Progress Report Team 11:

Music Classification Neural Network

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

Project Description

The end goal of our project is to identify instruments present in an audio file. A key part to appreciating music -- whether it be at a concert or online -- includes being aware of the numerous instruments coming together in the piece.

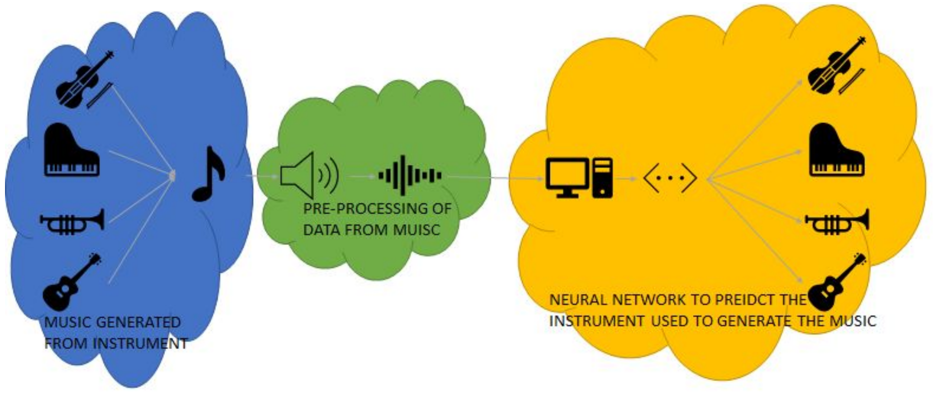

PRE-PROCESSING OF
DATA FROM MUISC
MUSIC GENERATED
FROM INSTRUMENT
NEURAL NETWORK TO PREIDCT THE
INSTRUMENT USED TO GENERATE THE MUSIC

Figure 1: Project motivation

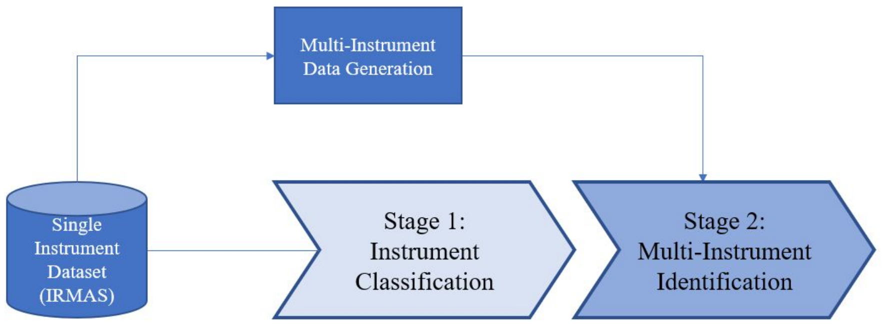


Figure 2: Overview of the project stages.

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 (IRMAS dataset) consist of only one instrument playing at once. 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.

