

ML Final Project Report -- Sound

Member: b05902002 李栢淵、b05902013 吳宗翰

b05902019 蔡青邑、b05902042 林瑋毅

Team: NTU_b05902013_到底要選哪一題QQ

1. Introduction & Motivation

在這次的作業中，我們打的是外面的kaggle比賽 -- Freesound General-Purpose Audio Tagging Challenge。在競賽中我們會分別得到9000多筆的 `.wav` 音訊檔及其label去預測9000多筆的音訊檔屬於哪個類別。

雖然這只是一個基本的classification問題，不過有幾點問題是困難的：

1. 音訊檔的data preprocessing是重要的，其中的issue包括要選取的秒數、要不要先preprocessing、training的時候要不要random crop data等等問題
2. 在這個classification問題中有相當多個class(41個)，而且在簡單的investigate後會發現labels並不平衡，因此這可能會造成在訓練的時候沒有那麼簡單；此外雖然這是音訊辨識，不過由於音訊是有時間性的，因此要不要使用RNN也是一個issue。
3. MAP@3 Evaluation和我們常見的MAP@1在根本上有著不同之處。在MAP@1中，我們只需要minimize cross entropy即可，但在MAP@3中，我們還要多考慮如何在第一個答錯的狀況下，盡可能的讓自己的model在第2, 3比預測中趕快補救以增加Accuracy。

以上的三點問題我們將分別在Section 2中的preprocessing、Section 3中的Model Description以及Section 4中的Experiment and Discussion中提到以及說明。

2. Data Preprocessing/Feature Engineering

2-1 Raw Data

為處理個音訊檔長度不等的問題，訓練時，對於長度超過 2 秒的檔案，我們會隨機截取一段長度為兩秒的區間，而對於長度未達 2 秒的檔案，我們則在其前後補零（所以補上的區間是沒有聲音的），使長度曾為兩秒；預測時，對於長度超過 2 秒的檔案，我們一樣會隨機截取一段長度為兩秒的區間，而對於長度未達 2 秒的檔案，我們也是在其前後補零，但是為避免 2 秒太短，可能截取到代表性不足的區間，對於同一個音訊檔，我們會做十次，並取其機率的平均值作為輸出。

2-2 MFCC

由於我們在raw data上面好像train不太起來，因此就考慮使用簡單的preprocessing，希望可以extract出好的feature，而其中一個就是MFCC。由於聲音的時間不定，因此我們嘗試了crop多種不同的長度(詳細內容寫在Section 4中)，對於過長的音訊就crop掉後面的，至於過短的音訊則是用 `np.resize` 複製補足。待我們得到相同長度的音訊後，我們便把這些音訊送去MFCC preprocessing，產生40個MFCC。

3. Model Description

3-1 CNN on raw data

此 model 的輸入為 2 秒 44100Hz 的單聲道音訊檔，使用的 model 結構如下：

```
1D Conv (channels=16, kernel_size=9), ReLU()  
1D Conv (channels=16, kernel_size=9), ReLU()  
1D Max Pooling (kernel_size=16), Dropout(p=0.2)  
  
1D Conv (channels=32, kernel_size=3), ReLU()  
1D Conv (channels=32, kernel_size=3), ReLU()  
1D Max Pooling (kernel_size=4), Dropout(p=0.2)  
  
1D Conv (channels=32, kernel_size=3), ReLU()  
1D Conv (channels=32, kernel_size=3), ReLU()  
1D Max Pooling (kernel_size=4), Dropout(p=0.2)  
  
1D Conv (channels=256, kernel_size=3), ReLU()  
1D Conv (channels=256, kernel_size=3), ReLU()  
1D Global Max Pooling, Dropout(p=0.2)  
  
Linear(in=256, out=1024), ReLU()  
Linear(in=1024, out=1024), ReLU()  
Linear(in=1024, out=41)
```

其中，optimizer 使用 Adam (Pytorch 預設參數)，batch size 為 64，epoch 數為 300。另外，data preprocessing 的部分如 Section 2. Raw Data 所述，本節不在提及。

