eSTUDENT

Instrument Detection in Audio Signals Using Machine Learning Methods

Project Documentation

LUMEN Data Science Competition 2023
at the University of Zagreb

Contents

1 Introduction

The development of automated systems that can detect and classify musical instruments from audio recordings has become increasingly important in recent years due to the growing popularity of music in various industries. For example, in music information retrieval, such systems can be used to help users search for and retrieve specific pieces of music based on the instruments used. This can be especially useful in contexts where traditional metadata-based searches may not be sufficient, such as when searching for pieces of music with specific timbral characteristics or when searching for music that features uncommon or obscure instruments.

In music production, automated instrument detection can be used to help producers and engineers isolate and manipulate individual tracks or instruments within a mix, allowing for more precise editing and processing. For example, a producer may want to adjust the volume or EQ of a specific instrument in a mix, or may want to replace a particular instrument with a different sample or recording. Automated instrument detection can help streamline these processes by automatically identifying and separating individual tracks or instruments from a mix. This can save producers and engineers a significant amount of time and effort, especially in large, complex mixes with many different instruments.

In music education, instrument detection can be used to provide students with real-time feedback on their playing, helping them to improve their skills and technique. For example, a student may use an instrument detection system to record themselves playing a piece of music, and the system can then analyze the recording and provide feedback on areas where the student may need to improve, such as intonation or rhythm. This can be a valuable tool for music educators, as it allows them to provide targeted feedback and guidance to students, even when they are not physically present.

WAV format files are a popular choice for instrument detection because they are capable of storing uncompressed audio data, which means that the audio signal is not subjected to any lossy compression techniques that could degrade its quality. Additionally, WAV files can store a variety of audio metadata, such as sample rate, bit depth, and channel count, which can be useful for analyzing the audio signal. However, WAV files can also be quite large and can require significant processing power to analyze, which can be a challenge for some applications. As a result, researchers and developers are continually working to develop more efficient algorithms and techniques for instrument detection that can handle large audio files with greater ease. These techniques may involve using more advanced machine learning algorithms, more efficient signal processing techniques, or a combination of both.

Overall, the development of automated systems for instrument detection represents a significant advancement in the field of music technology, with potential applications in a variety of contexts, from music production to education. The task at hand is now to develop a solution capable of detecting/classifying musical instruments within an audio recording. Given a WAV format audio file, the objective is to identify all the instruments present in the recording. The possible instruments include cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin, and human vocal. The results of the model need to be returned in the JSON format, where the keys are strings representing the 11 listed instruments, and the values are either 0 or 1, depending on whether that instrument is present in the audio file.

However, the data used to train the model presents a challenge, as the training data differs from the test and validation data. The training data set will contain a certain number of audio clips for each instrument, in which only that instrument is present (more on this later). On the other hand, the test and validation data contain audio recordings with multiple instruments, and detecting them is the main task. The approach and strategy to utilize the training data to the fullest are left to the imagination of the developer.

In this work, we will use a ML Approach similar to [Bla22], modularising the problem at hand into many binary classifications. For each instrument we construct a model that either detects the presence of the instrument or not. This architecture has threefold benefits:

- In principle, we can use different models for different instruments
- The complexity requirements of each model is lower, as each model only solves a smaller task
- Training can be executed on multiple devices at the same time

While all three points can come in handy, we really only make use of the second one and third one, choosing the same model for all instruments. We are using a CNN for our task, more on this later though.

The training data for this dataset consists of audio recordings, each lasting 3 seconds, containing only one instrument. Although some recordings may have additional instruments playing in the background, only the instrument that is present throughout the entire recording will be labeled. The possible instruments and their corresponding labels are cello (cel), clarinet (cla), flute (flu), acoustic guitar (gac), electric guitar (gel), organ (org), piano (pia), saxophone (sax), trumpet (tru), violin (vio), and human vocals (voi).

In addition, some of the files have notes in their file names indicating the presence ([dru]) or absence ([nod]) of drums, as well as the music genre: country-folk ([cou-fol]), classical ([cla]), pop-rock ([pop-roc]), and latino-soul ([lat-sou]). We will, however, not make use of these additional lables.

There are a total of 6705 audio files in 16-bit stereo wav format sampled at 44.1 kHz in the training set. The validation data, on the other hand, consists of audio recordings lasting between 5 to 20 seconds with one or more instruments playing, and their corresponding labels are provided in a .txt file listing all the instruments present. It is important to note that only the instruments listed in the training phase (as mentioned above) will be considered, which means instruments such as drums will not be listed.

There are a total of 2874 files in 16-bit stereo wav format sampled at 44.1 kHz in the validation set. The test set, which is unknown to the participants, will be used to evaluate the final accuracy of the model.

2 Signal Processing of Sound Waves

2.1 The Physics of Sound

Sound is a type of mechanical wave, which means it requires a medium to propagate. Mechanical waves are created when a disturbance or vibration is introduced into a medium, causing the particles in that medium to oscillate back and forth. This oscillation creates a wave that travels through the medium, carrying energy with it. Sound waves have several properties that define their behavior, including wavelength, frequency, amplitude, and speed.

2.1.1 Wavelength

Wavelength is the distance between two consecutive points on a sound wave that are in phase. It is measured in meters and is denoted by the Greek letter lambda (λ). The wavelength of a sound wave is related to its frequency and speed by the following equation:

$$\lambda = \frac{v}{f} \tag{1}$$

where v is the speed of sound in the medium and f is the frequency of the wave.

2.1.2 Frequency

Frequency is the number of cycles of a sound wave that occur in one second and is measured in Hertz (Hz). The frequency of a sound wave is related to its pitch - higher frequency waves have a higher pitch, while lower frequency waves have a lower pitch.

2.1.3 Amplitude

Amplitude is the maximum displacement of particles in a medium from their equilibrium position as a result of the passage of a sound wave. It is measured in meters and is denoted by the letter A. The amplitude of a sound wave is related to its loudness - higher amplitude waves are perceived as being louder, while lower amplitude waves are perceived as being quieter.

2.1.4 Speed

The speed of sound is the rate at which a sound wave travels through a medium and is measured in meters per second. The speed of sound depends on the properties of the medium through which it is traveling, such as its density and temperature. In air at room temperature, the speed of sound is approximately 343 meters per second.

2.1.5 The Wave Equation

The behavior of sound waves can be described using the wave equation:

$$\frac{\partial^2 p}{\partial x^2} = \frac{1}{v^2}\frac{\partial^2 p}{\partial t^2} \tag{2}$$

where p is the pressure of the sound wave, x is the position of a particle in the medium, t is time, and v is the speed of sound in the medium. This equation describes how the pressure of a sound wave varies with time and space.

2.1.6 Sound Propagation

When a sound wave is produced, it travels outwards from the source in all directions. As the sound wave moves away from the source, it loses energy due to the spreading out of the wavefront and the absorption of energy by the medium. This causes the amplitude of the sound wave to decrease as it travels further from the source.

In addition, sound waves can be reflected, refracted, and diffracted as they propagate through a medium. Reflection occurs when a sound wave encounters a boundary between two media and bounces back. Refraction occurs when a sound wave changes direction as it passes through a medium with varying properties, such as air with different temperatures or pressures. Diffraction occurs when a sound wave bends around obstacles or through openings in a barrier.

2.2 Audio WAV File Format

In the real world, sound is created by the vibration of an object, causing the surrounding air particles to vibrate and create a pressure wave that travels through the medium, usually air. This continuous waveform is then approximated in digital form through a process called sampling. Sampling involves taking regular measurements of the sound wave at fixed intervals of time and representing each measurement as a numerical value (what we are measuring really is the amplitude of the wave at specific times). These numerical values are then stored in a digital format, such as a WAV file, which can be processed and played back by a computer.

To convert the analog sound wave into digital audio, we use pulse code modulation (PCM), a method of digitally representing analog signals by sampling the signal at regular intervals and quantizing each sample to the nearest value in a finite set of levels. In PCM, the analog sound wave is sampled at a certain rate, called the sample rate, and each sample is represented by a binary number with a certain number of bits, called the bit depth.

The sample rate and bit depth play crucial roles in determining the quality and fidelity of the digital audio signal. The sample rate determines the frequency range of the audio signal that can be accurately represented in the digital domain, while the bit depth determines the resolution and dynamic range of each sample. A higher sample rate and bit depth generally result in better audio quality, but also require more storage space and processing power.

The Nyquist frequency is a limit to the frequency range that can be represented in the digital audio signal and is defined as half the sampling rate. Any frequencies above the Nyquist frequency will be incorrectly represented in the digital audio signal and will result in a phenomenon known as aliasing.

In the context of sound, audio refers to an electrical representation of sound waves that can be stored, transmitted, and manipulated. Digital audio is created by converting an analog sound signal into a series of binary numbers that can be stored as a digital file. WAV (Waveform Audio File Format) is a commonly used digital audio format that stores audio data using PCM. WAV files can store mono or stereo audio data, and can be compressed or uncompressed. Uncompressed WAV files store the raw PCM data without any compression, while compressed WAV files use various compression algorithms to reduce the file size.

