EE541 - Computational Introduction to Deep Learning

Musical Chord Classification Using Deep Learning

Project Group 26

Sanskar Tewatia, Naman Rajendra Joshi

May 8, 2023

ABSTRACT

The experimentation for this project was conducted for the classification of musical chords using audio recordings of the chords being played on a guitar and a piano. The project uses the dataset from Kaggle, which consists of two types of audio recordings from a piano and a guitar. The clips contain musical chords, which are either major chords or minor chords. This project aims to train machine learning and deep learning models that take these audio clips as inputs and classify them as major or minor chords. First, some analysis and background information about musical notes and chords was required to achieve this. Understanding notes, their frequencies, and how chords are composed of such notes was a requisite for this course. After this, the dataset was analyzed, the audio clips were sampled, and features were extracted. Two separate approaches were tried for this, and accordingly, machine learning models were trained on these numerically extracted frequencies from the audio clips. Finally, since this was a classification problem, we printed confusion matrices to see how the models performed and achieved over 80% accuracy on a section of the dataset set aside for testing.

INTRODUCTION/BACKGROUND

Definitions of some keywords in music theory.

• **Note**: the most fundamental building block of music. There are 7 natural notes, represented as do-re-mifa-sol-la-si or A-B-C-D-E-F-G, which can be played with white keys on a piano. In between these natural notes are five half-step notes (e.g., A# between A and B) except B-C and E-F. They can be played with black keys on a piano. There are 12 notes listed as follows, and each pair of adjacent notes are a half-step apart.

A A# B C C# D D# E F F# G G#

• **Frequency**: Each note has a corresponding frequency (Hz) that can be calculated as

$$f = 440 \times 2^{d/12}$$

where d is the number of half-steps away from the reference note A.

• **Chord**: A harmonic set of notes/frequencies heard as if sounding simultaneous. A major chord consists of a root note, a major third (4 half-steps higher than the root), and a perfect fifth (7 half-steps higher than the root). A minor chord consists of a root note, a minor third (3 half-steps higher than the root), and a perfect fifth. For example, if the root is C, the major chord is C-E-G, and the minor chord is C-D#-G.

• **2D CNN (Convolutional Neural Network):** A 2D CNN performs convolution operations on two-dimensional input data, such as images, to extract relevant features. The CNN architecture consists of multiple layers, including convolutional, pooling, and fully connected layers. In the convolutional layer, the network learns to detect local patterns and features from the input data by sliding a set of filters across the image, which performs element-wise multiplication and accumulation of the filter weights and the corresponding input pixels. The resulting feature maps capture the most relevant features of the image at that particular location.

- **Transfer Learning -** Transfer learning is a machine learning technique where a pre-trained model is used as a starting point for a new model trained on a different but related task. In other words, the knowledge learned by a model on a certain task can be used to perform better on another related task.

For example, if we have a pre-trained model on a large image dataset such as ImageNet, we can reuse the lower-level features learned by the model, such as edges, curves, and textures, for a new image classification task. Instead of training a new model from scratch, we can remove the pre-trained model's last layer(s), add a new classification layer, and fine-tune the whole network on our new task.

LITERATURE REVIEW

[1] This paper implements Automatic Chord Recognition using a deep convolutional neural network. They have extended the commonly used major/minor vocabulary (24 classes) to an extended chord vocabulary of seven chord types with 84 classes.

This paper implements a joint classification model to classify chord type and chord root pitch in isolated instrument recordings. At the same time, we intend to have a separate classification for the musical chords.

[2]This paper utilizes deep learning to learn high-level features for audio chord detection. It uses HMMs and SVM for chord classification. It can be a helpful starting point for our project, and we can extend it by trying out newer models and deploying the same.

