Final Report Team 11:

Music Classification Neural Network

Ahmed Rosanally (1003204999)
Tianxu An (1003088393)
Tiffany Yau (1004191867)
Reynold Chan (1002954797)

Word Count: 2462

Table of Contents

Introduction

Motivation

One of the five senses that humans have is the gift of hearing. With the gift of hearing , one can enjoy music of all different genres. With a cacophonic melody that is composed of many instruments, one might find themselves intrigued in the instruments themselves. Thus, the goal of this project is to identify instruments present in a mixed audio file. Refer to Figure 1 for an illustration of this project.

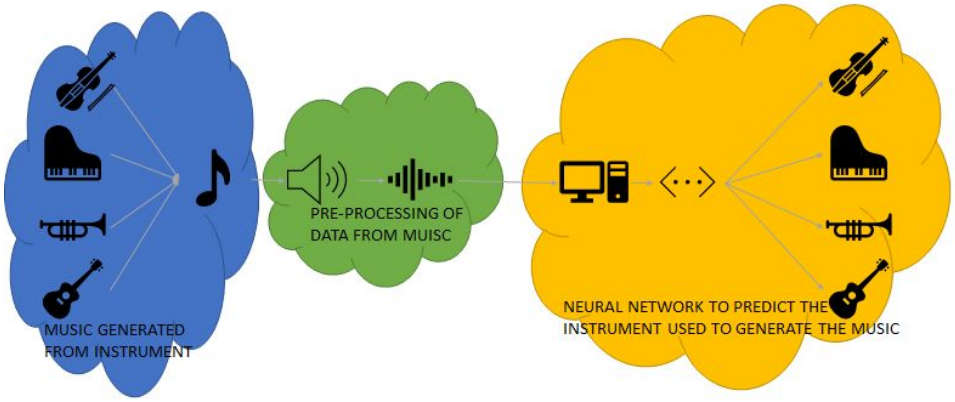


Figure 1: Project motivation

Why Use Machine Learning?

Algorithmic programming methods are insufficient to address this instrument classification task due to the complexity in instrumental audio signals. The problem is hard since the algorithm needs to distinguish multiple instruments in a mixed setting. Machine learning is necessary to identify slight patterns in instrument classes and differentiate between them.

Project Structure

The project was divided into two key stages. This process is outlined in Figure 2. **Stage 1** involves building a classifier to determine the single instrument that is playing in an audio file. All samples in the single instrument dataset consist of only one instrument playing at a time. **Stage 2** expands on the idea of Stage 1 and involves building a neural network model that can identify which instrument(s) is/are present in a multi-instrument sample. These multi-instrument samples are generated based on the single instrument dataset by overlapping those audio samples. Refer to Figures 2 to 4 for an illustration of the stages.

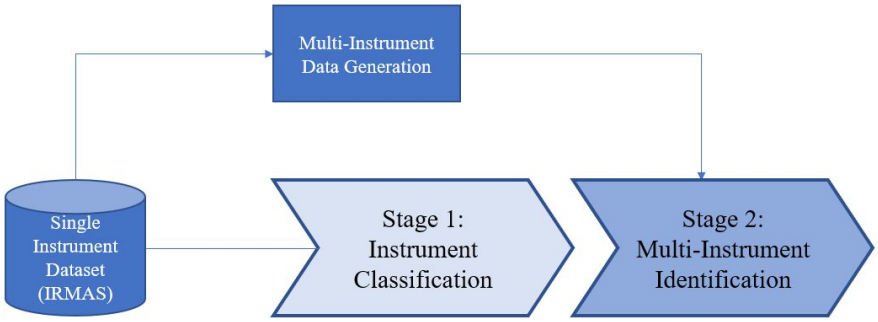


Figure 2: Overview of the project stages.

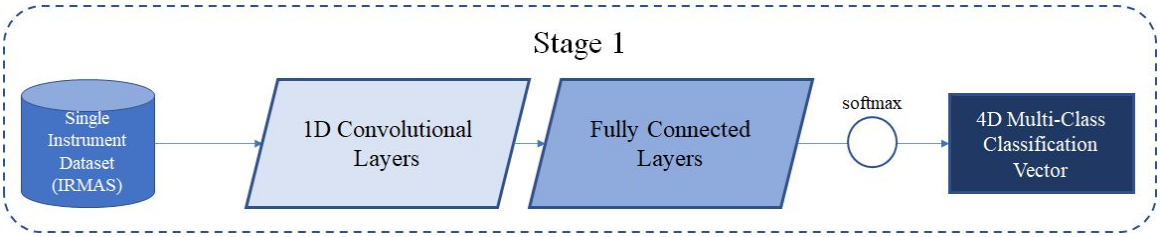


Figure 3: Detailed framework of Stage 1

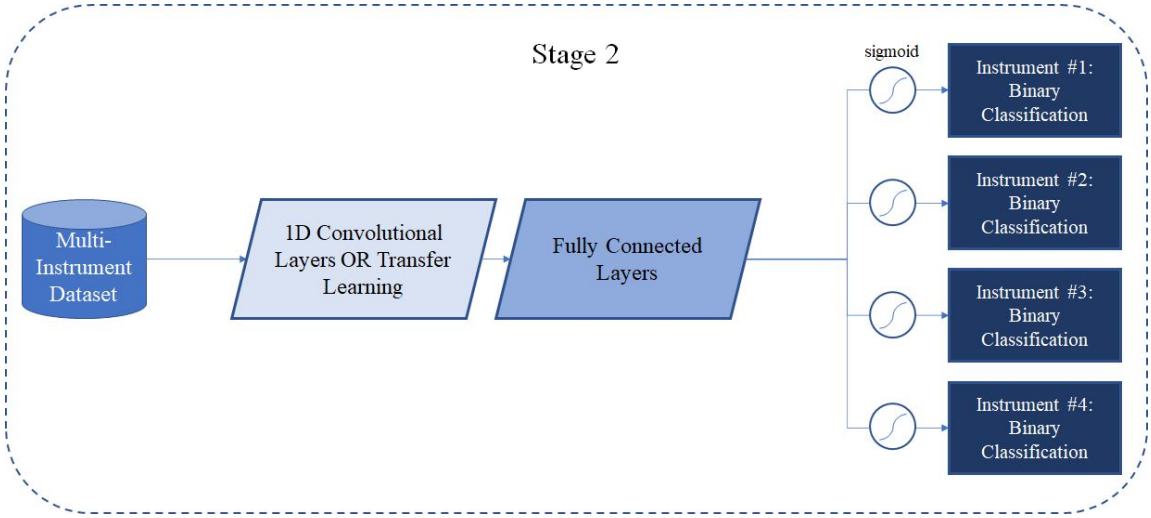


Figure 4: Detailed framework of Stage 2

The model will take multi-instrument audio samples as input, and generate four outputs, each identifying whether a particular instrument is present in the sample. Stage 2 identifies which instruments are present in an audio sample containing multiple instruments. Transfer learning was explored to extract the features, which would then be flattened and passed into each of the four classifiers with a sigmoid activation.

Background & Related Work

There are two related projects found online. The first project is Audio AI [1]. It can detect and isolate vocals from music signals. The approach is to first build a Vocal Activity Detector (VAD). This was achieved by converting the audio signal into time-frequency graphs using Short-Time Fourier Transform, then splitting the graphs into equal-sized images, each with 25 STFT frames (~300 milliseconds). A CNN was used to classify each 25 frames as "containing vocal" or not (Figure 5).

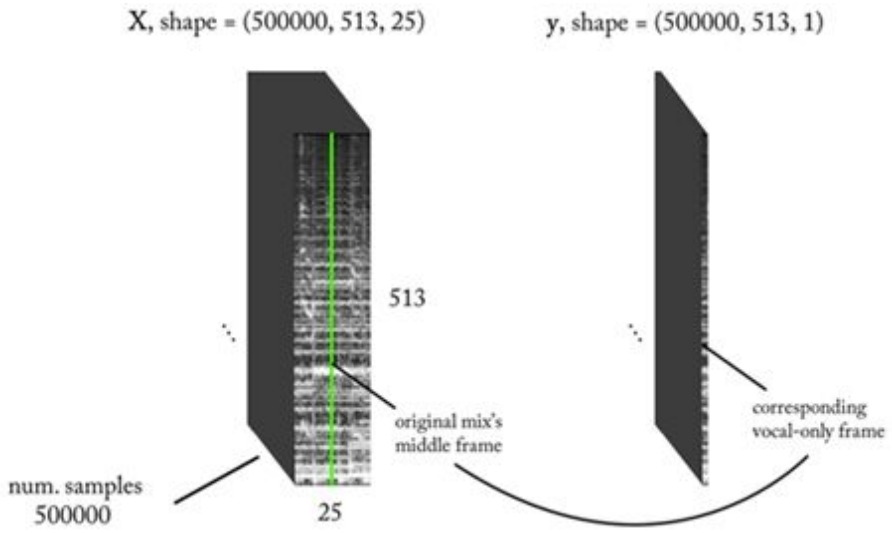


Figure 5 Input & Output of CNN [1]

Then the output from the CNN was passed to a regression network to learn a 513-dimensional bitmask vector indicating if the vocal has a predominant presence at each pixel (Figure 6).