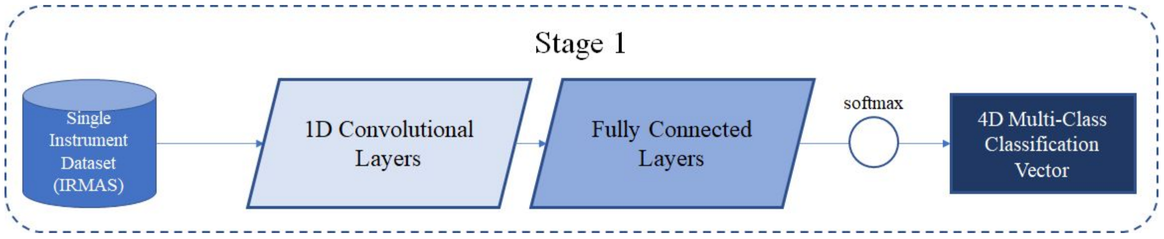
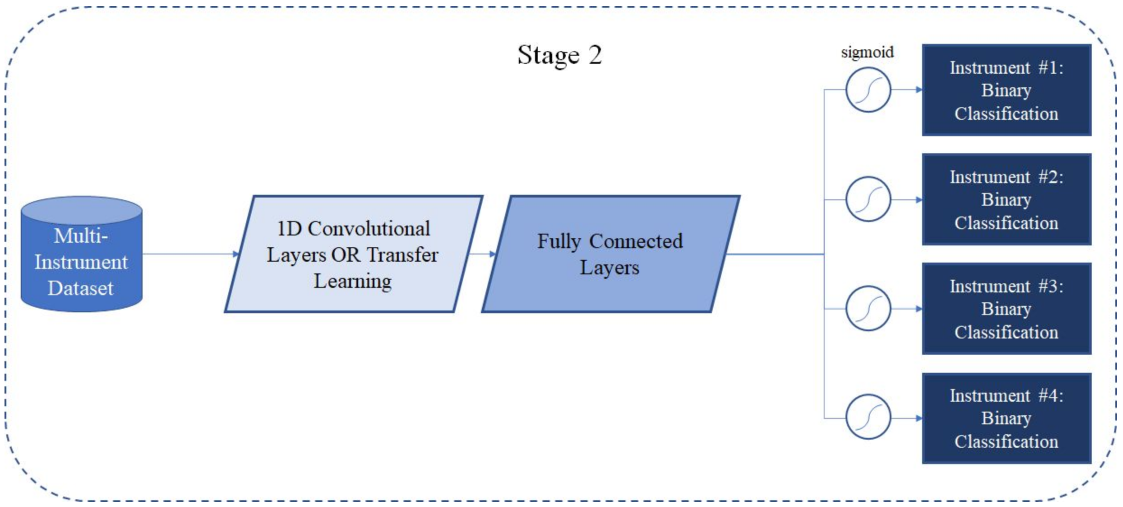


Figure 3: Detailed framework of Stage 1

The single instrument audio samples are used as input and a four-dimensional multi-class classification vector as output. The vector will be used for differentiating between piano, violin, trumpet, and acoustic guitar samples. Algorithmic programming methods are insufficient to address this instrument classification task due to the diversity and complexity in instrumental audio signals. Machine learning is necessary to identify slight nuances and patterns in instrument classes and differentiate between them.


Stage 2
sigmoid
Multi-Instrument Dataset
1D Convolutional Layers OR Transfer Learning
Fully Connected Layers
Instrument #1: Binary Classification
Instrument #2: Binary Classification
Instrument #3: Binary Classification
Instrument #4: Binary Classification
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. We plan to explore using transfer learning to extract the features, which would then be flattened and passed into each of the four classifiers with a sigmoid activation.

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 Figure 5 and Figure 6:

