Unveiling the Orchestra: A Novel System for Audio Separation and Instrument Identification in Musical Recordings

Group Members: Manato Ogawa, Juan Almanza, Brad Ma, Rakshithaa V. Jaiganesh, Danielle Wang

Mentors: Quincy Delp, Alan Wong

Abstract

Audio source separation is a widely applicable field that aims to utilize signal processing techniques to extract separate sources from a piece of audio. While the human brain can easily discern between audio sources, computers require source separation systems to achieve this task. It is difficult for a computer to identify which instrument is playing directly from audio – but we can create a unique frequency "signature" for each instrument based on qualities such as timbre to accomplish this task. The Fast Fourier Transform (FFT) lets us translate audio signals from the time domain to the frequency domain. This creates a unique visualization from which we can extract a musical signature to discern which instruments are playing at a given time. Conventional approaches run slowly and often fail to accurately separate sound. Using the FFT, we are developing an efficient and effective model to detect, categorize, and separate multiple audio sources based on relative magnitudes of harmonic frequency peaks. We hope that our results will demonstrate the ability of a model to differentiate between instruments using features we extract from audio. The instrument feature characterization could be applied to various domains, including music production, automated transcription, and music recommendation systems.

Background

When we listen to music, we are able to identify and process information about what we're hearing at incredible speeds. We can identify, for instance, what instruments are playing, how high or low their pitches are, etc. But can a computer accomplish this task to the extent humans can? This is the challenge we are confronted with in this research paper. There are various sounds that instruments create in a song – the sharp strumming of a guitar, the smooth melodies of a flute, or the low thuds of a drum. Our goal is to teach a computer, using artificial intelligence (AI), how to understand and identify these instruments when it encounters them in an audio recording.

This technology has the potential to change the way music is produced and consumed. Musicians could compose music more easily by allowing computers to organize and arrange the sounds of different instruments; music teachers could use the model to help students learn how each instrument contributes to a piece of music. By analyzing what types of instrumentals are most popular, music recommendation systems can be further refined and customized for each individual user.

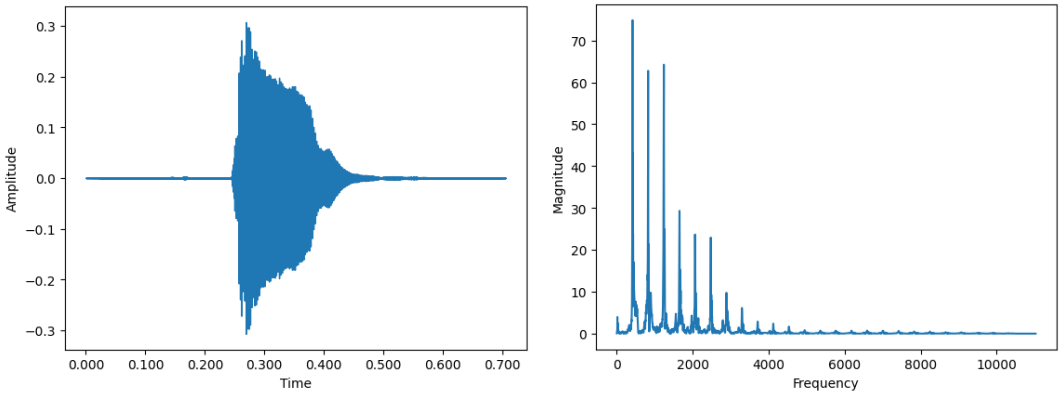
To tackle this challenge, we have split the project into two components: creating special tools that help computers understand & process sound, and engineering an intelligent model that learns from examples. The first tcomponent is used to prepare the data for the model to use – it converts audio (which the model cannot process) into numerical data, which the model *can* process. The second tool is our model, which we train with the prepared data. We'll be using these tools to explore whether a computer can truly grasp the unique voices of instruments hidden within audio files. By doing this, we will not just teach AI to recognize instruments; we will also learn more about how we understand and enjoy the music that fills our lives.

