ML Final Report

Sound Classification Problem

組別：大家都去當大神惹 QQ

組員：R06725035 陳廷易 R06725041 彭証鴻 R06725054 蔡鈞 B03902110 毛偉倫

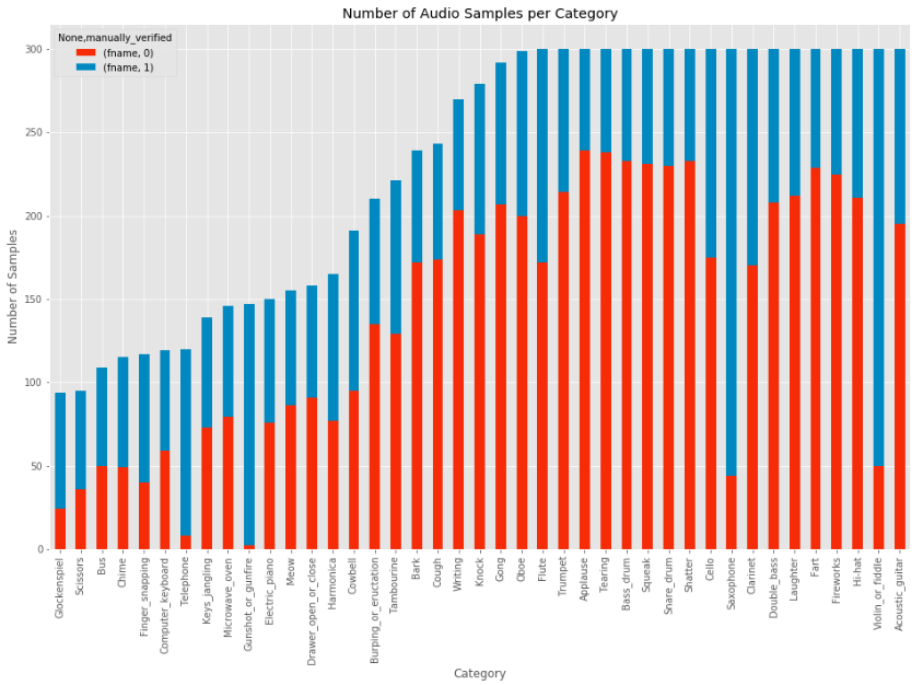
- ## Introduction & Motivation
  - 有些聲音用聽的可以很容易被辨識出來，像小孩的叫聲。但有些聲音，如果閉著眼睛去聽，基本上人很難去辨識，像是電鋸與攪拌機的聲音相似度極高。
  - 至今為止，沒有一個系統可以有效的辨別出這些聲音，而在辨識聲音類別多數都是以人工標籤為主。因此這個比賽的目的就是要參賽者去建立一個 model 達到音檔辨識及分類的效果。
  - 給定 9473 筆的音檔，每個音檔都有自己的類別，把這些檔案當 Training data，要我們去 predict Testing data 的音檔屬於哪一個類別。
  - 總共有 41 種類別
    - "Acoustic_guitar", "Applause", "Bark", "Bass_drum", "Burping_or_eructation", "Bus", "Cello", "Chime", "Clarinet", "Computer_keyboard", "Cough", "Cowbell", "Double_bass", "Drawer_open_or_close", "Electric_piano", "Fart", "Finger_snapping", "Fireworks", "Flute", "Glockenspiel", "Gong", "Gunshot_or_gunfire", "Harmonica", "Hi-hat", "Keys_jangling", "Knock", "Laughter", "Meow", "Microwave_oven", "Oboe", "Saxophone", "Scissors", "Shatter", "Snare_drum", "Squeak", "Tambourine", "Tearing", "Telephone", "Trumpet", "Violin_or_fiddle", "Writing"
  - **Training data**：
    - 3710 筆有人工驗證的 data (有 label 的 data)

- 5763 筆無人工驗證的 data (有 label 但不一定正確的 data)