In addition to the Nyquist frequency, the bit depth also affects the digital audio signal by determining the maximum amplitude that can be represented by a given bit depth. The formula to calculate the maximum amplitude that can be represented by a given bit depth is $\mathrm{A}_{max} = \mathrm{p} \cdot 2^{n-1}$, where n is the bit depth and p is the scaling factor. For example, with a 16-bit depth and a scaling factor of 1, the maximum amplitude that can be represented is $\pm$ 32767. We are working with 16-bit WAV files with a sample rate of 44.1kHz. A 16-bit WAV file with a sample rate of 44.1kHz is a commonly used format for

CD-quality audio. This means that the sound wave is sampled 44,100 times per second, and each sample is represented by a 16-bit binary number. With a bit depth of 16 bits, there are 65,536 possible levels for each sample, resulting in a dynamic range of approximately 96 decibels (dB). This is sufficient for most music and audio applications, as the human ear can typically only perceive sounds within a dynamic range of about 120 dB. However, for more demanding applications such as professional audio recording and mastering, higher bit depths and sample rates may be used to achieve even greater fidelity and accuracy.

2.3 Basic Signal Processing

In the context of digital audio, an audio signal can be thought of as a discrete-time signal that is sampled from an analog sound wave. This digital representation of the sound wave is stored as a sequence of numerical values in a file format such as WAV. In this section, we will explore the mathematics behind basic signal processing of audio WAV files.

2.3.1 Time Domain and Frequency Domain

The time domain is a representation of the audio signal as a function of time. In other words, the audio signal is plotted on the y-axis against time on the x-axis. The time domain representation of an audio signal is useful for analyzing the signal's properties over time, such as amplitude, frequency, and phase.

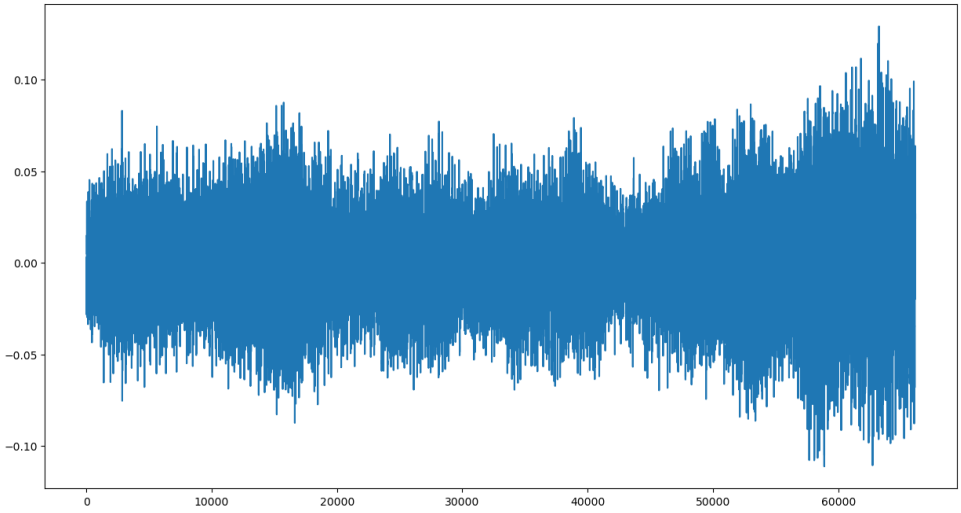

Figure 1: Time Domain Representation of one of the Audio Files of the Training Data

The frequency domain, on the other hand, is a representation of the audio signal as a function of frequency. The frequency domain representation of an audio signal is obtained by applying the Fourier transform to the time domain representation of the signal. The Fourier transform converts a time domain signal into a frequency domain signal, allowing us to analyze the signal's properties in terms of its frequency content.

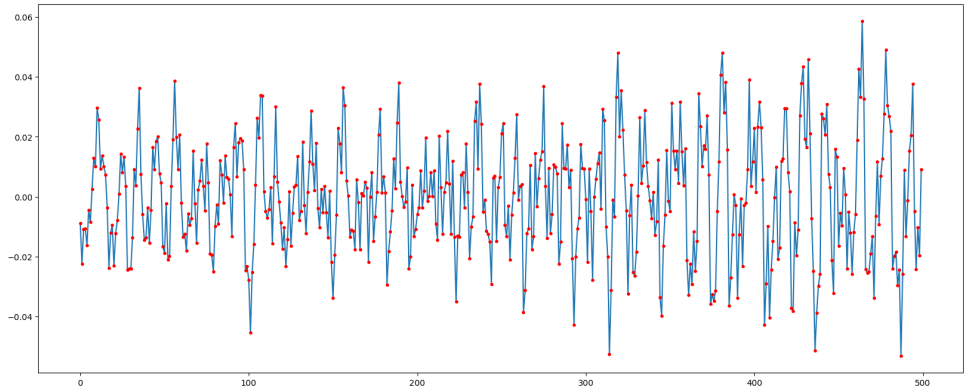

Figure 2: Frequency Domain Representation of the same Audio File

2.3.2 Fourier Transform

The Fourier transform is a mathematical tool that is used to decompose a signal into its constituent frequencies. For an audio signal $x(t)$, the Fourier transform is defined as:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i f t} dt$$

where $X(f)$ is the frequency domain representation of the signal, f is the frequency, and i is the imaginary unit.

The inverse Fourier transform is used to convert the frequency domain representation of a signal back into the time domain representation:

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{2\pi i f t} df$$

where $x(t)$ is the time domain representation of the signal.

2.3.3 Discrete Fourier Transform

In practice, we cannot perform the Fourier transform on a continuous signal, as the integral may not converge. Instead, we use the discrete Fourier transform (DFT), which is a numerical approximation of the Fourier transform that is suitable for analyzing digital signals.

For an N-point sequence $x[n]$, the DFT is defined as:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-2\pi i k n / N}$$

where $X[k]$ is the frequency domain representation of the signal and k is an integer between 0 and $N - 1$.

The inverse DFT is used to convert the frequency domain representation of a signal back into the time domain representation:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{2\pi i k n / N}$$

where $x[n]$ is the time domain representation of the signal.

2.4 Spectrograms

A spectrogram is a visual representation of the frequency content of a signal over time. It is obtained by dividing the signal into short overlapping segments, and computing the Fourier transform of each segment to obtain its frequency content. The resulting frequency spectrum is then plotted against time to create a two-dimensional representation of the signal's frequency content over time.

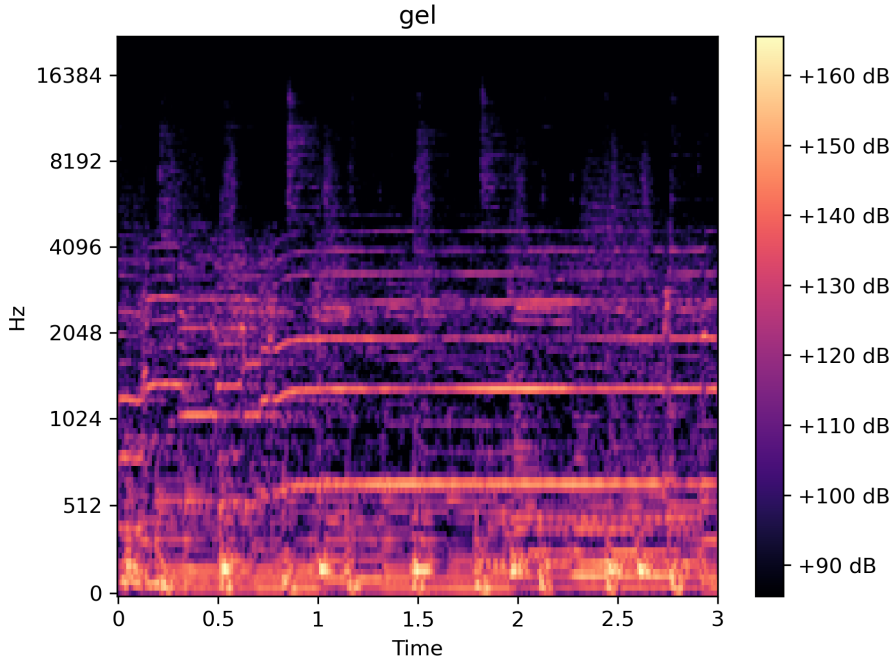

Figure 3: Spectogram of an Electric Guitar File

To obtain a spectrogram from a WAV file, we can use a library like Librosa in Python. The process involves several steps:

1. Load the WAV file and extract the audio signal as a one-dimensional array.

2. Apply a windowing function to each segment of the signal to reduce spectral leakage. A commonly used windowing function is the Hann window:

$$w[n] = 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right), \qquad 0 \leq n \leq N-1 \tag{3}$$

where N is the length of the window.