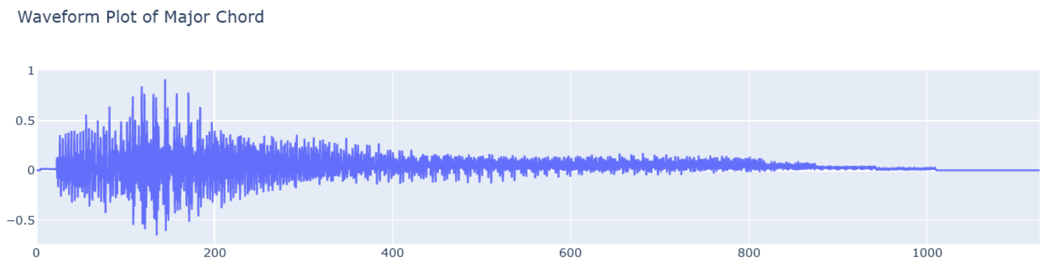
METHODOLOGY

DATASET

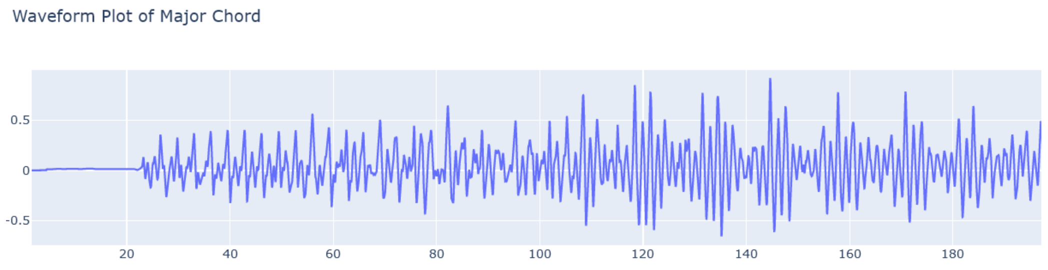
In this project, the dataset was gathered from Kaggle. The dataset consists of 859 musical chord recordings stored in 2 folders - A major folder containing 502 audio clips of major chords being played on the piano & guitar and a Minor folder containing 357 audio clips of minor chords. All these audio clips have a duration of approximately 2 seconds, but it is not the same for all clips. All audio clips are of ".wav" type, and we used Scipy.io's wavfile library to read all these clips, along with Librosa Library to gather information about the sampling rate, exact duration, and the number of channels for data exploration. All audio clips were sampled at 22Khz and had varying durations, so for processing, they had to be clipped to have the same duration. These audio clips were accordingly processed differently for the two approaches we experimented with.

AUDIO VISUALIZATION & FEATURE EXTRACTION

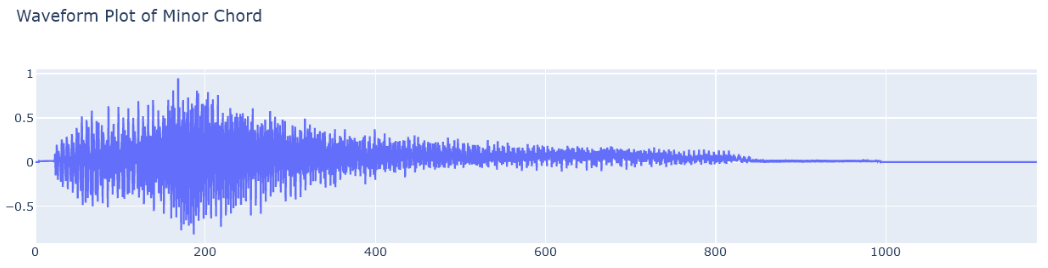
To visualize these audio clips, there are multiple options. We can analyze the waveform plot where the audio clip's scaled amplitude (in dB) is plotted on the Y axis, and time is chosen as the X axis. Such a plot shows how loud or soft the sound was in that particular audio clip and at what times.