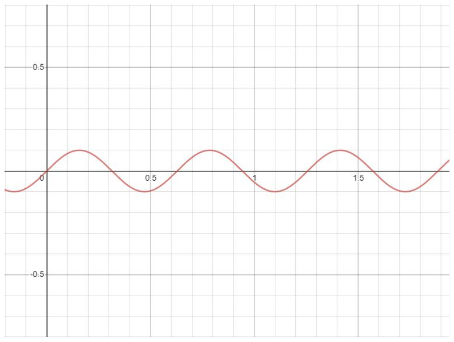


Figure 5: Original Wave Data

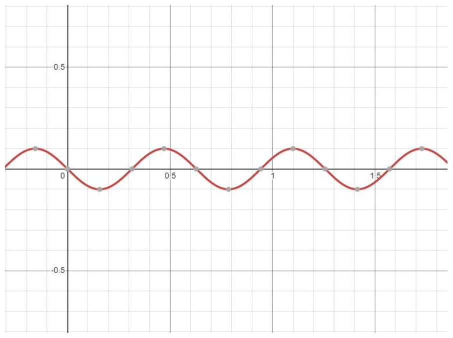


Figure 6: Augmented Wave Data

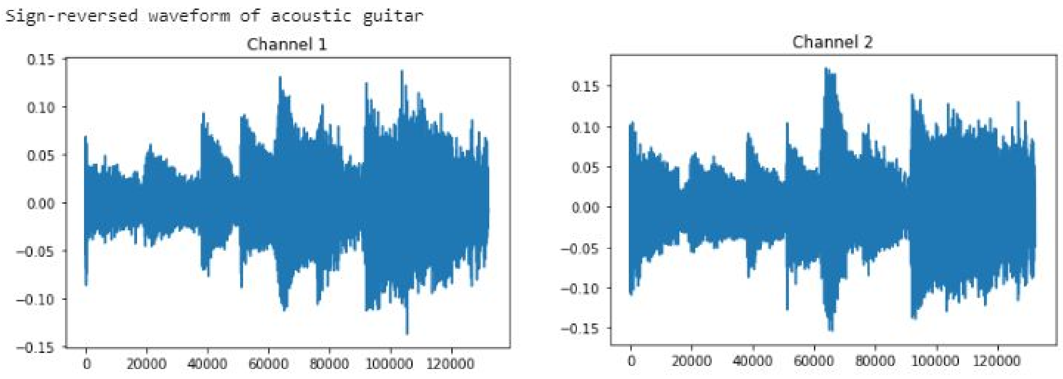


Figure 7: An Example of Sample Data of Augmented Acoustic Guitar Audio

Table 2: Number of Samples in Balanced Data, split into [0.6, 0.2, 0.2] for training, validation and testing. Total data number for each class is 721

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 Figure 8 and Figure 9:

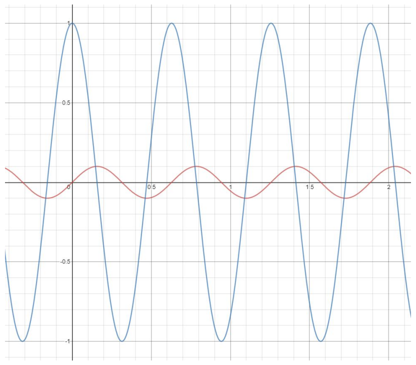


Figure 8: Original Wave Data

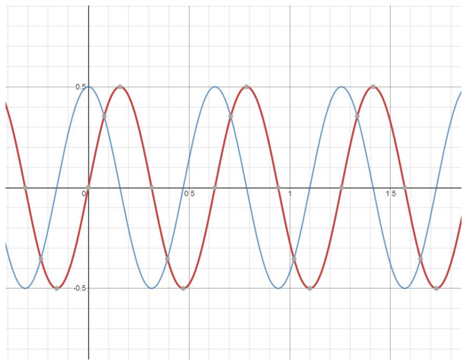


Figure 9: Normalized Wave Data

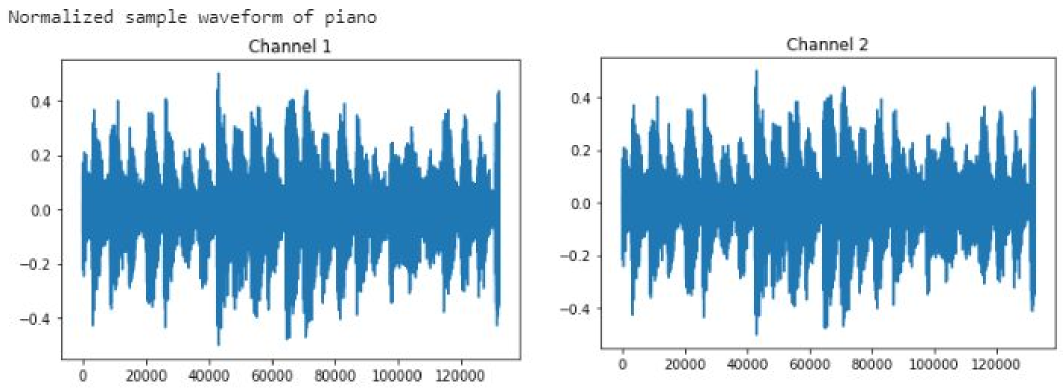


Figure 10: An Example of Sample Data of Normalized Piano Audio

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 11 and Figure 12:

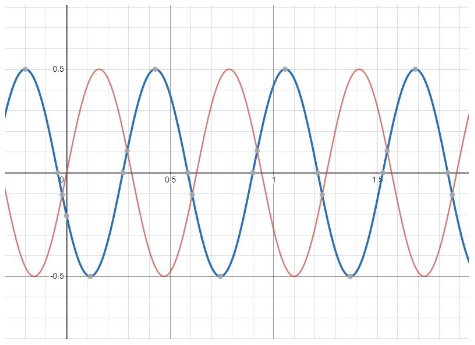


Figure 11: Two Normalized Waves

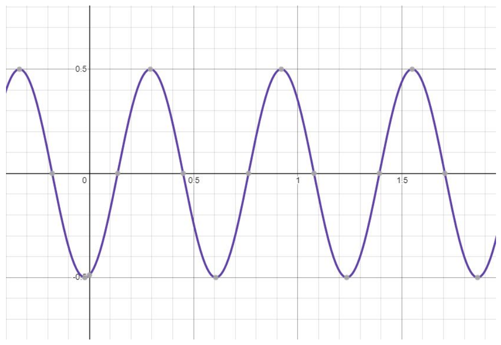


Figure 12: Summed Wave (Normalized)