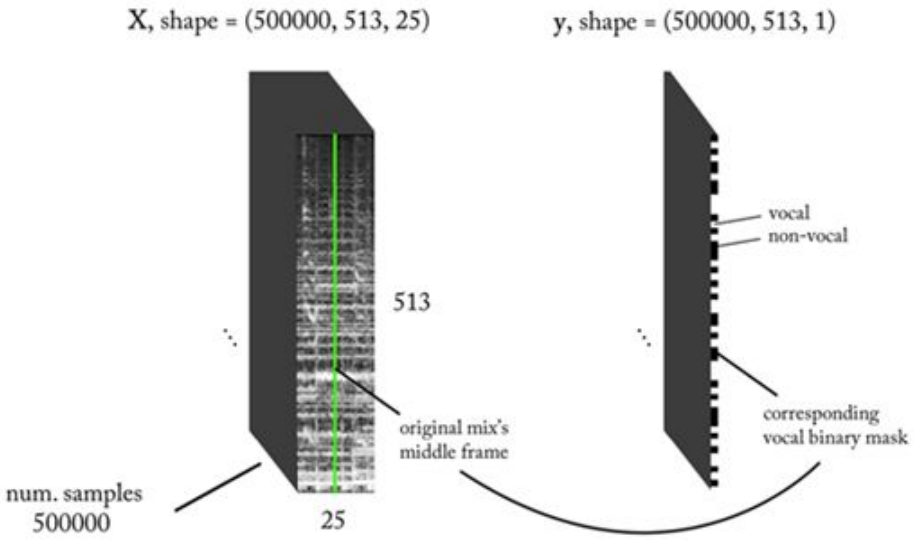


Figure 6: Input & Output of Regression Network [1]

Combining all bitmask vectors, the input STFT graph was converted to a bitmask image with 1 representing vocal presence and 0 otherwise (Figure 7). The bitmask image was then applied to the source audio to filter out vocals.

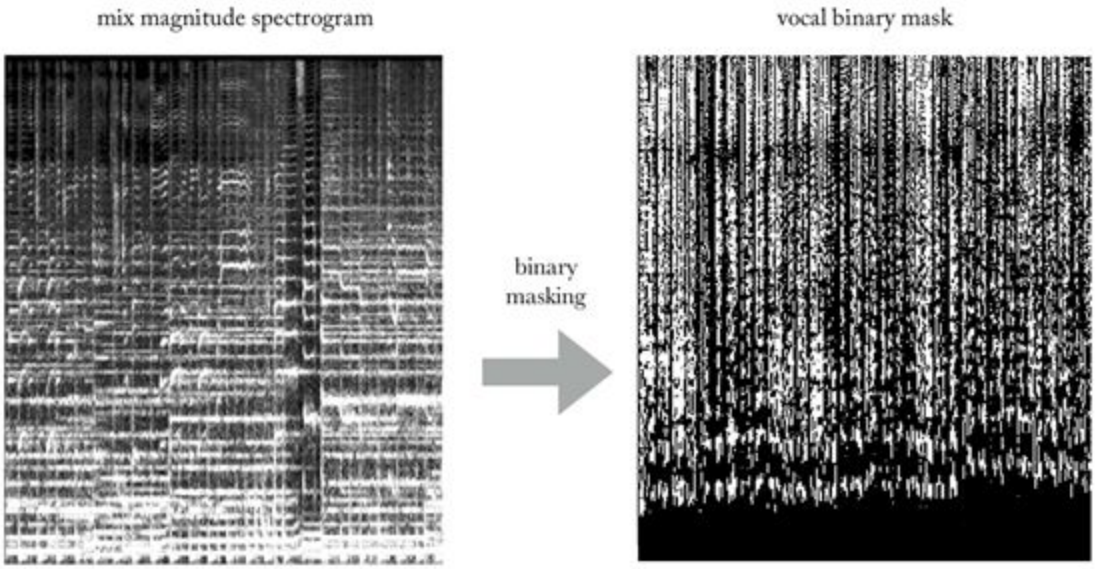


Figure 7: Converting STFT image to binary bitmask [1]

The second project is PixelPlayer [2]. It takes the audio-visual data as two separate inputs. Three networks were trained within the system, as discussed below:

1. *Video Analysis Network*: extracts visual features from video frames.

2. *Audio Analysis Network*: splits the input audio into multiple components based on their distinct sounds features.

3. *Audio Synthesizer Network*: associates visual features and audio features, then builds bit masks to separate different instrument audios.

These audio-visual inputs are parsed into the model for classification. The model uses a self-supervised learning model to separate mixed audio into specific components and spatially localize them in each pixel in the video (Figure 8).

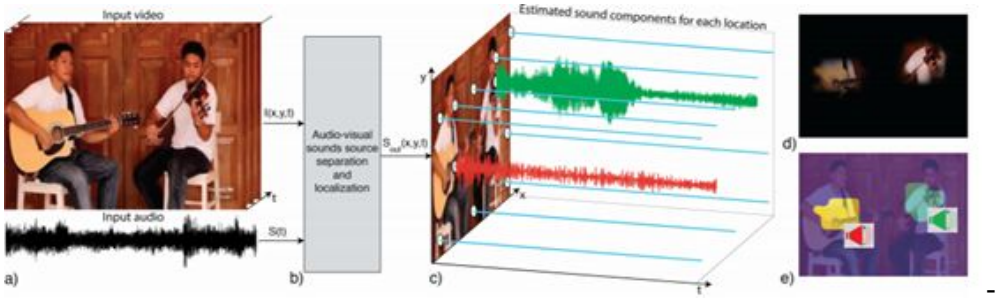


Figure 8: Introduction to PixelPlayer [2]

Data Processing

Raw Data

Four instrument classes were selected for classification: acoustic guitar, piano, trumpet and violin. The raw data for each .wav file is a tensor of size [2, 132299], with sample rate 44100 Hz. The examples of raw data are presented in Appendix A.

Table 1: Number of Samples in Raw Data

Class	Number of Samples
Acoustic Guitar	637
Piano	721
Trumpet	577
Violin	580

Data Balancing

Data balancing was done by adding augmented audio to the classes with fewer data. The augmented data was generated by reversing the signs of all audio data. An example of data augmentation is illustrated in Figures 9 to 11. The number of samples after data balancing is shown in Table 2, this data is split into [0.6, 0.2, 0.2] for training, validation and testing. Total data count for each class is 721. The class labels are randomized to make sure they are evenly distributed across three datasets, so that each class can be evenly represented in the three datasets.

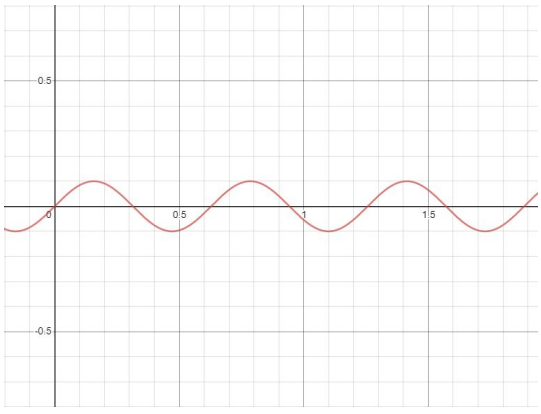


Figure 9: Original Wave Data

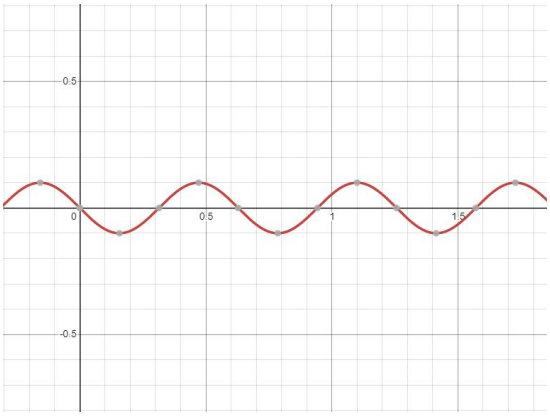


Figure 10: Augmented Wave Data

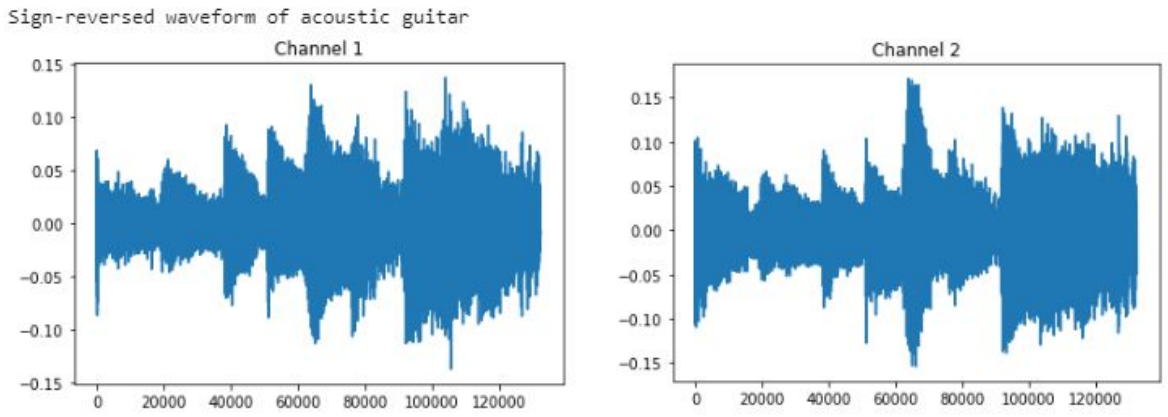


Figure 11: An Example of Sample Data of Augmented Acoustic Guitar Audio

Table 2: Number of Split Samples in Each Balanced Data

Class	Training Count	Validation Count	Testing Count
Acoustic Guitar	433	144	144
Piano	433	144	144
Trumpet	433	144	144
Violin	433	144	144

Data Normalization

The raw data are normalized in range [-0.5, 0.5] to unify audio magnitude. This can avoid loud instruments overshadowing others when generating multi-instrument audio. An example of data augmentation is illustrated in Figures 12 to 14.

