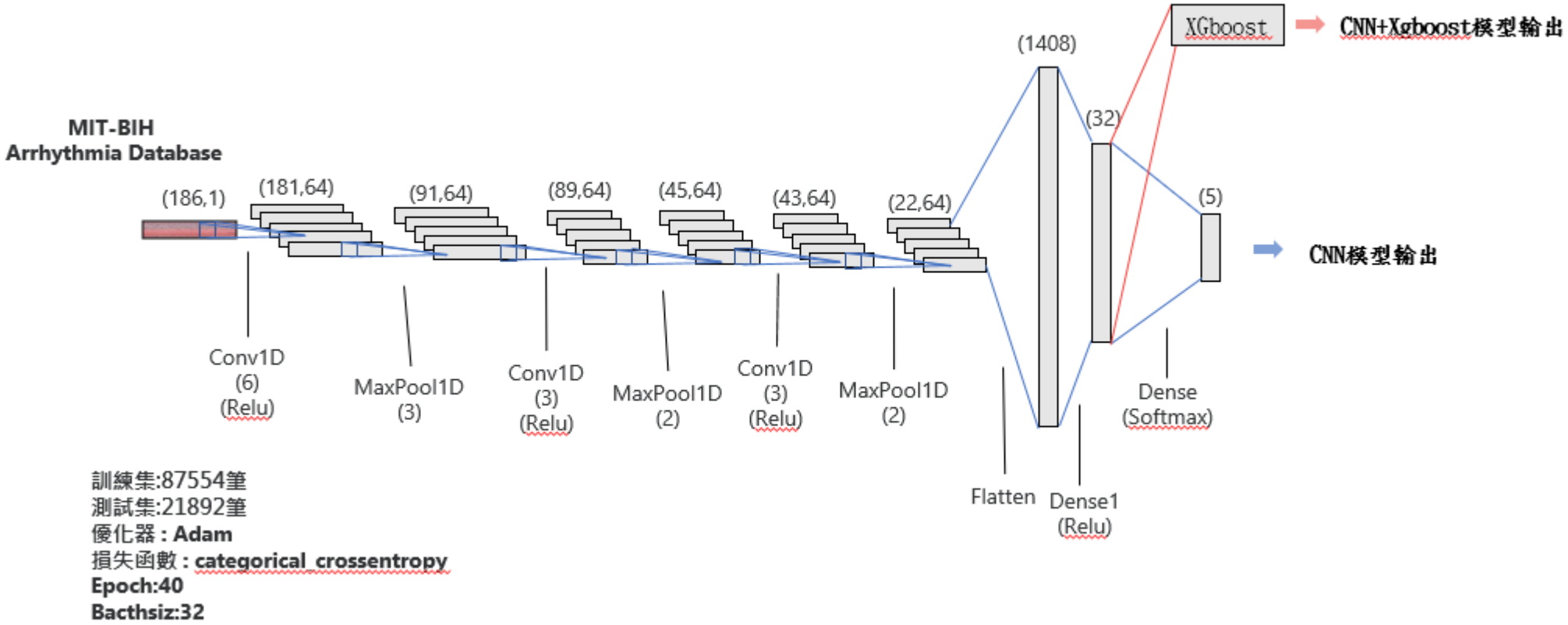


04

分析與改善

CNN+XGboost

訓練集數據 → 資料前處理 → CNN模型 → 提取特徵向量 → XGboost模型 → 訓練結束



訓練集:87554筆
測試集:21892筆
優化器 : **Adam**
損失函數 : **categorical_crossentropy**
Epoch:40
Bacthsiz:32

04

分析與改善

XGboost(Extreme Gradient Boosting)

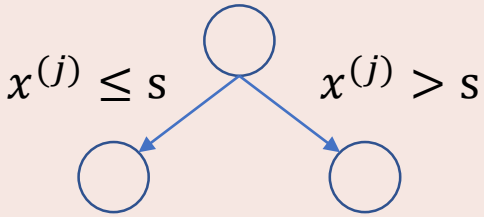
- XGboost是boosting集成學習的其中一種，是將許多樹模型(弱分類器)集成在一起，形成一個很強的分類器。而所用到的樹模型則是CART回歸樹模型。

集成學習

- Boosting流派 : 各分類器之間有依賴關係，例如Adaboost、GBDT(Gradient Boosting Decision Tree)、XGboost
- Bagging流派 : 各分類器之間沒有依賴關係，例如隨機森林（Random Forest）

