SGN-24007 Advanced Audio Processing

Final Project Report

Student: Ton Duc An (293203), Quan Do (277650)

Topic: Instruments classification using CNN and Tensorflow Keras

1. Introduction

The main task of this project work is to classify 11 different instruments class based on the Nsynth instruments dataset by Tensorflow Magenta. As mentioned in the website, there are 11 different instruments classes collected at different qualities and sources. There are three main sources, which are acoustic, electronic and synthetic. The instrument families are bass, brass, flute, flute, guitar, keyboard, mallet, organ, reed, string, synth lead and vocal.

As can be seen in the dataset, the records in the instrument families are quite imbalanced. For instance, there are many bass records as different qualities and sources, but the flute records is not that much compare to the others.

Family	Acoustic	Electronic	Synthetic	Total
Bass	200	8,387	60,368	68,955
Brass	13,760	70	0	13,830
Flute	6,572	35	2,816	9,423
Guitar	13,343	16,805	5,275	35,423
Keyboard	8,508	42,645	3,838	54,991
Mallet	27,722	5,581	1,763	35,066
Organ	176	36,401	0	36,577
Reed	14,262	76	528	14,866
String	20,510	84	0	20,594
Synth Lead	0	0	5,501	5,501
Vocal	3,925	140	6,688	10,753
Total	108,978	110,224	86,777	305,979

Moreover, because of the project time and the computer resource, it would take several hours only running the model to pre-process the data and run the machine learning model. Therefore, the solution is to take only a train subset and train on it. After getting a good result, other methods to run on larger data will be achieved.

The train subset is taken by taking about 90 audio files of each instrument family, and the audio files are taken so that it can cover different sources. For example, if the bass has three sources which are acoustic, electronic and synthetic, hence, 30 audio files will be at the acoustic, 30 will be at the electronic and the last 30 will be the synthetic. Therefore, the data will be more balanced and the audio files would be sufficient enough for the training and validation split process.