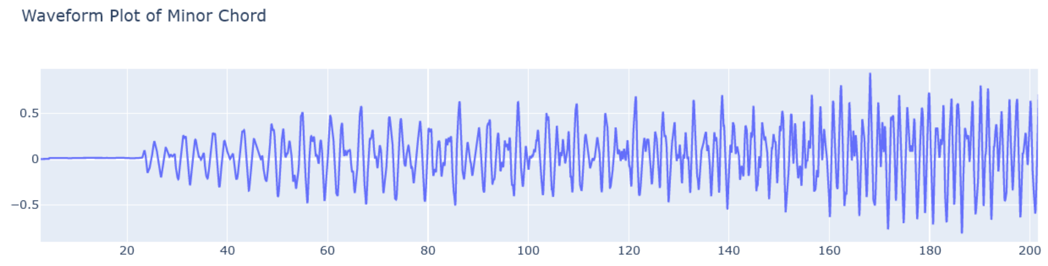
Waveform Plot of a sample audio clip from the Major Chord Directory



Zoomed-in plot of the first 200ms of the same audio clip(Major)



Waveform Plot of a sample audio clip from the Minor Chord Directory



Zoomed-in plot of the first 200ms of the same audio clip(Minor)

As we can see from the plots, even though these plots give information about the loudness of the audio clip, it is impossible to distinguish between a major and a minor chord purely from the waveform plot since loudness is not a measure of the underlying frequencies. Hence, it is not feasible to get insights into the characteristics of the underlying notes or their harmonics from this plot.

To achieve that, we can try a different approach and try to plot the loudness of the audio clip against frequency. Doing that would give a sense of how loud specific frequencies are, which frequencies or peaks of frequencies are mainly present in the audio clip, and which frequencies or their harmonics are most prevalent in the chord composition of that clip.

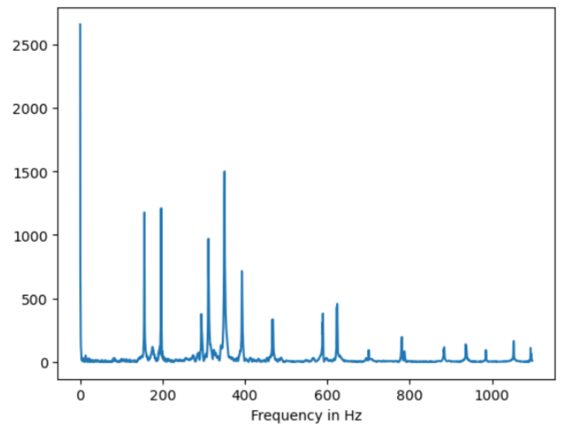
For that purpose, a Python function called find_harmonics2 was implemented that takes a path to an audio file as input and identifies the harmonics in the audio clip. The function uses the scipy library's wavfile module to read the audio file and determine the sample rate (fs) and the audio signal (X) as a one-dimensional array of amplitude values.

Next, the function applies the Fast Fourier Transform (fft) to convert the audio signal from the time domain to the frequency domain. It then calculates the one-sided Fourier transform of the signal and the corresponding frequency values using the np.abs and fftfreq functions. The function then selects the first half of the frequency domain and stores it as X_F_onesided.

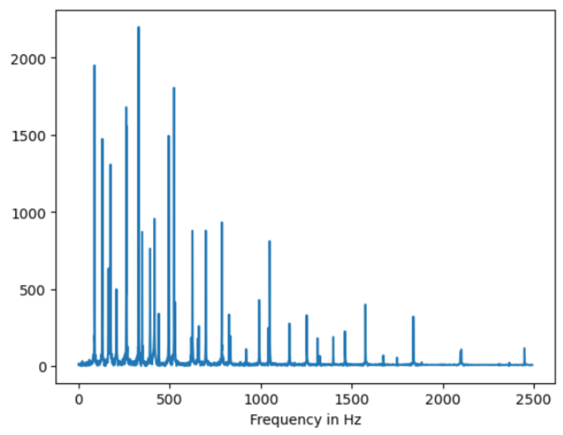
The function then identifies potential harmonics using the find_peaks function from the scipy library to find local maxima in X_F_onesided that are at least 5% of the maximum peak height and have a minimum distance of 10Hz between them. The function stores the indices of these peaks in the peaks array and their corresponding heights in the ht array.