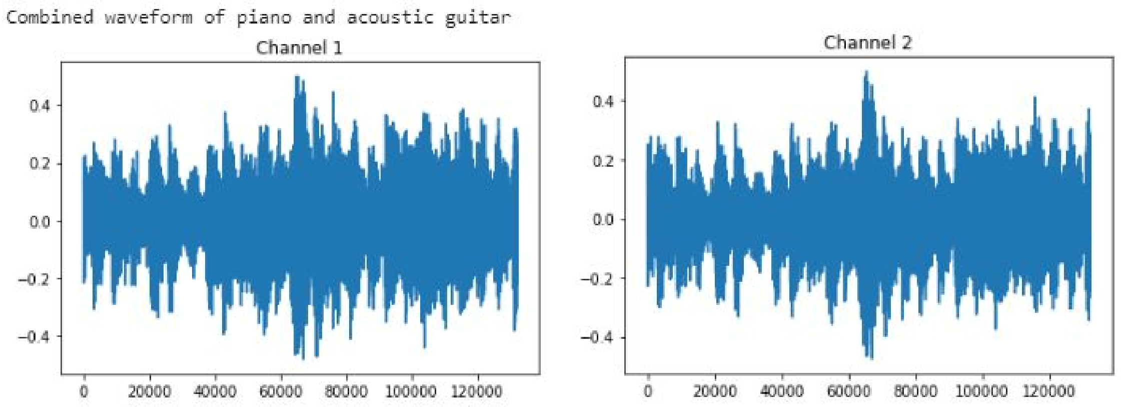


Figure 13: An Example of Combined Wave of Piano Audio and Guitar Audio

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's sci-kit learn. [1] 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. (i.e. is it percussive, is it loud or quiet, does it sound like a chord is being played.) A random forest would thus be best suited as a baseline model to compare to the neural network. Refer to Figure 14 for how random forest classification works.

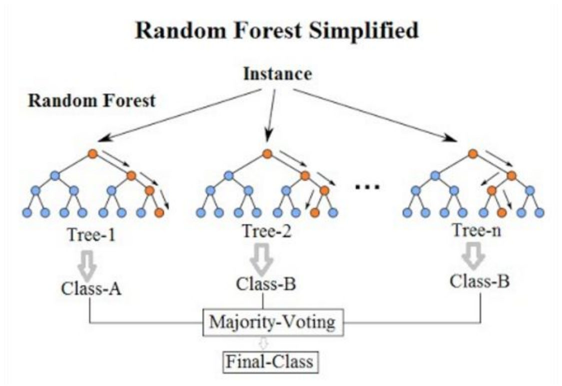


Figure 14: Random Forest Classifier Diagram [2]

A random forest tree works by generating different decision trees randomly. These decision trees result in certain predictions based on features. The final prediction is the majority in the individual predictions. Sci-kit learn requires the tuning of the number of estimators which represents the decision trees. This parameter is easy to tune.

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 random forest was trained on the training data. The validation accuracy is shown below in Table 3. The amount of trees do not seem to guarantee a large increase in accuracy.

Table 3: Validation accuracy of the Baseline Model.

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

Primary Model

The best architecture used for this consists of four 1D convolutional layers and one fully connected layer. Many aspects of the original model were modified during the hyperparameter tuning. For instance, overfitting techniques were employed such as adding dropout layers, batch normalization, adding more convolutional layers, reducing parameters in the linear layers,... etc.

The figure below gives a simple visual representation of one of the CNN architectures used. The audio input consists of two channels with 132,299 frames. Since this input is large, we reduce dimensionality in the convolutional layers so that fully connected layers would not have that many parameters. This is done by choosing larger kernel sizes and introducing pooling layers. Note, stride length and padding amount were kept default. Finally, a softmax function is applied to the output of the model to determine the predicted instrument by the classifier.

