Project Title: Audio files classification of real-world sounds

Song Gao,* Jingyi Yang,† Zhirui Li,‡ and Project Supervisor: Prof. Hantao Zhang§
Department of Computer Science
University of Iowa
(Dated: May 12, 2020)

This document is the Final Report of the project work which was performed in Spring 2020 in fulfillment of CS:5980:0002. Project submitted on our website.

I. PROJECT OVERVIEW & DATA SET DETAILS & PROBLEM UNDER INVESTIGATION

The goal of this project is to use the wav audio file to train neural network for future classification. The web link for this dataset is `www.kaggle.com/c/freesound-audio-tagging`. The dataset contains 5.8GB .wav files, and those wav files belong to 41 categories, like "Cello", "Bus", and "Fart". Each wav file is very short, about 2 to 3 seconds length. Part of the files comes with labels which are already verified by human. While the remain part are labeled by machine. So, the data might contain some mal-verified label. In other words, the dataset contains some dirty data.

Problems under this dataset is: the existence of the dirty wav files, the data is in high dimensions form with 41 categories, and it is a large file taking 5.8GB on disk. At this point, we agree that traditional NN network (Artificial neural network) would not perform efficiently. We have to use CNN (Convolutional neural network) to extract information from the dataset. We decide to generate various data form, and train various models on each generated data. An overview between the relationship of generated data and the models we will address in our project is in the following table:

Data Type	Without filtering out the dirty .wav file	After Filtering out the dirty .wav file
1D raw audio file	Model 1 (CNN ensemble)	Model 4 (CNN ensemble) Model 5 (CNN-GRU ensemble)
2D MFCC 44100Hz or 30000Hz (results do not differ)	Model 2 (CNN ensemble) Model 3 (CNN-LSTM ensemble)	Model 6 (CNN ensemble)
2D data augmentation 44100Hz	NA	Model 7 (CNN ensemble) Model 8 (CNN ensemble)
2D data augmentation 25000Hz	NA	Model 9 (CNN ensemble)

For all of the models, we use 10 folds of cross validation for validating accuracy and ensembles. We start with 1D raw audio data. We used a library called "Librosa" to convert wav files into times series, then we use Keras with Tensorflow-gpu as backend to train and evaluate our models. Next in 2D data, we use Librosa library to convert wav files to vectors. The name of this technique is called MFCC (Mel-Frequency Cepstral Coefficient), and it transforms the original data to log scale, and takes the max peak in the audio. Using this technique, we reduce the size of our dataset from 5.8GB to 1.7GB. Meanwhile, the performance of the prediction accuracy also improved. In 2D data augmentation, we use some features of audio to generate some new data, our methods includes stretching and compressing the sound wave, shifting the sound wave to left and right, and adding white noise. We also tried to add echoes, and Harmonic-percussive source separation, but it seems that such methods confuses the neuron network, so that the prediction accuracy achieved decreased. After all operations above, we discovered that the dirty wav files indeed hurt our model. The prediction performance was not better even when we generate huge 8GB 2D augmentation data.

We build new models to filter out dirty wav files. We train our new models recurrently, track the wav files that seem hard to predict during the validation, and then we move these files into a blacklist. We repeat this process three

* song-gao-1@uiowa.edu
† jingyi-yang@uiowa.edu
‡ zhirui-li@uiowa.edu
§ hantao-zhang@uiowa.edu

times, and then take the intersection of these three blacklists. After filtering the dirty data out, we repeat the 1D, 2D, and augmented models for purified dataset. This time, we are going to trim the silence from audio.

In the attached output, we put all the models Without filtering out dirty wav files into "part1.ipynb", and the remaining models are putted in "part2.ipynb".

II. SOFTWARE USAGE

Our project is performed on Jupyter notebook with python 3.6. We used the following libraries: numpy, os, shutil, time, seaborn, pandas, tqdm, sklearn, librosa, random, copy, scipy, tensorflow-gpu, tensorflow_addons, and imblearn.