The function rounds off the frequencies of the identified peaks and returns them in the harmonics array along with their corresponding heights stored in ht. If the optional argument print_peaks is set to True, the function also plots the audio signal's frequency spectrum using the matplotlib library and marks the identified peaks.

Overall, the find_harmonics2 function provides a helpful tool for identifying the harmonics in an audio signal, which can be valuable for tasks such as music transcription, sound analysis, and noise reduction. The function uses standard scientific computing libraries and techniques to transform the audio signal from the time domain to the frequency domain and then identify the local maxima in the frequency spectrum. Following is the output plot for a sample of major and minor chords.



Fourier Transform Frequency Plot for a Major Chord Audio Recording

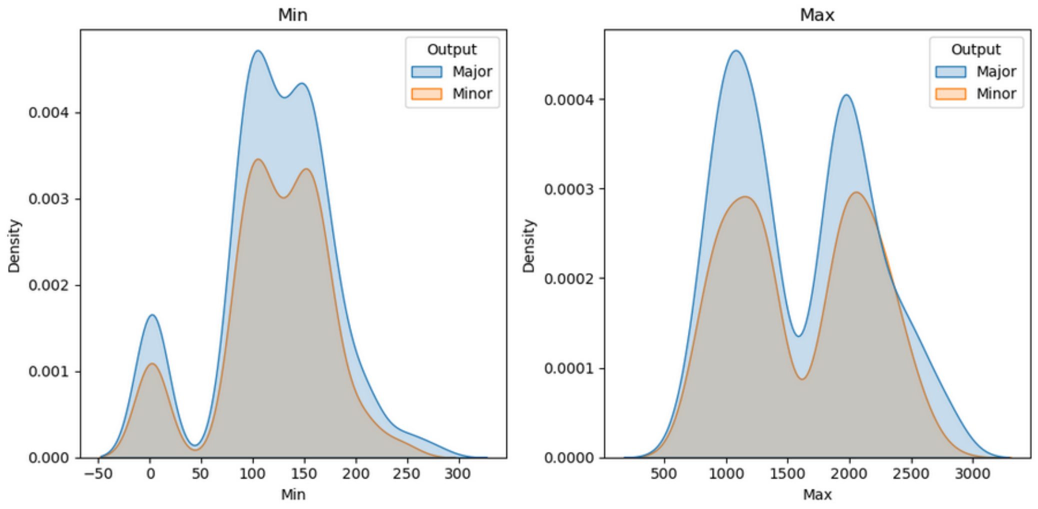


Fourier Transform Frequency Plot for a Minor Chord Audio Recording

This function is similar to homework 2, where the intensity was plotted against the wavenumber. In contrast, in this function, it was plotted against the frequency to view the peaks and store the coordinates so that we add them to arrays and save the peak height and the corresponding frequencies at which the peak was observed.

DATA PREPARATION FOR METHOD 1

Using the function from the previous section, we can actually get insights about various frequencies and their effects on the type of chord. Since we have 859 audio clips, and each of them is variable in length, and loudness after converting the number of harmonics returned by this function are also variable in number. Since we also returned the height of intensities of all these frequencies, we created a dictionary with the key being the height of the frequency and the value being the actual frequency in Hz. We then sorted this dictionary by keys and chose the values(i.e., the frequency) of the 8 largest harmonics since the 8 were the lowest harmonics returned by one of the audio clips. That way, we ended up with a list of 8 frequencies with the highest intensities in the Fourier transforms, sorted them in ascending order, and added them all to a data frame in 8 columns for that particular audio file. Apart from these 8 harmonics, the maximum and the minimum harmonic were also stored as two columns, along with the number of harmonics returned by the function. Following is a hue plot showing the relation between the minimum and the maximum harmonics with the output class.



