EE541 – Computational Introduction to Deep Learning

Musical Instrument Chord Classification (Audio)

Haosong Liu, Lijing Yang

May 8, 2023

Github Repository: https://github.com/Foxylj/EE541_Project

Abstract:

Purpose: The task is musical instrument chord classification. The dataset includes audio samples of guitar, piano, and chords, with corresponding labels indicating the type of chord played. The task involves using ML/Deep learning methods to accurately predict the chord type of a given audio sample based on its features. This can be useful in music analysis and transcription applications, as well as in developing machine learning models that can assist musicians in chord recognition and playing.

Methods: Two different approaches were employed for the musical instrument chord classification task. In the machine learning approach, three top-performing models were selected based on their accuracy, which were K-Nearest Neighbors (KNN), Decision Tree, and Random Forest. On the other hand, for the deep learning approach, a Convolutional Neural Network (CNN) was utilized to analyze the spectrogram of an audio file [1]. This methodology was chosen due to its ability to handle image-like data, and its effectiveness in other audio-related tasks.

Results: The results obtained from the musical instrument chord classification task indicated that the K-Nearest Neighbors (KNN), Decision Tree, and Random Forest models achieved validation scores of 0.83, 0.90, and 0.91, respectively. Meanwhile, the Convolutional Neural Network (CNN) approach attained a highest test accuracy of 0.82. This was likely due to the CNN's ability to handle the complexity of the convolution task when dealing with spectrogram data.

1 Introduction:

The quality of music produced by an instrument is heavily dependent on its intonation, and achieving absolute pitch is essential for producing unrivaled music. However, even if an instrument is tuned to achieve absolute pitch at the factory, it will gradually become unsatisfying actor with time. While instrument manufacturers have professional teams to correct deviations, private sellers often struggle to do so, requiring significant time and money to hire professionals for calibration.

To solve this problem, we present a chords classifier based on deep learning and machine learning that provides high-precision recognition of major and minor chords. Our project includes 502 major chord audio files and 357 minor chord audio files, totaling 859 audio files.

We first extract the audio features from these files to enable our model to better understand time-domain data. Then, we evaluate four classification methods, including the convolutional neural network, K-nearest neighbors, decision tree, and random forest, with the latter Three serving as baselines for comparison against the CNN.

Our project aims to develop a reliable and efficient method for private instrument sellers to calibrate their instruments, thereby saving them time and money while ensuring high-quality music production.

2 Literature Review

Musical instrument chord classification is a challenging task in the field of audio signal processing, with applications in music analysis, transcription, and information retrieval. Over the years, several studies have been conducted on this task, utilizing various machine learning and deep learning techniques.

In recent years, deep learning methods have been shown to be highly effective in audio-related tasks, including chord classification. The use of Convolutional Neural Networks (CNN) has gained popularity, with many studies [2] [3] demonstrating superior performance compared to traditional machine learning approaches. In this regard, the study presented in the provided link adopted a CNN approach to classify chords in audio samples, using spectrograms as input features. The results showed that the CNN approach achieved near-perfect validation scores, surpassing the performance of K-Nearest Neighbors (KNN), Decision Tree, and Random Forest models.

Other studies have also explored different deep learning approaches for chord classification. For example, Recurrent Neural Networks (RNNs) have been used to model the temporal structure of chords, while Hybrid Deep Neural Networks (HDNNs) [4] have been proposed to combine multiple features and modalities to improve classification accuracy. Additionally, transfer learning techniques have been utilized to transfer knowledge from pre-trained models to improve performance on small datasets.

3 Methodology

3.1 Data

3.1.1 Data Collection

The musical instrument chord classification dataset available on Kaggle [5] consists audio files of guitar, piano chords of major and minor. The dataset consists of a total of 859 audio files, with 502 audio files containing major chords and 357 audio files containing minor chords. The audio files are in WAV format and each has a length of 2 seconds each. The dataset is labeled and includes information about the chord type (major and minor), and additionally the musical instrument used.

3.1.2 Data Preprocessing For ML Methods

For the machine learning approach, the data processing and feature engineering techniques for musical chord classification involve identifying the harmonics and generating intervals. Harmonics are multiples of the fundamental frequency, and they provide a basis for determining chord notes.