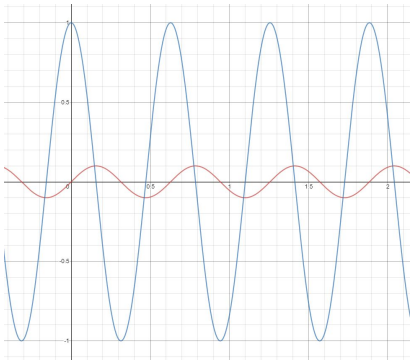


Figure 12: Original Wave Data

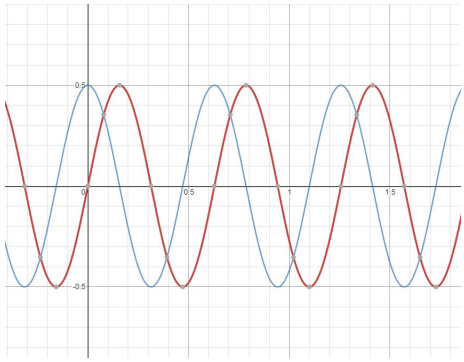


Figure 13: Normalized Wave Data

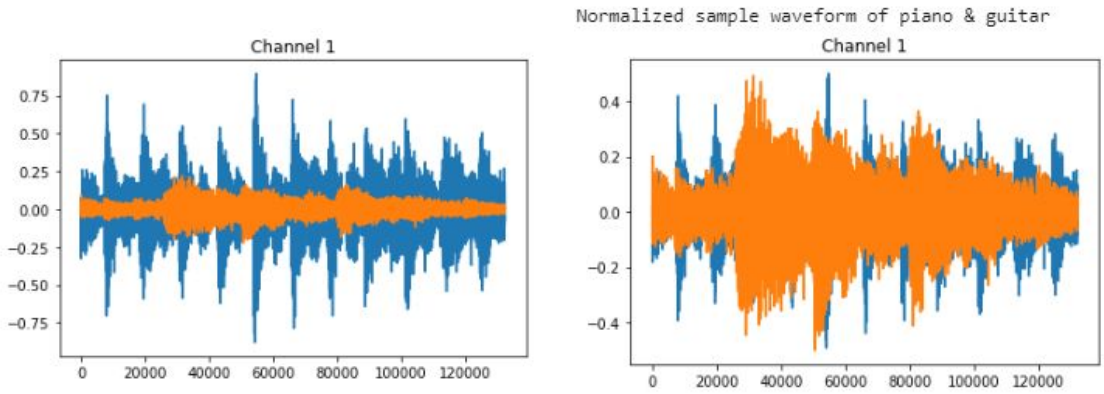


Figure 14: An Example of Sample Data of Normalized Piano & Guitar Audio

Generate Multi-instrument Audios for Stage 2 and Testing data

Multi-instrument audio files are generated by summing the tensors of different instrument audios and normalizing it. These new data will be used to both train and test new models in Stage 2. An example of data augmentation is illustrated in Figure 15 and Figure 17:

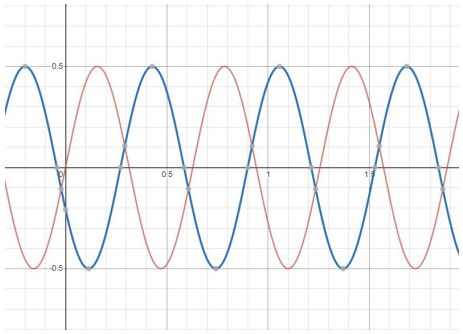


Figure 15: Two Normalized Waves

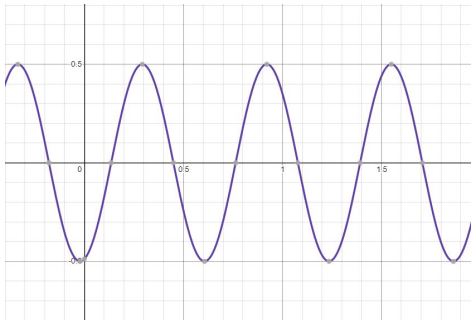


Figure 16: Summed Wave (Normalized)

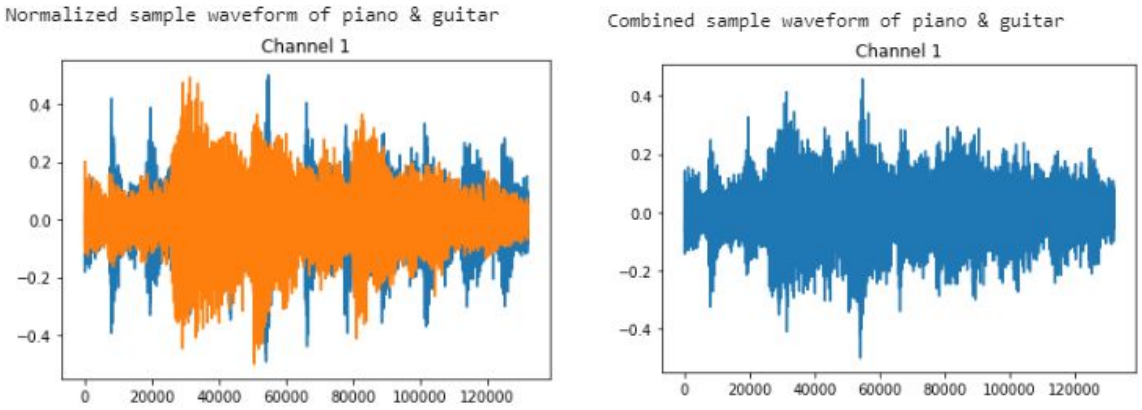


Figure 17: An Example of Combined Wave of Piano Audio and Guitar Audio

Every two classes will be combined to form six multi-instrument labels. For each label, the total number of data will also be 721 to match the number of samples in each single-instrument class. Similar to Stage 1, these data will be split into [0.6, 0.2, 0.2] for training, validation and testing.

Baseline Model

The baseline model was built off random forest trees in Python Scikit-learn. [3] The reasoning for a random forest classification is because if one tries to classify instruments, one might look at many features to arrive at a decision on what type of instrument it would be. A random forest would thus be best suited as a baseline model to compare to the neural network. Refer to Figure 18 for how random forest classification works. Sci-kit Learn requires the tuning of the number of estimators which represents the decision trees. This parameter is easy to tune.

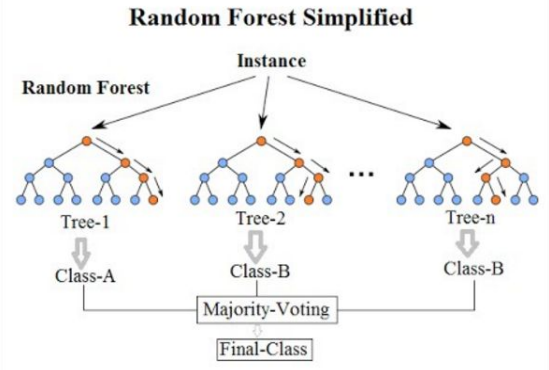


Figure 18: Random Forest Classifier Diagram [4]

Since a random forest requires the information to be presented in the tabulated form, the waveforms from channel 1 and channel 2 were concatenated. The validation accuracy is shown below in Table C.1 in Appendix C. The amount of trees do not seem to guarantee a large increase in accuracy.

For Stage 2, the baseline model was still parsed with the same information but converted the encoding of the labels from a multilabel encoding to a bit encoding. Refer to Table C.2 in Appendix C for the compiled results of the baseline in Stage 2 and its performance.

Architecture

In **Stage 1**, the model output is a four-dimensional vector representing the probability across the four instrument classes. The training labels are one-hot encoding vectors identifying the true class the sample belongs to. The cross entropy loss was used during training, which also accounts for the softmax activation at the output.

In **Stage 2**, the model identifies the instruments present in data samples. The output of the model is a 4D multi-label classification vector, where each value represents the probability of the corresponding class being positive. The target vector is a similar "two-hot" encoding vector indicating two instruments in the sample (e.g. [1 0 0 1] represents the instrument at index 0 and 3 are present in the sample). In this stage, the *multi-label soft margin loss* is used to compute the loss of multi-label output. It accounts for the sigmoid activations at the output.