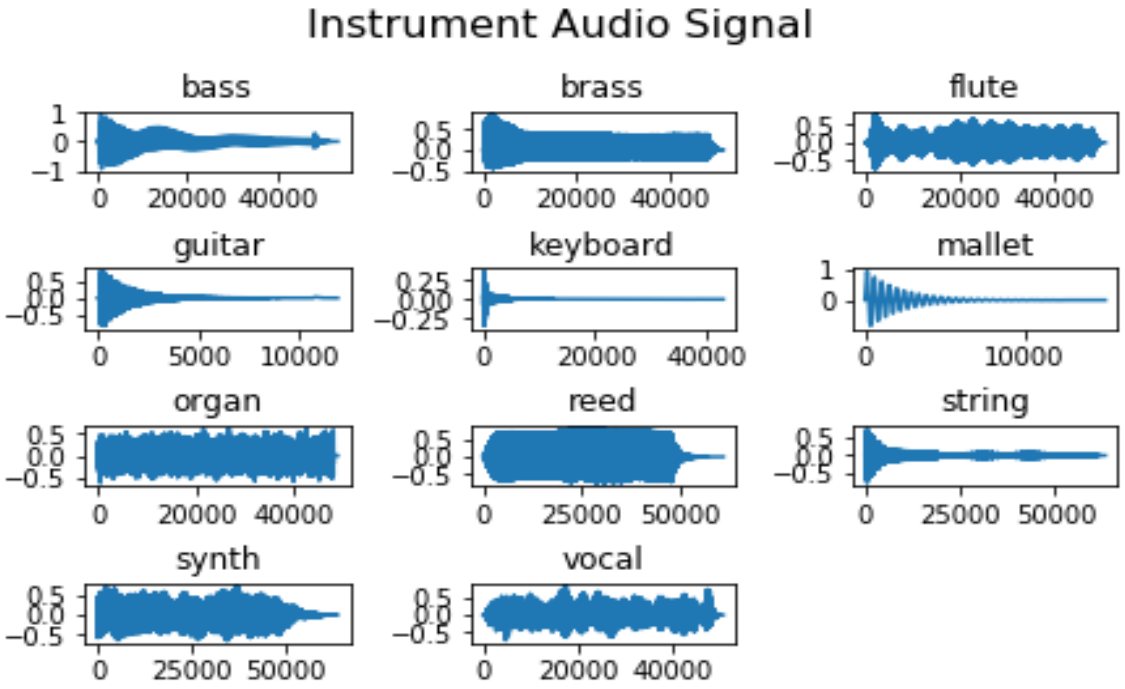
2. Approach method and solution

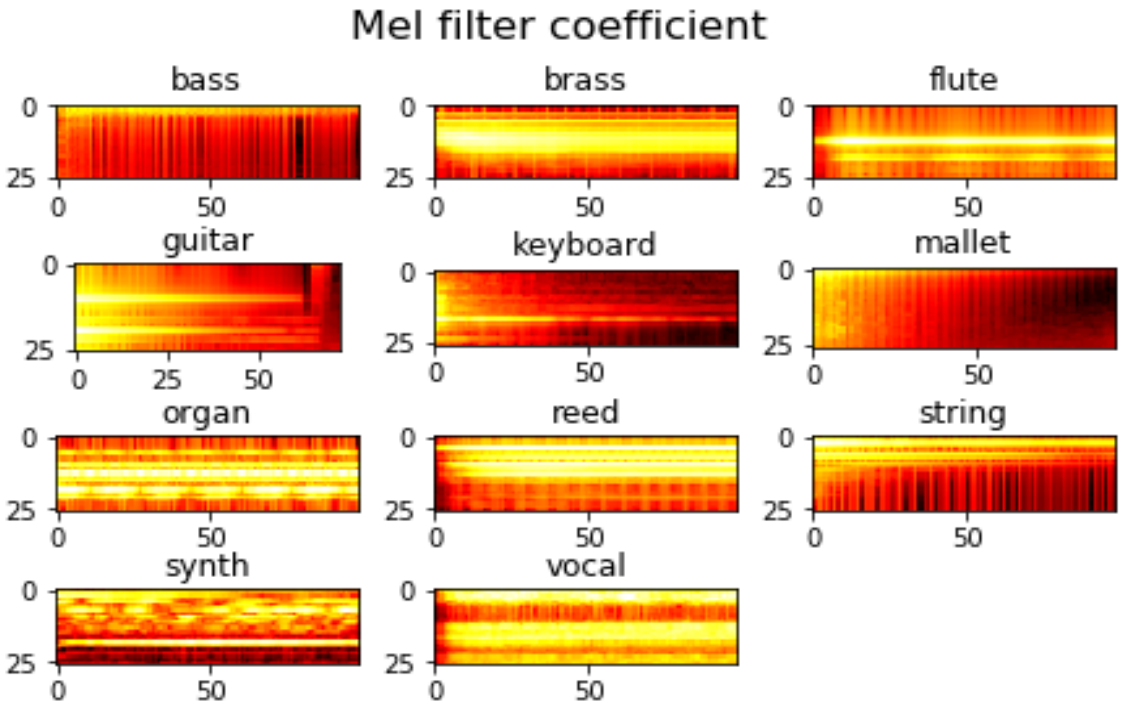
Because this classification task can be done in several ways, therefore, convolution neural network (CNN) is a good approach. For the classification task, CNN is a good model to build and approach the classification task. It basically contains convolution layers to take the features of the audio data, and pooling layers to reduce the size and take the most significant features.

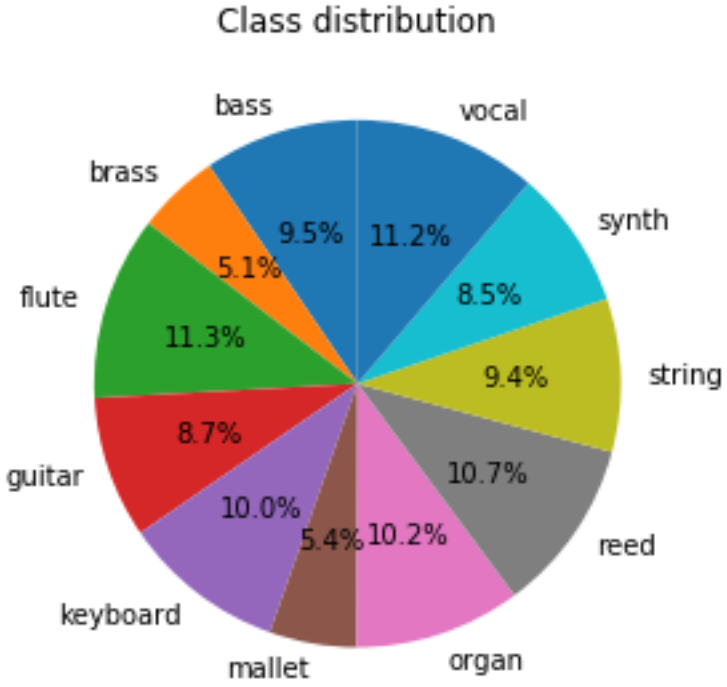
For the audio data, as referenced, the normal data needs to transfer into mel-scale and take the mfcc to get the good classification. Moreover, because the data is recorded in different qualities and sources, some pre-processing tasks need to be done as well.
For this task, Tensorflow Keras is used. Although PyTorch is used in the course implementation, however, because of the convenience in using Tensorflow, the convolution model will be done in this package.