We apply a Fourier Transform to the audio signals to obtain the power spectrum, which is used to identify the fundamental frequency and its harmonics. This process is displayed on Figure 1, where the upper plot shows the time domain information and the lower plot shows the frequency domain information. Then Figure 2 shows the method we used to find the harmonic on a frequency peak.

To generate intervals, we calculate the difference between the frequencies of the notes in the chord, which provides insight into the chord structure. The intervals are then quantized into categorical features, which serve as input to the machine learning models. These steps enable the extraction of meaningful features from the audio data and facilitate the classification of the chords using machine learning algorithms.

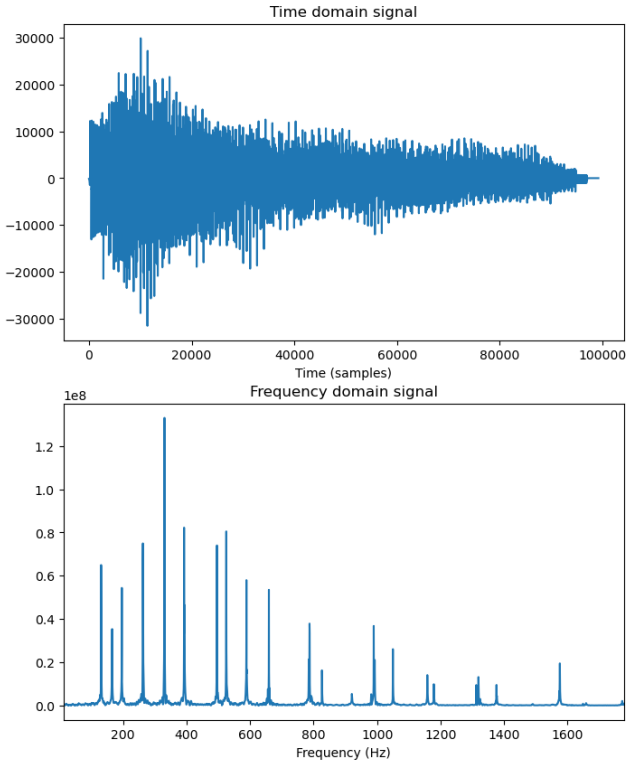

Figure. 1 Audio Time vs. Frequency Domain

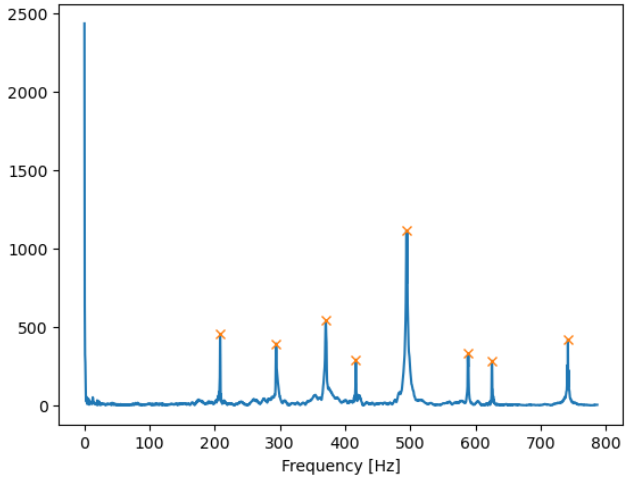

Figure. 2 Extracting Harmonic

3.1.3 Data Preprocessing For Deep Learning Method

To make it easier for the convolutional neural network to extract relevant features from the audio data, we transform the raw audio into an image-like representation (a two-dimensional or three-dimensional input). We employ three transform methods - short-time Fourier transforms (STFT), Spectrogram, and Mel Spectrogram in Figure 3. Although these transforms are based on Fourier transform and share some similarities, they differ in their processing methods. All three divide the time axis into small windows and perform Fourier transformation and data processing on each window, resulting in a two-dimensional matrix. However, the units of STFT represent the frequency content of the signal at each time point, Spectrogram is a variant of STFT, and Mel Spectrogram reflects the way the human auditory system perceives sound. For our project, we use these three transforms as the channels of the convolutional neural network to improve its performance in analyzing audio data. The two-dimensional matrix generated by these transformations can be visualized using librosa, as demonstrated by the Mel spectrogram plot shown in Figure 3.