CART回歸樹模型

CART回歸樹是假設樹為二叉樹，通過不斷將特徵進行分裂。比如當前樹結點是基於第j個特徵值進行分裂，設該特徵值小於s的樣本劃分為左子樹，大於s的樣本劃分為右子樹。

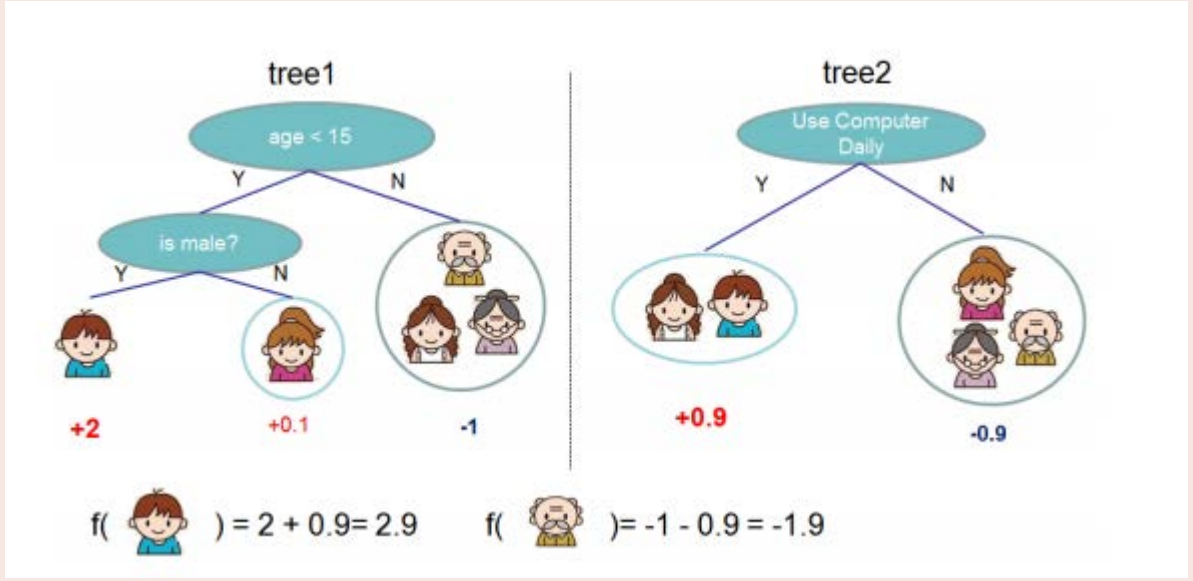


XGboost算法思想

- 該算法思想就是不斷地添加樹，不斷地進行特徵分裂來生長一棵樹，每次添加一個樹，其實是學習一個新函數，去擬合上次預測的殘差

舉例

預測一家人對電子遊戲的喜好程度



04

分析與改善

CNN+XGboost

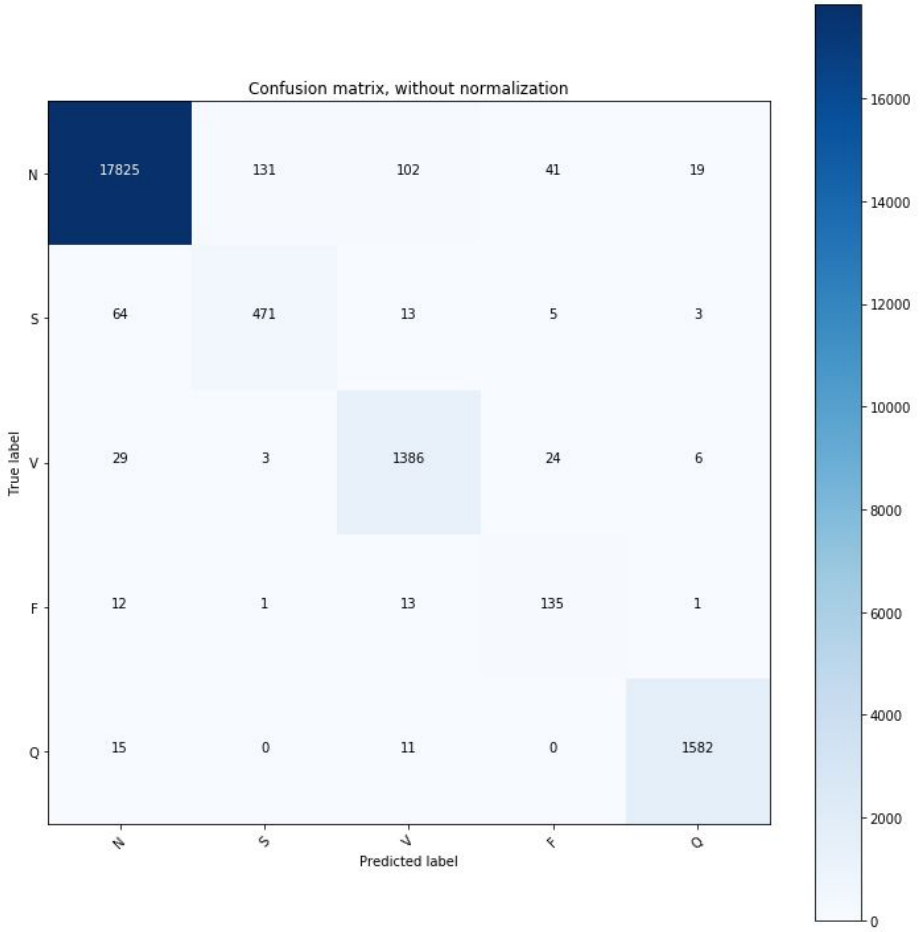
```
X_train = X_train.reshape(len(X_train), X_train.shape[1],1)
X_test = X_test.reshape(len(X_test), X_test.shape[1],1)

im_shape=(X_train.shape[1],1)
inputs_cnn=Input(shape=(im_shape), name='inputs_cnn')
conv1_1=Convolution1D(64, (6), activation='relu', input_shape=im_shape)(inputs_cnn)
conv1_1=BatchNormalization()(conv1_1)
pool1=MaxPool1D(pool_size=(3), strides=(2), padding="same")(conv1_1)
conv2_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool1)
conv2_1=BatchNormalization()(conv2_1)
pool2=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv2_1)
conv3_1=Convolution1D(64, (3), activation='relu', input_shape=im_shape)(pool2)
conv3_1=BatchNormalization()(conv3_1)
pool3=MaxPool1D(pool_size=(2), strides=(2), padding="same")(conv3_1)
flatten=Flatten()(pool3)
dense_end1 = Dense(64, activation='relu')(flatten)
dense_end = Dense(32, activation='relu')(flatten)
main_output = Dense(5, activation='softmax', name='main_output')(dense_end)