3. Divide the signal into overlapping segments of length L, with a hop size of H. This means that adjacent segments overlap by $L - H$ samples.

4. Compute the discrete Fourier transform (DFT) of each segment using the fast Fourier transform (FFT) algorithm:

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N}, \qquad k = 0, 1, \ldots, K-1 \tag{4}$$

where N is the length of the segment, K is the number of frequency bins in the FFT (usually $N/2+1$), and $x[n]$ is the windowed segment.

5. Take the magnitude of the complex Fourier coefficients to obtain the power spectrum:

$$P[k] = |X[k]|^2 \tag{5}$$

6. Repeat steps 4 and 5 for each segment of the signal, and concatenate the resulting power spectra to obtain a matrix of size $K \times T$, where T is the number of segments.

7. Apply a logarithmic scaling to the power spectrum to compress the dynamic range of the values:

$$S[k,t] = 10 \log_{10} \left(\frac{P[k,t]}{P_{\text{ref}}} \right) \tag{6}$$

where P_{ref} is a reference power level (usually set to the maximum power value in the spectrogram).

8. Optionally, apply a frequency weighting function to the spectrogram to mimic the frequency response of the human ear. One commonly used weighting function is the mel scale, which is a nonlinear transformation of frequency that approximates the perceptual distance between sounds. The resulting spectrogram can be visualized as a heatmap, with time on the horizontal axis, frequency on the vertical axis, and the color representing the power level at each point in the time-frequency plane.

We will now show typical spectrograms from each of the instruments.

float

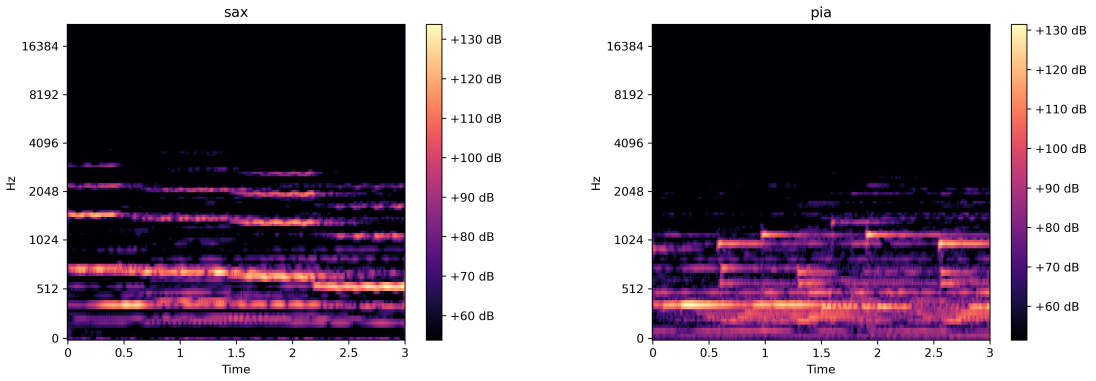

Figure 4: Spectrogram of a Saxophone File (left) and a Piano File (right)

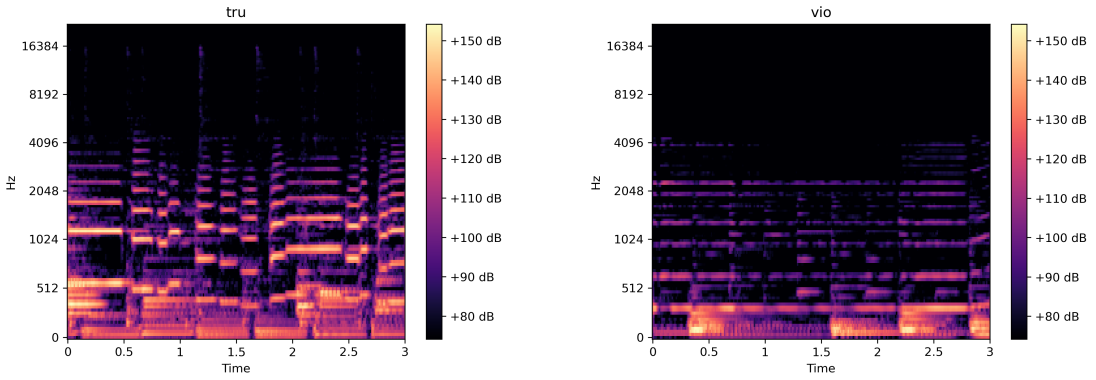

Figure 5: Spectrogram of a Trumpet File (left) and a Violin File (right)

Spectrograms will make up the backbone of our ML models, as we will ultimately use the spectrograms

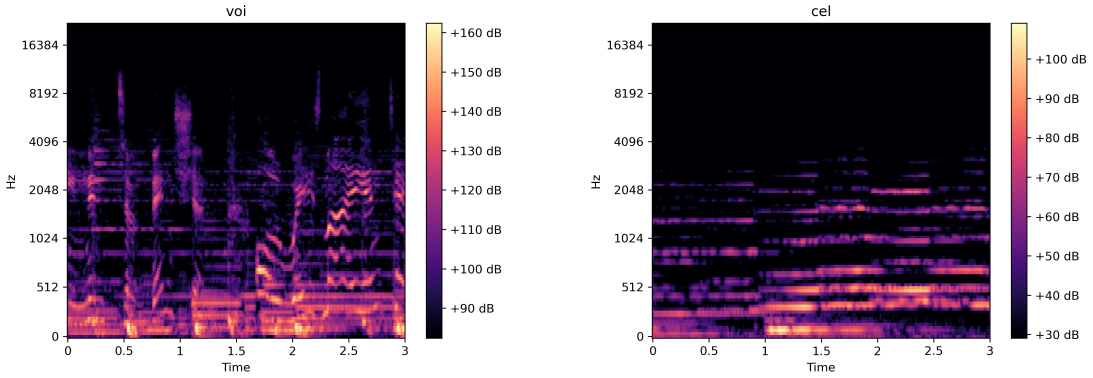

Figure 6: Spectrogram of a Voice File (left) and a Cello File (right)

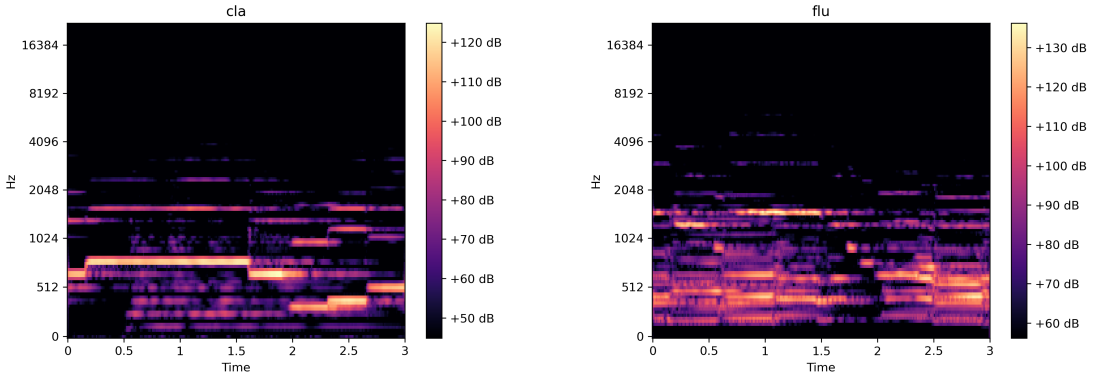

Figure 7: Spectrogram of a Clarinet File (left) and a Flute File (right)

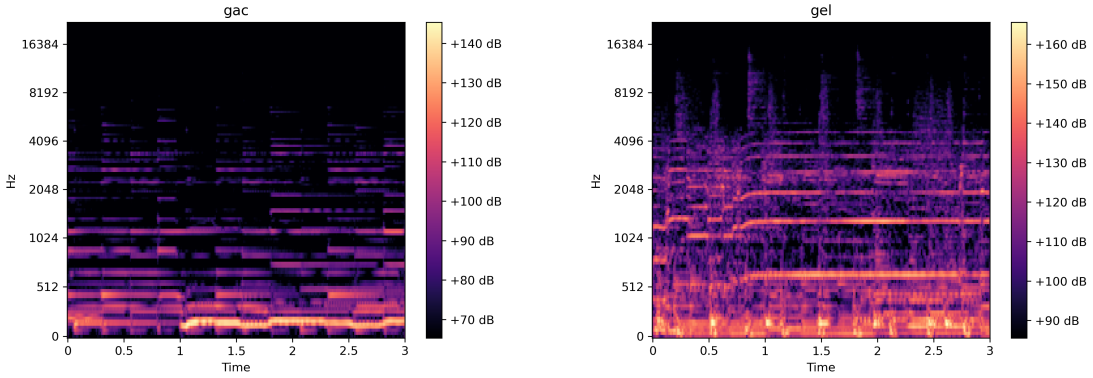

Figure 8: Spectrogram of an Acoustic Guitar File (left) and an Electric Guitar File (right)

of the audio signals as the data to train the CNNs on. Much of the techniques covered in this chapter can also be found in [Dos21] and in [Kaw19].