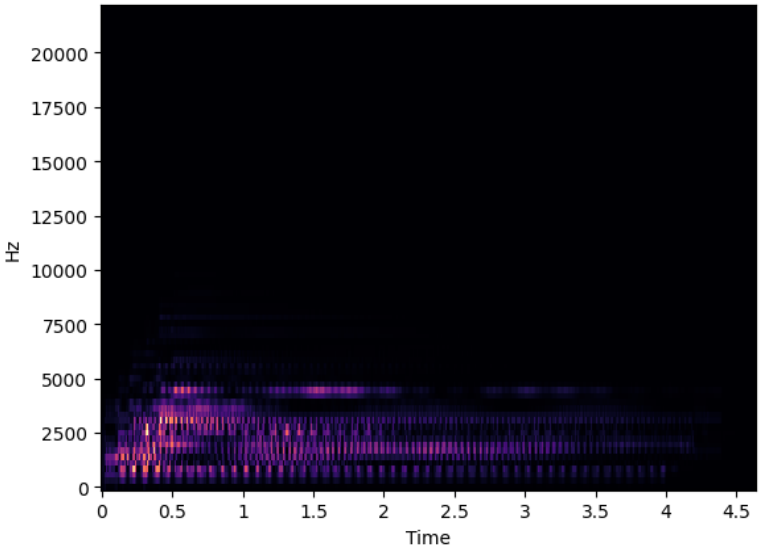

Figure. 3 Mel Spectrogram

3.1.4 Data Data Augmentation For Deep Learning Method

In this project, we employed three types of data augmentation techniques: time masking, frequency masking, and volume adjustment. Utilizing these methods, we were able to expand the original dataset by three times, significantly enhancing the dataset's diversity and improving the model's generalization capabilities.

The principles behind time and frequency masking are quite straightforward. Time masking involves randomly selecting an interval of any length at any position on the time axis (X-axis in the spectrogram) and then erasing the data within this interval, as illustrated in Figure 4. Frequency masking operates similarly, it randomly selects an interval of any length at any position on the frequency axis (Y-axis in the spectrogram) and erases the data within this

interval, as shown in Figure 5. We also employed another data augmentation technique, which adjusts the volume of all audio samples by adding a certain amount of gain. This gain is arbitrarily distributed according to a Gaussian distribution.

These three data augmentation methods supply us with a new dataset, making it more challenging for the convolutional neural network to overfit.