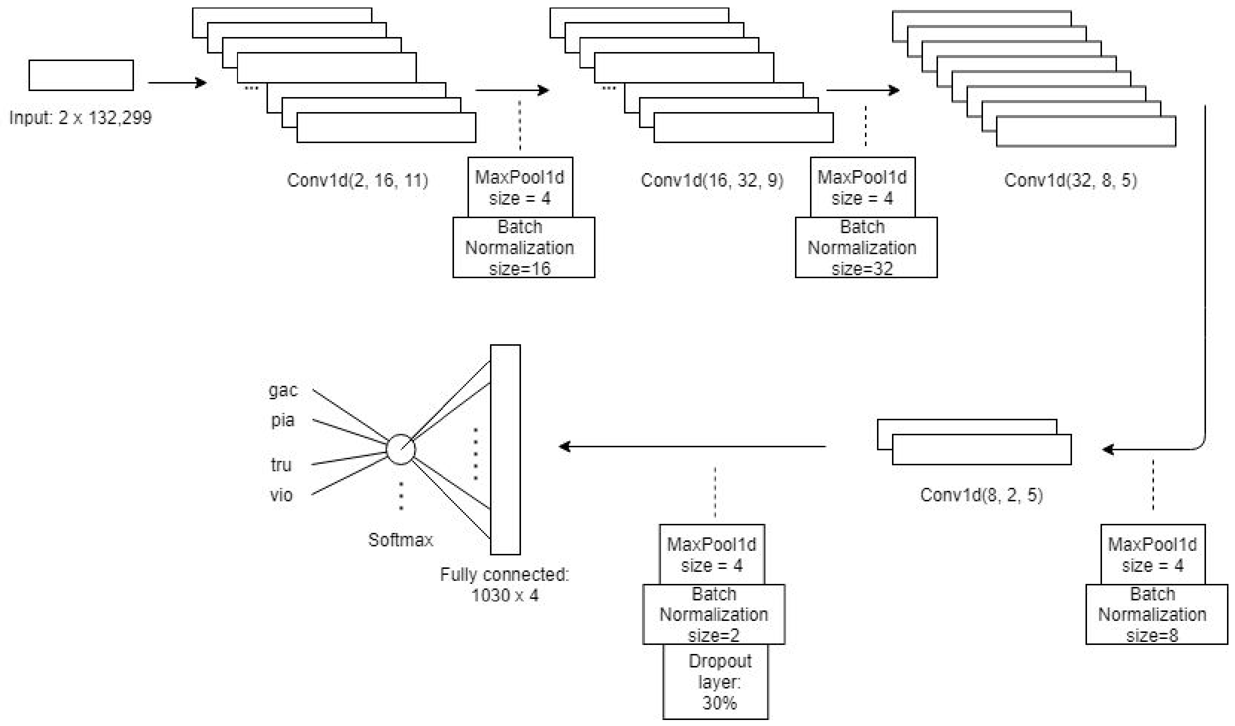


Figure 15: Sample CNN Architecture In Hyperparameter Tuning

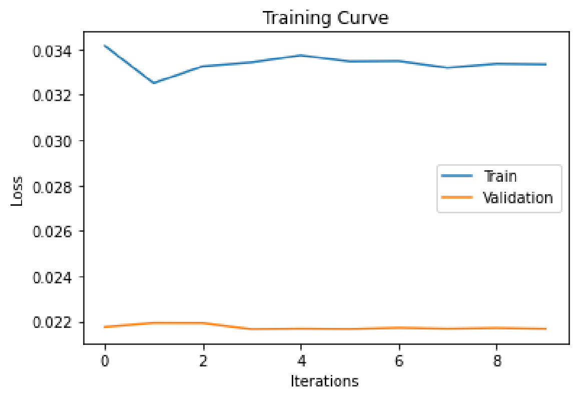
Results

Table 4: Summary of architecture-related hyperparameters tested

Model Name	# Conv. Layers	# Filters	#Fully Connected (FC) Layers	# Units in FC Layer(s)	Activation Function	Batch Normalization	Dropout
MyNet	3	20, 5, 4	2	8266, 500	ReLU	No	None
MyNet2	3	20, 5, 4	2	8266, 100	ReLU	No	None
MyNet3	3	20, 5, 4	1	8266	ReLU	No	None
MyNet4	3	20, 5, 4	1	8266	Leaky ReLU	No	None
MyNet5	3	20, 5, 4	1	8266	Leaky ReLU	Yes	None
MyNet6	3	20, 5, 4	1	8266	ReLU	Yes	None
MyNet7	4	16, 32, 8, 2	1	1030	ReLU	Yes	0.2
MyNet8	4	16, 32, 8, 2	1	1030	ReLU	Yes	0.3

Table 5: Abbreviated summary of results from hyperparameter testing -- best result from each architecture. Some training curves are shown in Figures 16 to 18. Additional training curves are shown in Appendix B. Complete summary shown in Appendix C.

Model Name	Learning Rate	Batch Size	Number of Epochs	Final Training Loss	Final Validatio n Loss	Final Training Accuracy	Final Validatio n Accuracy	Best Validatio n Accuracy