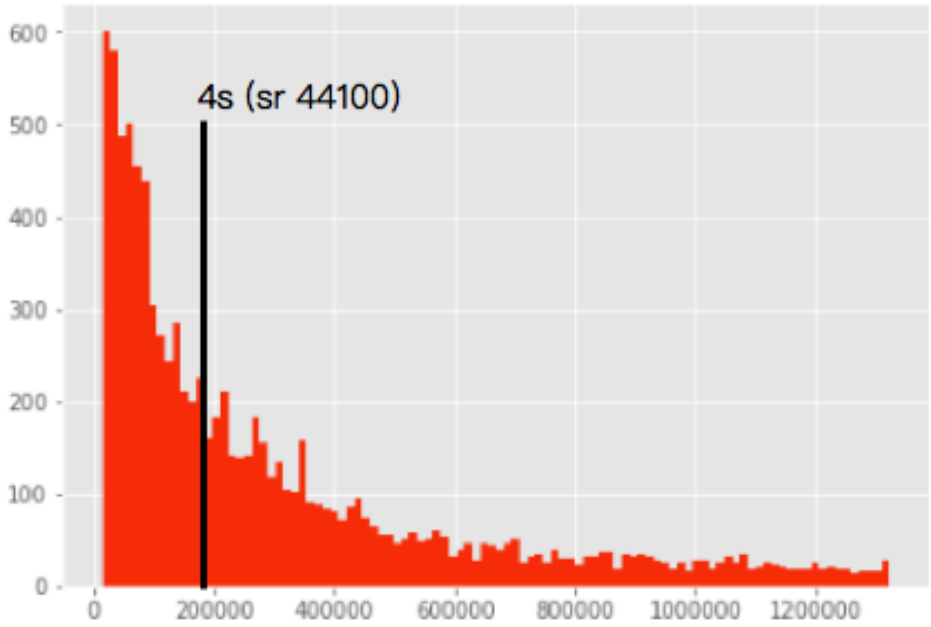
○ **Testing data**：

- 9400 筆有人工驗證的 data (沒有 label 的 data)

○ 資料分析：



▲共 41 個 unique label，此外近 40% training data 的標籤有經人工驗證



▲16-bit 位元深度、Sample Rate 44100。多數音檔不足 5 秒，但有少數可達 10~20 秒

- # Data Preprocessing/Feature Engineering
  - 依據不同時間長度(橫軸)、頻率維度(縱軸)，總共抽取五種 **MFCC Feature**：
    - 四秒、四十維 *(以下簡稱:mfcc1)*
    - 六秒、二十五維 *(以下簡稱:mfcc3)*
    - 八秒、二十維 *(以下簡稱:mfcc4)*
    - 五秒、六十四維 *(以下簡稱:mfcc6)*
    - 三秒、六十維 *(以下簡稱:mfcc7)*
  - 由於每個音檔的長度不一樣，但 ResNet 的 input 維度必須相同，因此必須將 data 做**前置處理**：
    - 太短的 data 做前後 padding（前後補 0）
    - 太長的 data，去頭去尾抽取中間的值
  - Training data 只有 3710 筆有正確 label 的 data，資料量明顯不夠，因此我們利用 mixup algorithm[3][4][9]來做 data augmentation。
- # Model Description
  - **Phase1 (10-fold resnet model with different feature):**
    - ResNet-18/34/50/101/152: 我們 model 主要為使用 keras resnet[2]的 model，如下圖五種不同深度的 resnet，model 架構如下，而我們再最後多增加 dropout 與 fully connected layer。

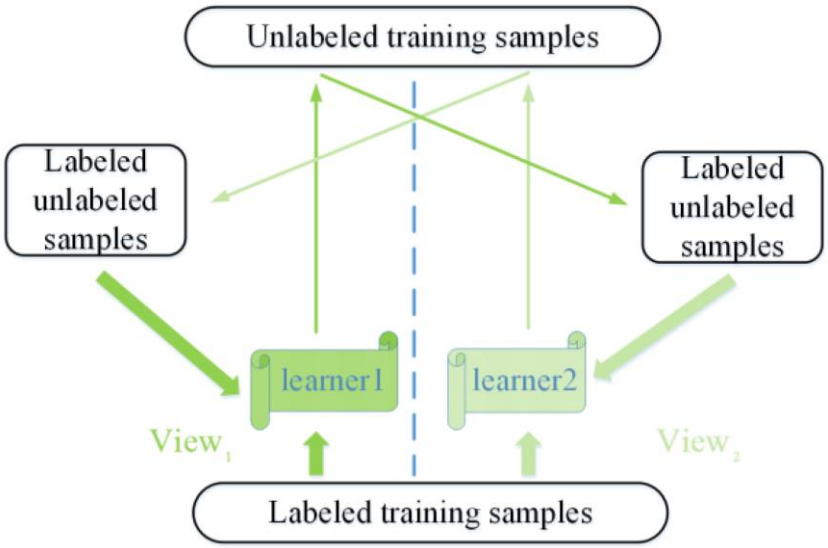
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

我們十種 model 將採用不同的 mfcc feature，互相搭配情況製表如下：

	resnet_18	resnet_34	resnet_50	resnet_101	resnet_152
mfcc	V				V
mfcc3		V			V
mfcc4	V				V
mfcc6		V	V		
mfcc7		V		V	