Comparison of maximum and minimum harmonics against the output classes

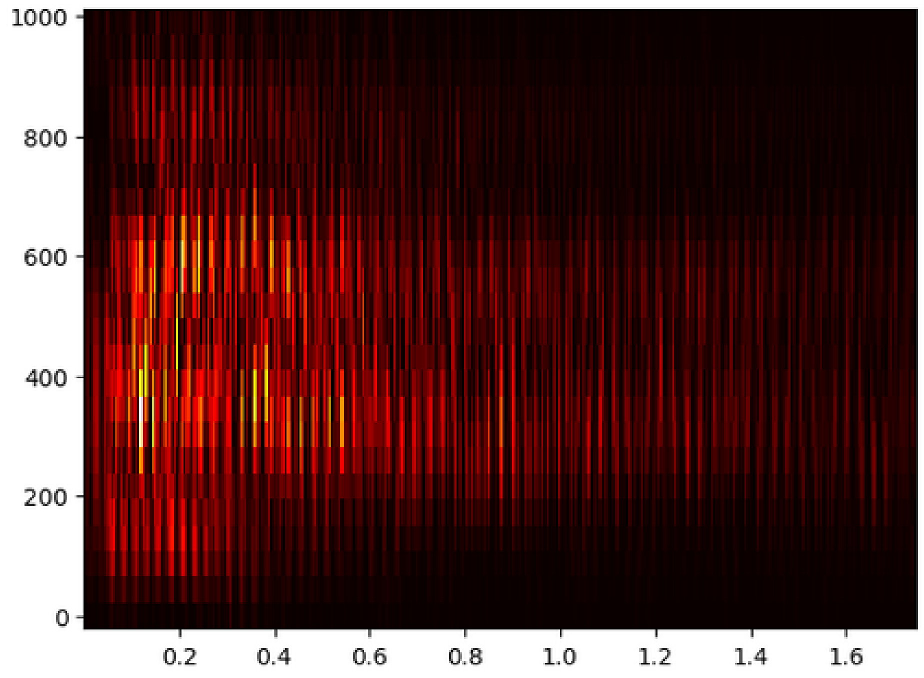
It is observed that the density of such observations was different and that these can be incorporated as predictors for the machine learning models. Hence, they are added as predictors for the corresponding audio clip in the data frame.

Also, since frequencies of various notes are of form 440*2^(d/12), so we decided to divide the harmonic (i+1) by harmonic (i) to get the value of that specific multiple of harmonics, and this way, 7 more columns of predictors are added to the DataFrame for each audio clip. After plotting hue plots, choosing just the first four intervals was decided since they were correlated to the output class. This data frame formed the input to the first machine-learning method.