3-2 CNN on MFCC

```
2D Conv (channels=32, kernel_size=(2, 5)), BN, ReLU()  
2D Max Pooling (kernel_size=(2, 2))  
  
2D Conv (channels=64, kernel_size=(2, 5)), BN, ReLU()  
2D Max Pooling (kernel_size=(2, 2))  
  
2D Conv (channels=128, kernel_size=(2, 5)), BN, ReLU()  
2D Max Pooling (kernel_size=(2, 2))  
  
2D Conv (channels=256, kernel_size=(2, 5)), BN, ReLU()  
2D Max Pooling (kernel_size=(2, 2))  
  
2D Conv (channels=384, kernel_size=(2, 5)), BN, ReLU()  
2D Max Pooling (kernel_size=(2, 2))  
Linear(in=flatten_size, out=1024), ReLU(), Dropout(0.5)  
Linear(in=128, out=41)
```

其中，optimizer 使用 Adam ($lr = 10^{-4}$, 其餘都是keras預設參數)，batch size 為 32，epoch 數為 200。(不過有用Early Stop所以根本跑不到那麼久)

另外，data preprocessing 的部分如 Section 2. MFCC 所述，本節不在提及。

3-3 RNN + CNN on MFCC

想使用RNN的動機是因為考慮到音訊檔有其時間順序的關係，因此想說先拿LSTM cell去提取音訊資訊，接著再把帶有時間順序的東西送給CNN做feature extraction。

```
LSTM (input_size=40, hidden_Size=256)

2D Conv (channels=16, kernel_size=(4, 4), stride=2), BN, ReLU()
2D Max Pooling (kernel_size=(2, 2))

2D Conv (channels=32, kernel_size=(4, 4), stride=1), BN, ReLU()
2D Max Pooling (kernel_size=(2, 2))

2D Conv (channels=64, kernel_size=(3, 3), stride=1), BN, ReLU()
2D Max Pooling (kernel_size=(2, 2)), Dropout(p=0.2)

2D Conv (channels=128, kernel_size=(3, 3), stride=1), BN, ReLU()
2D Max Pooling (kernel_size=(2, 2)), Dropout(p=0.2)

2D Conv (channels=128, kernel_size=(3, 3), stride=1), BN, ReLU()
2D Max Pooling (kernel_size=(2, 2)), Dropout(p=0.2)

Linear (in=1024, out=256), Dropout(p=0.2), BN, ReLU()
Linear (in=256, out=41)
```

Optimizer的話使用Adam， $lr = 10^{-4}$ 以外都用pytorch預設的參數，batch_size=32。至於預處理已經在上面提及。

3-4 XGBOOST extract feature

1. Feature Extraction

xbboost背後的原理是gradient boosting算法，因此並沒有NNet提取feature的能力，必須由我們自己手動提取。以下是我們選取的feature：

```
mfccs = librosa.feature.mfcc(y=X, sr=sample_rate, n_mfcc = 20)
tempo, beats = librosa.beat.beat_track(y=X, sr=sample_rate)
cr = librosa.feature.zero_crossing_rate(y=X)[0]
roll = librosa.feature.spectral_rolloff(y=X, sr=sample_rate)[0]
cen = librosa.feature.spectral_centroid(y=X, sr=sample_rate)[0]
con = librosa.feature.spectral_contrast(y=X, sr=sample_rate)[0]
band = librosa.feature.spectral_bandwidth(y=X, sr=sample_rate)[0]
f1 = np.concatenate([np.mean(mfccs, axis=1), np.std(mfccs, axis=1),
np.max(mfccs, axis=1), np.max(mfccs, axis=1), skew(mfccs, axis=1)])
f2 = []
for x in [cr, roll, cen, con, band]:
    f2 += [np.mean(x), np.std(x), np.max(x), np.max(x), skew(x)]
feature = np.concatenate([f1, f2])
```