III. PROJECT PROCEDURE AND ARCHITECTURE OF ALL MODELS

A: **Without filtering out dirty wav file**

1. 1D music data:
   Model 1 architecture:

Layer	Filter size	Kernel size	Activation function	Padding
Conv1D	16	9	relu	valid
Conv1D	16	9	relu	valid
MaxPool 1D	16			
Dropout	Rate=0.1			
Conv1D	32	3	relu	valid
Conv1D	32	3	relu	valid
MaxPool 1D	4			
Dropout	Rate=0.1			
Conv1D	32	3	relu	valid
Conv1D	32	3	relu	valid
MaxPool 1D	4			
Dropout	Rate=0.1			
Conv1D	256	3	relu	valid
Conv1D	256	3	relu	valid
GlobalMaxPool 1D				
Dropout	Rate=0.2			
Dense	Unit=64			
Dense	Unit=128			
Dense	Unit=41			

   The highest validation accuracy is 0.7244, Kaggle private test score is 0.63829. We then also tried to add or remove a layer, but the validation accuracy does not change much.

2. 2D music data: Model 2 architecture:

Layer	Filter size	Kernel size	Activation fun	Padding
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Dense	Units=64			
Dense	Units=41			

Validation Accuracy: $\sim$ 1.000; Kaggle private test score: 0.68116. Since the validation accuracy is close enough to 1.00 and Kaggle private test score is 0.68, we want to train alternate model to improve Kaggle private test score.
Model 3 architecture:

Layer	Filter size	Kernel size	Activation fun	Padding
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
GlobalMaxPool2D				
LSTM	Units=64			
BatchNormalization				
Dense	Units=64			
Dense	Units=41			

Validation Accuracy: 0.8933; Kaggle private test score: 0.58904. In model 3 we also tried to replace Conv2D with LSTM to see if we can improve the Kaggle private test score. The result gave worse private test score. So we realized it might not be a good idea to train an RNN layer for 2D data.

3. 2D data with data augmentation

   We have a failure in this setting, since we made some mistakes during the process of generate new data. Although our validation accuracy is 1.00, the accuracy for Kaggle submission is extremely low. We also found that if we are running float 32 on 8GB samples, we are likely to blow up our 32GB memory plus 6GB video memory. So, it would be better in the future runs to reduce the generated sample to 4GB.

Above all, We can see if we do not clean up our dataset, a real accuracy 0.68 is going to be the upper limit for our models.

B: **After filtering out the dirty wav files**

1. 1D music data:
   Model 4 architecture:

Layer	Filter size	Kernel size	Activation fun	Padding
Conv1D	32	9	relu	valid
Conv1D	32	9	relu	valid
MaxPool 1D	16			
Dropout	Rate=0.1			
Conv1D	32	3	relu	valid
Conv1D	32	3	relu	valid
MaxPool 1D	4			
Dropout	Rate=0.1			
Conv1D	32	3	relu	valid
Conv1D	32	3	relu	valid
MaxPool 1D	4			
Dropout	Rate=0.1			
Conv1D	256	3	relu	valid
Conv1D	256	3	relu	valid
GlobalMaxPool 1D				
Dropout	Rate=0.2			
Dense	Units=64			
Dense	Units=128			
Dense	Units=41			

   Validation accuracy: 0.5912; Kaggle private test score: 0.51634. Now we observe that the model performance became worse. We suspect that the decreasing of the accuracy was due to that the data changed. Therefore, we needed to fit a new model.

