ML Final Project Report

Freesound General-Purpose Audio Tagging Challenge

電機四 b03901098 王建翔
電機四 b03901109 陳緯哲
電機四 b03901015 梅希聖
電機三 b04901092 詹鈞凱

一、題目介紹與動機

這次選擇 Freesound General-Purpose Audio Tagging Challenge 為題目，是去考驗我們對於真實存在現實環境中的聲音，是否訓練出有辨識能力的 model，這些聲音的來源非常多樣，有可能是長度短且容易被辨別的嬰兒笑聲、也有可能是長度長一點且難以區別二者的伐木機和攪拌機、還有長度到 50 秒又有許多聲源混在一起的市場背景音，我們要根據 Freesound 和 Google Research's Machine Perception Team 所提供的 dataset 做為 training data，去預測是 41 個 label 裡頭的哪一個，最後可以預測的時候能夠對每一筆音訊檔上傳三個 label 做為自己的答案。

關於為什麼會選擇這個題目，一方面是因為在資料量越來越大的時代，要人工為每一段資料去做 label 是 cost 越來越大的一件事，所以能夠自動且精確的把資料，不管是圖片或是音訊檔做 label，都是愈發重要的事，我們認為這一題目的研究可以讓我們試著去學習建立精確的 label model；另一方面是有組員修習過李琳山老師的數位語音處理概論這門課，在找資料方面會比較有方向，對於將在該堂學到的理論和在機器學習這門課上所學習的 model 應用做結合也有濃厚的興趣，所以便決定以此題目做為我們的 Final Project。

二、資料處理

1. 音訊處理

在這次的 final project 中，訓練資料一共有 9500 筆，其中 3700 筆有經過人工 label，剩下的 5800 筆並沒有經過人工 label，因此正確率只有 60-70%；在此我們將資料分為 verified 與 non-verified，並同時對兩者進行頻率轉換，轉換完後因為每個音訊檔的長度不一，所以也對各音訊檔 padding 到一定的長度，之後再找出 value 最大的時間，取得該時間點前後 150 單位的資料進行訓練；test data 的部分也是按照以上的過程進行處理，再輸入模型之中進行預測。

2. 增加訓練資料

若只有 3700 筆完全正確的資料，對於訓練來說可能還是太少了，沒辦法為 41 種類別都找到 fit 的模型，因此我們使用了兩個方式來增加訓練的資料，並且希望能保持音訊本身的特性：