model_dense_end = Model(inputs= inputs_cnn, outputs=dense_end)
model = Model(inputs= inputs_cnn, outputs=main_output)
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics = ['accuracy'])
model.summary()
```

```
import xgboost
import pandas as pd
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC,LinearSVC

X_train_xg = model_dense_end.predict(X_train)
X_test_xg = model_dense_end.predict(X_test)

z = np.concatenate([np.array(X_train_xg).reshape(100000,32),np.array(target_train).reshape(100000,1)],axis=1)
z = pd.DataFrame(z)
z.to_csv('train_xg.csv',index=False)

z = np.concatenate([np.array(X_test_xg).reshape(21892,32),np.array(target_test).reshape(21892,1)],axis=1)
z = pd.DataFrame(z)
z.to_csv('test_xg.csv',index=False)

train = pd.read_csv('train_xg.csv')
train_y = train['32'].astype('int')
train_x = train.drop(['32'],axis=1)
dataset = xgboost.DMatrix(train_x, label=train_y)
watchlist = [(dataset, 'train')]
params = {'max_depth':6, 'eta':0.15, 'silent':1, 'num_class':5,'objective':'multi:softmax' }
model_xg = xgboost.train(params, dataset, num_boost_round=250, evals=watchlist)

test = pd.read_csv('test_xg.csv')
test_y = test['32'].astype('int')
test_x = test.drop(['32'],axis=1)

test_x = xgboost.DMatrix(test_x)
result = model_xg.predict(test_x)
```