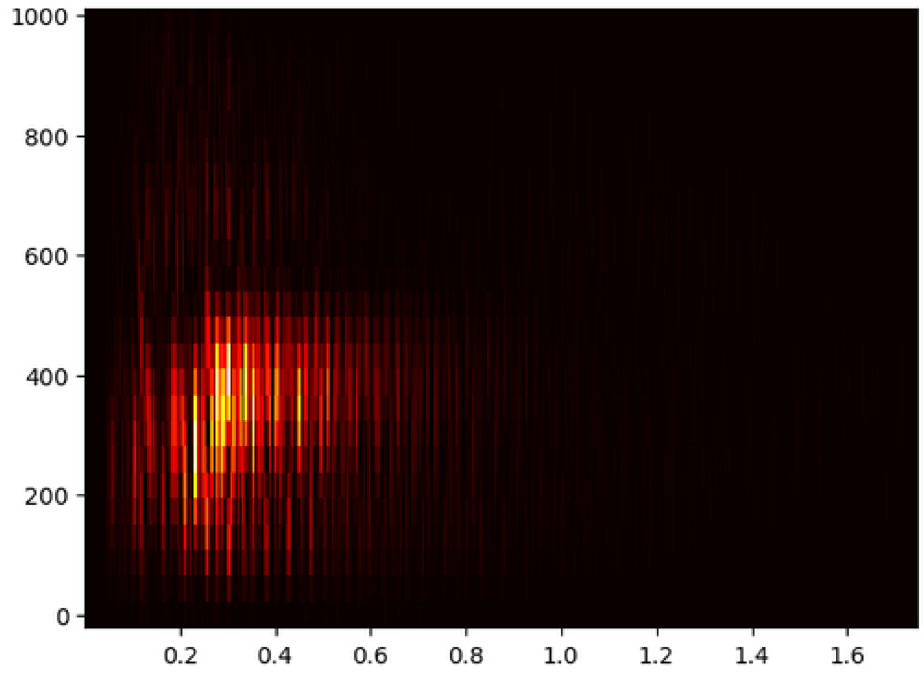
DATA PREPARATION FOR METHOD 2

For this method, a different approach was used. Instead of extracting and engineering components separately, we decided to use spectrograms. The code segment reads a wav file using the wavfile module from the scipy library. It defines a segment length of 1.75 seconds and extracts the first segment of the audio file to compute its spectrogram. The spectrogram is calculated using the spectrogram() function from the scipy.signal module, which returns the frequencies, times, and spectrogram data.

Then, the code plots the spectrogram using the matplotlib library by creating a figure with a specified size and using the pcolormesh() function to create a color mesh plot of the spectrogram data. The plot is limited to frequencies below 1000 Hz to remove high-frequency noise. The axis is turned off, and the plot is saved as a JPEG file into separate folders for minor and major chords.



Spectrogram Plots for a sample Major Chord Audio File



Spectrogram Plots for a sample Minor Chord Audio File

This code segment helps visualize the frequency content of an audio signal in a spectrogram format. A spectrogram is a time-varying representation of the frequencies in a signal, with time on the x-axis, frequency on the y-axis, and color representing the amplitude or power of the signal at each time and frequency. Spectrograms help analyze the frequency content of a signal, identify patterns and features, and detect changes over time. They are commonly used in speech recognition, music analysis, and acoustic signal processing.

Once these images corresponding to major and minor chords are saved in the appropriate folders, they can be given as inputs to a 2d CNN for classification.

MODELLING

BASELINE METHOD

Since the dataset consists of 502 major chords and 357 minor chords, if we predict all inputs as major chords, then for any train-test split, we will get an accuracy of 58%. So we will consider this our baseline system and compare other models to this.

METHOD 1

For this method, we will use the features engineered in the previous section, where a DataFrame was created for each audio recording. We have 14 predictors - {'Min', 'Max', 'Harmonic number 1', 'Harmonic number 2', 'Harmonic number 3', 'Harmonic number 4', 'Harmonic number 5', 'Harmonic number 6', 'Harmonic number 7', 'Harmonic number 8', 'Int 1', 'Int 2', 'Int 3', 'Int 4'}.

We train a neural network with the following definition -

```
Deep(

(layer1): Linear(in_features=14, out_features=128, bias=True)

(act1): ReLU()

(layer2): Linear(in_features=128, out_features=32, bias=True)

(act2): ReLU()

(output): Linear(in_features=32, out_features=1, bias=True)

(sigmoid): Sigmoid()

(dropout): Dropout(p=0.2, inplace=False)

)
```

For training and testing, we use an 85:15 split and use StratifiedKFold to get the average test accuracy from 2 runs. Following were the hyperparameters for training -

Table 1: Table listing all hyperparameter values

Optimiser	Adam
Loss	Binary Cross Entropy Loss
Learning Rate	0.0001
Epochs	200
Batch-Size	5

METHOD 2

To compare the performance of this deep learning-based approach with machine learning methods, we decided to train SKLearn's XGBoost Classifier on this same data. It is a compelling implementation of the gradient boosting algorithm, which works by building multiple decision trees sequentially, and each tree tries to correct the mistakes of the previous tree. Datasets with many features or datasets with missing values are suitable for this since XGBoost is one of the most robust machine learning methods and also incorporates regularization techniques to prevent overfitting. Due to this reason, this method performed the best out of all the methods we tried.