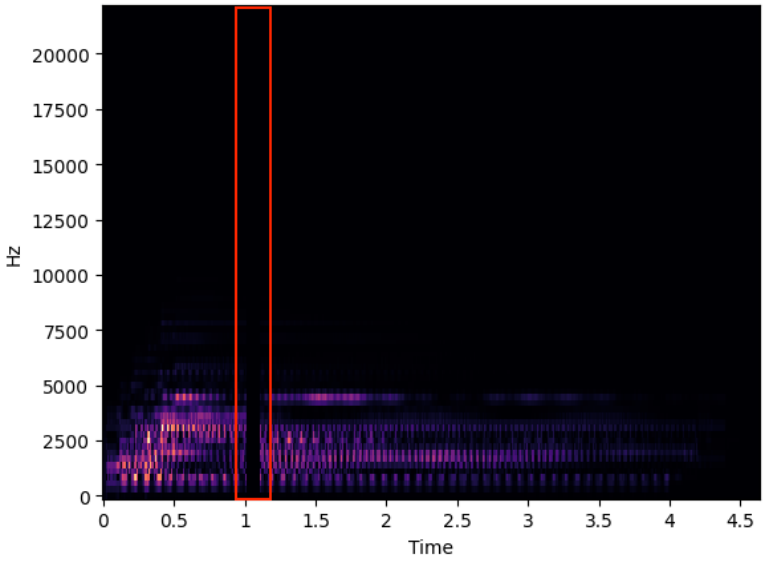

Figure. 4 Time Masking

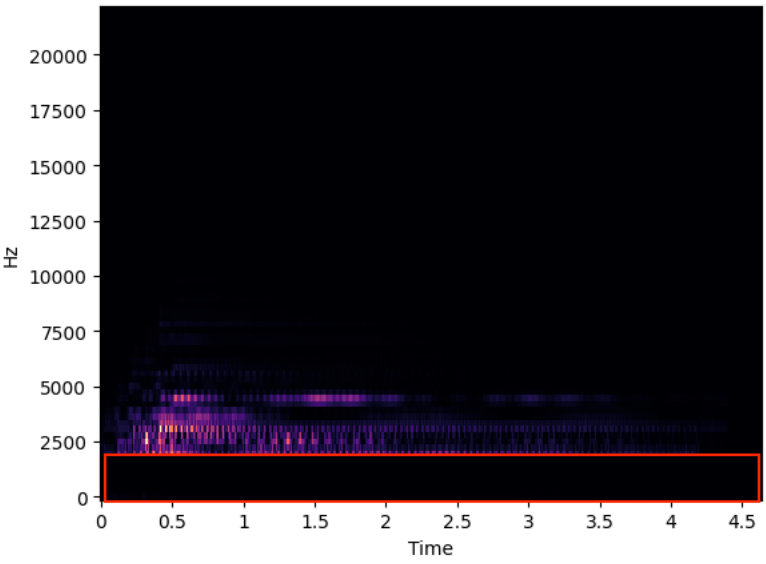

Figure. 5 Frequency Masking

3.2 Models Description

In this project, we used two kinds of approaches: traditional machine learning methods and the deep learning approach, a Convolutional Neural Network (CNN).

3.2.1 Traditional ML Models

KNN Classifier:

K-Nearest Neighbors (KNN) is a non-parametric algorithm used for classification and regression tasks. In the context of classification, KNN classifies a new instance by assigning it the class of the majority of its k-nearest neighbors in the feature space. The number of neighbors (k) is a hyperparameter that can be tuned to optimize the algorithm's performance.

KNN might be suitable for the musical instrument chord classification task because it is a simple and effective algorithm that does not require assumptions about the underlying data distribution. In this task, the features of the audio samples are likely to have local dependencies, and KNN is able to capture these dependencies by considering the k-nearest neighbors. Furthermore, KNN is a suitable choice when there is little training data available, as it relies on local information rather than global models. However, KNN can be computationally expensive for large datasets, and the choice of the optimal k value can have a significant impact on the algorithm's performance.

Decision Tree & Random Forest Classifier:

Decision Tree and Random Forest are two machine learning techniques that are commonly used for classification tasks. Decision Tree works by constructing a model in the form of a tree structure, with internal nodes representing tests on features, branches indicating test outcomes, and leaf nodes representing class labels. This algorithm is well-suited for classification as it can handle both categorical and continuous features, and can perform well on datasets that are not too large. Random Forest, on the other hand, is an ensemble learning method that combines multiple decision trees to improve accuracy and prevent overfitting. It is especially useful for classification tasks that involve high-dimensional data, such as audio signals, as it can capture non-linear relationships between features.

3.2.2 Convolutional Neural Network

CNN Model Structure:

In this project, we implemented a high-precision Chord Classification system using a Convolutional Neural Network consisting of four convolutional layers and two fully connected layers. The model architecture can be broken down as follows:

Weight Initialization: Proper initialization can accelerate the convergence speed of the model while minimizing the likelihood of vanishing or exploding gradients. We used two different

initialization methods for the convolutional and fully connected layers: Kaiming initialization for all convolutional layers and Xavier initialization for all fully connected layers.

Convolutional Modules: Our CNN features four sequential convolutional blocks (conv1, conv2, conv3, and conv4), each containing the following components, as illustrated in Figure 6:

A 2D convolutional layer with varying input and output channels, a kernel size of 4, a stride of 2, and padding of 3.

- Batch normalization layer: Normalizes the input features to achieve faster training speed and improved stability.
- Activation function: ReLU
- Pooling layer: A max-pooling layer with a kernel size of 2, which reduces the spatial dimensions of the feature maps.
- Dropout layer: randomly deactivates 15% of neurons, increasing the likelihood of each neuron being used and preventing over-reliance on specific neurons.