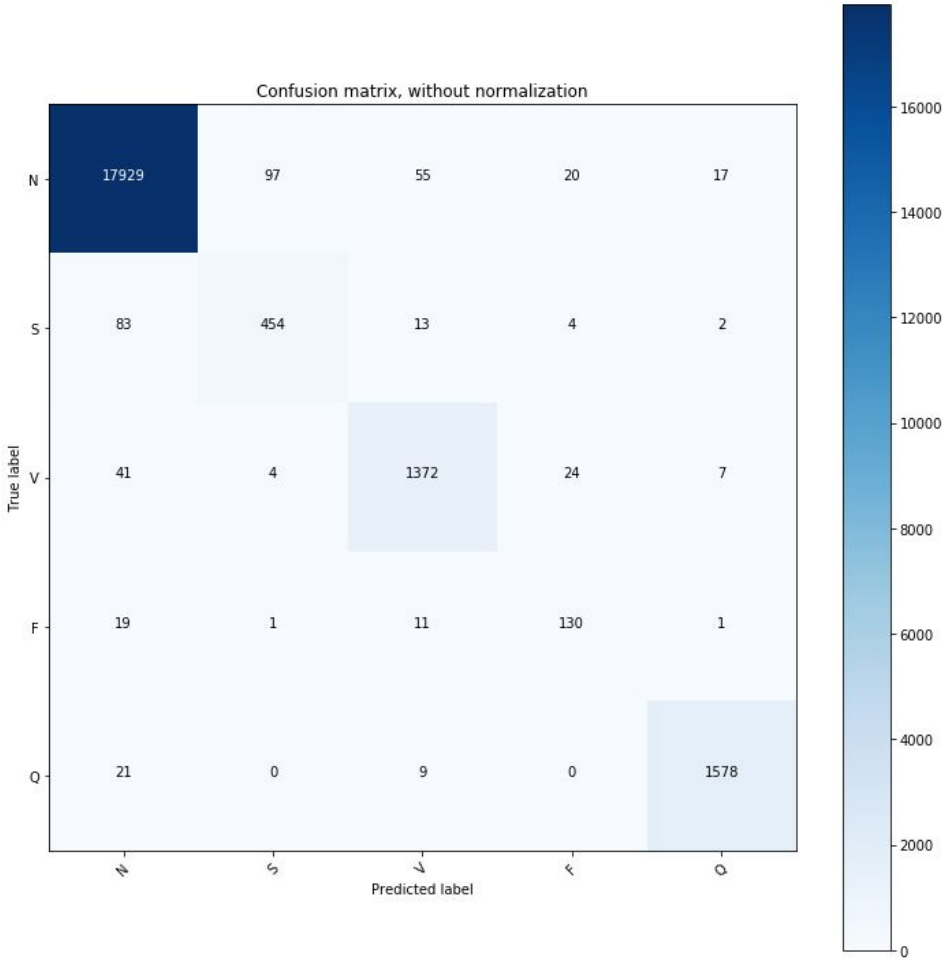
04

分析與改善

原始MODEL



CNN

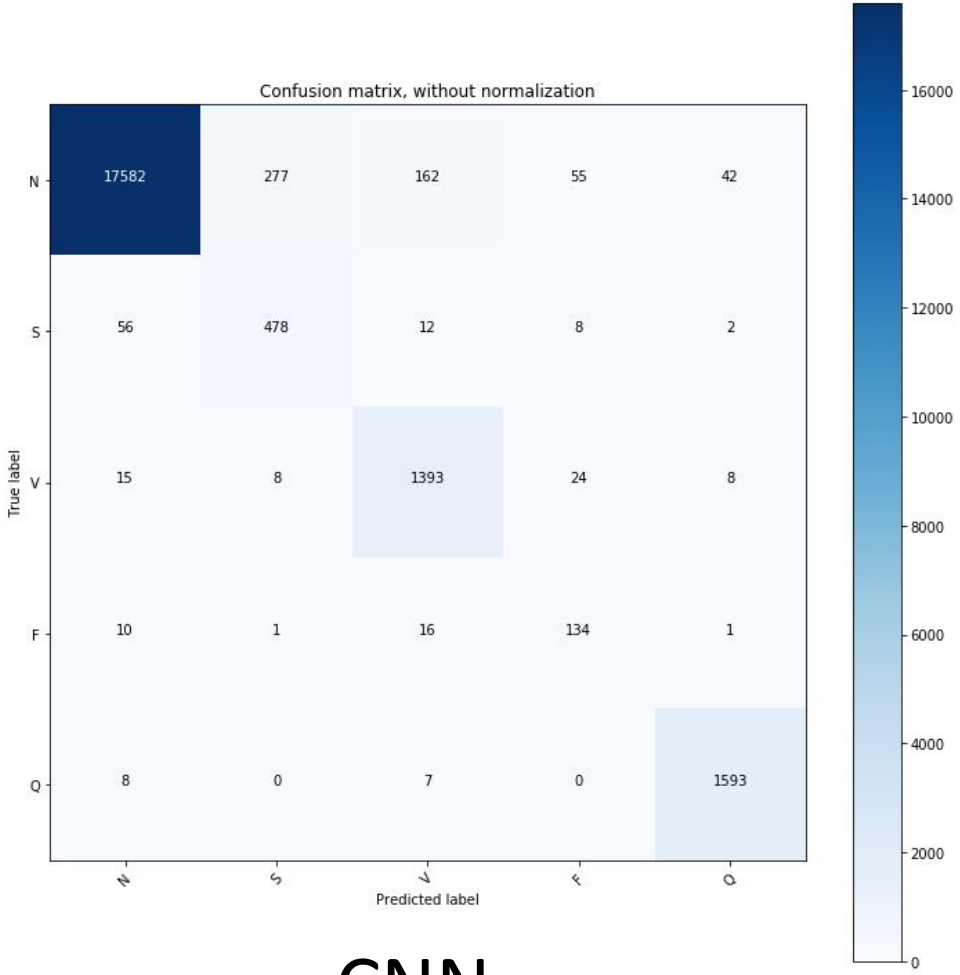


CNN+XGboost

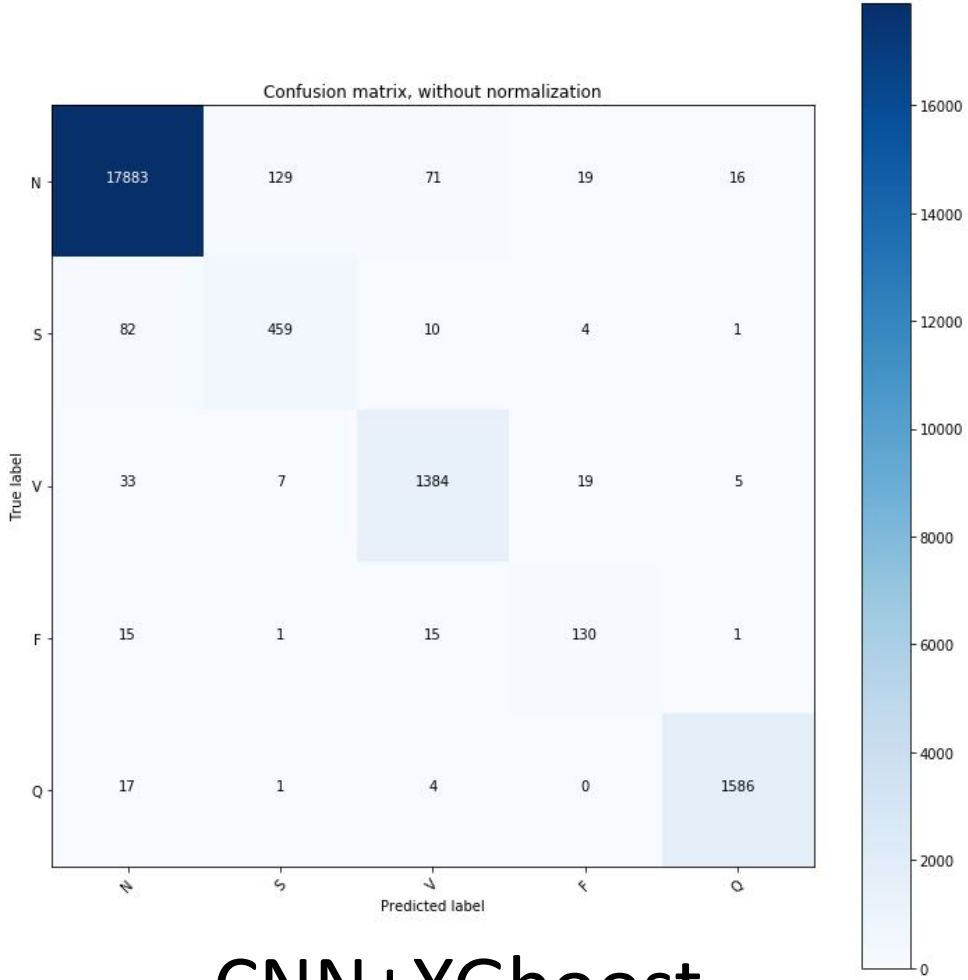
04

分析與改善

解決過擬合的MODEL



CNN