METHOD 3

In this method, we used the spectrogram images generated using the spectrogram() function from scipy.

Here, we used ResNet (Residual Network), a type of CNN (Convolutional Neural Network) that addresses the problem of vanishing gradients, which can occur during the training of very deep networks. The idea behind ResNet is to add skip connections, which allow the gradient to flow more easily through the network and help to prevent the gradients from getting too small. This allows for the training of much deeper networks than was previously possible. The main advantage of ResNet over traditional CNNs is its ability to train very deep networks. Traditional CNNs can suffer from the vanishing gradient problem, making it challenging to train deep networks.

For this method, the training, validation, and testing split was 75:15:10. For transfer learning, we unfreeze the fully connected layers of the ResNet just like we did in the last homework and then train them for 10 epochs using the following parameters and also save the loss and accuracy curves -

Table 2: Training Parameters for Fully Connected Layers of ResNet

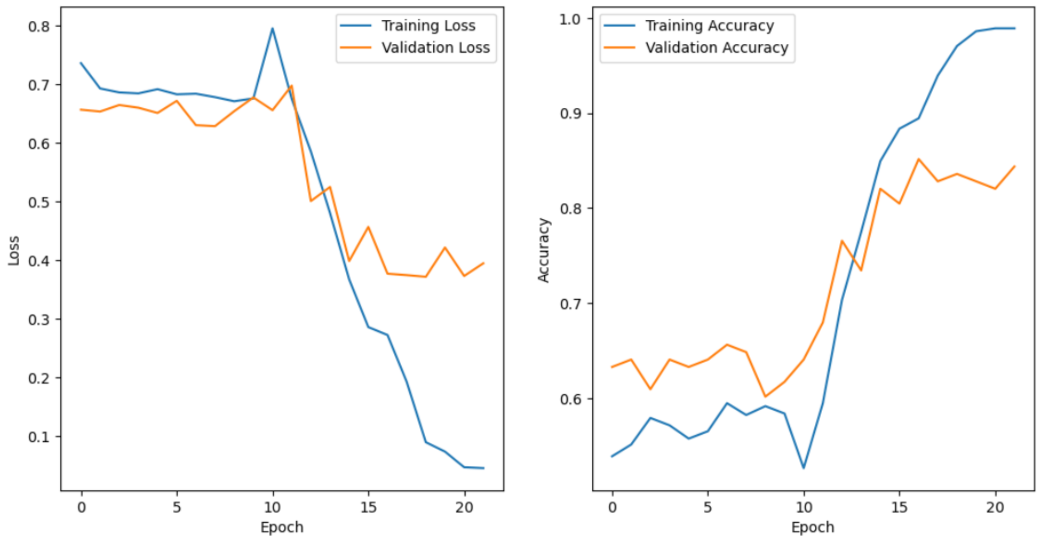
Optimiser	Adam
Loss	Cross Entropy Loss
Learning Rate	0.0001
Weight_Decay	0.0001
Epochs	8
Batch-Size	4

Next, we unfreeze Layer 3 and Layer 5, train for another 12 epochs with the following parameters, and append the loss and accuracy to the lists to plot the learning curves.