2. xgboost的參數如下：

- number of estimator: 1200
- learning rate: 0.05
- max_depth: 8

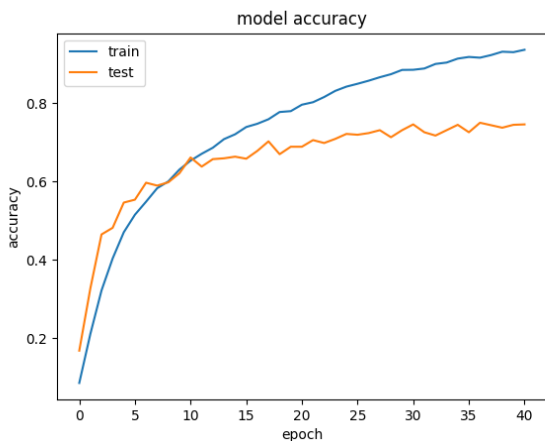
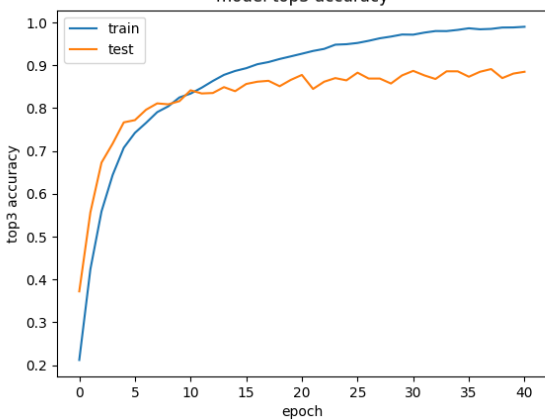
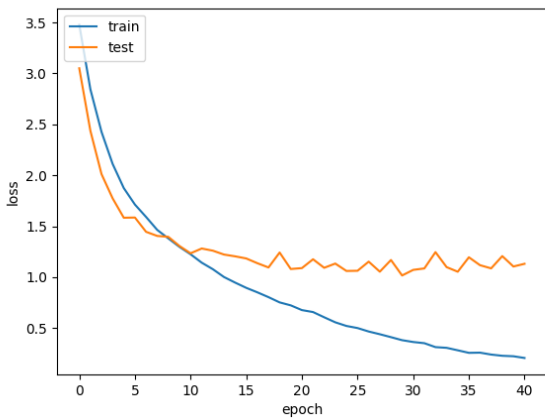
4. Experiment and Discussion

4-1 Experiment

4-1-1 Training Plot

以下大概是我們用單一model並使用前面Section 3-2所提及的model的training plot，顯然地，在十幾個epoch時，這個model就已經獲得還不錯的成果。如圖所示，MAP@1 Accuracy大概只能到0.7而已，不過training data的Top3 Accuracy卻可以到達1，這個model很容易overfit，所以其中的Dropout設得很高。