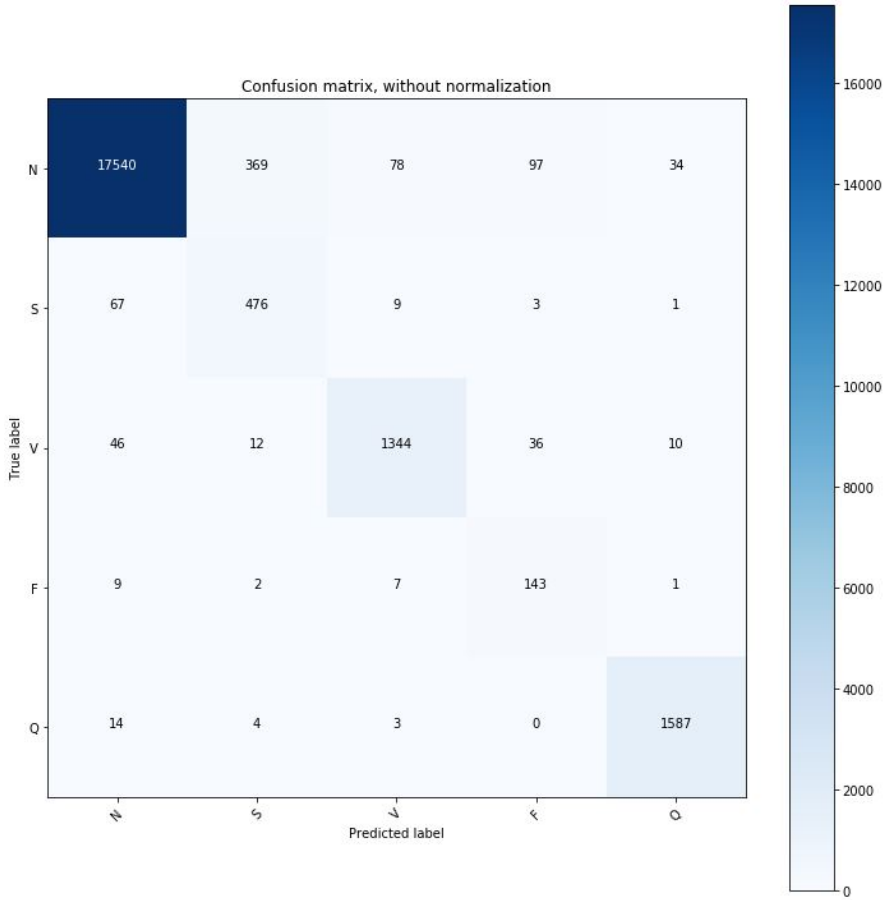


CNN+XGboost

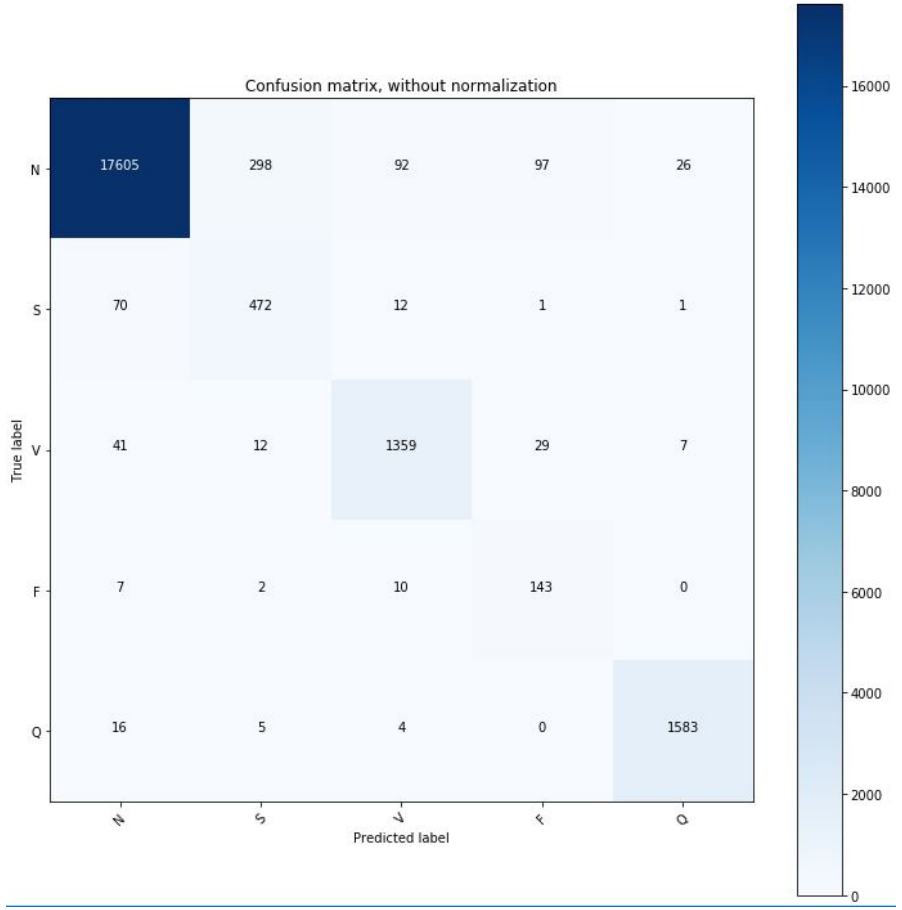
04

分析與改善

增加噪音的MODEL



CNN

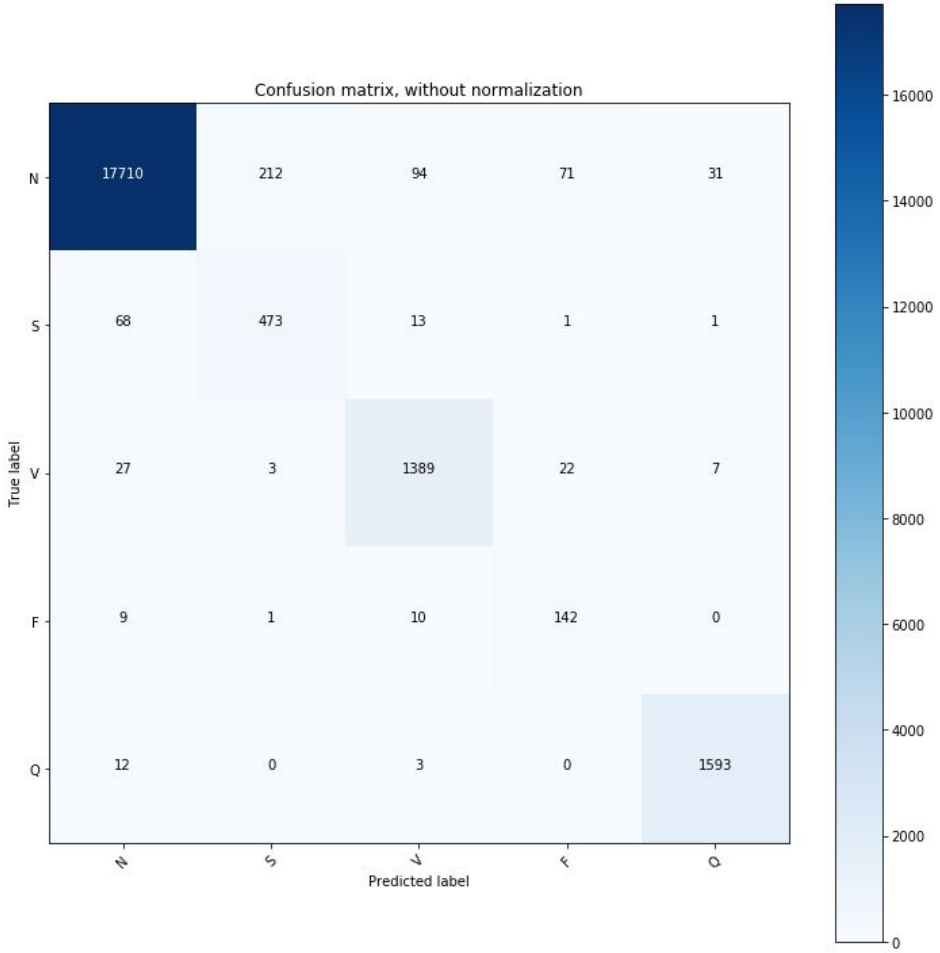


CNN+XGboost