a. 增加雜訊

在拿到時域的訊號後，為其加入隨機且大小適中的雜訊，讓音訊聽起來的雜音增加但又不影響判別其分類，並希望將其加入訓練後，能增加模型抗噪的能力。

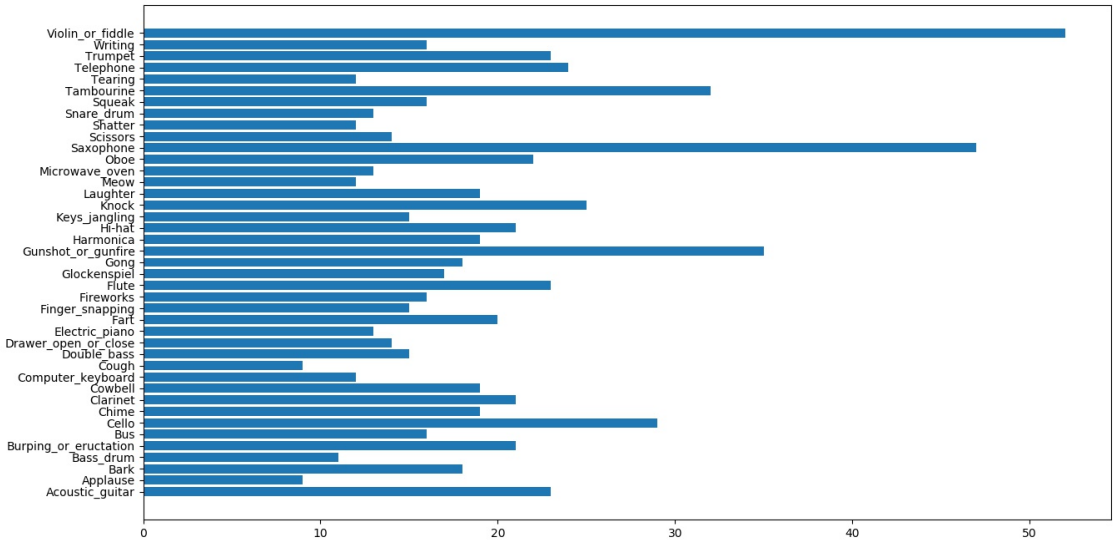
b. 移動音訊

將音訊進行移動，例如往後 5 秒才開始，並將最後 5 秒放到前 5 秒。

透過以上兩個方法，我們便有了足夠且可信賴的訓練資料，在 supervised learning 下便能獲得足夠的準確率。

3. Valid Data

我們將 verified data 的前 800 筆作為 valid data，這 800 筆資料的分類分布可以參考下圖，並確認其分布符合 training data 的分布：



三、模型敘述

1. 架構

我們一共使用兩種架構，分別為 CNN 與 RNN

a. RNN

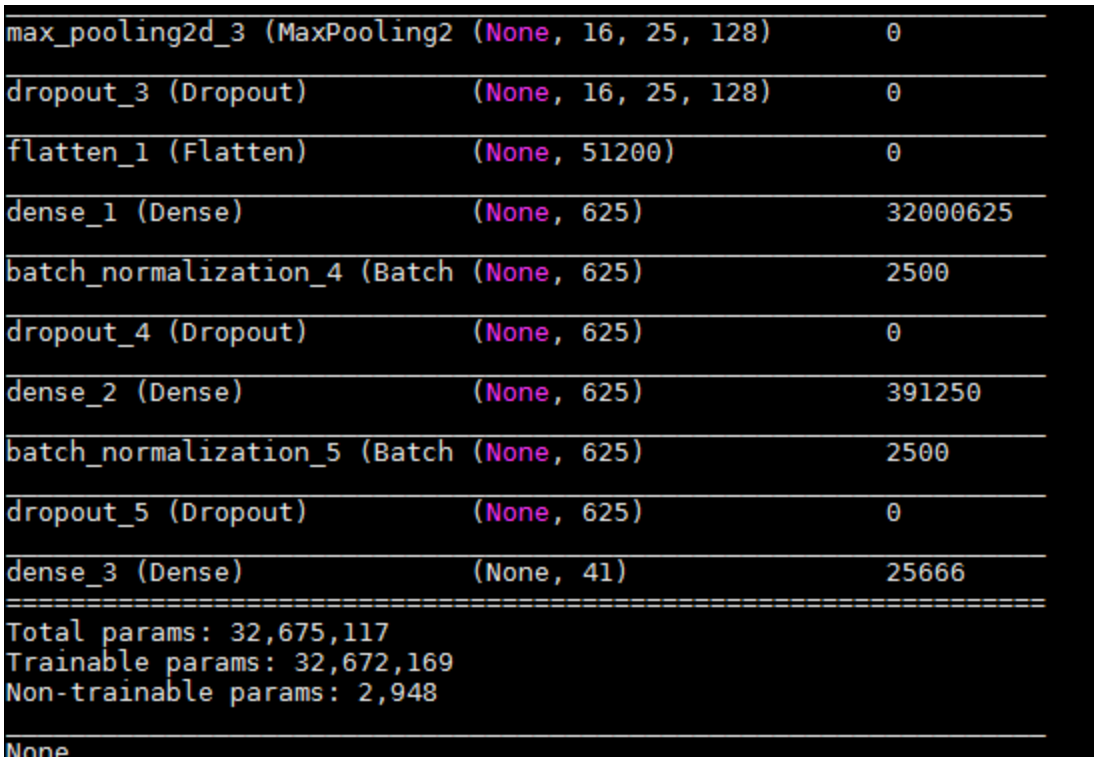
RNN 的話則採用 LSTM 為我們的 cell，Time_Step 和 Input_Size 分別為 spectrogram 的長和寬，以 Batch_Size = 50 來做訓練，model summary 如下圖：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 129, 200)          400800
_________________________________________________________________
bidirectional_1 (Bidirection (None, 400)               641600
_________________________________________________________________
dense_1 (Dense)              (None, 41)                16441
_________________________________________________________________
activation_1 (Activation)    (None, 41)                0
=================================================================
Total params: 1,058,841
Trainable params: 1,058,841
Non-trainable params: 0
_________________________________________________________________
```

b. CNN

我們採用 2D 的 CNN，共有 3 層 CNN，3 層 Fully Connected Layer，詳細架構如下：