The training process for both stages is very similar. Coincidentally, it was found that the optimal hyperparameters and architecture for Stage 2 are the same as those of Stage 1. The best architecture used for this consists of six 1D convolutional layers and one fully connected layer. Elements including dropout layers, batch normalization, adding more convolutional layers, and reducing parameters in the linear layers are added as well.

Figure 19 is a visual representation of the best CNN architecture. The audio input consists of two channels with 132,299 frames. Max pooling layers are used to reduce dimensionality for the fully connected layer. Various kernel sizes are used to capture audio features of different frequencies. A stride of one and no padding is used throughout.

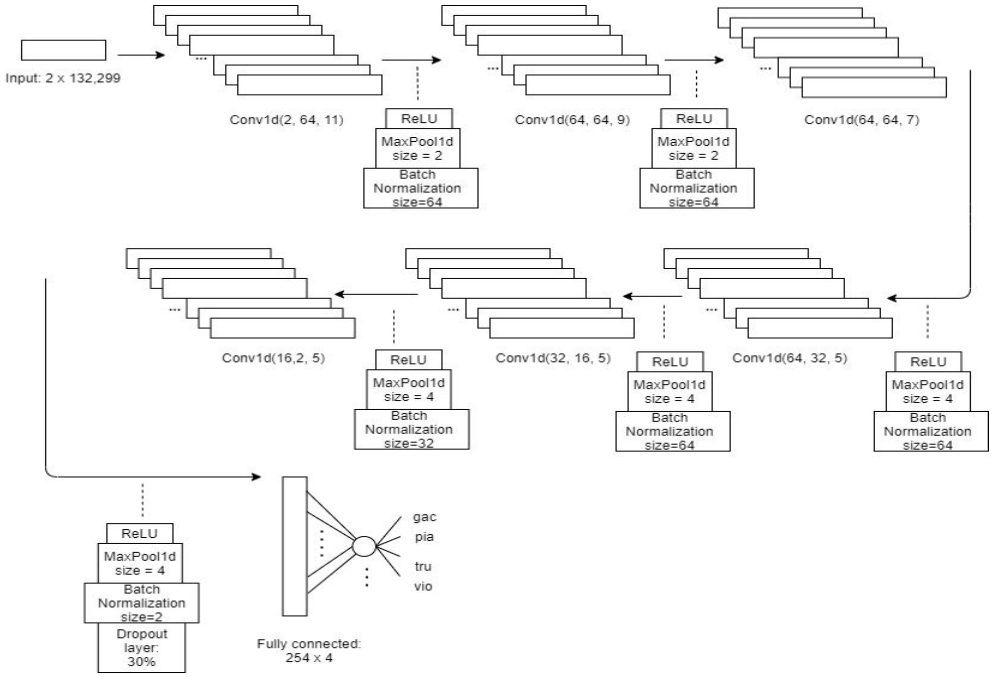


Figure 19: Final 1D CNN Architecture for Both Stages In Hyperparameter Tuning.

In addition to training a CNN from scratch, a transfer learning model was also applied in **Stage 2**. Since no existing transfer learning model was found to distinguish instruments, audio features are extracted using the weights of the network found in Stage 1. After features were extracted from the CNN model from Stage 1, the ANN shown in Figure 20 was used for classification.

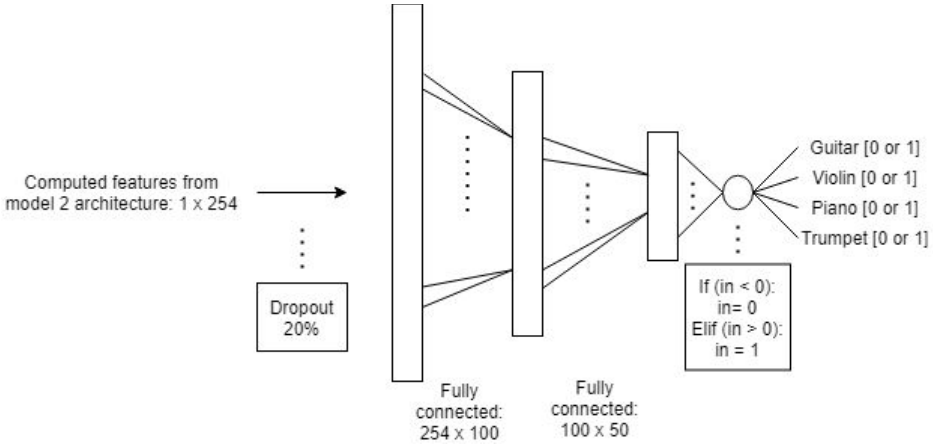


Figure 20: Transfer Learning ANN classifier portion

In terms of computation time and memory required, the transfer learning model has a clear edge, as shown in Table 3. Transfer learning would allow us to train less than half the number of parameters. Both the computation time per epoch as well as overall computation time are significantly less with transfer learning. From these metrics, it is clear that there is a huge potential with this concept. This could be further explored as the focus of future projects.

Table 3: Summary of Stage 2 CNN and Transfer Learning Comparison

	CNN	**Transfer Learning**
# Learnable Parameters	81166	30754
Computation Time per Epoch (seconds)	93	0.33
Overall Average Computation Time	~30 mins	3.3 sec

Results and Discussion

The models for both stages performed well and significantly outperformed the baseline, as outlined in Table 4. From the Stage 2 results it is clear that the transfer learning validation accuracy is comparable with that of the initial CNN. However, the test accuracy on the holdout dataset differs by 5%. This reflects some limitations in the transfer learning model's ability to generalize to new data. Since the model with the higher validation accuracy in Stage 2 is the 1D CNN, all further analysis in this section will pertain to the CNN model.

Table 4: Summary of Validation and Test Results for All Stages. Note: Training curves and training and validation accuracy and loss values are included in Appendix B and C.

	Validation Accuracy	**Test Accuracy**
Stage 1: 1D CNN	72.57%	70.66%
Stage 1: Baseline Model Accuracy	43.3%	40.3%
Stage 2: 1D CNN	69.05%	69.74%
Stage 2: Transfer Learning	68.21%	64.16%
Stage 2: Baseline Model Accuracy	38.7%	36.7%

Stage 1 Results and Analysis

Analyzing the confusion matrix (Table 5) for the **first Stage**, the trumpet class stands out to be the best, with the highest true positive rate (88.55%) along with lowest false positive (4.49%) rate. This makes sense because the trumpet is the only brass instrument. However, the guitar class, which has the second highest true positive rate (74.34%), also has the highest false positive rate (20.28%). Additionally, the piano and violin classes account for almost all of the guitar's false positives and negatives. Considering the numerous styles of playing the guitar, piano, and violin, these results are reasonable. Also, these three instruments are stringed, creating inherent similarities between their audio data. This idea of similarity between these instruments is further evidenced by the piano's false negatives, which are mostly misclassified as guitar samples.

Table 5: Confusion Matrix for Stage 1 Test Data

		Predicted Label			
True Label		**Guitar**	**Piano**	**Trumpet**	**Violin**
	Guitar	113	23	1	15
	Piano	57	81	7	3
	Trumpet	4	8	116	3
	Violin	25	11	12	97

Stage 2 Results and Analysis

With the **second Stage**, the test accuracy for individual binary classifications was 69.74%. However, looking at the confusion matrix (Table 6), there are some interesting results when it came to classifying all four binary labels for each sample. Apart from the fact that around half the multi-label predictions classified an incorrect number of instruments (as shown in the rightmost column), a pattern that segregates the multi-label classes into two parts -- a group for the samples containing the guitar and another for the samples without the guitar. Within each subgroup of multi-label classes, there are high false positive and negative rates. It appears that the network has trouble differentiating between instrument pairs within the same subgroup. Additionally, misclassifications outside of those subgroups are rare in comparison. This interesting observation suggests that it is much easier to detect the presence of the guitar than it is to detect the presence of the piano, violin, or trumpet, when in a mixed setting. Perhaps in the future, experimentation could be done with more instruments or other instrument combinations to see whether this pattern occurs with other instruments.

Table 6: Confusion Matrix for Stage 2 Test Data

		Predicted Labels						
True Labels		**Guitar + Piano**	**Guitar + Trumpet**	**Guitar + Violin**	**Piano + Trumpet**	**Piano + Violin**	**Trumpet + Violin**	**Other**
	Guitar + Piano	21	35	32	2	1	1	65
	Guitar + Trumpet	20	25	19	2	0	0	75
	Guitar + Violin	11	13	70	0	3	1	44
	Piano + Trumpet	4	2	1	53	13	5	60
	Piano + Violin	1	4	11	25	13	15	70
	Trumpet + Violin	7	3	18	11	18	10	81
	Other	0	0	0	0	0	0	0

Model Performance on Novel Data - Demo

The limitations of the dataset used to train the model could have a negative impact on how well the model performs on real life data. For instance, the dataset in **Stage 2** is composed by combining audio samples from individual instrument recordings in the IRMAS dataset. This often results in the different instruments in one sample to be playing different pieces of music. Additionally, the music is normalized so that the instruments have the same volume, which is not always the case in live music. To ensure our model performs as expected on real-world data, audio files taken from YouTube are used to test the model.