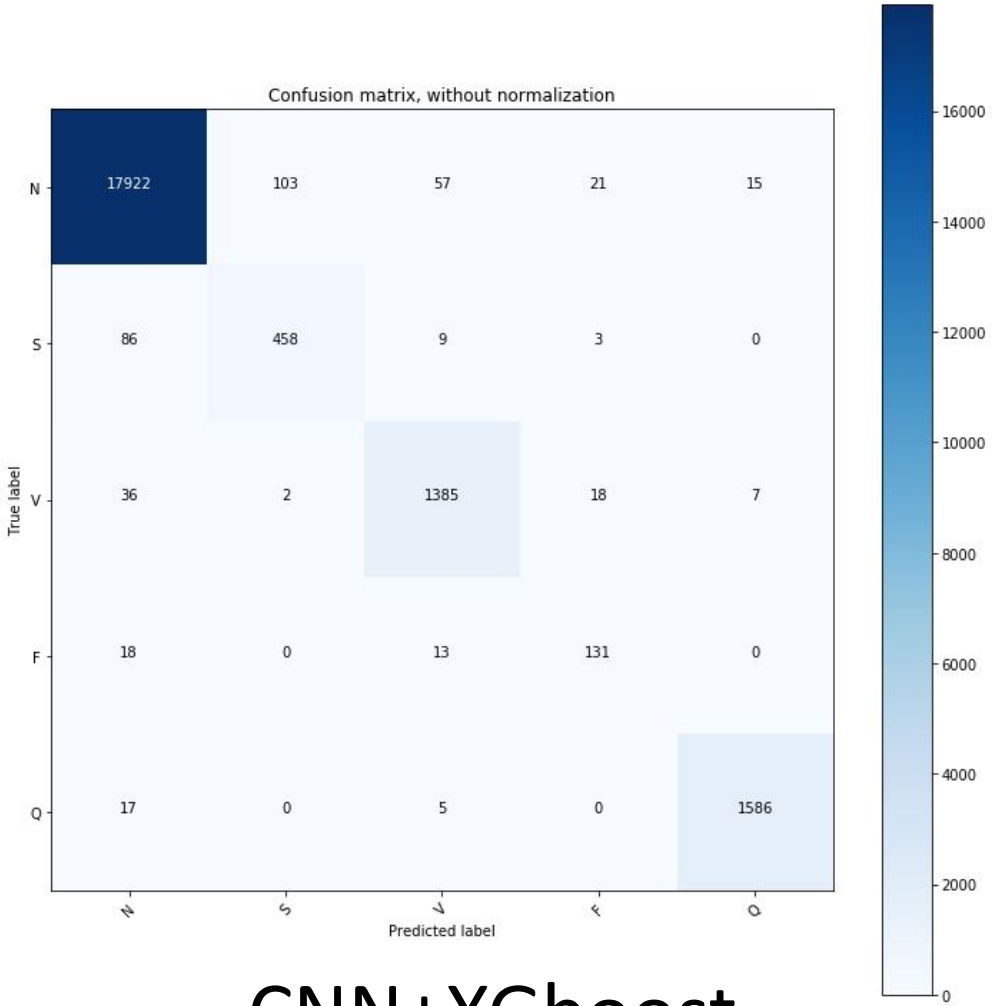
04

分析與改善

增加Batch Normalization的MODEL



CNN

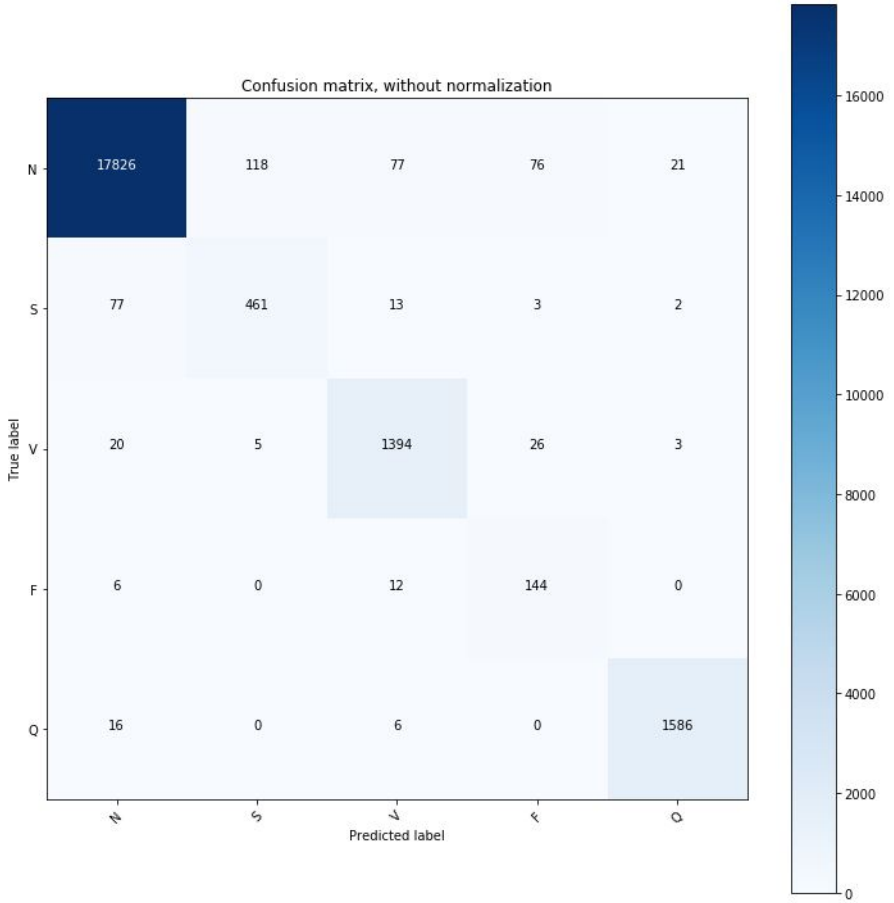


CNN+XGboost

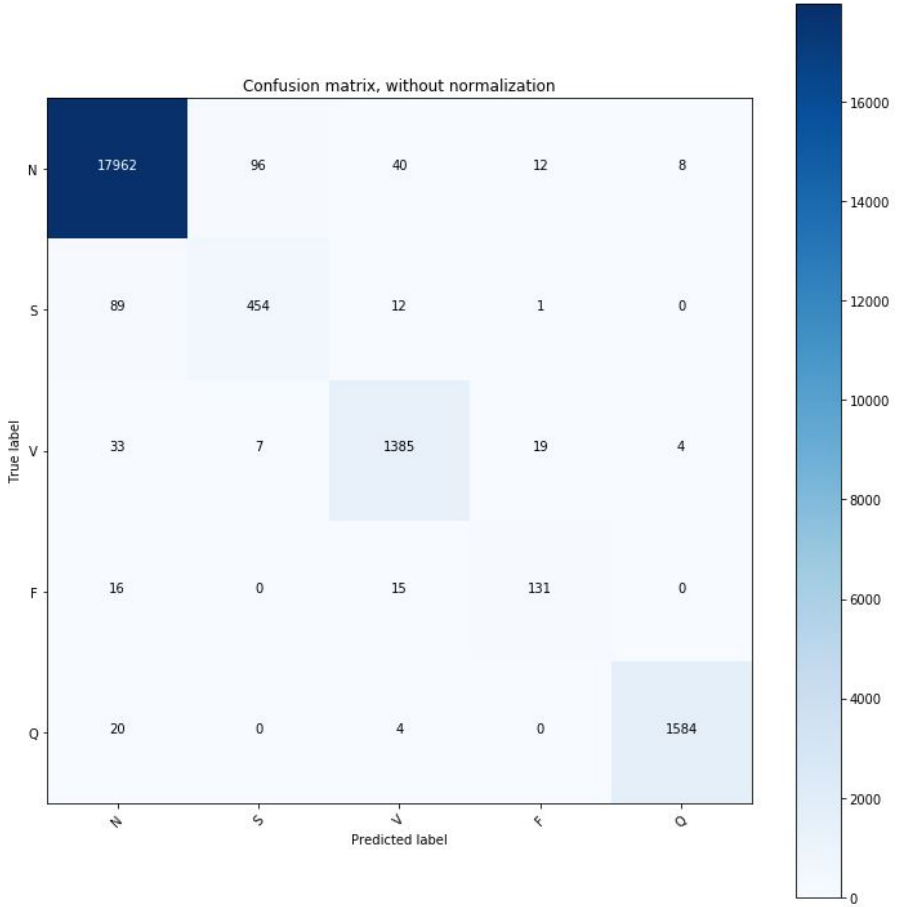
04

分析與改善

最終的MODEL



CNN