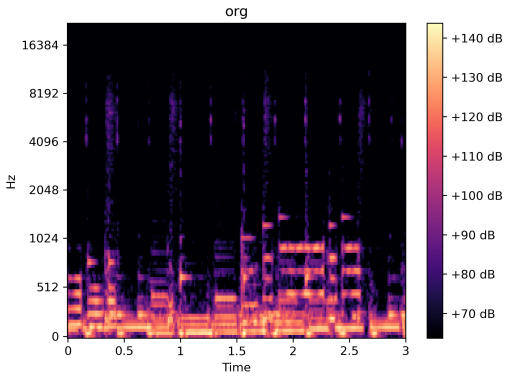

Figure 9: Spectrogram of an Organ File

2.5 Mel Frequency Cepstral Coefficients (MFCC)

Mel Frequency Cepstral Coefficients (MFCCs) are a commonly used feature extraction technique for speech and audio processing. They are derived from the power spectrum of a signal, and are particularly useful for speech recognition tasks.

MFCCs are obtained by applying a series of transformations to the power spectrum of a signal, as follows:

Divide the signal into overlapping segments, and apply a windowing function to each segment to reduce spectral leakage.

Compute the discrete Fourier transform (DFT) of each segment using the fast Fourier transform (FFT) algorithm, to obtain the power spectrum.

Apply a filterbank of triangular filters spaced on the mel scale to the power spectrum, to obtain a set of mel-frequency spectral coefficients.

Take the logarithm of each mel-frequency spectral coefficient to obtain the log-mel-spectrum.

Apply the discrete cosine transform (DCT) to the log-mel-spectrum, to obtain the cepstral coefficients.

The resulting MFCCs are a set of real-valued coefficients that capture the spectral envelope of the signal, and are typically used as input features for speech recognition algorithms. They are particularly useful because they are relatively insensitive to additive noise and distortions in the signal.

To compute MFCCs in Python, we can use a library like Librosa, which provides a convenient function called mfcc that implements the above steps. The resulting MFCCs can be visualized as a matrix, with time on the horizontal axis and the MFCC coefficients on the vertical axis.

In our instrument detection project, we can use MFCCs as an alternative or complementary feature to spectrograms, and train our CNNs on a combination of both types of features. Further details on MFCCs and their applications can be found in [Dos21] and in [Kaw19].

Now let us look at a couple MFCC examples.

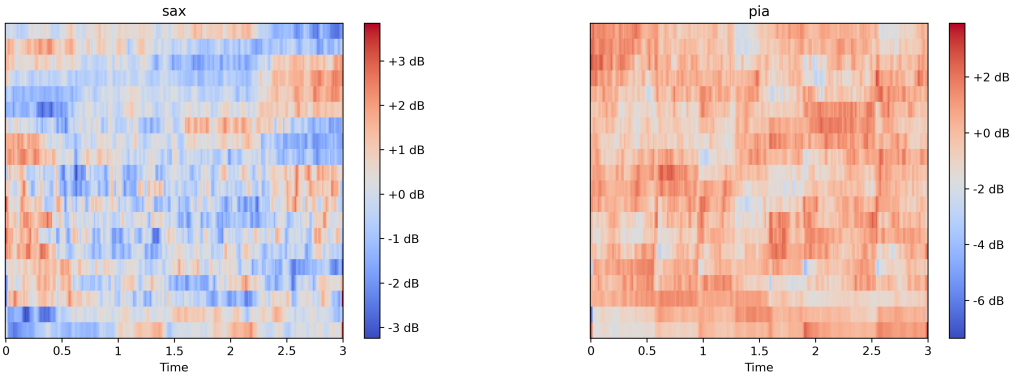

Figure 10: MFCCs of a Saxophone File (left) and a Piano File (right)

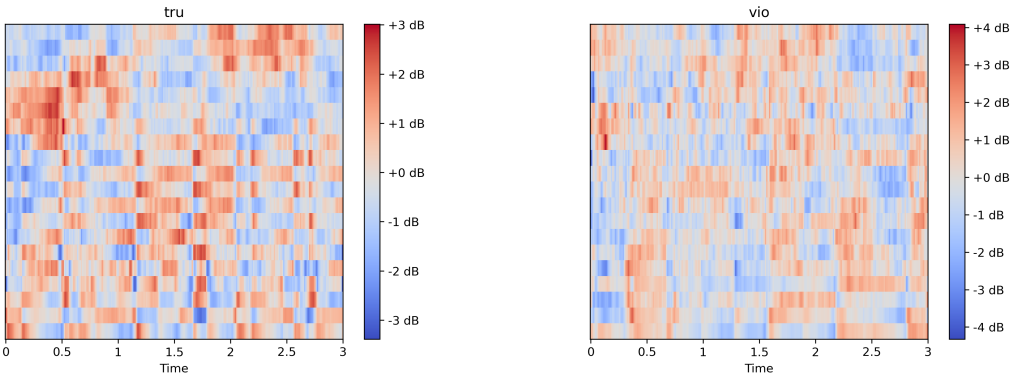

Figure 11: MFCCs of a Trumpet File (left) and a Violin File (right)

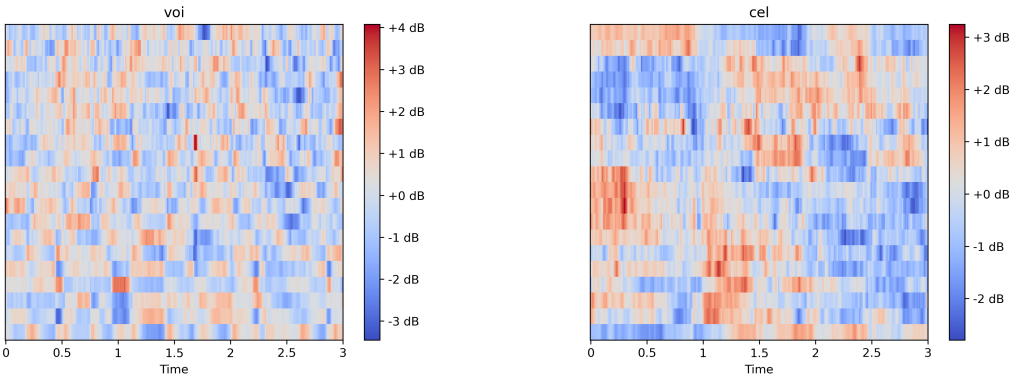

Figure 12: MFCCs of a Voice File (left) and a Cello File (right)

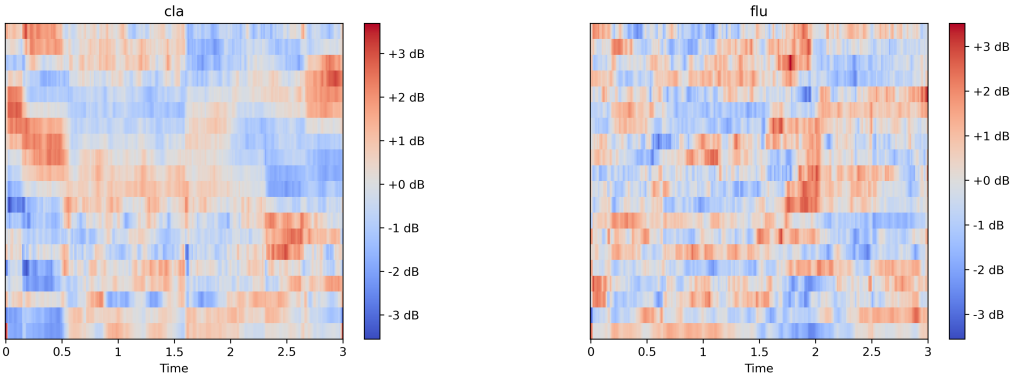

Figure 13: MFCCs of a Clarinet File (left) and a Flute File (right)

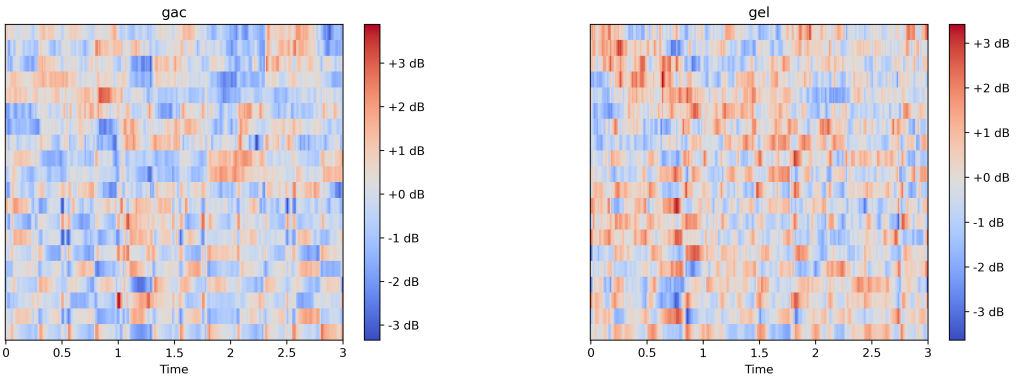

Figure 14: MFCCs of an Acoustic Guitar File (left) and an Electric Guitar File (right)