In **Stage 1**, a solo trumpet recording was taken from YouTube and split into multiple samples [5]. The trumpet class was chosen for this demo to emphasize the difference in performance of this distinct instrument versus the other three similar instruments.

```
Prediction for sample 1: ['trumpet']
Prediction for sample 2: ['trumpet']
Prediction for sample 3: ['trumpet']
Prediction for sample 4: ['trumpet']
Prediction for sample 5: ['trumpet']
Prediction for sample 6: ['trumpet']
Prediction for sample 7: ['trumpet']
Prediction for sample 8: ['trumpet']
Prediction for sample 9: ['trumpet']
Prediction for sample 10: ['trumpet']
Prediction for sample 11: ['trumpet']
Prediction for sample 12: ['trumpet']
Prediction for sample 13: ['trumpet']
Prediction for sample 14: ['trumpet']
Prediction for sample 15: ['trumpet']
Prediction for sample 16: ['trumpet']
Prediction for sample 17: ['trumpet']
Prediction for sample 18: ['trumpet']
Prediction for sample 19: ['trumpet']
Prediction for sample 20: ['trumpet']
Prediction for sample 21: ['trumpet']
Prediction for sample 22: ['trumpet']
Prediction for sample 23: ['trumpet']
Prediction for sample 24: ['trumpet']
Prediction for sample 25: ['trumpet']

Accuracy: 100.0% | Number of predictions for each class: tensor([ 0,  0, 25,  0])
```

The model exceeds expectations, providing 100% accuracy on this new data. This matches the analysis of Stage 1 and the relative performance of different classes shown in the confusion matrix.

In Stage 2, a recording containing violin and trumpet was taken from YouTube [6]. The music clip was split into 23 music samples and passed through the model. The following summarizes the *qualitative output*:

```
Prediction for sample 1: ['acoustic guitar', 'piano', 'trumpet']
Prediction for sample 2: ['piano', 'trumpet']
Prediction for sample 3: ['piano', 'trumpet']
Prediction for sample 4: ['acoustic guitar', 'violin']
Prediction for sample 5: ['trumpet', 'violin']
Prediction for sample 6: ['trumpet', 'violin']
Prediction for sample 7: ['acoustic guitar', 'piano']
Prediction for sample 8: ['piano', 'trumpet']
Prediction for sample 9: ['violin']
Prediction for sample 10: ['piano', 'violin']
Prediction for sample 11: ['piano', 'trumpet', 'violin']
Prediction for sample 12: ['piano', 'trumpet', 'violin']
Prediction for sample 13: ['trumpet', 'violin']
Prediction for sample 14: ['piano', 'trumpet', 'violin']
Prediction for sample 15: ['piano', 'trumpet']
Prediction for sample 16: ['trumpet', 'violin']
Prediction for sample 17: ['acoustic guitar', 'piano', 'trumpet']
Prediction for sample 18: ['piano', 'violin']
Prediction for sample 19: ['piano', 'violin']
Prediction for sample 20: ['piano', 'trumpet']
Prediction for sample 21: ['trumpet', 'violin']
Prediction for sample 22: ['piano', 'trumpet', 'violin']
Prediction for sample 23: ['acoustic guitar', 'piano', 'trumpet']
```

The fully correct outputs are highlighted in green. Partial marks were given to calculate the model accuracy. Quantitative performance on this data has 60.86% accuracy, which is expected considering the original dataset limitations.

```
Accuracy: 60.86956521739131%
Number of predictions for each class: tensor([gui: 5, pia: 16, tru: 17, vio: 14]
```

Out of the 23 samples, the model identified the presence of the trumpet in 17 and violin in 14 of the slices, with many piano and minimal guitar false positives. This further illustrates the Stage 2 model analysis.

Ethical Considerations

In this project, good machine learning practices were maintained by proactively preventing model bias. It was ensured that the data used to train, validate, and test the model is balanced across classes. This is necessary during training to give the model equal opportunity to learn and generalize for classes.

For the project, the model was trained on western instruments. However, training the model on only a set of certain instruments will only enable classification of those instruments. In particular, training on different genres of music would have an effect since different cultures would have different dominant instruments. During data processing, all data was normalized to be equal and then mixed them equally. However, in a live concert setting, this would not be the case. Therefore, if this model were to be further developed, it should be ensured that data collected is representative of a more diverse group of instruments and genres.

Additionally, an ethical issue could arise from a similar project that includes voice classification. Privacy concerns behind gathering people's voices would exist. Different languages have varying tones that would also need to be represented within the data.

Project Difficulty and Quality

The project was more difficult than expected. The initial plan was to distinguish the instruments within any live concert piece. However from the data processing it became evident that it would be hard to distinguish between instruments when they are mixed. Thus it was reduced to recognizing only two instruments for Stage 2.

Also, testing the model against new data proved to be challenging in Stage 2. Since most real data on the web does not balance each instrument's volume evenly, it was a challenge to get the same accuracy. Despite this, the model's accuracy on the demo data was 60.9%.

It was evident from the results that the models performed significantly better than the baseline model. It was decided to explore transfer learning in Stage 2 using Stage 1 weights due to the similarities between the two models. This reduced training time significantly and still ensured a similar quality.

Individual Contributions

Table 7: Ahmed's Contribution Summary (25% overall contribution)

Completed Tasks	**Incomplete Tasks**
Research existing literature on current machine learning models used for similar types of problems	Designing the RNN architecture as well as other promising model architectures to obtain better results. Evaluating performance of each model & finding trends in existing models to leverage in the project
Designing & implementing the 1D CNN network architectures for Stages 1 & 2	
Tuning model hyperparameters for Stages 1 & 2	Creating a front-end for users to interact with the single instrument classifier
Providing suggestions/explanations for why certain hyperparameters should be used to improve the original model architecture. For instance, adding more CNN layers, reducing the number of parameters in the single ANN layer & using overfitting techniques.	Using existing techniques for audio machine learning problems such as extracting frequency components by using the Fast Fourier Transform as a filter.

Table 8: Reynold's Contribution Summary (25% overall contribution)

Completed Tasks	**Incomplete Tasks**
Collected nsynth dataset as the backup dataset	Testing out different instrument classes in the IRMAS dataset to see if models for different classes would perform better in both stages
Baseline Model Training for Stage 1 and 2	
Final Report (50%)	
Research into existing literature	

Table 9: Tianxu's Contribution Summary (25% overall contribution)

Completed Tasks	**Incomplete Tasks**
Collected IRMAS dataset as the training and validation dataset	Building alternative RNN model for Stage 1
Data Preprocessing	Researching on existing network models to distinguish instruments for transfer learning
Built and trained Transfer Learning model	Making comparison between human accuracy and prediction accuracy of Stage 2 model
Helped tuning model hyperparameters for Stages 1 & 2 (10%)	
Wrote the Data and Data Processing sections in the presentation script; made the corresponding presentation slides	
Presentation video editing (70%)	

Table 10: Tiffany's Contribution Summary (25% overall contribution)

Completed Tasks	**Incomplete Tasks**
Presentation video editing (30%)	Trying to train Stage 2 on more than two instruments to see if model could handle these cases
Wrote Stage 1 training and accuracy functions	
Wrote code for 12 alternative architectures tested in Stage 1 during hyperparameter tuning	Tuning hyperparameters for the multi-output model (which didn't end up being used in our final results) → The model was learning to predict constant values regardless of input, giving exactly 50% accuracy. The model was too complex and this idea was rejected after the multi-label model showed promising results.
Tuned hyperparameters for Stage 1, improving validation accuracy from ~25% to 72% (ran tests for >20 hyperparameter combinations)	
Wrote the model architecture for a multi-output model that was tested	Testing out different instrument classes in the IRMAS dataset to see if models for different classes would perform better in both stages
Wrote training and accuracy functions for the multi-output model	
Wrote Stage 2 training and accuracy functions for multi-label model	Further training for transfer learning model (experimenting with the number of layers of weights to transfer to Stage 2 instead of just taking all convolutional layers, or perhaps allowing some of the transferred weights to be updated during training)
Tuned hyperparameters for Stage 2 (ran tests for >15 hyperparameter combinations), reaching 69% validation accuracy	
Wrote code to find the test accuracy and generate the confusion matrices of both stages	Looking at the Stage 2 samples where the model predicted more or less than two instruments present in the data, to see if there are any patterns.
Wrote the model and results sections in the presentation script; made the corresponding presentation slides	

Works Cited

[1] A. Koretzky, Audio AI: isolating vocals from stereo music using Convolutional Neural Networks, towards data science, 2019.

[2] P. DAR, PixelPlayer – Identify and Extract Musical Instrument Sounds from Videos with MIT's AI, Analytics Vidhya, 2018.

[3] "3.2.4.3.1. Sklearn.ensemble.RandomForestClassifier." *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

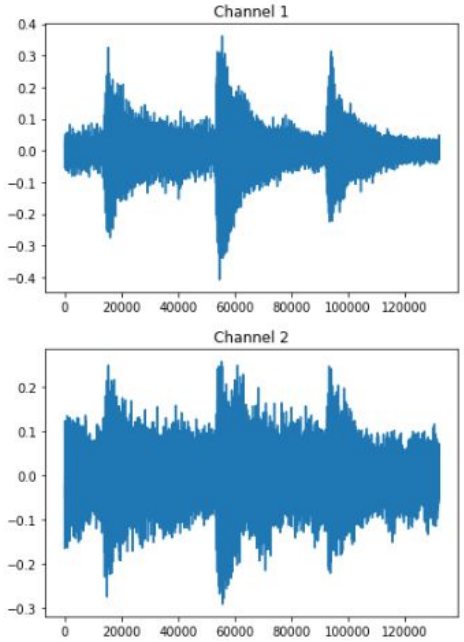
[4] "Random Forest." *Wikipedia*, Wikimedia Foundation, 11 July 2020, en.wikipedia.org/wiki/Random_forest.