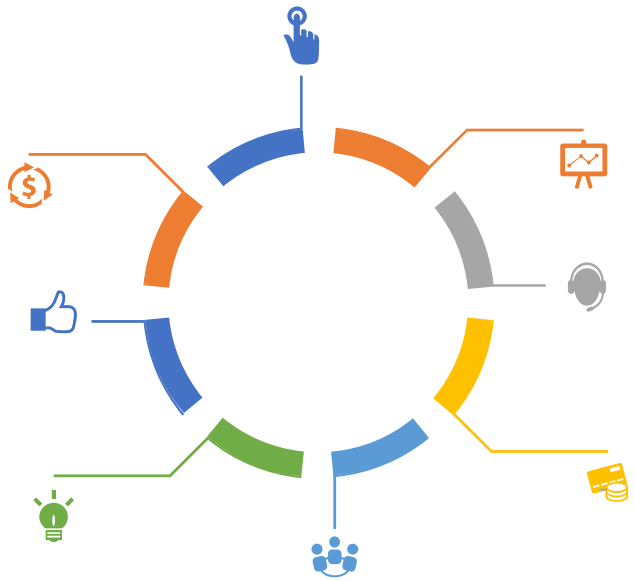


CNN+XGboost

05 結論

- 從CNN+XGboost的分析結果來看，只有樣本數最多的N類的泛化能力有提升，有可能是因為XGboost可以提升模型的可解釋力，但樣本數少的類別因為CNN模型訓練本來就不太好，反而導致模型在經過XGboost後就更差。

- 透過此模型可以協助醫生在評斷心律不整的病徵時，能夠多一個客觀的分析，讓醫生能夠快速判斷病徵，使病人及早治療。



- 根據本研究改善後所提出的模型，測試集準確率達99.2%，比起原先所提出的測試集96.65%有微幅的提升，而看到confusion matrix上，比起原先模型，有小部分的改善，其中僅有類別S的病徵的預測能力下降。

- 未來可以考慮單使用XGboost來建構模型，並進行比較。

Thank for your listening