Table 3: Training Parameters for Layer3, Layer4, and Fully Connected Layers of ResNet

Optimiser	Adam
Loss	Cross Entropy Loss
Learning Rate Scheduler	0.0002, Step Size - 8, Gamma - 0.1
Weight_Decay	0.0001
Epochs	12
Batch-Size	4

Following are the plots for the learning curves -



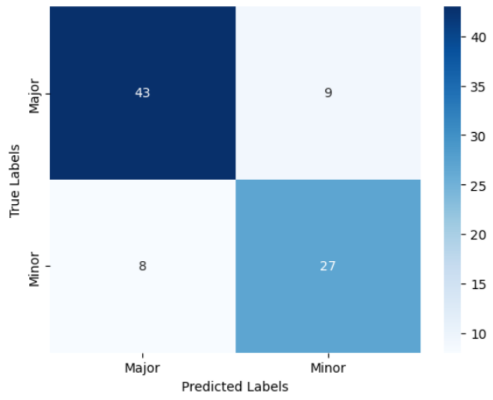
Learning Curves for ResNet Transfer Learning

RESULTS

The baseline model that predicts all inputs as major chords give an accuracy of 58.44% for any train-test split.

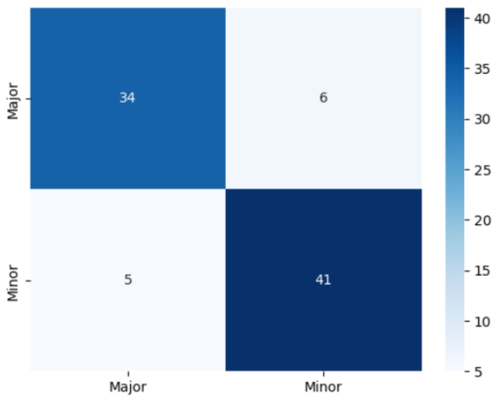
In the model where we used artificial neural networks for binary classification, the harmonics data from the DataFrame performed slightly better than the baseline model. They had an average accuracy of 63.01% on the test set, which was 15% of the shuffled dataset.

The model where we used transfer learning on ResNets gave an accuracy of 80.46%.



Confusion Matrix on this split of test-set for the ResNet Model

Finally, the XGBoost Model that performed the best out of all had an accuracy of 87.21% on the test set, and printed below is the confusion matrix.



Confusion Matrix on this split of test-set for the XGBoost Model

CONCLUSION

In this project, we explored the classification of musical chords using machine learning and deep learning models. We used a dataset of audio recordings of chords played on a guitar and a piano. After analyzing the dataset, we extracted features and trained models to classify the chords as major or minor. Two separate approaches were experimented with to convert the audio data into data suitable for the machine learning and deep learning models- including a 2D CNN and transfer learning on ResNets, and a harmonic feature extraction-based approach where ANNs and XGBoost were used to train and test the performance.

Our results showed that the XGBoost model outperformed all other models with an accuracy of 87.21% on the test set. Transfer learning on ResNets proved to be a successful approach, achieving an accuracy of 80.46%. This project highlights the potential of machine learning and deep learning techniques in analyzing and classifying audio data, particularly in music. Overall, we have demonstrated which models can effectively classify musical chords, and future research could focus on expanding this work to include more specific types of chords, and also, 1D CNNs can also be used to extract valuable information from the audio clips and might prove to be more successful than the current approaches.

REFERENCES

1. Nadar, Christon & Abeßer, Jakob & Grollmisch, Sascha. (2019). Towards CNN-based Acoustic Modeling of Seventh Chords for Automatic Chord Recognition.
2. Zhou, X., & Lerch, A. (2015). Chord Detection Using Deep Learning. *International Society for Music Information Retrieval Conference.*
3. https://www.kaggle.com/code/ahmetcelik158/mathematics-of-music-chord-classification
4. https://www.kaggle.com/code/alejandroguaranguay/mathematics-of-music-chord-classification
5. https://xgboost.readthedocs.io/en/stable/

Appendix

GITHUB LINK OF THE PROJECT

See https://github.com/sanskartewatia/EE541$_Project.git$ for more information.