Dataset

We used a dataset from the Royal Philharmonic, which was uploaded to a shared Google Drive. Our dataset consists of all the standard symphony orchestra instruments playing every note in their range at several dynamics. There were roughly 10,000 audio files in total, with the files ranging in length from 1 second to 15 seconds. Each audio file was labeled with its respective instrument, making the training method a supervised learning method. Once the dataset for the audio files was gathered, all of the audio files were then passed into an FFT method which converted the audio files into analyzable data. All of the audio files had a specific FFT graph displaying the peak magnitudes, peak locations, and general shape the instrument's frequency possessed. The peak magnitudes represented the importance a specific frequency had on the overall instrument, with a higher value indicating a more influential frequency, and the peak locations were the frequency values itself. (In music theory, the tallest such peak is referred to as the *fundamental*, which is the note our ears hear; the smaller peaks, which are each a fixed distance away from the fundamental and each other, are referred to as the *overtones*). Then, we created a threshold so that only the most significant peaks were obtained (to avoid passing in too much data). From this, we were able to group together specific audio files from the same instrument with the same peak numbers, so that the ML algorithm could be trained using the general shape of each instrument's FFT graph with the same highest peak number value. Once all of the audio files were gathered into separate groups, the files were passed through a method which extracted valuable information, specified later in the paper. This data was then saved onto a CSV file, with which we trained our TensorFlow model .

Methods

For our project, we needed to create certain elementary methods to process our data. First, we created a python class named *AudioFile,* which housed the FFT method. As previously mentioned, the FFT method allowed us to convert audio files from the time domain to the frequency domain. The method utilizes a python library called *Librosa*. Librosa is a Python package that is commonly used for audio and music analysis. It provides tools for analyzing and processing audio signals, allowing users to work with audio data in various ways, such as extracting features, visualizing audio characteristics, etc. Librosa's FFT function was then used to display the magnitude v. frequency graph of a sample audio signal. To retrieve the FFT graph, we loaded in the audio file and passed it through the librosa.load() function.

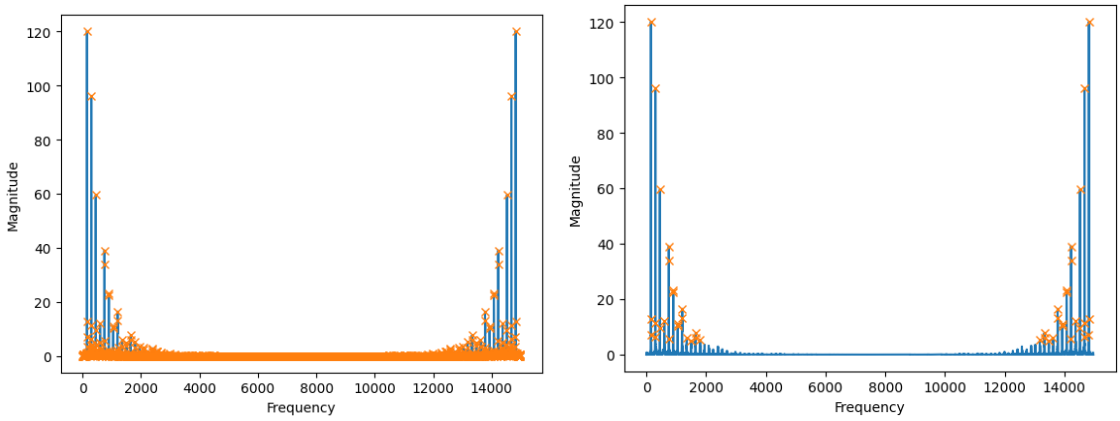


A. Librosa visualization ***B. FFT graph***

Figure 1

Figure 1A, shown above, represents a visual representation of the audio file of an instrument (in this case, a trumpet). As shown, the librosa method displays the waveform of the instrument, and the file is represented with the time domain. Figure 1B, also pictured above, shows the FFT representation of the same audio file after the file data was passed into the FFT method. As shown, the FFT contains many peaks with different magnitudes, all spaced apart relatively evenly. Once the FFT was retrieved, we were able to use it to visually identify valuable information such as peak height and peak magnitude.

As previously mentioned, the peak height and magnitude are key components of the data we used to train our AI model. To find these values, we used the *SciPy* library, an open-source scientific computing library for Python that provides a wide range of mathematical, scientific, and engineering functionalities. It builds on top of the NumPy library and offers additional tools for tasks such as optimization, signal processing, statistics, linear algebra, and more. We used the *find_peaks* method, which was contained within the SciPy submodule *scipy.signal.* The find_peaks method took an input array of data (from our FFT graph) and returned the indices where peaks (local maxima) were found. The function allowed us to customize the criteria for peak detection using various parameters such as height, threshold, distance, and prominence.