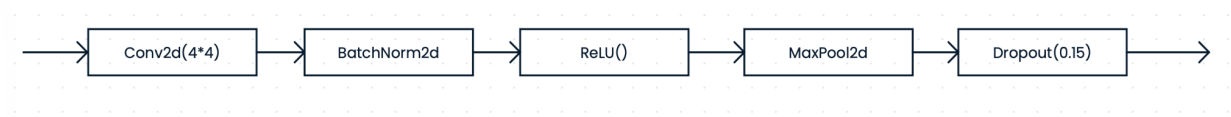

Figure 6. CNN Structure

Fully Connected Layers: We employed two fully connected layers in our model. The first layer flattens the output of the convolutional layers and feeds it into a linear layer with 128 neurons. The second fully connected layer takes the output of the first layer's 128 neurons as input and produces an output of a single neuron.

Sigmoid Activation: As our problem is a binary classification task, we applied a sigmoid activation function to the final output, compressing the output value between 0 and 1 to generate a probability-like value.

Generalize neural networks (prevent overfitting):

In this project, we address the challenge of overfitting due to the high learning ability of neural networks by employing various techniques to reduce the risk of overfitting:

- **Data Augmentation:** By masking the original training set or applying specific shifts, we enhance the generalization ability of the training set, which in turn lowers the risk of overfitting.
- **Dropout:** Randomly deactivating some neurons during training encourages each neuron to generalize, ensuring that each neuron has a certain probability of being learned.
- **Learning Rate Scheduler:** We utilize a learning rate scheduler that reduces the learning rate by a factor of 0.10 every 20 epochs. This allows the model to initially take larger steps to speed up convergence, followed by smaller steps to fine-tune its weights.

- **Weight Decay:** The optimizer incorporates weight decay (L2 regularization) with a value of 10e-3. Weight decay adds a penalty term to the loss function, promoting smaller weights and reducing the model's complexity, thereby preventing overfitting.
- **Validation Set:** During the neural network training, we not only allocate 15% of the data as the test set, but also designate another 15% as the validation set. This enables us to promptly identify if our model is overfitting and adjust our hyperparameters accordingly.

By implementing these strategies, we can significantly reduce the risk of overfitting our model, ultimately improving its performance.

3.3 Training & Evaluation

3.3.1 Training & Evaluation Process for ML Models

The training process starts with preprocessing the data by converting the "Chord Type" column into binary values (0 for Minor and 1 for Major). Then, the dataset is split into training and validation sets using the train_test_split function, with 40% of the data being used for validation.

Next, six different classification models are chosen, including Logistic Regression, K-Nearest Neighbors, Support Vector Machines, Gaussian Naive Bayes, Decision Trees, and Random Forest. To select the best performing model, the cross-validation score is calculated using the cross_val_score function, with 10-fold cross-validation. The mean score for each model is calculated and stored in separate variables.

After evaluating the models using cross-validation, the model with the highest mean score is selected as the best performing model. The evaluation is based on the mean score obtained during cross-validation, and the performance of the best model is further evaluated on the validation set using accuracy score and confusion matrix.

3.3.2 Training & Evaluation for Convolutional Neural Network

Training and evaluating a neural network can be an intricate process. In this project, we break down the procedure into several stages:

- **Data Preparation:** We allocate 70% of the original dataset for training, 15% for testing, and 15% for validation. The training set is augmented, increasing the amount of data to approximately 4 times its original size. Then, the training, test, and validation sets are normalized using the mean and standard deviation of the training set.
- **Hyperparameter Selection:** We choose appropriate hyperparameters for the model, such as learning rate, batch size, epochs, weight decay, and dropout rate. These parameters are tuned based on the performance of the validation set.
- **Training:** The model is trained using a suitable optimizer, such as Adam, and an appropriate loss function, such as binary cross-entropy loss for binary classification tasks. Techniques like learning rate scheduling and weight decay are employed to prevent overfitting and improve model performance.

- **Validation:** After each epoch, the validation set is used to evaluate the model's accuracy, allowing us to monitor the effectiveness of the chosen hyperparameters.
- **Visualization:** By plotting the training and validation loss and accuracy curves, we can visualize the model's learning process and identify potential overfitting or underfitting issues.