[5] 2011. *Cherry Pink And Apple Blossom White Trumpet*. [online] Available at: <https://www.youtube.com/watch?v=GnbnSBcRy5U> [Accessed 8 August 2020].

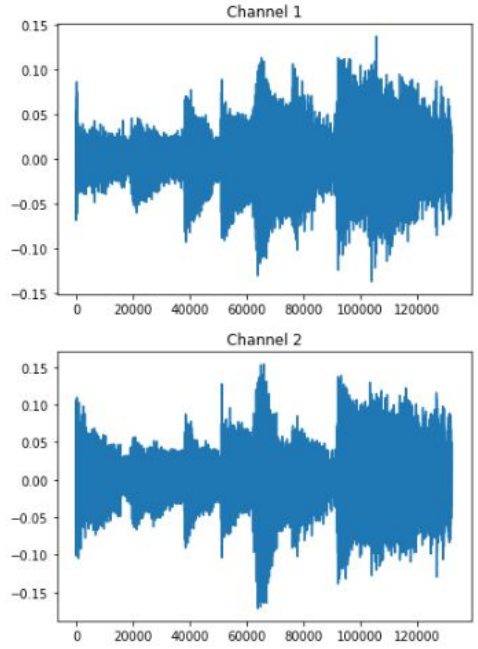
[6] Martinez, D. and Cadavid, M., 2018. *Violin And Trumpet Duet, N.º 10, KV. 487 (W. A. Mozart)*. [online] Available at: <https://www.youtube.com/watch?v=GnbnSBcRy5U> [Accessed 8 August 2020].

Appendix A: Raw Data of Four Instrument Classes

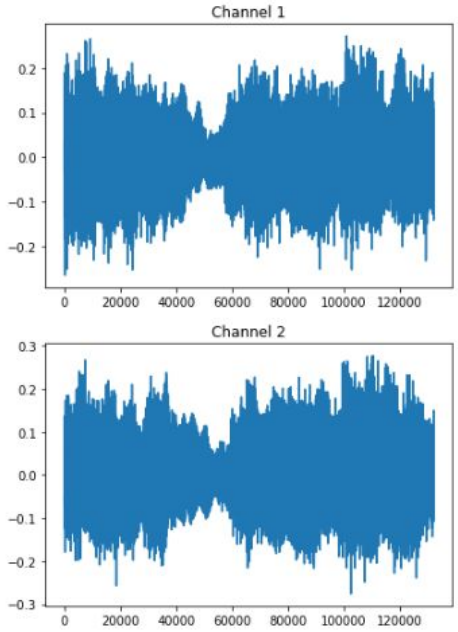
Sample waveform of violin
Shape of waveform: torch.Size([2, 132299])
Sample rate of waveform: 44100



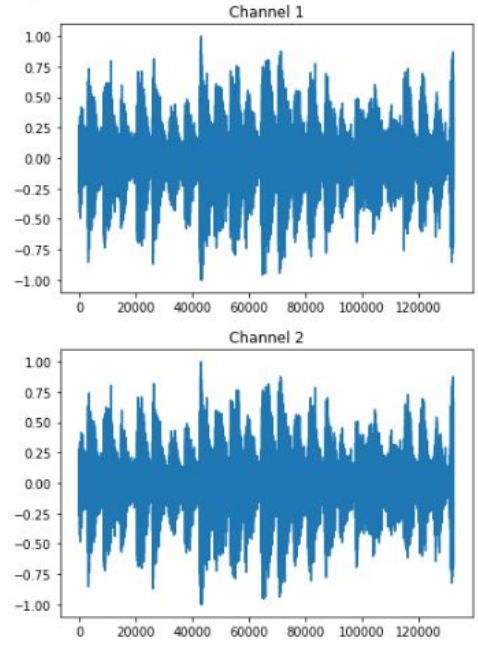
Sample waveform of acoustic guitar
Shape of waveform: torch.Size([2, 132299])
Sample rate of waveform: 44100



Sample waveform of trumpet
Shape of waveform: torch.Size([2, 132299])
Sample rate of waveform: 44100



Sample waveform of piano
Shape of waveform: torch.Size([2, 132299])
Sample rate of waveform: 44100



:

Appendix B: Training Curves

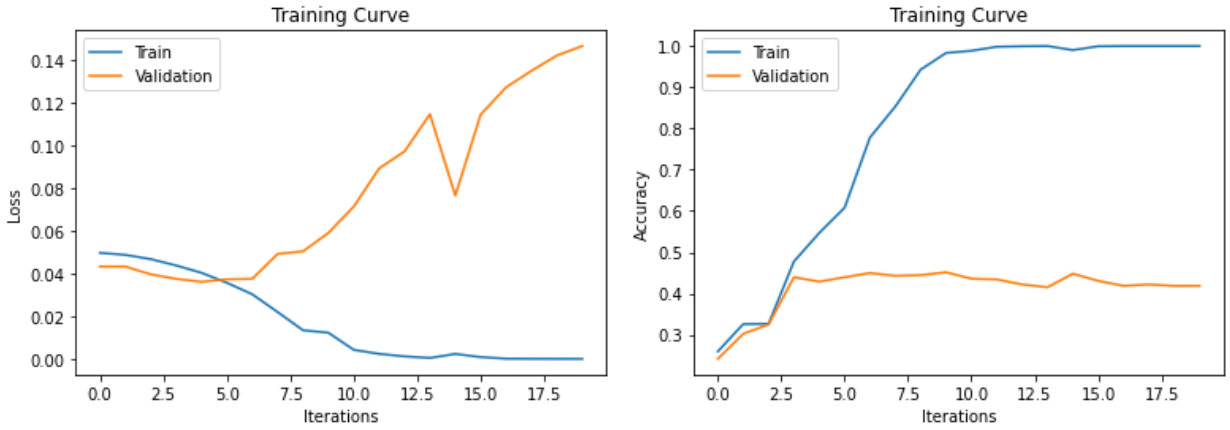


Figure B.1: Training curves for MyNet2, lr=0.001, batch_size=32, num_epochs=20

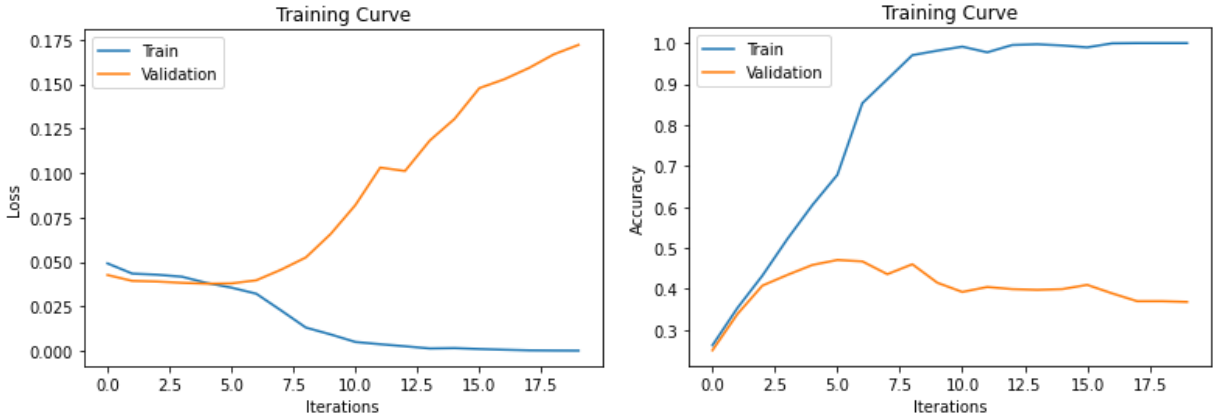


Figure B.2: Training curves for MyNet3, lr=0.001, batch_size=32, num_epochs=20

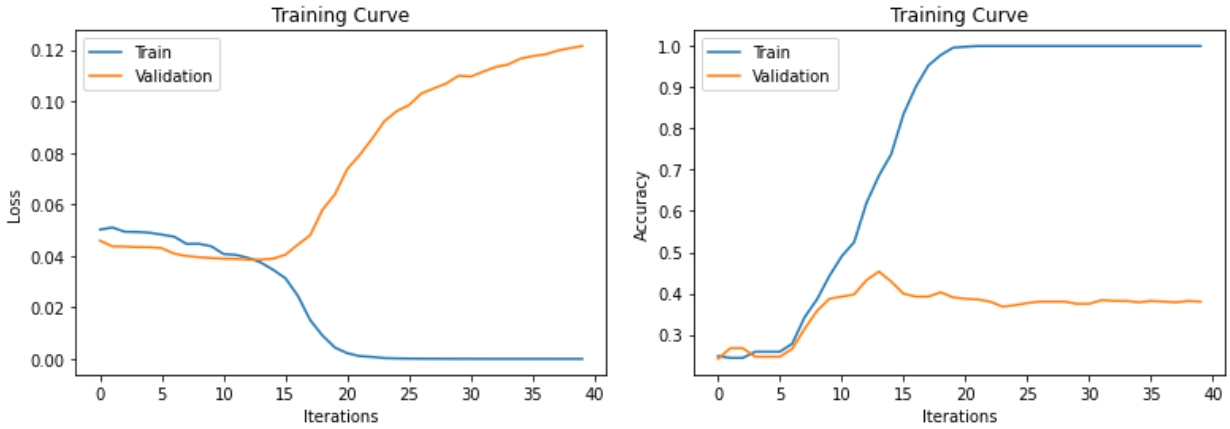


Figure B.3: Training curves for MyNet3, lr=0.0004, batch_size=32, num_epochs=40

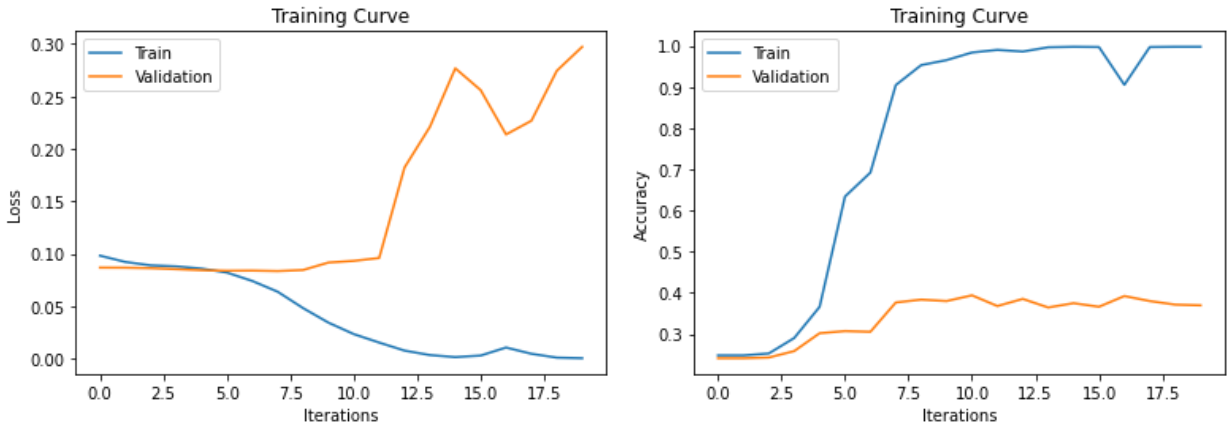


Figure B.4: Training curves for MyNet3, lr=0.0007, batch_size=16, num_epochs=20

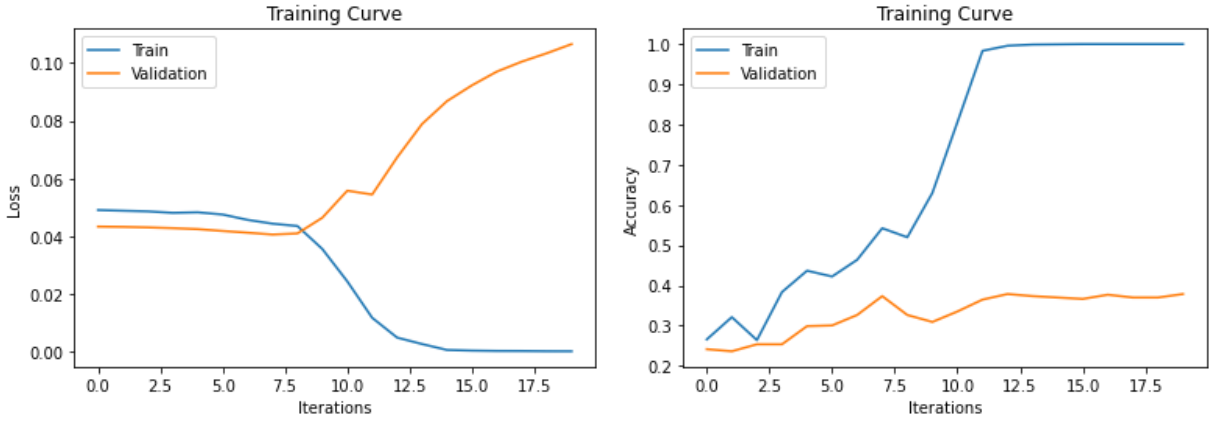


Figure B.5: Training curves for MyNet4, lr=0.0005, batch_size=32, num_epochs=20

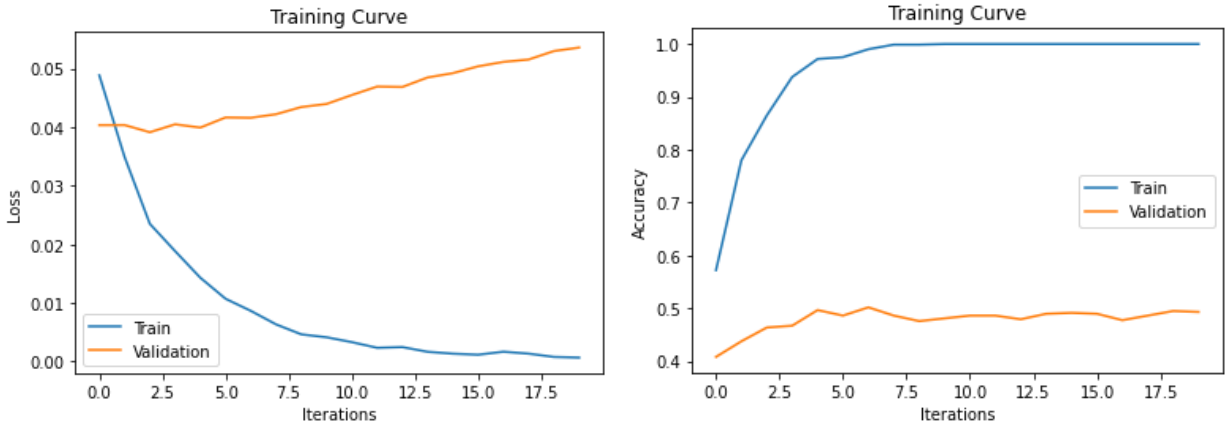


Figure B.6: Training curves for MyNet5, lr=0.0003, batch_size=32, num_epochs=20

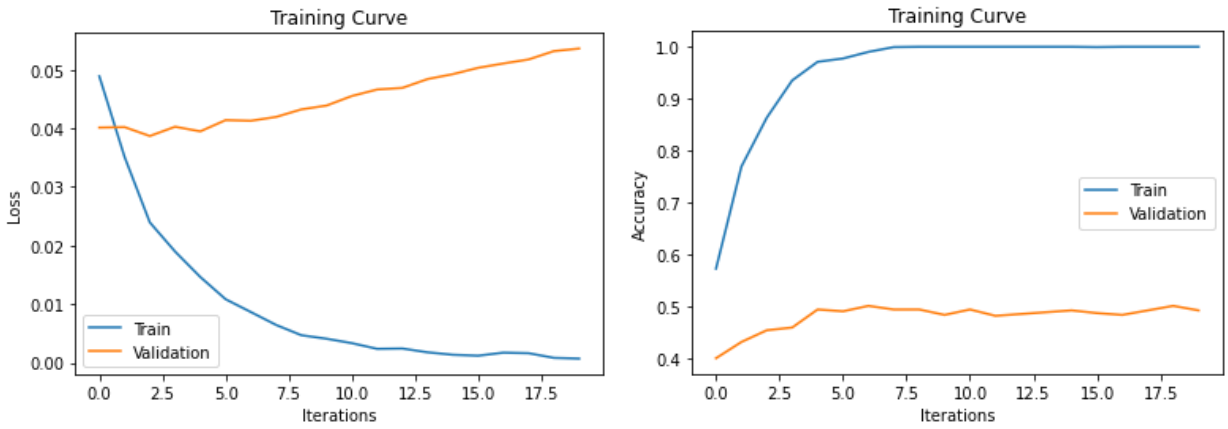


Figure B.7: Training curves for MyNet6, lr=0.0003, batch_size=32, num_epochs=20

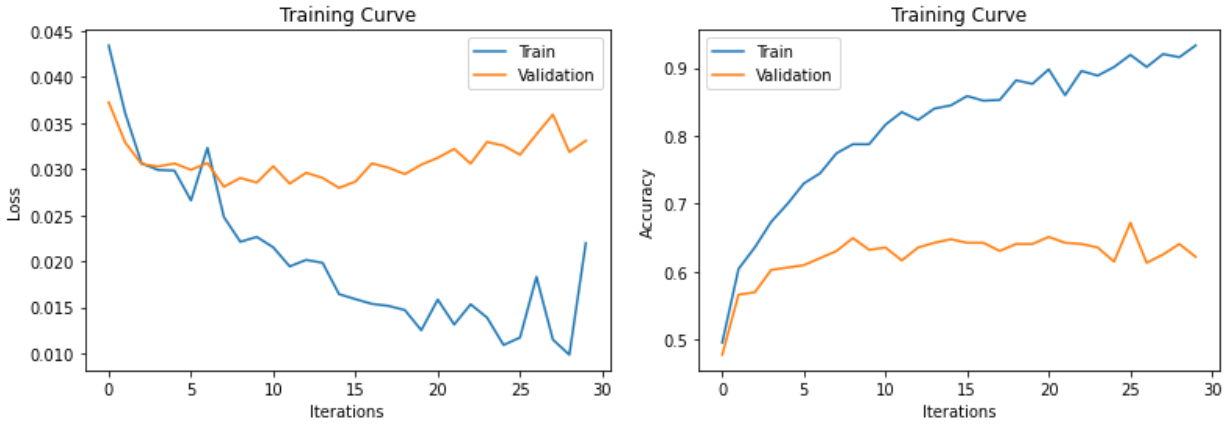


Figure B.8: Training curves for MyNet8, lr=0.0003, batch_size=32, num_epochs=30

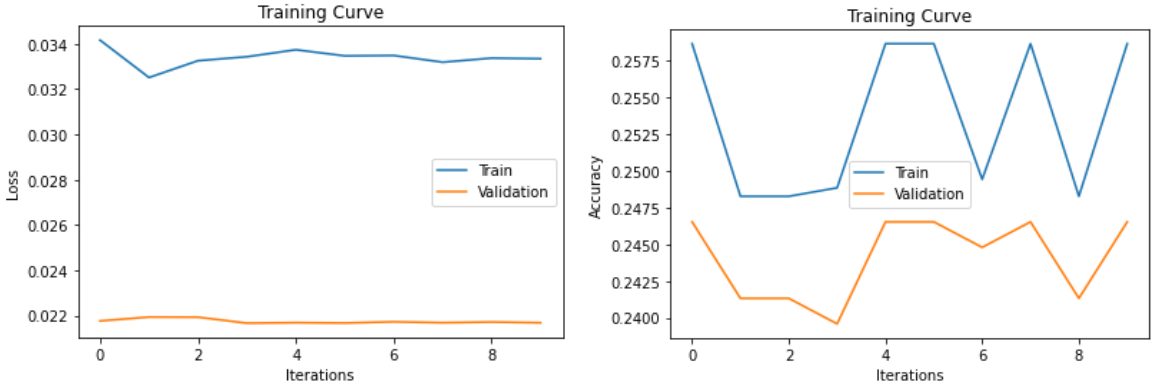


Figure B.9: Training curves for MyNet, lr=0.0001, batch_size=64, num_epochs=10

Observations: The result in Figure 16 shows poor performance as the model fails to learn from the training examples. Here, the complexity of the problem is too high. In the next phase, instruments with very similar waveforms were selectively removed to simplify the problem.

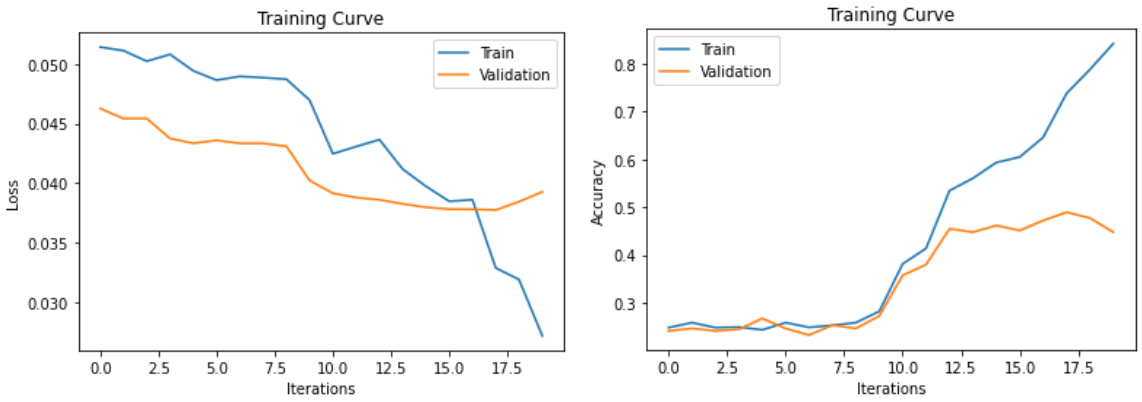


Figure B.10: Training curves for MyNet3, lr=0.0005, batch_size=32, num_epochs=20

Observations: After simplifying the problem, the model's performance started to improve. Distinct features were extracted from each instrument's waveforms. One fully connected layer was removed to reduce model complexity. However, it was anticipated that the model would be overfitting beyond the 15th epoch, as shown in Figure 17.

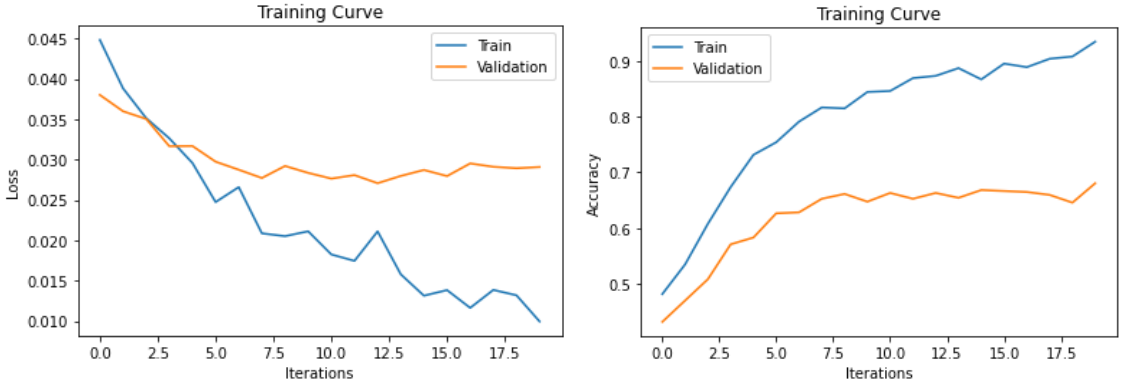


Figure B.11: Training curves for MyNet7, lr=0.0003, batch_size=32, num_epochs=20