A. FFT graph without threshold ***B. FFT graph with threshold***

Figure 2

Figure 2A, shown above, is a visual representation of an FFT graph after the find_peaks method. Each orange 'x' represents a peak in the graph – which is problematic, because of the amount of unnecessary data. To filter out this unnecessary data, we placed a threshold to limit the find_peaks method from drawing an extensive amount of peaks. The threshold was determined by a specific percentile depending on the overall size of the audio file. Longer audio files resulted in a higher overall frequency value, resulting in the percentile increasing. Figure 2B, shown above, is a visual representation of the same FFT, but with a set threshold. This process was subsequently repeated for all of the instruments and all of the audio files.

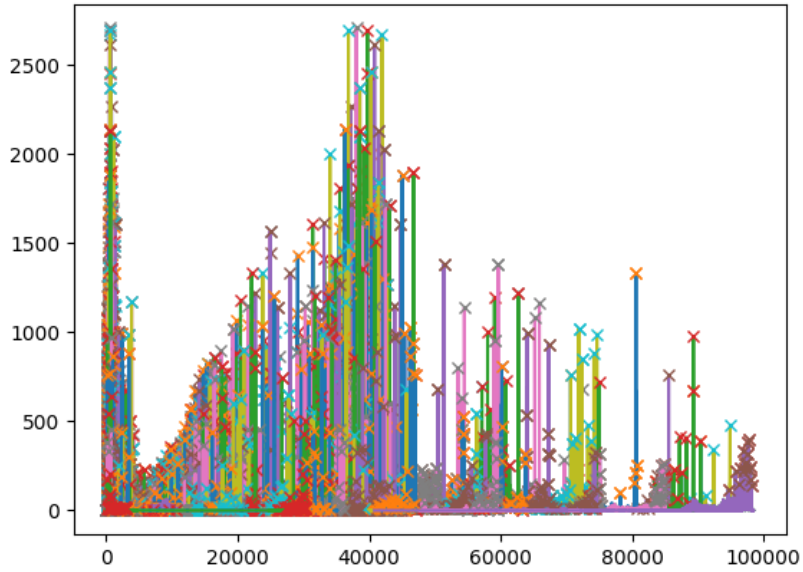
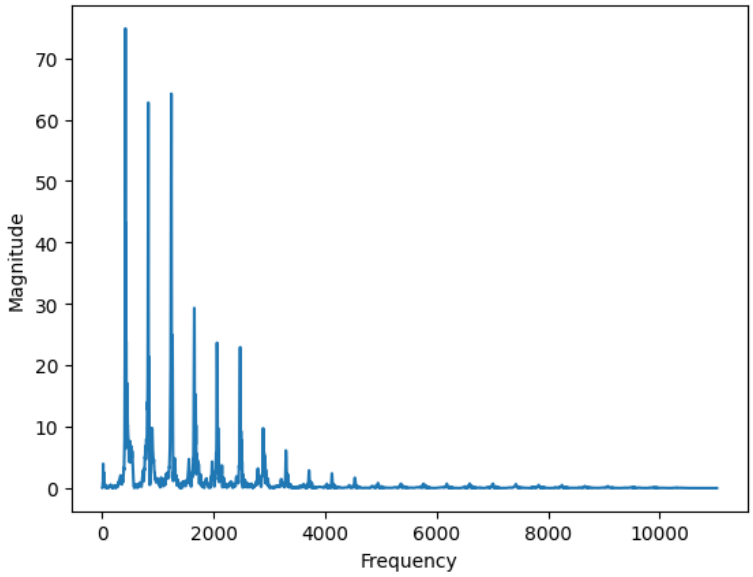


Figure 3

Figure 3, shown above, is a graph depicting all of the peaks detected for every audio file for a single instrument.