- 10-fold: 十種 model 會從 3710 筆中隨機切出不同的 10 fold，進行 10-fold cross validation。在 phase1 只會對有人工 verified 的 data 進行訓練，訓練完成後每個 model 的每個 fold 要 predict 全部的 semi data(沒有人工 unverified 的以及 testing data)共 15163 筆。

Phase2 (Co-Train)：



- 依據各 model 所 predict 出來的 semi data，並依據各 fold 之 validation accuracy 進行 weighted sum ensemble。再依 ensemble 的結果求出每筆 fname 的 argmax 作為 label，而該 argmax 值做為信心指數。
- 求出信心指數的平均值與標準差的和做為 threshold， 而將信心指數超過 threshold 的 fname 及新給的 label 記錄下來。因該 kaggle 於 data description 有提到，未經人工 verified 的約有 30~40%資料是錯誤的，而設此 threshold 可確保資料僅會剩下

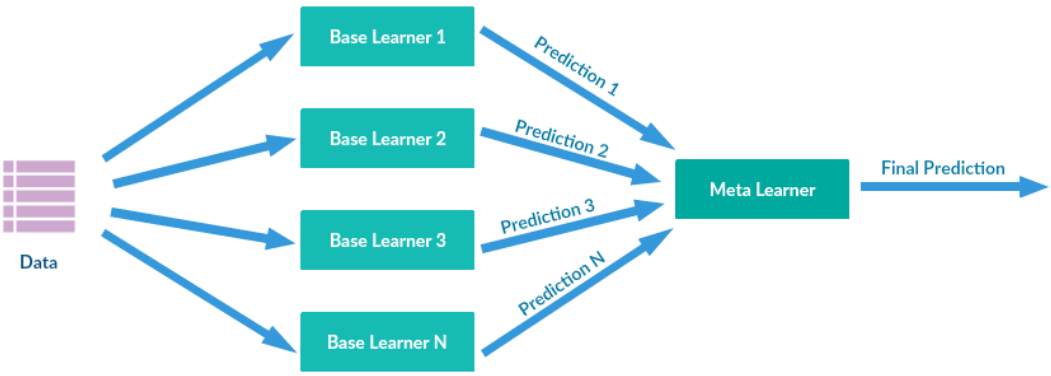
16~32%，進而確保所給的 label 皆為正確的。

- 不同 model 間交換彼此記錄下來的 fname 與新 label，進行 fine tune。期望藉此方式降低各 model 的 bias error。
- fine tune 後各 model 各 fold 再一次 predict 全部 15163 筆 semi data。

- **Phase3 (Self-Train):**

- 將十個 model 共 100 個 fold 的 semi data，利用其所對應之 validation accuracy 進行 weigted sum ensemble。
- 求出 ensemble 後的 argmax 做為 label，其所對應之值做為信心指數。
- 求出信心指數的平均值與標準差的和作為 upper threshold，而頻均值作為 lower threshold。
- 若該 fname 經 ensemble 後所得之 label，其所對應的信心指數超過 upper threshold 則記錄下來。又或信心指數超過 lower threshold，但我們所 ensemble verified 得到的 label 與 dataset 所給的 unverfied label 一致的話，也記錄下該筆資料的 fname 與 label。
- 十個 model 皆拿相同經上述處理的 fname 與 label 對應至各自的 mfcc feature 進行 fine tune
- 最後，每個 model 的每個 fold 要 predict 全部的 testing data 輸出 softmax distribution 值。而每個 model 也要各 fold 也需 predict 自身的 validation fold data。

- **Phase4 (Stacking):**



- Stage1:
  - 將 phase3 十個 model 十個 fold 共一百個 softmax 輸出，當成兩種 model 的輸入。
  - 第一種 model 為將十種 model predict 出來的 softmax 41 維度 concate 在一起變成 410 維度作為輸入，經過四層 fully-connected(neurons_unit= 256、256、64、64；activation_function='relu'；optimizer=adam；loss_function=categorical_crossentropy)後以 sigmoid 輸出。利用 verified data 3710 筆切 training 與 validation(9:1)，並記錄 validation accuracy。
  - 第二種 model 為將十個 model 的 softmax 值進行 weighted sum 在一起成 41 維度作為輸入，經過兩層 fully-connected(neurons_unit=41、41；activation_function='swish'；optimizer=Nadam；loss_function=categorical_crossentropy)後以 softmax 輸出。同樣以 dataset 中有經人工 verified 的資料作訓練，並記錄 validation accuracy。
- Stage2:
  - 將第一種及第二種 model 的輸出分別作為 label spreading[10]的輸入進行訓練。Predict 全部 9400 筆的

testing data。

- **Voting：**
  - 因此 task evaluation 方式為 MAP@3，可選擇三個 label 最為答案：
    1. 將 Phase4 stage2 的兩種 model 進行投票，要兩 model 皆說為同一 label，才對該 fname 的第一個 label 進行貼標。
    2. 將 Phase4 stage1 的 model 輸出，依照 validation acc 進行 weighted sum ensemble，作為該 fname 的第一或第二個 label。
    3. 將 Phase3 十個 model 共 100 個 fold 的輸出，依其 validation accuracy 進行 weighted sum，作為該 fname 的其餘二或第三個 label。
    4. 依此優先順序，若有 label 重複相同者，則往下遞移，最後進行輸出答案。