Model 5 architecture:

Layer	Filter size	Kernel size	Activation fun	Padding
Conv1D	16	3	relu	valid
Conv1D	32	3	relu	valid
MaxPool 1D	32			
GRU	Units=128	return_seq=True		
Dense	Units=128			
GlobalMaxPool 1D				
Dropout	Rate=0.2			
Dense	Units=64			
Dense	Units=41			

Validation accuracy: 0.7382; Kaggle private test score: 0.71234. Now we believe that the RNN is only useful for fitting 1D data. The result from RNN in 2D data does not differ much from the CNN models, since we are using MFCC for 2D data. MFCC probably removes time dependency. So, it would be meaningless to fit a 2D RNN model.

2. 2D music data
Model 6 architecture:

Layer	Filter size	Kernel size	Activation fun	Padding
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Dense	Units=64			
Dense	Units=41			

Validation Accuracy: ~ 1.000; Kaggle private test score: 0.74441. Due to the data cleaning and filtering process, the validation accuracy improved to 1.00 and the Kaggle private test score improved from 0.68 to 0.74.

3. 2D data with data augmentation
   Model 7 architecture:

Layer	Filter size	Kernel size	Activation fun	Padding
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Dense	Units=64			
Dense	Units=41			

Validation Accuracy: 0.8400; Under fit encountered, so we discarded this model. For each time, we add on new dense layer into the model, and see whether the validation scoring stopped to increase. We end up at model 8.
Model 8 using 44100 sampling rates, architecture:

Layer	Filter size	Kernel size	Activation fun	Padding
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	64	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	64	(2, 5)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	128	(1, 3)	relu	valid
BatchNormalization				
MaxPool 2D				
Dense	Units=256			
Dense	Units=128			
Dense	Units=64			
Dense	Units=41			
Weight Decay has been added to this model, rate=0.000011				

Validation Accuracy: 0.9717; Kaggle private test score: 0.76212.

Model 9 using 25000 sampling rates, architecture:

Layer	Filter size	Kernel size	Activation fun	Padding
Conv2D	32	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	64	(4, 10)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	64	(2, 5)	relu	valid
BatchNormalization				
MaxPool 2D				
Conv2D	128	(1, 3)	relu	valid
BatchNormalization				
MaxPool 2D				
Dense	Units=256			
Dense	Units=128			
Dense	Units=64			
Dense	Units=41			
Weight Decay has been added to this model, rate=0.000011				

Validation Accuracy: 0.9782; Kaggle private test score: 0.76583. We train the same model for different fineness of data, and this trick improves the ensemble model. The Kaggle private test score for ensemble model 8 and model 9 improves to 0.79134.

C: Final model decision:

We decided that we should choose the model after filtering out the dirty wav files. Also we decided to ensemble models 5, 8 and 9. We ignore model 6 and 7 since we found that model 6 and 7 are too similar to model 8 and 9, so they do not add much new information for the prediction. Also ensemble models 6 and 7 in possibly resulted in a lower Kaggle private score. The score for our final chosen model is 0.82589, better than any of the previous individual model runs.

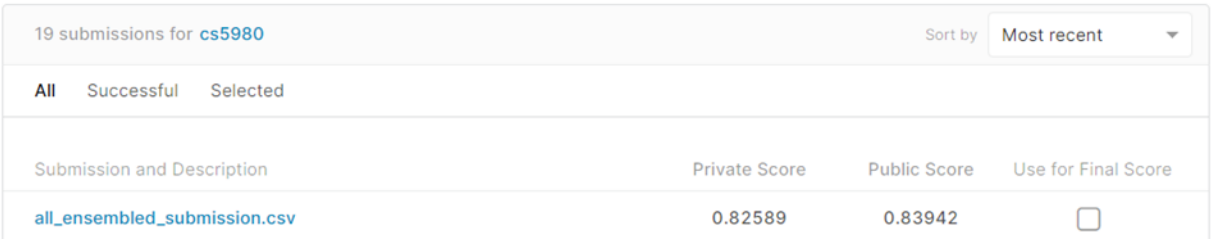

FIG. 1. Kaggle score screen shot

IV. OVERALL STATISTICAL ANALYSIS

We except the Kaggle private test score improves as our model becomes more complex and our data become cleaner and more augmented. We observed some positive feedback from data augmentation: a 5% improvement in Kaggle private test score. We are dissatisfied with the result of our filtering process used to clean up our data. It seems that there was only a 6% improvement, yet we had expected the cleaning would bring us an at least 10% improvement. We suspect we might still have some dirty data left in our data files not putted into the blacklist. The final ensemble

model brings us an accuracy of 0.82 in Kaggle private test score, comparing with the result of the first model we tried which was 0.63, we are happy with this ensemble-model result.