最終我們就用了諸多不同duration的MFCC，train出了多個mdoel，將特別差的幾個砍掉(在test不到0.86)，再做ensemble。

	Training History																														
MAP@1 Accuracy	 <p>This line graph, titled "model accuracy", plots accuracy against epoch number from 0 to 40. The y-axis, labeled "accuracy", ranges from 0.2 to 0.8. Two lines are shown: a blue line for "train" accuracy and an orange line for "test" accuracy. Both lines start at epoch 0 with an accuracy of approximately 0.1. The training accuracy increases steadily, reaching about 0.9 by epoch 40. The test accuracy also increases, reaching about 0.75 by epoch 40, with some minor fluctuations between epochs 10 and 40.</p> <table><tr><th>epoch</th><th>train</th><th>test</th></tr><tr><td>0</td><td>0.1</td><td>0.15</td></tr><tr><td>5</td><td>0.5</td><td>0.55</td></tr><tr><td>10</td><td>0.65</td><td>0.65</td></tr><tr><td>15</td><td>0.75</td><td>0.68</td></tr><tr><td>20</td><td>0.8</td><td>0.7</td></tr><tr><td>25</td><td>0.85</td><td>0.72</td></tr><tr><td>30</td><td>0.88</td><td>0.74</td></tr><tr><td>35</td><td>0.9</td><td>0.75</td></tr><tr><td>40</td><td>0.9</td><td>0.75</td></tr></table>	epoch	train	test	0	0.1	0.15	5	0.5	0.55	10	0.65	0.65	15	0.75	0.68	20	0.8	0.7	25	0.85	0.72	30	0.88	0.74	35	0.9	0.75	40	0.9	0.75
epoch	train	test																													
0	0.1	0.15																													
5	0.5	0.55																													
10	0.65	0.65																													
15	0.75	0.68																													
20	0.8	0.7																													
25	0.85	0.72																													
30	0.88	0.74																													
35	0.9	0.75																													
40	0.9	0.75																													
MAP@3的 Accuracy	 <p>This line graph, titled "model top3 accuracy", plots top3 accuracy against epoch number from 0 to 40. The y-axis, labeled "top3 accuracy", ranges from 0.2 to 1.0. Two lines are shown: a blue line for "train" top3 accuracy and an orange line for "test" top3 accuracy. Both lines start at epoch 0 with a top3 accuracy of approximately 0.2. The training top3 accuracy increases steadily, reaching about 1.0 by epoch 40. The test top3 accuracy also increases, reaching about 0.88 by epoch 40, with some minor fluctuations between epochs 10 and 40.</p> <table><tr><th>epoch</th><th>train</th><th>test</th></tr><tr><td>0</td><td>0.2</td><td>0.35</td></tr><tr><td>5</td><td>0.7</td><td>0.75</td></tr><tr><td>10</td><td>0.85</td><td>0.85</td></tr><tr><td>15</td><td>0.9</td><td>0.85</td></tr><tr><td>20</td><td>0.95</td><td>0.85</td></tr><tr><td>25</td><td>0.98</td><td>0.88</td></tr><tr><td>30</td><td>0.99</td><td>0.88</td></tr><tr><td>35</td><td>1.0</td><td>0.88</td></tr><tr><td>40</td><td>1.0</td><td>0.88</td></tr></table>	epoch	train	test	0	0.2	0.35	5	0.7	0.75	10	0.85	0.85	15	0.9	0.85	20	0.95	0.85	25	0.98	0.88	30	0.99	0.88	35	1.0	0.88	40	1.0	0.88
epoch	train	test																													
0	0.2	0.35																													
5	0.7	0.75																													
10	0.85	0.85																													
15	0.9	0.85																													
20	0.95	0.85																													
25	0.98	0.88																													
30	0.99	0.88																													
35	1.0	0.88																													
40	1.0	0.88																													
Categorical Crossentropy	 <p>This line graph, titled "model loss", plots loss against epoch number from 0 to 40. The y-axis, labeled "loss", ranges from 0.5 to 3.5. Two lines are shown: a blue line for "train" loss and an orange line for "test" loss. Both lines start at epoch 0 with a loss of approximately 3.2. The training loss decreases steadily, reaching about 0.2 by epoch 40. The test loss also decreases, reaching about 1.1 by epoch 40, with some minor fluctuations between epochs 10 and 40.</p> <table><tr><th>epoch</th><th>train</th><th>test</th></tr><tr><td>0</td><td>3.2</td><td>3.1</td></tr><tr><td>5</td><td>1.8</td><td>1.6</td></tr><tr><td>10</td><td>1.2</td><td>1.3</td></tr><tr><td>15</td><td>0.8</td><td>1.2</td></tr><tr><td>20</td><td>0.6</td><td>1.1</td></tr><tr><td>25</td><td>0.4</td><td>1.1</td></tr><tr><td>30</td><td>0.3</td><td>1.1</td></tr><tr><td>35</td><td>0.2</td><td>1.1</td></tr><tr><td>40</td><td>0.2</td><td>1.1</td></tr></table>	epoch	train	test	0	3.2	3.1	5	1.8	1.6	10	1.2	1.3	15	0.8	1.2	20	0.6	1.1	25	0.4	1.1	30	0.3	1.1	35	0.2	1.1	40	0.2	1.1
epoch	train	test																													
0	3.2	3.1																													
5	1.8	1.6																													
10	1.2	1.3																													
15	0.8	1.2																													
20	0.6	1.1																													
25	0.4	1.1																													
30	0.3	1.1																													
35	0.2	1.1																													
40	0.2	1.1																													

4-1-2 Result

以下是我們多次實驗所獲得的結果，在此先整理一下列成表格，待4-2我們會比較詳細的探討其中一些直得注意的issue。

Training Method	Accuracy
Raw data + 1D Conv	0.777
CNN on MFCC, dutation = 10	0.866
RNN + CNN on MFCC, duration = 10	0.849
XGBOOST self feature extraction	0.785
Ensemble Multiple CNN on MFCC model , duration = 1 ~ 13	0.889

4-2 Discussion

4-2-1 MFCC feature extraction有幫助

經由 [kernel sample code](#) 的啟發，我們曾嘗試直接對raw data作為model的input，然而如上表所示，在經過fine tune後實際上的accuracy表現都不超過80%。因此我們早早就放棄了raw data而改用MFCC preprocessing後的產物當成training data。