Observations: In the final hyperparameter tuning phase, techniques were employed to prevent overfitting. First more convolutional layers were added to reduce input dimensions to 1030 before feeding into the linear layer. Then batch normalization layers and one dropout layer was added to each of those convolutional layers. This significantly improved performance as well, as shown in Figure 18.

Appendix C: Tabulated Results

Table C.1: Validation accuracy of the Baseline Model in Stage 1

Number of estimators	Baseline Accuracy on Validation
2000	43%
1500	43%
1000	38%
500	39%
250	36%

Table C.2: Validation accuracy of the Baseline Model in Stage 2

Number of estimators	Baseline Accuracy on Validation
2000	38%
1500	37%
1000	36%
500	36%
250	36%

Table C.3: Complete summary of results from hyperparameter tuning

Model Name	Learning Rate	Batch Size	Number of Epochs	Final Training Loss	Final Validation Loss	Final Training Accuracy	Final Validation Accuracy	Best Validation Accuracy
MyNet	0.0001	64	10	0.0333	0.0217	0.2587	0.2465	0.2465
MyNet	0.001	64	10	0.0333	0.0217	0.2587	0.2465	0.2674
MyNet	0.01	64	10	0.0333	0.0217	0.2587	0.2465	0.2465
MyNet	0.01	32	10	0.0489	0.0434	0.2587	0.2465	0.2465
MyNet2	0.01	32	10	0.0489	0.0434	0.2587	0.2465	0.2465
MyNet2	0.001	32	20	0.0000	0.1467	1.0000	0.4184	0.4514
MyNet3	0.001	32	20	0.0001	0.1720	1.0000	0.3681	0.4705
MyNet3	0.0005	32	20	0.0272	0.0393	0.8424	0.4479	0.4896
MyNet3	0.0003	32	20	0.0152	0.0437	0.9758	0.3628	0.3750
MyNet3	0.0003	32	40	0.0027	0.0557	0.9977	0.3906	0.3906
MyNet3	0.0004	32	40	0.0000	0.1215	1.0000	0.3802	0.4531
MyNet3	0.0007	16	20	0.0004	0.2972	0.9977	0.3698	0.3941
MyNet4	0.0005	32	20	0.0001	0.1065	1.0000	0.3785	0.3785

MyNet5	0.0005	32	20	0.0002	0.0631	1.0000	0.4358	0.4410
MyNet5	0.0003	32	20	0.0007	0.0536	1.0000	0.4931	0.5017
MyNet6	0.0003	32	20	0.0007	0.0531	1.0000	0.4931	0.5017
MyNet7	0.0003	32	20	0.0100	0.0291	0.9342	0.6806	0.6806
MyNet8	0.0003	32	30	0.0220	0.0331	0.9330	0.6215	0.6719
MyNet9	0.0003	32	30	-	-	-	-	0.6667
MyNet10	0.0003	32	30	-	-	-	-	0.7014
MyNet11	0.0003	32	30	-	-	-	-	0.7118
MyNet12	0.0003	16	30	-	-	-	-	0.7170
MyNet13	0.0003	16	30	-	-	-	-	0.7257