```
=================================================================
conv2d_1 (Conv2D)            (None, 129, 300, 32)      1632
_________________________________________________________________
batch_normalization_1 (Batch (None, 129, 300, 32)      128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 100, 32)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64, 100, 32)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 100, 64)       102464
_________________________________________________________________
batch_normalization_2 (Batch (None, 64, 100, 64)       256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 50, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32, 50, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 50, 128)       147584
_________________________________________________________________
batch_normalization_3 (Batch (None, 32, 50, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 25, 128)       0
```



2. 訓練方式

以上兩個架構都是使用以下所提及之訓練方式：

先利用 verified data 進行訓練，產生一個準確率足夠的模型後，再使用 semi-supervised learning 繼續進行訓練，詳情請見 五、討論 3. semi-supervised learning，有較為詳細的解釋。

四、測試結果

將以上兩個架構進行訓練後，分別上傳到 kaggle 上，可以得到以下成績

	CNN	RNN
MAP@3	0.867	0.77

五、討論

1. 使用不同的音訊處理方式(spec 與 mfcc 之比較)

我們使用 spectrogram 來對音訊進行轉換，但在找尋音訊處理的資料時，我們發現有很多人在辨認音訊時都會使用梅爾倒頻譜係數(MFCC)來進行處理，該處理方式的特點為考慮到人耳對不同頻率的感受程度，因此特別適合用在語音辨識，也能區別聲音中主體與客體間的差異，但我們的訓練目標為一般性的，因此我們很好奇 MFCC 能不能達到比用 spectrogram 更好的結果。

我們將音訊檔進行 MFCC 的轉換，並同樣擷取長寬為(300,129)的資料陣列，並放入與 spectrogram 一樣架構的模型中進行訓練，再利用 verified data 訓練出第一個模型後，我們繼續使用 non-verified data 進行 unsupervised learning，最後的 valid data 準確率如下表

	spectrogram	MFCC
Valid MAP@3	0.8836	0.846

可以看出 spectrogram 的效果較好，我們討論後認為 MFCC 可能比較用於語音辨識等明顯主體的聲音，而 spectrogram 對一般性音訊的辨別性較高。

2. RNN or CNN

CNN 的應用有非常多，不管是在自然語言處理、圖像辨識等都有好的表現，而這次的語音辨識問題也可以交由 CNN 來處理，因為語音問題的.wav 可以轉成 Spectrogram，那一但有了圖片，CNN 就可以藉由訓練不同 feature 的 Classifier 來處理相關的辨識問題；除了 CNN 之外，RNN 因為對不同順序和時間的輸入敏感，而常用於機器翻譯、預測報告等和時間有密切關係的活動，我們也就想到這一次的語音辨識問題是否也可以透過 RNN 的結構來訓練 model，因為 RNN 可以考慮到序列輸入的前後順序來做 class 的判定，這一次總共有 41 個 class，每一個 class 的聲音在不同的時間點上也會有不同的輸入，我們認為是可以做為訓練的 feature，所以，我們也嘗試了使用 RNN 來訓練 model，以下將究其二者討論：

在這次的 Project 中，我們主要是以 CNN 為我們的主要架構，而我們將音訊由 wav 先轉成 spectrogram，再轉成 numpy 檔之後，將其輸入到 CNN，model summary 如之前模型敘述所述，CNN 的 model 在訓練時較為容易，val_error 也下降的比較快，準確率也比較高。

	CNN	RNN
Valid MAP@3	0.8497	0.78207

Valid data 取 800 筆，上表是兩者的準確率比較圖，可以看出 CNN 的準確率較 RNN 為高，model 在訓練時的收斂速度也比較快，我們認為對 spectrogram 取各式各樣的 feature 來做 classifier 比去根據 spectrogram 對時間取 feature 還要適合來解決數位語音辨識的問題，如果要以 RNN 來做語音辨識的問題，在以人類發出的語音是可能會有比較好的辨識結果，因為人類所使用的語言，先後順序對於語意的理解影響極大，所以我們後來認為這次的 data 在執行上對時間的敏感度不夠，所以在此 CNN 較 RNN 適用。

3. Semi supervised learning

在這次題目所提供的 data 中，有些是已經 Verified 過的資料，這些資料可以直接用來 train model，但是也有許多是尚未被 verified 的資料可以用來做 Semi supervised learning，首先我們先將在 Supervised learningr 階段 train 好的 model 給 load 下來後對 unverified data 做 predict，若超過一定的 threshold 就將其 label 為該 class，以下是分別是 RNN 和 CNN 做了 Semi supervised learning 後所呈現的結果：

	CNN	RNN
Valid MAP@3	0.88365	0.7887

由上表可以發現，是否有更多具一定信心水平的 data 對於 training 是很重要的，原本 verified 的 data 只有 3700 筆，但透過額外的 5800 筆 unverified data 也大大的改善了 model 的 performance。

4. Ensemble by single-class classifiers

此方法是對於 41 個不同類別，各自訓練一個 single-class classifier，最後在聚合成完整的 multi-class classifier。我們預期如果每個 single-class classifier 都能各自達到很高的準確率，則聚合成的 multi-class classifier 應該也會有極高的準確率。

我們使用與前述 CNN model 類似的架構，3 層的 convolution layer + pooling 接上 2 層 DNN layer，差別是最後一層改為 1 個 sigmoid 的 output。因為 single class 的緣故，訓練資料類別比例並不平均（label=0 的數量遠大於 label=1 的數量），預測之結果會傾向將所有資料都預測為 0，因此嘗試加入 weights 以平衡如此之偏差。

訓練過程：