MyNet	0.0001	64	10	0.0333	0.0217	0.2587	0.2465	0.2465
MyNet2	0.001	32	20	0.0000	0.1467	1.0000	0.4184	0.4514
MyNet3	0.0005	32	20	0.0272	0.0393	0.8424	0.4479	0.4896
MyNet4	0.0005	32	20	0.0001	0.1065	1.0000	0.3785	0.3785
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



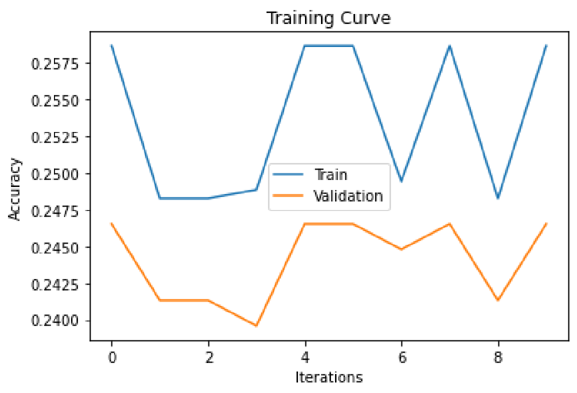


Figure 16: Training curves for MyNet, lr=0.0001, batch_size=64, num_epochs=10

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

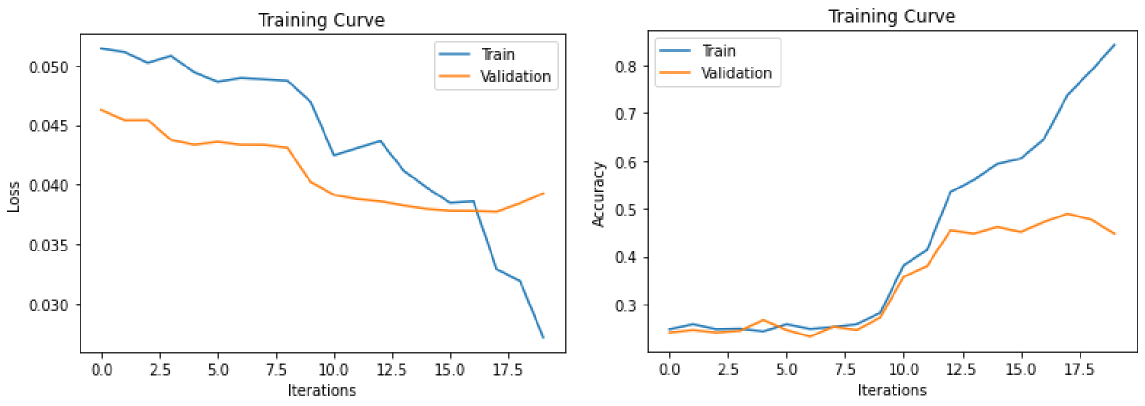


Figure 17: Training curves for MyNet3, lr=0.0005, batch_size=32, num_epochs=20

Observations: After simplifying the problem, the model's performance started to improve. Distinct features were extracted from each instrument's waveforms. We also removed one fully connected layer to reduce model complexity. However, we anticipated the model would be overfitting beyond the 15th epoch, as shown in Figure 17, and we were not satisfied with the model's performance.