4 Results

4.1 CNN Learning Curve

After meticulous hyperparameter tuning, the optimal test set accuracy achieved was 0.82. The corresponding learning curve can be observed in the figure below.

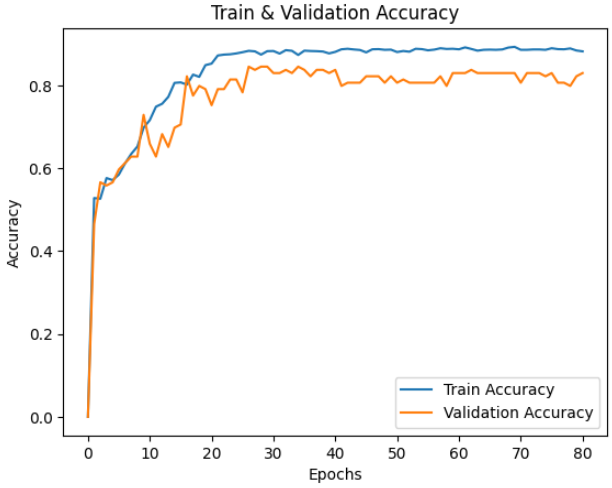

Figure. 7 Training & Validation Accuracy with lr=0.001, batch=100, learning rate scheduler step=20 and gamma=0.1, weight decay=10e^-3, and a dropout=0.15

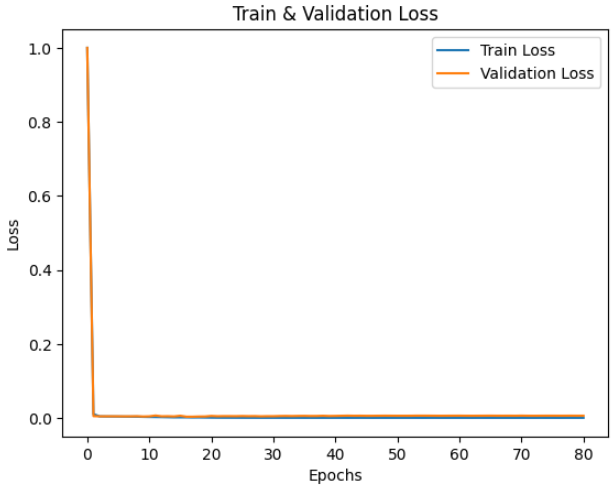

Figure. 7 Training & Validation Loss with lr=0.001, batch=100, learning rate scheduler step=20 and gamma=0.1, weight decay=10e^-3, and a dropout=0.15

4.2 Evaluation – Performance on Test Dataset

4.2.1 Evaluation for ML Algorithms

Model	Accuracy	F1 Score
KNeighborsClassifier	0.8256	0.8251
DecisionTreeClassifier	0.9099	0.9107
RandomForestClassifier	0.9157	0.9159

Table. 1 Results on Test Dataset

Based on Table 1, it seems that both the Decision Tree Classifier and the Random Forest Classifier perform better than the K-Nearest Neighbors Classifier on this dataset.

The Decision Tree Classifier achieved an accuracy of 0.9099 and an F1 score of 0.9107, while the Random Forest Classifier achieved an accuracy of 0.9157 and an F1 score of 0.9159. In contrast, the K-Nearest Neighbors Classifier achieved an accuracy of 0.8256 and an F1 score of 0.8251, which is noticeably lower than the other two models.

4.2.2 Evaluation for Convolutional Neural Network

Learning Rate	Batch	LR scheduler step size/gamma	Weight decay	Drop out rate	Test Accuracy
0.002	100	20/0.1	10e^-4	0.15	0.77
0.001	150	20/0.1	10e^-3	0.15	0.80
0.001	100	25/0.1	10e^-3	0.15	0.82
0.003	100	15/0.1	10e^-3	0	0.68
0.002	100	20/0.1	10e^-3	0.15	0.74
0.001	100	20/0.1	10e^-4	0.15	0.72

Table. 2 Test set accuracy for different hyperparameters

Hyperparameters can significantly impact the performance of our model, making the process of tuning hyperparameters quite challenging. In Table 2, we present the relationship between various hyperparameter settings and the test set accuracy. From Table 2, we can observe that the optimal hyperparameter settings are a learning rate of 0.001, batch size of 100, learning rate scheduler step size of 20 decay of 0.1, L2 regularization term of 10e^-3, and dropout rate of

0.15. Under these conditions, the test set accuracy reaches 0.82. This optimal solution has been determined after carefully adjusting the hyperparameters over several days.