3. Exploratory data analysis (EDA)

After getting the subset of the train data, take each audio file of each instrument families and analyze it to see the difference between them. As mentioned above, mel-scale are used to make the classification easier and it is easier to visualize, hence make it easier to tell the difference between the instruments.
Moreover, for the subset data, the length of each audio needs to be extracted and add together by instruments families to get the length percentage for later features building purpose.
Envelope audio signal is done to clear out the small values and the mute values at the beginning and the end of the audio data, making the classification easier.
The result after taking each audio of each instrument family when loading the file and change them into mfcc.







As can be seen in the figures, if considering only the audio file and its Fast Fourier Transform (FFT), it is difficult to tell the instrument apart. However, when taking the mfcc, the visualization of the instruments is good, and it is easy to tell them apart.

The class distribution figure tells the total length of each instrument in percentage. As can be seen, brass takes the smallest percentage compared to the others.

4. Pre-process and features building

The idea of this classification task is to clean the raw data and take the mfcc of it.
It is done by reading the raw training audio data, and then after the envelope process and re-write into another folder directory for later use.
Please refer to the eda_subset.py for the implementation of this task.
The feature is built by taking the tenth of a second in a random choice with the class probabilities mentioned above (the pie chart). Taking a random tenth of a second of the random audio file class, then transfer it into mfcc and append to the input data.
After getting all data, it will be normalized and change the shape so that it would fit with the CNN model.
Please refer to the build_features function in model_subset or model.py for the programming part of the features building.

5. The machine learning model

As mentioned in the solution part above, convolution neural network is considered in this task. The biggest advantage of using CNN model is that the model is easy implemented, and the members already had experiences using CNN model before. Moreover, using LSTM machine learning model would be implemented later after the project period is over, because of the time period of this project work, only one model is considered.
The model consists of five convolutional layers, which have different neurons, but have the same kernel size of 3x3, and the same with strides and padding. After these layers, there is one max pooling layer of 2x2 size, to reduce the size and parameters. A dropout layer is added in this model because based on experienced, it would avoid the overfitting issue and make the model run faster by turn off random neurons during the training process.
After the convolution block, three fully connected layer are added with the final activation function is the softmax function, as the task is multi-class classification.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 9, 13, 32)	320
conv2d_1 (Conv2D)	(None, 9, 13, 64)	18496
conv2d_2 (Conv2D)	(None, 9, 13, 128)	73856
conv2d_3 (Conv2D)	(None, 9, 13, 64)	73792
conv2d_4 (Conv2D)	(None, 9, 13, 32)	18464
max_pooling2d (MaxPooling2D)	(None, 4, 6, 32)	0
dropout (Dropout)	(None, 4, 6, 32)	0
flatten (Flatten)	(None, 768)	0
dense (Dense)	(None, 1028)	790532
dense_1 (Dense)	(None, 514)	528906
dense_2 (Dense)	(None, 11)	5665

Total params: 1,510,031
Trainable params: 1,510,031
Non-trainable params: 0

When fitting the model, a 0.2 percent of validation data is done because of the small size of the subset data. Moreover, there are two callbacks, which are early stopping, and the save the best model parameters.