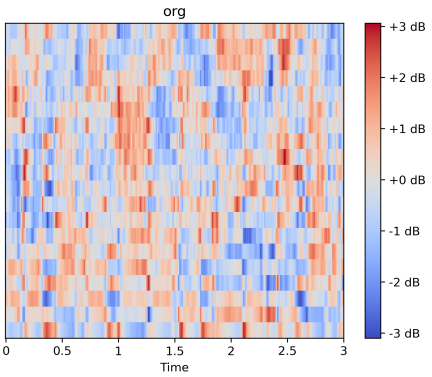

Figure 15: MFCCs of an Organ File

3 Our Solution

When constructing our solution, we came up with the following idea. We will take the dataset and enhace it by adding new audio signals with more than one instrument to them (this is easily done by adding the two files), such that for each of the instruments we will have about half the training data with them and half without them. This is also commonly referred to as "Synthetic Minority Over-sampling Technique" (SMOTE). We will then train binary classifier CNNs for each of the instruments. For the validation data then each binary CNN will either vote for or against the instrument in the sample. The validation data provided will no longer be be always 3 seconds long (like the training data). Hence we segment the audio into 3 second segments which we can then feed through the CNNs. For those 3 second segments we then have a majority vote of the segments (setting to 0 in a tie).

3.1 Enhancing the Data Set

In the context of digital audio, adding two signals means adding the numerical values of each sample at the corresponding time points. Mathematically, if we have two audio signals $x(t)$ and $y(t)$, where t is the time index, the resulting signal $z(t)$ obtained by adding them is:

$$z(t) = x(t) + y(t)$$

In practice, this means that we add the numerical values of each sample of $x(t)$ and $y(t)$ at the corresponding time points to obtain the numerical value of $z(t)$ at that time point.

It is important to note that the signals $x(t)$ and $y(t)$ need to have the same sample rate and bit depth in order to be added together. This however is given for our training data.

We enhance the data set for category S by randomly choosing any category

$$T \in \{\text{cel, cla, flu, gac, gel, org, pia, sax, tru, vio, voi}\} \setminus \{s\}.$$

For any $x \in S$ we then randomly choose $y \in T$ and add $z = x + y$ to the training data. SMOTE (Synthetic Minority Over-sampling Technique) is a data augmentation technique commonly used in machine learning to address class imbalance problems. It is specifically designed to address the issue of having imbalanced datasets, where the number of samples in one class is much lower than the number of samples in another class.

The basic idea behind SMOTE is to generate synthetic samples for the minority class by interpolating between existing samples.The algorithm then repeats this process for as many new samples as are required to balance the dataset. The SMOTE algorithm is typically applied as a preprocessing step before training a machine learning model. By generating new synthetic samples, it increases the number of samples in the minority class, making it easier for the model to learn to distinguish between the two classes.

3.2 CNN Architecture

3.2.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of neural network that are commonly used for image recognition and analysis tasks. CNNs consist of multiple layers, each of which performs a specific type of processing on the input image. In this section, we will provide an overview of the key components of CNNs.

Convolutional Layers The first layer of a CNN is typically a convolutional layer. Convolution is a mathematical operation that is used to extract features from an image. The idea is to take a small square of the input image (called the kernel or filter), and slide it over the entire image, computing the dot product of the kernel and the portion of the image under the kernel at each location. The result is a new image, called a feature map, which highlights regions of the input image that are similar to the kernel.

Formally, let X be the input image, K be the kernel, and Y be the feature map. The convolution operation can be defined as:

$$Y_{i,j} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{i+m,j+n} K_{m,n} \tag{7}$$

where M and N are the dimensions of the kernel. The feature map Y is typically passed through a nonlinear activation function, such as the Rectified Linear Unit (ReLU), to introduce nonlinearity into the network.

Pooling Layers After each convolutional layer, it is common to include a pooling layer. Pooling is a down-sampling operation that reduces the dimensionality of the feature maps. The most common type of pooling is max pooling, which takes the maximum value of each $p \times p$ region of the feature map, where p is the pooling size.

Given a $p \times p$ pooling window, we slide this window over the output feature map, where each element of the window is indexed by (m, n). The pooling operation takes the maximum value of each window and outputs a new feature map where the size is reduced by a factor of p. Mathematically, this operation can be represented as follows:

For a window centered at position (i, j) in the output feature map, the corresponding element in the pooled feature map Y is obtained by computing:

$$Y_{i,j} = \max_{m=0}^{p-1} \max_{n=0}^{p-1} X_{pi+m,pj+n} \tag{8}$$

where X is the input feature map.

This equation can be interpreted as finding the maximum activation within each non-overlapping $p \times p$ region of the input feature map. This has the effect of down-sampling the input feature map, while preserving the most important information in terms of the maximum activation values.

Fully-Connected Layers After the convolutional and pooling layers, the feature maps are typically flattened into a 1D vector and passed through one or more fully-connected layers. A fully-connected layer is a standard neural network layer, in which each neuron is connected to every neuron in the previous layer.

Formally, let X be the input vector, W be the weight matrix, b be the bias vector, and Y be the output vector. Then the fully-connected layer can be defined as:

$$Y = f(WX + b) \tag{9}$$

where f is the activation function.

Our CNN Architecture For our CNN we settled on the following architecture. The architecture consists of several layers of convolutional and pooling operations followed by some fully connected layers.

The input to the CNN is a spectrogram of audio data with a shape of (128, 259, 1), where 128 is the number of frequency bins, 259 is the number of time steps, and 1 is the number of channels.

The architecture of the CNN consists of five convolutional layers followed by max-pooling layers to reduce the dimensionality of the output. The first layer has 32 filters and a kernel size of (5, 5), and the subsequent layers have 64, 128, 256, and 512 filters, respectively, with the same kernel size. The ReLU activation function is used for each convolutional layer to add non-linearity to the output. Each convolutional layer is followed by a max-pooling layer with a pool size of (2, 2) to down-sample the output.

After the convolutional layers, a Flatten layer is added to flatten the output, followed by two fully connected (Dense) layers. The first Dense layer has 256 units with the ReLU activation function, and the second Dense layer has a single unit with the sigmoid activation function for binary classification.

The model is trained on the training set for 150 epochs with a batch size of 32, and the validation data is used to monitor the model's performance. Early stopping is applied with a patience of 30 to prevent overfitting. The model is then evaluated on the testing set to assess its performance.

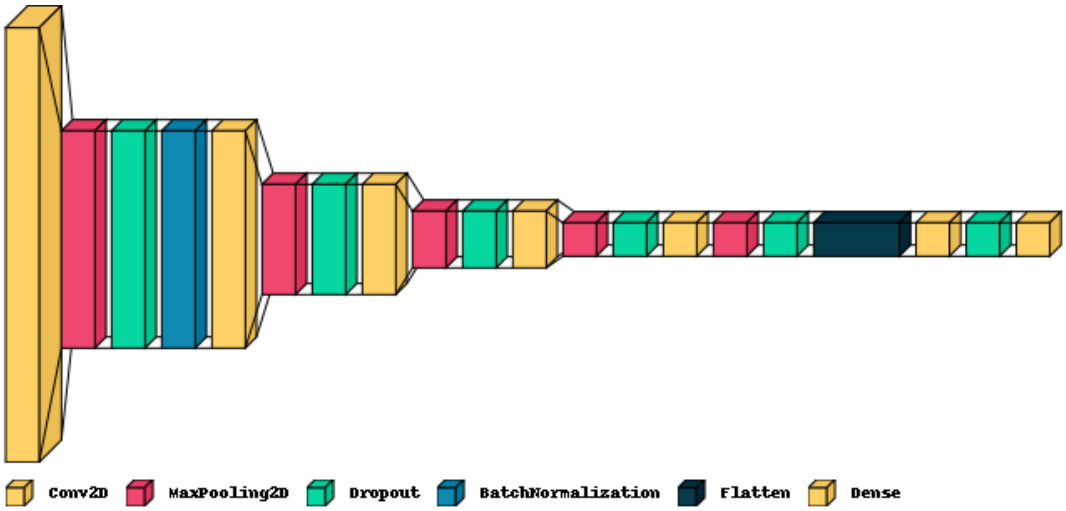

Figure 16: CNN Architecture of our Models

3.3 Methodology

We had several initial ideas on how to approach the task, and there were several important questions we needed to address. These questions included determining the appropriate form in which to observe audio files for training our models, selecting the suitable model architecture, handling the classification of 11 labels, and deciding whether to work with monophonic or polyphonic data while considering background noise.