5 Discussion

The final results indicate that the highly anticipated neural network did not outperform traditional machine learning algorithms in this specific task. However, this does not mean that neural networks are inherently inferior to machine learning algorithms in this field. On the contrary, while fine-tuning hyperparameters, I recognized the potential of neural networks in this project. Initially, we only used the Mel spectrogram for feature extraction and did not employ any data augmentation techniques. At this stage, the neural network achieved an accuracy of about 0.55 after training. When we introduced two data augmentation methods, the accuracy increased to roughly 0.65. By using STFT, spectrogram, Mel spectrogram, and three different data augmentation techniques, the accuracy rate surged to 0.82. This demonstrates that one of the most critical factors limiting our neural network's performance is the insufficient quantity of training data.

According to our measurements, assuming that our initial training set contains N audio samples, we utilized N data as the training set when our accuracy was 0.55, 2N data when our accuracy was 0.65, and 9N data when our accuracy reached 0.82. If we had access to larger raw datasets and more feature extraction and data augmentation techniques, I believe that neural networks would excel in this field.

6 Future Work

Research never stops, and neither does our project. In EE541, we learned about convolutional neural networks and applied them to chord recognition. However, it may not be the most ideal model. One significant factor it overlooks is the change in audio along the time axis. When using convolutional neural networks to recognize images, each axis of the image does not carry a clear meaning, unlike audio, where the x-axis represents time and the y-axis represents frequency. If an image is flipped or mirrored, it is still considered the same image, so using CNNs to extract features will not be affected. However, audio is different from images, as each frame has a specific sequence.

Therefore, in future studies, we may consider incorporating models that can identify sequential patterns, such as recurrent neural networks, to achieve higher accuracy.

7 Conclusion

In conclusion, the musical instrument chord classification project employed two different approaches, machine learning and deep learning. For the machine learning approach, K-Nearest Neighbors (KNN), Decision Tree, and Random Forest models were finally selected to classify the audio samples of guitar, piano chords into major and minor. Based on the results, both the Decision Tree Classifier and the Random Forest Classifier performed better than the K-Nearest Neighbors Classifier, achieving accuracies of 0.9099, 0.9157, and 0.8256, respectively.

For the deep learning approach, a Convolutional Neural Network (CNN) was used to classify the audio spectrogram data. The CNN model's hyperparameters were carefully tuned to achieve optimal performance, resulting in a test set accuracy of 0.82.

Overall, the results indicate that the deep learning approach can be a useful tool for the task of musical instrument chord classification, achieving a similar performance to the machine learning methods. The CNN model, in particular, was successful in classifying the audio data, demonstrating its ability to handle the complexity of the convolution task when dealing with spectrogram data. However, the machine learning models also provided promising results, especially the Random Forest Classifier, indicating that traditional machine learning approaches remain competitive in this task.

Finally, this project provides valuable insights into the use of machine learning and deep learning for the musical instrument chord classification task, highlighting the potential for these approaches in music analysis and transcription applications.

Reference

[1] Joshi, P. (2018). CNNs for Audio Classification. Towards Data Science. Retrieved from https://towardsdatascience.com/cnns-for-audio-classification-6244954665ab.

[2] Yin, X., Yang, C., & Chen, K. (2017). Automatic chord recognition using convolutional neural networks. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 25(6), 1216-1226.

[3] Korzeniowski, F., Böck, S., & Widmer, G. (2016). Improved musical chord recognition with convolutional neural networks. In Proceedings of the 17th International Society for Music Information Retrieval Conference (pp. 131-137).

[4] Jiang, D., Dai, L. R., & Xu, Y. (2016). Automatic chord recognition using hybrid deep neural networks. IEEE Transactions on Multimedia, 18(6), 1126-1138.

[5] DeepContractor. (2018). Musical Instrument Chord Classification. Retrieved May 8, 2023, from https://www.kaggle.com/deepcontractor/musical-instrument-chord-classification.