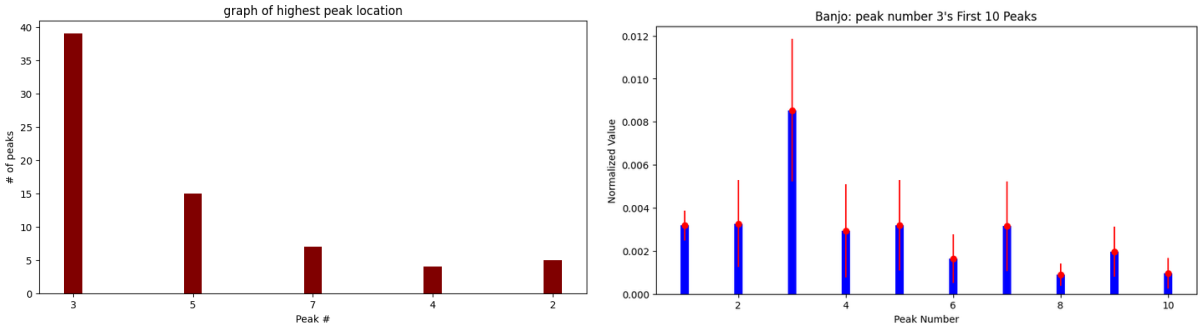


Same as figure 1; shown for clarity

Figure 4

Once all of the information was gathered, we could move onto the preparation of the training process. For the training process, we created training data with 5 parameters: average distance between peaks, highest peak magnitude, highest peak frequency, highest peak number, and name of instrument. These parameters were important because it allows the model to understand the unique shape of an instrument, which is necessary when training a machine learning model to detect an instrument. To obtain this information, 5 methods were needed. The first method was a method which would return the average distance between each peak in the FFT. As shown above in Figure 4, the spread of each peak is very uniform. Therefore, we determined that the average distance between each peak is a good indicator to differentiate the instruments. The method simply returned a value representing the average distance between each peak. For example in Figure 4, shown above, the method would print a value close to 400, as each peak is equally spaced about 400 frequencies apart. The second and third method returned the value of the highest peak frequency and magnitude of the FFT. These methods woud return the highest peak magnitude, in Figure 4's case, around 75 and the frequency correlating to it, around 400. The fourth method used was similar to methods two and three. However, the method returned the peak number of the highest magnitude. Peak number simply means the amount of peaks in the FFT. As shown above in Figure 3, there are 9 peaks above the magnitude of 5. Each peak is assigned a peak number, so the first peak is peak 1 and etc. The method simply returns the peak number with the highest magnitude, which is peak number 2 in Figure 3's case. Similar to the 2 methods described prior, we believe that the model can better understand and

differentiate between instruments based on the highest peak number. The last method simply returns the name of the instrument. This information is necessary to the training process so the model knows the instrument it is learning. Once all 5 of the necessary information was gathered, we separated audio files with the same highest peak numbers, and created graph based on the audio files of each separated groups.



A. Graph representing peak # ***B. Graph of shape of Banjo at peak #3***

Figure 5

Figure 5A, shown above, diagrams the frequencies at which the magnitudes are highest (indicating a peak). As shown, the banjo had the highest magnitude at peak #3 as the number of instances is nearly 40. All of the audio files with the same highest peak number were separated into different groups, so the banjo had a total of 5 groups. Figure 5B, shown above, diagrams all of the FFTs from the group containing audio files with the highest peak magnitude of 3. As shown, peak #3 is the highest compared to all of the other peaks. The red line indicates the standard deviation of each peak. The longer the red portion of the graph, the more diverse the data was.

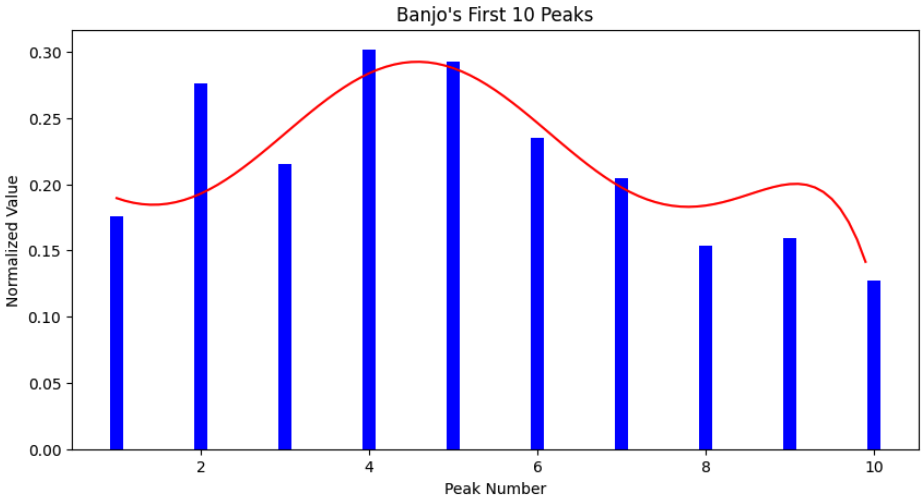


Figure 6

Figure 6, shown above, is a chart depicting the average magnitudes of the first 10 peaks for the Banjo instrument. The curve that you see is a result of the curve-fitting algorithm we created. Using a predefined polynomial and sinusoidal template function, the scikitlearn library was able to find the best constants that shape the curves of the bars. This curve fitting will be one characterization factor used later for our machine learning training. Once all of this information was gathered, we created a CSV file containing all of the data produced and proceeded to the training process.