To streamline our approach, we made several critical decisions early on, without extensive experimentation. We opted to utilize imaging techniques to represent audio files and combine them with neural networks. Additionally, we chose to create a separate model for each instrument, allowing each classification to indicate the presence or absence of a specific instrument in the soundtrack. The progress of the methods used will be explained individually for each modeling element, followed by an overview of the entire analysis history.

3.3.1 Prediction process

One immediate challenge we encountered was the variation in duration between the audio files in the given training and validation datasets. The training dataset exclusively consisted of monophonic audio files with a fixed duration of 3 seconds, whereas the validation dataset primarily contained polyphonic audio files ranging from 5 to 21 seconds.

To address this issue, we made a decision to represent audio files using either MFCC or mel-spectrogram images. However, the difference in duration posed a challenge. Since both MFCC and mel-spectrogram images have a time domain, we chose to train our models on 3-second audio files and split the audio files from the validation dataset into 3-second intervals. We then utilized the predictions for each interval to make a prediction for the entire audio file based on majority.

Overall, these decisions enabled us to tackle the challenge of variable audio file durations and paved the way for further analysis and model development.

3.3.2 Training datasets

The transition from using a monophonic dataset to incorporating polyphonic audio files was a decisive move based on thorough testing, as it provided a significant advantage. Initially, we created the *MIX* folder, which contained approximately 6,700 new audio files. These files were generated by combining two random polyphonic audio files from the original training dataset provided by the organizers. The combinations were created using the addition feature of the *librosa* library, ensuring that the distribution of instruments remained consistent. The resulting dataset consisted of around 13,000 audio files, with label 1 representing approximately 10-15% of the data, and label 0 representing the remainder, varying depending on the instrument.

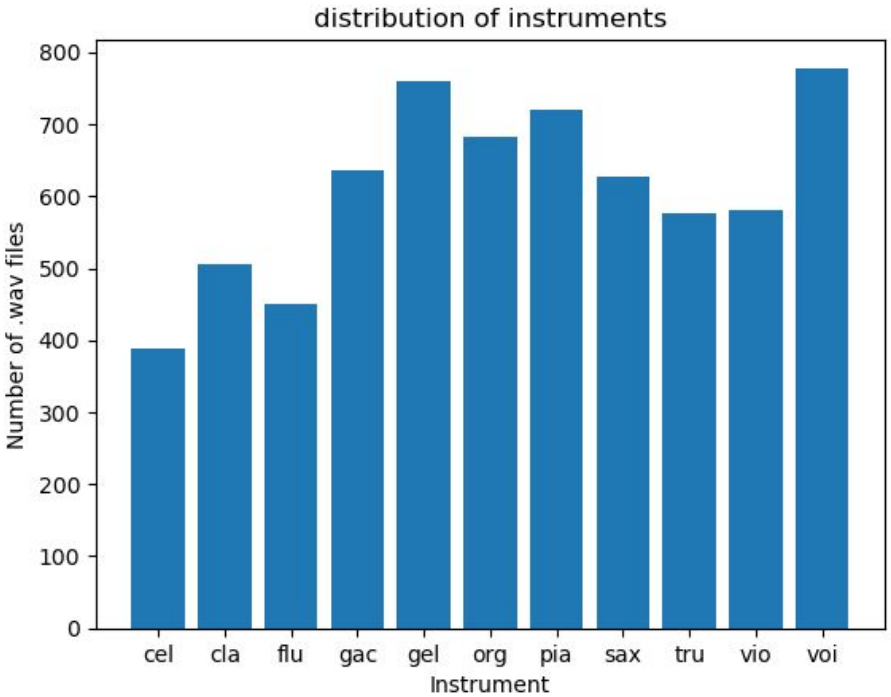

Figure 17: Distribution of Instruments in the Original IRMAS Training dataset

However, our models trained on MFCC images and mel spectrograms for this dataset encountered difficulty in learning patterns and achieved only 70-71% accuracy in recognizing the electric guitar on

validation datasets 1 and 2. To address this, we focused on class balancing, particularly for the electric guitar classification accuracy on validation dataset 2. Initially, we balanced the classes by creating the *MIX2* folder, which consisted of approximately 10,000 samples. These samples were created by randomly combining two monophonic audio files, with most having label 1 and a few having label 0. By merging the two datasets and selecting a subset randomly, we improved our accuracy to 75-76%, resulting in more correct positive predictions along with a slightly increased number of false positives.

Subsequently, the *MIX2* folder was replaced by the *MIX2_InstrumentShortcut* folder. This new folder followed a similar approach, but it also included audio files generated from three and four monophonic audio files, instead of just two as before. Although this modification did not have a significant impact on the results, we continued working with this dataset for our final models.

Another dataset approach we experimented with involved using the validation datasets for training. We divided these datasets into 3-second intervals, as described in the technical documentation of the project. If an instrument was labeled as 1 in an audio file, we assumed that it was labeled as 1 in all the 3-second intervals created from it. We used validation sets 1 and 3 to create this dataset, resulting in approximately 11,000 3-second audio files. We kept validation set 2 separate to report our results. We made some new data using data augmentation techniques appropriate for audio images to balance the classes in the dataset, so that 40-50% of the data is labeled as 1. Although using these datasets for training only slightly improved performance on validation set 2, we concluded that this approach was more suitable, considering its similarity to the actual test dataset compared to the previously created training datasets. It is important to note that this dataset does not cover every possible combination of instruments. Despite achieving slightly better performance on validation dataset 2 with this training dataset, our model exhibited great learning on validation datasets 1 and 3, achieving approximately 98% training accuracy in electric guitar classification. Therefore, we decided that using this dataset to train our final model would be the optimal choice. Really high training accuracy is somewhat expected, considering some of the training and validation samples are taken from the same song.

3.3.3 Audio file representation

Motivated by the insights gained from the first competition workshop and extensive research presented in published articles on Music Instrument Recognition, we promptly recognized the importance of selecting an appropriate representation for audio data. After careful consideration, mel spectrograms emerged as the most intriguing choice due to their demonstrated success in previous studies. To explore its efficacy, we initially employed a simple convolutional network and evaluated its performance on the *MIX2* dataset using mel spectrograms. However, we soon discovered that training with mel spectrograms incurred significant time costs. Despite enduring lengthy training sessions for a few tests, we failed to observe any substantial improvement compared to using MFCC images.

In comparison, when employing mel spectrograms with default settings, the resulting image resolution for each training sample was 128x259, whereas MFCC images, with default settings, had a resolution of 20x259. After conducting brief yet insightful experimentation, we opted to utilize 20 MFCC coefficients with the *hop_length=1024* parameter configuration to generate MFCC images with a resolution of 20x130. Notably, the default hop length value of 512 is commonly paired with 22.05kHz audio files, as observed in relevant literature. Given that our audio files have a sample rate of 44.1kHz, we chose a hop length value of 1024 to maintain consistent image characteristics and reduce dimensionality. This decision ensures the compatibility of our models with the 20x130 resolution MFCC images.

Although mel spectrograms often yield superior results in the papers we consulted [Ash23], the disparity in dimensionality allows us to work with a significantly smaller number of samples during training while

maintaining reasonable training times.

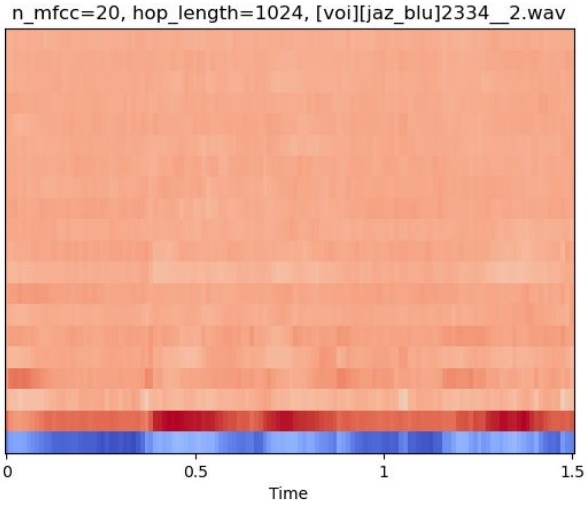

Figure 18: MFCCs of 44,1kHz Audio File with Hop Length of 1024

3.3.4 Model choice

Recognizing its strong performance in music instrument recognition tasks and its frequent mention in research papers, we opted to employ Convolutional Neural Networks (CNNs) for our project. However, as novices in the field of deep learning, we faced challenges in determining the optimal architecture. Each training attempt demanded several hours to complete, further complicating the process. To identify the most effective model, we conducted experiments with varying numbers of convolutional layers, diverse layer sizes, and different kernel sizes. Additionally, we explored the impact of l2 regularization layers, dropout layers, and LSTM layers based on a recent research paper [Ash23][Yun17]. Despite our attempts to optimize the model through adjustments to the layers and training dataset sizes, we did not observe significant improvements in our results. We also compared the Adam and SGD optimizers, but their selection did not yield substantial differences. Although we encountered incremental enhancements throughout our experimentation, we eventually reached a plateau, acknowledging the flaws of our approach. Ultimately, we settled on a CNN architecture featuring five convolutional layers, accompanied by max pooling and dropout layers. A single batch normalization, flatten, and dense layer were included, with a concluding dense layer. We intentionally kept our model relatively simple to ensure reasonable training times.