V. COMPARISON WITH RELATED SIMILAR WORK

Here we address our answer to the following questions: Has the same problem been accomplished by Deep Learning or not? How is our approach different from others?

Without deep learning, the traditional machine learning model might be extremely difficult to train. For example, in MNIST dataset, SVM and KNN takes 75 min to train with 10 folds of cross validation on a demo CPU train. MNIST dataset is about 209 MB on our disk, yet our audio data set is 5.8GB. As a direct comparison, it would take 27 hours to train a traditional network with our dataset. That is way too time-consuming. With GTX 980ti, our team is able to train the CNN for audio dataset in 1.8 hours. With RTX 2060, we can train the CNN in 6.6 mins. We could use Half precision on 2060. However, Half precision is too frigate. Our model loss some validation accuracy on it. We believe that indeed, the problem is attacked by Deep Learning.

The significance of our approach is using different methods to create various data form, and for each created data set, find best model with highest validation score. In the end, we use the best model from each created data set to create an ensemble model. We also use models to clean our dataset, an idea that is similar to and inspired by the idea of doing regression and locating the outliers.

VI. CONCLUSION

We trained many models for different generated data as well as tried different models. We observed a number of good practical tips. For example, we should add one layer for each fold to check whether the validation accuracy is improved. We also take advantage of many traditional tools in different fields that help with Deeping Learning. For instance, MFCC is the traditional tool in audio processing, and we benefited from this technique. Also, if we take different roughness of our data, we can use this difference to create some ensemble models. We realized the importance of data managing and as a result we need to carefully check whether our data is contaminated by previous generated data. We employed lots of mathematical theory and statistical analysis. In the end, our final ensemble achieves 0.82 in Kaggle private test score. We believe there is still a long way to go.

In the future, we could also train a 3D CNN model. We can see the audio data in 1D limits the performance of the model. Therefore, it is expected and it is indeed the case that the audio data with 2D models performed a lot better than the 1D models. We therefore expect it is worth to try some 3D models. Also, we could try some different approaches for filter out the dirty wav files, for example, trying to use some deep models as well as some sallow models as the filtering models.

Note that output files could all be found at our project website.

VII. TEAM CONTRIBUTIONS

Team leader: Song Gao

- Code writing and maintenance of "part2.ipynb" (Filtering out dirty files and perform network training)
- Section II & III of final report
- Statistical analysis of part 2. Kaggle private test.

Team member #2: Jingyi Yang

- Code writing and maintenance for network training part of "part1.ipynb" (All data, no blacklist)
- Section VI of final report
- Project web page master and maintenance

Team member #3: Zhirui Li

- Code writing and maintenance for data generating part of "part1.ipynb" (All data, no blacklist)
- Section I, IV & V of final report. Typesetting of report.
- Statistical analysis of part 1.

VIII. APPENDIX

We will not paste tons of pages of outputs here, rather, we recommend readers to go to our project website for .ipynb file containing full outputs.

IX. REFERENCES

[1] J Fan, Beginner's Guide to Audio Data, https://www.kaggle.com/fanhaobei/beginner-s-guide-to-audio-data-fan/notebook?from=groupmessage&isappinstalled=0, 2019
[2] Zou, Introduction to image convolution and filter, https://blog.csdn.net/zouxy09/article/details/49080029, 2019
[3] H Zhang, Deep learning using python notes, 2020.