avg_distance	highest_peak_fr	highest_peak_m	highest_peak_nu	instrument
1822.833333	737.4289935	377.9519266	3	banjo
1841.061667	744.8032834	381.7314459	3	banjo
1859.29	752.1775734	385.5109651	3	banjo

Figure 7, shown above, represents the first three rows of the CSV file. The first column represents the average distance between the peaks, the second column represents the highest peak frequencym the third column represents the highest peak magnitude, the fourth column represents the highest peak number and the fifth column represents the name of the instrument

Figure 7

We utilized a TensorFlow model for the research paper. TensorFlow is an open-source machine learning framework developed by the Google Brain team, and is widely used for building and training various machine learning models, including neural networks. For this research paper, we decided to train the model for 1000 epochs, as the values showed little variation or change after that amount

TensorFlow utilizes 4 layers to train the model. The first layer, known as the *input layer*, is the layer that receives the data or input and passes it on to the other layers. In our case, there were 5 input nodes.

The second layer, known as the collective *hidden layers*, are the layers between the input and output layers of the neural network. These layers perform computations on the input data using weights and loss functions. Weights are the parameters that the model learns during the training process. They determine how strongly the input signals from one layer influence the output of the subsequent layer. Each connection between neurons in different layers has an associated weight. These weights are initially set to small random values, and as the model is trained, they are adjusted iteratively to minimize the difference between the predicted outputs and the actual target values. The *loss function* (also known as the *cost function*) is a key component of a neural network that quantifies how well the model's predictions match the actual target values. It is a mathematical measure of the difference between predicted outputs and true outputs. The goal during training is to minimize this loss function. One key component of the hidden layer are the *activation functions*. An activation function is applied to the output of each neuron in the hidden layer, which allows the model to learn complex patterns and relationships in the data. For this paper, we used a specific activation function called *'ReLu'* (Rectified Linear Unit).

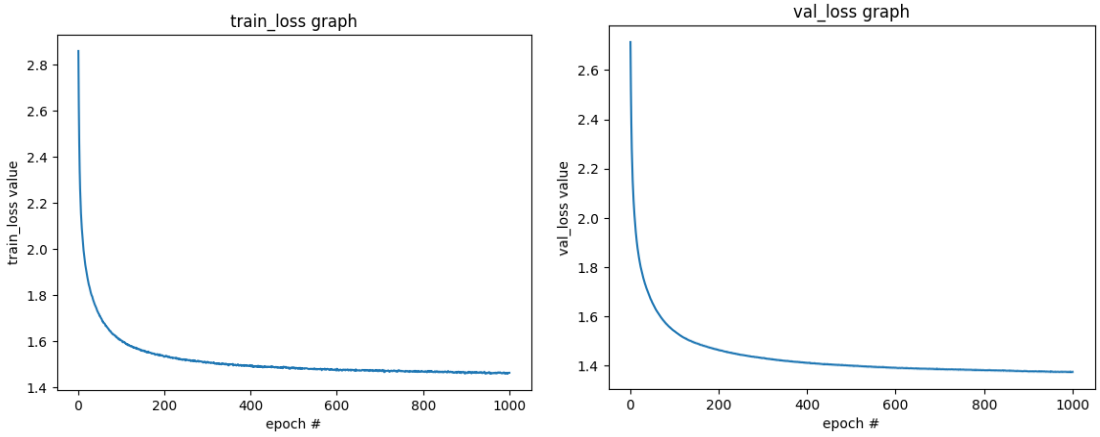
The third layer, known as the *dropout layer*, is a technique aimed to prevent the model from overfitting. Overfitting is a phenomenon caused when the model uses the same data over

and over to train, causing it to memorize the data, instead of learning patterns. To prevent this, the dropour later randomly deactivated neurons after each epoch, helping to make the network more robust and less reliant on specific neurons and thereby enhancing its generalization to new data.