To prevent overtraining and ensure optimal model performance, we implemented the early stopping feature in our training process. This technique allowed us to monitor the validation loss metric during training and halt the process if no improvement was observed for a consecutive 30 epochs. By saving the model from the epoch with the best validation loss, we ensured that we retained the most optimal weights and architecture configuration. The early stopping would always occur in the first 100-120 epochs. For our final models, we used the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.005 and a momentum of 0.9.

3.3.5 Results and reports

In the final stages of our project, we encountered challenges in achieving successful classification results with many of our models. We observed variations in the predictability of different instruments, where some instruments were easier to classify compared to others. To ensure consistency, we primarily focused

on reporting results based on the validation dataset 2, since the other two validation datasets were used for training for some models.

Due to the inherent class imbalance within our dataset, accuracy alone did not provide a comprehensive metric to report our results. Thus, we opted to present confusion matrices for our later models, which offer a detailed overview of the classification performance. It is important to note that some of these models may not represent our final selected models, as they were utilized for research purposes.

Throughout our research, we observed that the models' ability to generalize varied significantly depending on the instrument being recognized. Notably, voice, electric guitar, piano, and acoustic guitar proved to be relatively easier to classify compared to other instruments. As such, we present below the confusion matrices for some of the models developed during our research.

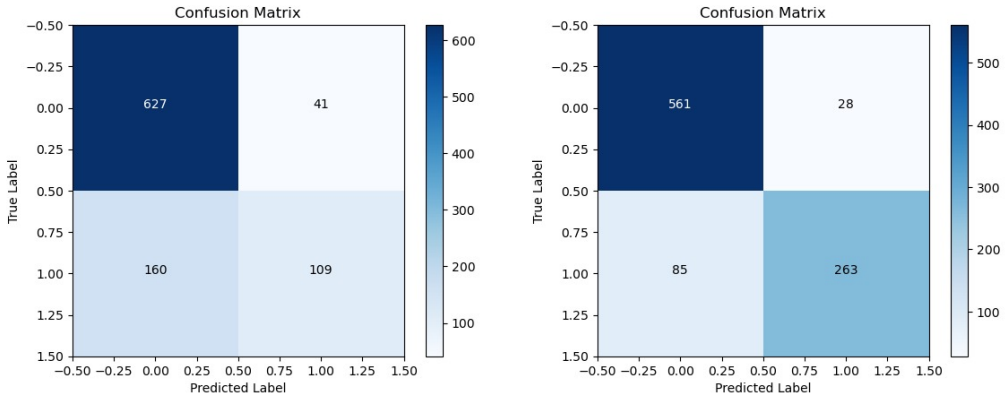

Figure 19: Confusion Matrices for Piano and Voice Recognition Models

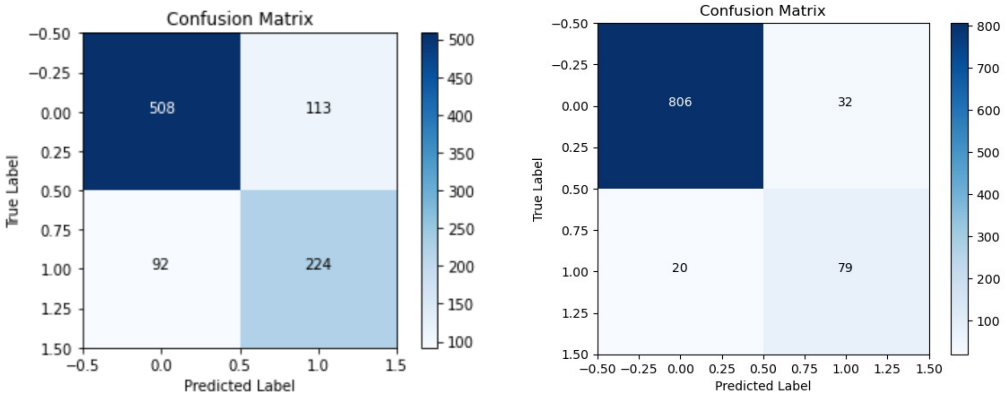

Figure 20: Confusion Matrices for Electric Guitar and Flute Recognition Models

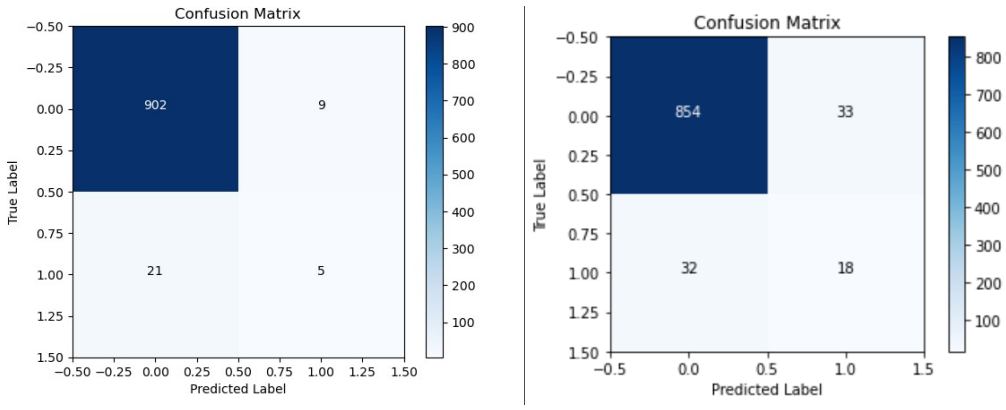

Figure 21: Confusion Matrices for Clarinet and Saxophone Recognition Models

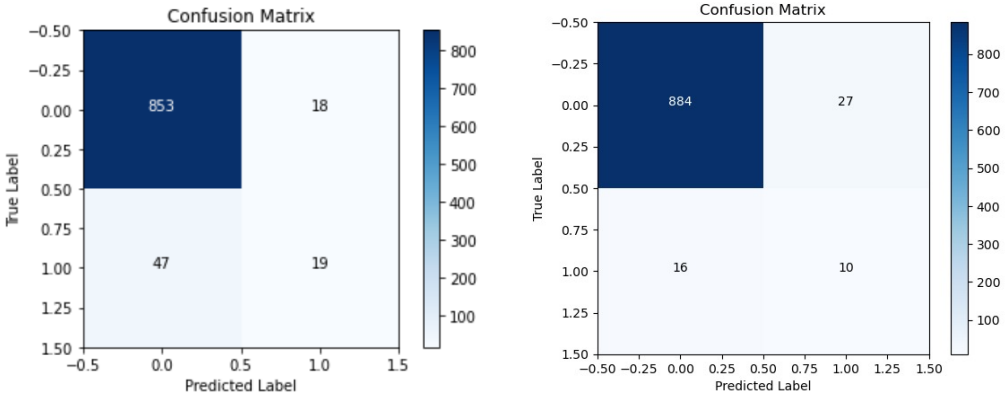

Figure 22: Confusion Matrices for Flute and Trumpet Recognition Models

The models demonstrated impressive learning capabilities across all of our training datasets, particularly those derived from the validation audio files. They consistently achieved near-perfect validation loss and accuracy during training. However, we encountered challenges in their generalization performance on the third validation set, indicating that the variations among audio files posed a significant obstacle for training our models using this approach.

To provide a visual representation of our training progress, we have included images below showcasing the performance of some of our models.