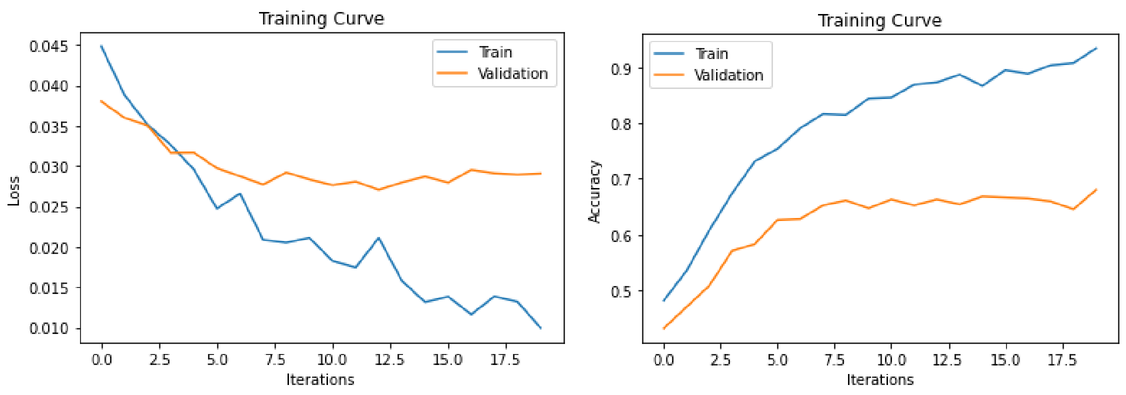


Figure 18: Training curves for MyNet7, lr=0.0003, batch_size=32, num_epochs=20

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

Project Progress

Table 6: Summary of each team member's completed tasks so far, which are all on track with the originally proposed project plan. Everyone completed their respective tasks on time and we consistently communicated our progress with each other.

Team Member	Completed Tasks
Ahmed	● Wrote code for our Stage 1 CNN model architecture ● Tuned the hyperparameters of the CNN model
Reynold	● Wrote code for the baseline ● Tuned the baseline model
Tianxu	● Wrote code to load and process the data ● Tuned the hyperparameters of the CNN model
Tiffany	● Wrote code for model training and accuracy calculation ● Tuned the hyperparameters of the CNN model

Table 7: Summary of each team member's upcoming tasks, with deadlines presented in Table 8

Team Member	Upcoming Tasks
Ahmed	● Fine tuning current architecture to have a higher validation accuracy than the baseline model of stage 1 ● Write code to take embedding from stage 1 and pass it into four sigmoid functions + outputs ● Tune hyperparameters for stage 2
Reynold	● Baseline for stage 2 ● Tune baseline model ● Tune hyperparameters for stage 2
Tianxu	● Building the dataset for stage 2 ○ Combining IRMAS dataset to generate the dataset for stage 2 ● Try using RNN for stage 1 ● Tune hyperparameters for stage 2
Tiffany	● Fine tuning current architecture to have a higher validation accuracy than the baseline model of stage 1 ● Write training code for stage 2 ● Tune hyperparameters for stage 2

Table 8: Summary of group deadlines for each upcoming task

Week	Tasks to be Completed
July 11 - July 17	● Finish making improvements to Stage 1 neural network ● Write data processing code for Stage 2 ● Make baseline for Stage 2 ● Write code for possible Stage 2 model architectures

July 18 - July 24	● Tune hyperparameters for Stage 2 neural network ● Resolve any issues that might have come up thus far
July 25 - July 31	● Run some final tests and finish up the model tuning ● Compile results into some presentable format ● Start working on the presentation video and aim to finish most of it
August 1 - August 8	● Finish the presentation video by the August 5 deadline ● Write the final report by the August 9 deadline

Table 9: Summary of project management platforms

Platform	**Usage**
Messenger Chat	● For ongoing communication throughout the week ● For any issues that may come up as each team member works on their individual tasks
Messenger Call	● For all arranged meetings to discuss progress, provide code demonstrations, and plan ahead for upcoming weeks
Google Drive	● Used to store all our code, datasets, and reports as the project progresses
Google Colab	● One group Colab file which contains combined code from all team members' individual tasks ● Personal Colab files are used when each team member is testing and writing their own code ● Working code is copied over to the group file when complete