The fourth layer, known as the *output layer*, is the final layer of the neural network. It produces the model's predictions or outputs based on the computations from the hidden layer. It is in this layer where the model predicts which instrument is being inputted.

Results

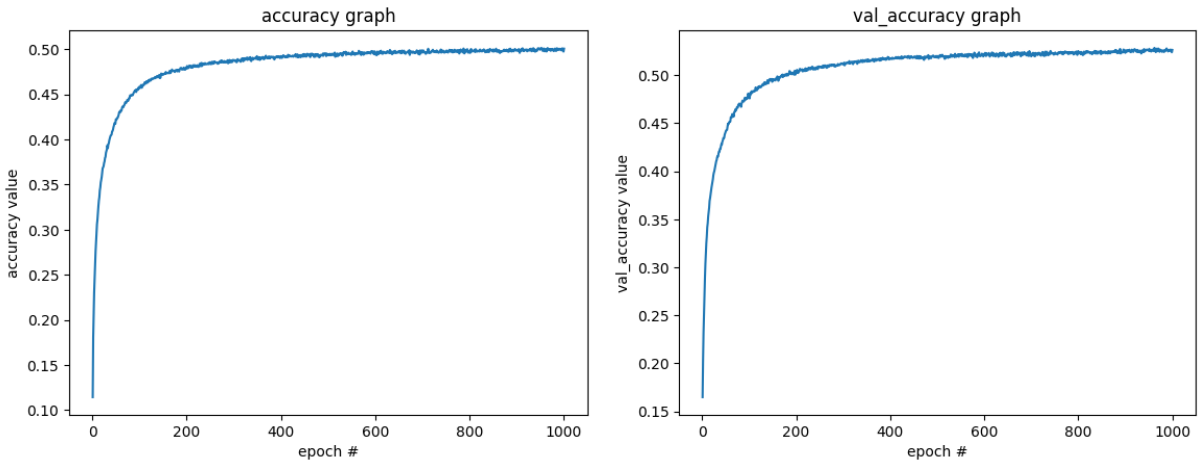
Once we concluded the training process, we selected a few audio files from outside the dataset. These files were then passed into our model, yielding the following results:



A. Graph of train_loss data ***B. Graph of val_loss data***

Figure 8

Figure 8, shown above, are visual representations of the loss functions over time. As shown above, the loss value slowly decreases over the 1000 epochs. The training loss is a measure of how well a machine learning model's predictions match the actual target values on the training dataset. It represents the difference between the predicted outputs and the ground truth. Validation loss is similar to training loss but is computed on a separate validation dataset that the model has never seen during training. The purpose of using a validation dataset is to assess the model's performance on unseen data. As shown above, both the validation and training loss has decreased, indicating the model has improved over time.



A. graph of accuracy ***B. graph of val_accuracy***

Figure 9

Figure 9, shown above, represents the graph of the model's accuracy over the 1000 epochs. Although the points are slightly scattered towards the end, the overall accuracy from the 1000 epoch training is around 50%. Once we determined the accuracy of our model, we then prepared the testing data to see how our model performs under audio files from the internet. Firstly, we picked out 10 audio files and passed them through a predicting method.

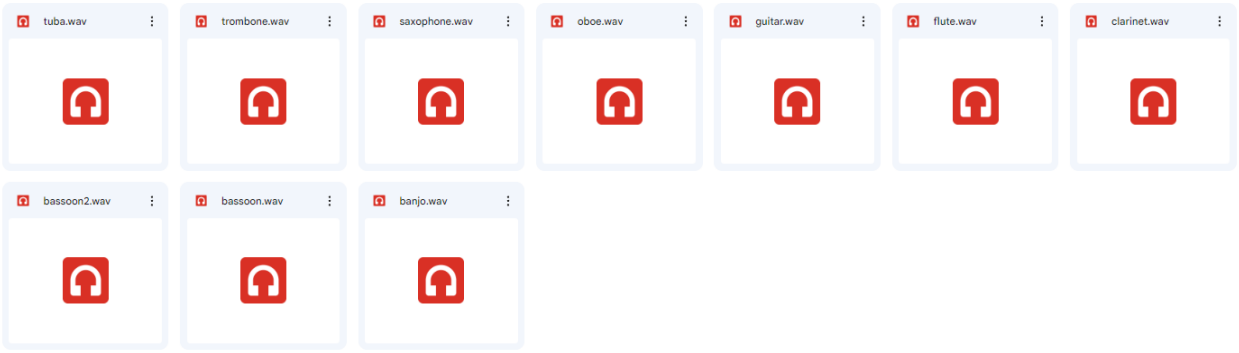


Figure 10, shown above, shows the files loaded into the testing folder into the google drive