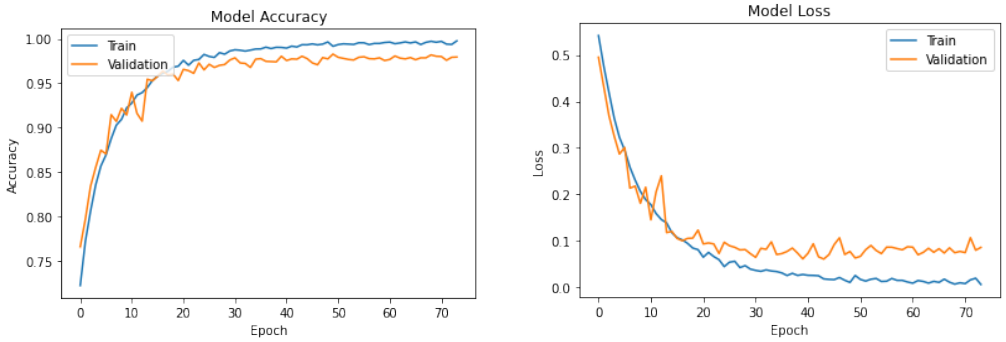

Figure 23: Training Progress for Electric Guitar Recognition Using Dataset Created from Validation Files

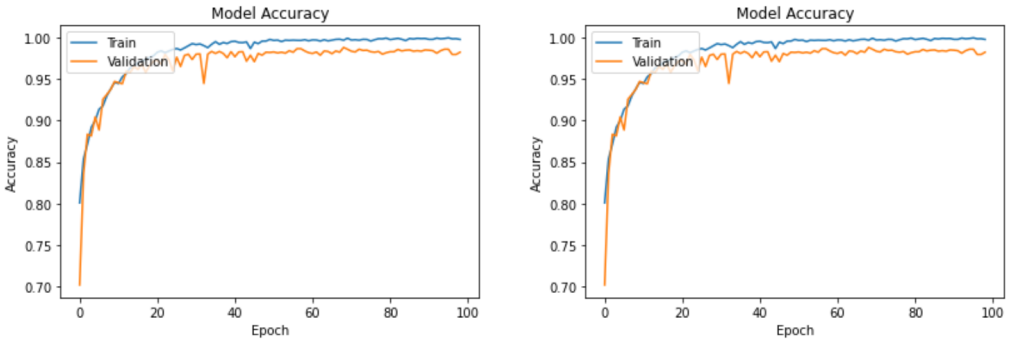

Figure 24: Training Progress for Saxophone Recognition Using Dataset Created from Validation Files

The images below show the training progress of our models trained on the *MIX2_InstrumentShortcut* datasets.

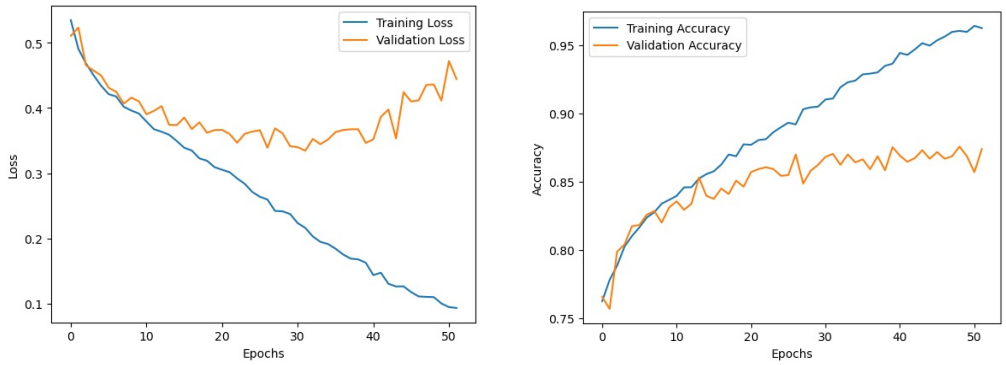

Figure 25: Training Progress for Trumpet Recognition Using *MIX2_tru* Dataset

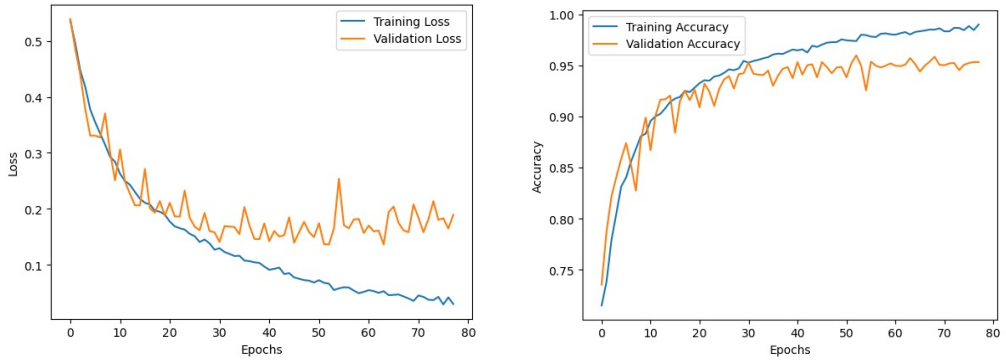

Figure 26: Training Progress for Voice Recognition Using *MIX2_voi* Dataset

We ensured to use early stopping feature for each model, so that the model with the best validation loss until the stopping is saved.

Final Prediction For the final prediction we then segment the file accordingly into 3 second intervals. For each of the intervals and each of the instruments we then predict if they are part of the intervals or not. Then all of the intervals vote either 0 or 1, depending on the respective confidence of the model for the interval. For different models we vote with 1, if the confidence is higher than some threshold. Here are the thresholds for the different instruments.

Instrument	**Confidence Threshold**
cel	0.9
cla	0.7
gac	0.35
pia	0.3
tru	0.6
vio	0.5
voi	0.25
flu	0.7
org	0.8
gel	0.5
sax	0.5

These thresholds can be regarded as hyperparameters of our final model. We decided on optimising these hyperparameters manually through educated guesses.

4 Conclusions and Future Work

Throughout the course of this project, we experimented with numerous different approaches and methods, drawing inspiration from recent publications in the field, as well as exploring our own novel ideas. Despite our thoroughness and dedication, we were unable to achieve the desired results. Our attempts to modify various aspects of our approach resulted in stagnant prediction performance, leading us to suspect that MFCC images may not be the optimal solution for music instrument recognition tasks.

While MFCCs exhibited promising results in recognizing certain instruments, they showed limitations in their recognition capabilities for others. Given that MFCCs, as image representations, possess limited information and can yield similar values for different instruments, particularly in polyphonic audio files, we acknowledge their inherent constraints. We chose to use MFCCs due to computational limitations and time constraints. Furthermore, we reduced their dimensionality, believing that investing computational power in more diverse and extensive training datasets would yield better results.

Considering the wealth of options available, we contemplated the idea of developing prediction models based on alternative audio processing techniques such as mel-spectrograms and spectral statistics. We recognized that relying on a single representation alone may not capture the comprehensive information required for complex polyphonic audio files. Additionally, there are likely numerous other audio processing methods that could prove valuable for addressing such challenges.

The optimization of neural networks also presents ample room for exploration. Due to the time-consuming nature of training, we were only able to explore a limited number of configurations for our CNN network, and thus, did not have the opportunity to extensively optimize the hyperparameters or investigate alternative network architectures. Recent publications have also highlighted the potential of transfer learning by utilizing pre-trained models trained on millions of image classification samples and training models in stages - from the simpler to the more complicated ones [Sze22].

Despite the challenges encountered, we are immensely gratified by the invaluable knowledge and experience gained throughout this project. The topic of this year's competition, **Lumen Data Science 2023**, proved to be captivating and thought-provoking, and we are sincerely grateful for the opportunity to participate.

5 References

[Ash23] Moshin Ashraf. *A Hybrid CNN and RNN Variant Model for Music Classification*. https://www.mdpi.com/2076-3417/13/3/1476, 2023.

[Bla22] Maciej Blaszke. *Musical Instrument Identification Using Deep Learning Approach*. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9025072/, 2022.

[Dos21] Ketan Doshi. *Audio Deep Learning Made Simple (Part 1, 2, 3): State-of-the-Art Techniques*. https://towardsdatascience.com/audio-deep-learning-made-simple-part-1-state-of-the-art-techniques-da1d3dff2504, 2021.

[Gur19] Siddharth Gururani. *An Attention Mechanism for Musical Instrument Recognition*. https://arxiv.org/abs/1907.04294, 2019.

[Han16] Yoonchang Han. *Deep Convolutional Neural Networks for Predominant Instrument Recognition in Polyphonic Music*. https://arxiv.org/pdf/1605.09507.pdf, 2016.

[Kaw19] Nadim Kawwa. *Can We Guess Musical Instruments With Machine Learning?* https://medium.com/@nadimkawwa/can-we-guess-musical-instruments-with-machine-learning-afc8790590b8, 2019.

[Li15] Peter Li. *Automatic Instrument Recognition in Polyphonic Music Using Convolutional Neural Networks*. https://arxiv.org/abs/1511.05520, 2015.

[Sze22] Dominika Szeliga. *Musical Instrument Recognition with a Convolutional Neural Network and Staged Training*. https://www.sciencedirect.com/science/article/pii/S1877050922011966, 2022.

[Tae21] Michael Taenzer. *Deep Learning-Based Music Instrument Recognition: Exploring Learned Feature Representations*. https://cmmr2021.github.io/proceedings/pdffiles/cmmr2021_24.pdf, 2021.

[Yun17] Mingqing Yun. *Deep Learning for Musical Instrument Recognition*. https://www.semanticscholar.org/paper/DEEP-LEARNING-FOR-MUSICAL-INSTRUMENT-RECOGNITION-Yun/ad4201d862fd0952d8028697d505ad7697337292, 2017.