```
20224/20370 [============================>.] - ETA: 0s - loss: 0.9036 - acc: 0.6571Epoch 00001: val_loss improved from inf to 0.45
20370/20370 [==============================] - 53s 3ms/step - loss: 0.9035 - acc: 0.6578 - val_loss: 0.4585 - val_acc: 0.9712
Epoch 2/20
20224/20370 [============================>.] - ETA: 0s - loss: 0.5463 - acc: 0.8276Epoch 00002: val_loss improved from 0.45853 to
20370/20370 [==============================] - 105s 5ms/step - loss: 0.5446 - acc: 0.8280 - val_loss: 0.1483 - val_acc: 0.9712
Epoch 3/20
20370/20370 [==============================] - 108s 5ms/step - loss: 0.3329 - acc: 0.9132 - val_loss: 2.3916 - val_acc: 0.0437
Epoch 4/20
20370/20370 [==============================] - 99s 5ms/step - loss: 0.2298 - acc: 0.9476 - val_loss: 0.7033 - val_acc: 0.6763
Epoch 5/20
20370/20370 [==============================] - 93s 5ms/step - loss: 0.1797 - acc: 0.9625 - val_loss: 0.2098 - val_acc: 0.9425
```

然而，雖然 val_acc 很高，validation set 的 true positive rate 卻只有 0.09。將 41 個 model 聚合之後，其 MAP@3 score 也十分不理想。

雖然加入了 weights 試圖平衡偏差，model 依然無法成功的學習。原因有可能是單一 class 之 data 之數量不夠，model 並無法找出類別之共同特徵，或者是從音訊資料抽取出之 feature 本身就不具有很明顯的共同特徵，導致此方法之失敗。

六、結論與未來方向

在這次 project 中，我們運用了這學期學到的各種知識與技巧，不斷的試圖改善我們的準確度，而適逢期末，在時間有限的情況下，我們仍試著挪出時間來做出各種不同的嘗試，spectrogram 與 MFCC、CNN 與 RNN、Ensem，甚至是做出 41 個 model 的 ensemble，雖然最後在排名上不盡理想，但這次 project 學到很多東西，收穫良多；而看向 kaggle 的排名，我們其實十分不甘

心，而距離 kaggle 上的期限還有一陣子，也許我們還有時間，繼續進行各種不同的嘗試，讓我們的排名可以再往上一些。

七、參考資料

1. https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data
2. https://www.kaggle.com/CVxTz/audio-data-augmentation
3. https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
4. http://python-speech-features.readthedocs.io/en/latest/
5. https://www.svds.com/tensorflow-rnn-tutorial/