Experiment and Discussion

- **raw**
  - 原我們欲使用兩秒的 raw data 來透過 CNN1D 或 LSTM 方式來進行訓練，然或許因為我們硬體設備較差，記憶體不足而無法進行嘗試。若縮短至一秒，則表現效果過差，準確率未達 80%便作罷。
- **fbank**
  - 我們也曾使用 Fbank feature (melspectrogram[11])來進行嘗試，但在同樣前處理、padding 的情況下，以我們的 CNN model 來說表現效果不盡理想，且相比於 mfcc 更常有 overfitting 的情況發生。因準確率皆未達 80%便作罷，或許應將音檔切開為多份進行訓練才會有比較好的效果。
- **增加 n-fold 數量**
  - 最初使用 6-fold 來 train model，後來更改架構後改成用 10-fold 來 train model，validation accuracy 提升 5~10%。

        - 非常直觀，每個 fold 的 training data 變多了，model 理想來說都會比較好。
    - **Training data**
        - 最初使用全部 training data (unverified + verified data)進行 training。後來改成只用 3710 筆 verified data 進行 training，validation accuracy 有提升 1~2%。
        - 由於 kaggle 上說明那些 unverified data 是由舉辦單位的 model 來 predict 的，因此我們估計會有很多 label 是錯的，如果把這些錯的 data 加進來會影響 model 的 accuracy。
    - **Semi-supervised training**
        - 用 verified data 進行 training 得到的 model 來 predict 那些 unverified data，然後把 softmax 值超過一定 threshold 的 data 加進來進行 semi-supervised training，validation accuracy 會提升 1~5%。
    - **Data augmentation**
        - 利用 mixup algorithm[3][4][9]與 random eraser[3][4]，validation accuracy 提升 1~5%。
    - **Data normalization**
        - 我們僅在抽取 MFCC 時對資料進行 sample-wise 的 min-max normalization。
        - 我們也曾嘗試將 data 做 mean-std normalization，但 validation accuracy 反而會降低 1~2%。我們猜測此種 normalization 會破壞 MFCC 原有的性質。
- Conclusion
    - Self-train 與 Co-train 在 semi-supervised learning task 中相當實用。
    - 將聲音訊號轉為 image 的 task，搭配 image augmentation 的技巧可大幅提升準確率。
    - 在我們 feature 搭配 resnet，若搭配 normalization 反而效果不好。
    - 如何在一段音檔中擷取最重要的聲音片段出來，是相當重要的環節。

- fold 數增加，ensemble 的結果越好。
- fbank 或 raw data 搭配 resnet 在此 task 表現中理想度較差一點。
- 應可採用 CRNN model 進行訓練嘗試。
- 最後 voting 或許可改採幾何平均數的方式，應會比算術平均數有更好的結果。

Reference

- [1] Kaggle Kernel - Beginner's Guide to Audio Data https://www.kaggle.com/fizzbuzz/beginner-s-guide-to-audio-data
- [2] Keras Resnet https://github.com/raghakot/keras-resnet
- [3] Kaggle Kernel - Mixup & cutout or random erasing to augment https://www.kaggle.com/daisukelab/mixup-cutout-or-random-erasing-to-augment
- [4] Mixup Generator https://github.com/yu4u/mixup-generator
- [5] Python for Scientific Audio https://github.com/faroit/awesome-python-scientific-audio
- [6] Keras Audio Preprocessors https://github.com/keunwoochoi/kapre
- [7] Muda Document https://muda.readthedocs.io/en/latest/examples.html
- [8] Kaggle kernel – Stacking(XGBoost+LightGBM+CatBoost) https://www.kaggle.com/mainya/stacking-xgboost-lightgbm-catboost
- [9] Mixup: BEYOND EMPIRICAL RISK MINIMIZATION https://arxiv.org/pdf/1710.09412.pdf
- [10] Scikit-learn Label Spreading Document http://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelSpreading.html
- [11] Librosa Document https://librosa.github.io/librosa/generated/librosa.feature.melspectrogram.html