6. Training and validation process

```
Train on 43889 samples, validate on 10973 samples
Epoch 1/10
43889/43889 [==============================] - 92s 2ms/sample - loss: 0.6679 - accuracy: 0.8374 - val_loss: 0.3282 -
val_accuracy: 0.9464
Epoch 2/10
43889/43889 [==============================] - 90s 2ms/sample - loss: 0.3311 - accuracy: 0.9411 - val_loss: 0.2705 -
val_accuracy: 0.9544
Epoch 3/10
43889/43889 [==============================] - 91s 2ms/sample - loss: 0.2671 - accuracy: 0.9561 - val_loss: 0.2041 -
val_accuracy: 0.9732
Epoch 4/10
43889/43889 [==============================] - 90s 2ms/sample - loss: 0.2264 - accuracy: 0.9629 - val_loss: 0.1759 -
val_accuracy: 0.9795
Epoch 5/10
43889/43889 [==============================] - 91s 2ms/sample - loss: 0.1985 - accuracy: 0.9677 - val_loss: 0.1654 -
val_accuracy: 0.9766
Epoch 6/10
43889/43889 [==============================] - 89s 2ms/sample - loss: 0.1772 - accuracy: 0.9712 - val_loss: 0.1663 -
val_accuracy: 0.9758
```

As can be seen when running the model with 0.2 validation split in the Python file model_subset.py, after 6 epochs, the early stopping is called because the validation loss of epoch 6 is higher than epoch 5 (patience = 1). Therefore, the training process stops. After 6 epochs, the training accuracy is 0.97, and the validation accuracy is 0.9758, which is good for a small subset with a 0.2 validation split. However, because the validation split would split 20 percentage of the

training data to validate each epoch. Therefore, the validation data is quite similar with the training data, hence it would make the validation accuracy as high as the training data.

Hence, another way to test the model is to run with unseen validation data. The Nsynth dataset also provides with a separate folder of validation and test data for a good validation record of the model. However, because of the small size of the subset, the model cannot give out a good validation result because of the small training data size. Therefore, the model is trained again, and it would run with the whole training folder and the separate validation folder to test the result.

```
217928/217928 [==============================] - 280s 1ms/sample - loss: 0.9844 - accuracy: 0.6844 - val_loss: 0.9075 -
val_accuracy: 0.7121
Epoch 27/32
217928/217928 [==============================] - 276s 1ms/sample - loss: 0.9775 - accuracy: 0.6863 - val_loss: 0.8916 -
val_accuracy: 0.7184
Epoch 28/32
217928/217928 [==============================] - 280s 1ms/sample - loss: 0.9713 - accuracy: 0.6874 - val_loss: 0.8942 -
val_accuracy: 0.7161
Epoch 29/32
217888/217928 [==============================>.] - ETA: 0s - loss: 0.9669 - accuracy: 0.6901
Epoch 00029: ReduceLROnPlateau reducing learning rate to 9.11250062927138e-05.
217928/217928 [==============================] - 277s 1ms/sample - loss: 0.9669 - accuracy: 0.6901 - val_loss: 0.8933 -
val_accuracy: 0.7161
Epoch 30/32
217928/217928 [==============================] - 277s 1ms/sample - loss: 0.9388 - accuracy: 0.6980 - val_loss: 0.8617 -
val_accuracy: 0.7276
Epoch 31/32
217928/217928 [==============================] - 277s 1ms/sample - loss: 0.9294 - accuracy: 0.7007 - val_loss: 0.8579 -
val_accuracy: 0.7286
Epoch 32/32
217928/217928 [==============================] - 277s 1ms/sample - loss: 0.9243 - accuracy: 0.7019 - val_loss: 0.8558 -
val_accuracy: 0.7289
```

After getting all the train data in the Nsynth dataset and training it with 32 epochs, the validation accuracy is 0.7289.

After that, get the accuracy score with the unseen test data.

```
100%|██████████| 100/100 [03:04<00:00,  1.85s/it]
The accuracy score for the task is: 0.5918367346938775
```

Another csv file is extracted to compare the true class and the predicted class of the test data (please refer to the test_df.csv file).

7. Programming

In the submission, there are three files which are eda_subset.py, model_subset.py and model.py.

The eda_subet.py is the file that uses for EDA process, to see the mfcc images of the instrument families. Moreover, it is used to extract the clean audio data (after envelope process) into a new directory called “clean_train”. This process is used for train and validation audio file, because the test data cleaning will be done in the model_subset and model Python files.

The model_subset.py and model.py are quite similar, except for the the model.py is more functional built, which means the program can be seen easier in functions, and the model.py is used to test for a separate validation folder and test folder as well, while model_subset.py stops at the training and validation process by using the validation_split parameter in model.fit. However, because of the computer resources, the training process cannot be done with a whole dataset, and random choice in the build_feature function would make the process better

because it chooses the audio file and the index randomly, which makes the training dataset not biased.