Figure 10

Once the audio files were loaded into a google drive folder, we created our own predicting method. The predicting method imputed a list of 4 details as explained before: average distance, highest peak frequency, highest peak magnitude, and peak number. Then, the method would input the list into the model we created which results in predictions being made.

```
1/1 [==============================] - 0s 18ms/step
Predicted Instrument: tuba
```

A. Example of predicting process

```
correct:  ['bassoon', 'tuba', 'trombone', 'oboe', 'flute', 'clarinet', 'bassoon', 'banjo', 'guitar', 'saxophone']
predicted:  ['tuba', 'tuba', 'bass clarinet', 'saxophone', 'tuba', 'saxophone', 'tuba', 'banjo', 'guitar', 'saxophone']
```

B. Full list of instruments predicted from model and the correct answers

Figure 11

Figure 10A, shown above, represents the model predicting one audio file from the google drive folder. Figure 10B, shown above, represents the correct vs predictions made from the model. As shown, the model predicted, 4 out of 10 correctly, having an accuracy of 40%. Although this is lower than the initial 50% accuracy from the training, the model was somewhat successful in being able to determine the instrument being played.

Conclusion

Audio source separation is a complex domain, where the goal is to dissect the amalgamation of sounds that constitute a musical recording. While the human brain performs this feat effortlessly, replicating this ability in machines necessitates the development of advanced algorithms. Our approach relied on the FFT to transition audio signals from the time domain to the frequency domain, ultimately revealing distinctive frequency "signatures" for individual instruments based on their unique timbral qualities.

In the present study, our goal was to develop an effective machine learning model to detect, categorize, and separate multiple audio sources based on peaks. We had a comprehensive dataset with over 10,000 audio files from 20 unique musical instruments. Employing the FFT technique, we transmuted audio data into visually interpretable graphs, depicting harmonic frequency peaks that define each instrument's auditory identity. These peaks not only served as the crux of our analysis but also facilitated the categorization and differentiation of instruments, defining peak features such as magnitude, frequency, and peak number. This quantitative features database was used to create a robust classification machine learning model.

The algorithm was proficient in differentiating between instruments using features we extracted from audio files. It underwent rigorous training spanning 1000 epochs, each iteration honing its ability to discern instrument-specific features, ultimately exhibiting an appreciable accuracy of approximately 50% on the presented dataset.

Future Research

Our future plans are to complete the separation aspect of the project, which will require further utilization of machine learning and signal processing techniques. While we were able to successfully extract information on the instruments' audio files and characterize them and train an accurate ML model that could recognize them, we hope to develop a full system that is able to separate a symphony orchestra's playing into individual parts and detect each of the instruments

playing at any moment. We envision a comprehensive system that can dissect a symphonic performance into its constituent parts and isolate each instrument's contribution.

Our research unveils a pioneering system that bridges the gap between machine perception and musical composition. Through advanced signal processing techniques and machine learning, we have shed light onto the intricacies of audio source separation and instrument identification. By integrating AI and audio, our research uncovers new avenues for efficient music composition, enhanced music recommendation systems, and much more.

References

[1] E. Cano, D. FitzGerald, A. Liutkus, M. D. Plumbley and F. -R. Stöter, "Musical Source Separation: An Introduction," in IEEE Signal Processing Magazine, vol. 36, no. 1, pp. 31-40, Jan. 2019, doi: 10.1109/MSP.2018.2874719.

[2] Grg, Pema. “Audio Signal Processing.” *Medium*, EKbana, 21 Mar. 2022, blog.ekbana.com/audio-signal-processing-f7e86d415489.

[3] Parker, Michael. *Digital Signal Processing 101 : Everything You Need to Know to Get Started*, Elsevier Science & Technology, 2010

[4] Talebi, Shawhin. “The Fast-Fourier Transform (FFT).” *Medium*, The Startup, 1 Apr. 2023, medium.com/swlh/the-fast-fourier-transform-fft-5e96cf637c38.