4-2-2 CNN不是越多層越好

在這次作業上我們嘗試了各種層數的CNN來extract feature，而在此過程中我們發現了CNN的層數並不是越多越好。經由多次實驗發現，如果是吃MFCC當成input的話，最好的CNN層數應該是介於4~5之間。

如果在大於5層的話收斂速度將變得非常的慢，另外也收斂不到好的local minima；反觀如果我們只使用5層的話，收斂速度就相當的快，大約5個epoch後MAP@3 Accuracy就可以高於60%(在K80上面跑約10分鐘)，另外最終的結果也比多層CNN來得好。

4-2-3 RNN在這個task上並沒有太大幫助

在上表有發現，RNN在這個task上面並沒有辦法達到非常高的Accuracy，另外如果考慮進training時間的話，由於RNN並沒有像CNN一樣完全平行的training，因此多方考量之下覺得RNN在這個task上其實並不好用。

詳細考慮過原因以後，我們覺得原因有二：

- 資料特性不同：由於我們在preprocessing中，只要遇到長度不足的音訊就會用 `np.resize` 進行補足，所以其實他早就已經打破了timestamp的觀念了，也因此RNN就沒有太大的幫助我們。另外MFCC也是透過在時間軸上不同windows切割出來的feature，也有包含前後相關順序的feature，所以應該也不會特別好。
- 問題不同：在Mahcine Learning HW5中，我們使用了RNN進行了文字情緒的判斷，這是因為RNN可以把時間順序考慮進去。然而在這次語音辨識的task中，我們可能從中取出一小段音訊即可判斷其音訊是哪個種類，也可能判斷音訊是分貝大小，跟時間一點關係都沒有。

4-2-4 Ensemble相關Issue

本次作業最後我們也有試著對多個Model做Uniform Blending，經由此Ensemble，我們的Accuracy從原本單一Model最高的86%提升到最後的88.9%。然而在Ensemble多個Model的時候，我們也發現了以下幾點issue值得我們注意：

1. 很難把xgboost跟NNet的model ensemble在一起

經由觀察發現，xgboost模型所預測出來每個class的機率差異相當的大，機率最高的class通常都有20%甚至30%以上而其餘每個可能都只有不到5%。反觀NNet model預測出來的機率往往最高的也不超過20%，因此如果使用uniform blending就會變成xgboost會dominate。

解決方法大概就是用weighted aggregation，不過有鑒於本次作業我們每天只能至多上傳兩次，因此我們很難這樣調整weighted參數，後期就比較少嘗試把xgboost模型ensemble進多個NNet裡面了

2. Ensemble多個NNet model

我們的Ensemble就是把多個不同 `train_test_split` 以及不同參數、音訊duration的NNet model分別對每個class softmax後的probability相加，接著取出前三高的出來做為我們的最終預測。經過多次實驗結果，我們發現了一個重要的事情：取最好的幾個相加起來的平均不一定會最好，因為如果越diverse的model可以有越好的generalization，因此相較於只挑好的那幾個做Ensemble，還不如每一種model都聽幾個拿起來做Aggregation反而有比較好的成效。

5. Conclusion

1. Data preprocessing以及該領域的domain know how是Machine Learning領域中相當重要的一環，以這次的task而言，使用MFCC preprocessing後的資料當成training data的表現明顯比raw data下去train還來得好。
2. 經過我們組內上面的實驗以及看了同學們的報告後，我們更加確定Aggregation Model確實是可以增加其accuracy，即使只是最簡單的uniform blending。
3. Machine Learning的問題還是要回到根本，不要妄想只是無腦train下去就會得到好的Accuracy，如同本次的task仔細想想後就會發現RNN其實並不合理。
4. 由於如上所見MAP@1 Accuracy最高都只能到0.7而已，因此在聽到別的組別報告後，我們認為未來可以試著對於model預測出來的probability再做two-level-learning，用一個NNet硬去correct第一層model所做出的偏差或許是可行的。

6. Reference

- Preprocessing
 - <https://zh.wikipedia.org/wiki/%E6%A2%85%E5%B0%94%E9%A2%91%E7%8E%87%E5%80%92%E8%B0%B1%E7%B3%BB%E6%95%B0>
 - <https://librosa.github.io/librosa/index.html>
- Training Model
 - <https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data>
 - <https://www.kaggle.com/amlanpraharaj/xgb-using-mfcc-opanichev-s-features-lb-0-81>