Table 10: Risk Analysis and Alternate Plans

Risk Description	**Risk Likelihood** (1 - highly unlikely, 5 - highly likely)	**Alternate Plan**
Current CNN model cannot yield high test accuracy (0.8+)	4	Try building a RNN model for Stage 1 and compare its validation accuracy with the current best model
Teammate unforeseen medical situation: (Covid19)	3	Other team members split that person's tasks
Two instruments sounds similar and the model finds it hard to distinguish them	4	Replace one of them with a more distinguishable instrument

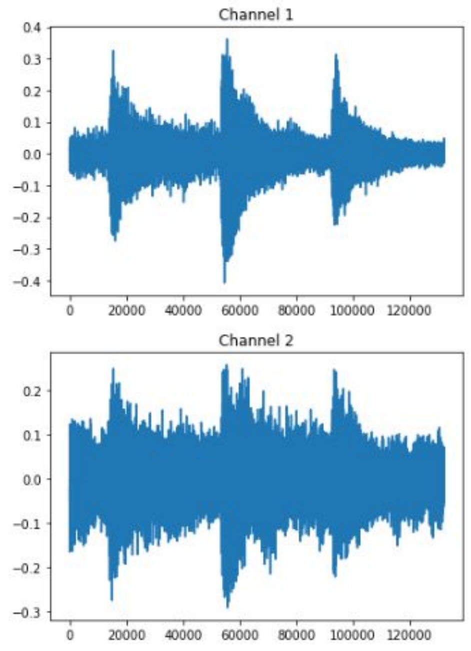
Works Cited

[1] “3.2.4.3.1. Sklearn.ensemble.RandomForestClassifier.” *Scikit*, scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

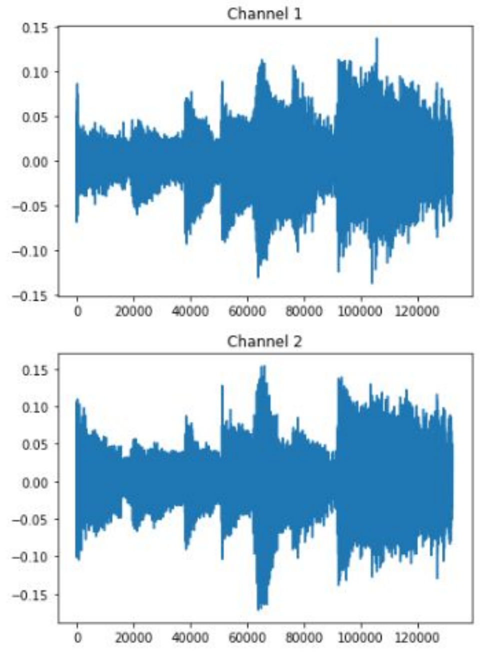
[2] “Random Forest.” *Wikipedia*, Wikimedia Foundation, 11 July 2020, en.wikipedia.org/wiki/Random_forest.

Appendix A: Raw Data of Four Instrument Classes

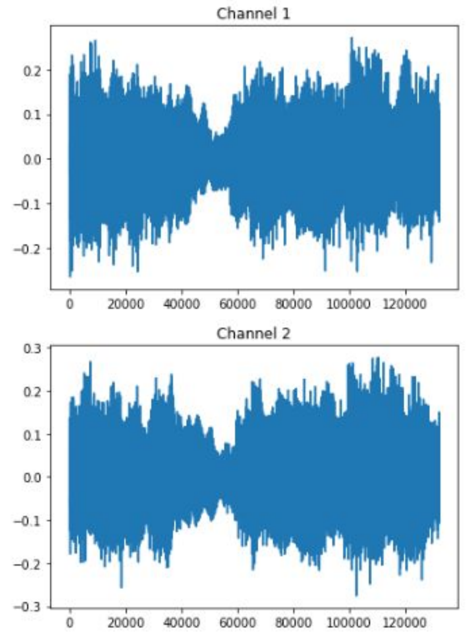
```
Sample waveform of violin
Shape of waveform: torch.Size([2, 132299])
Sample rate of waveform: 44100
```



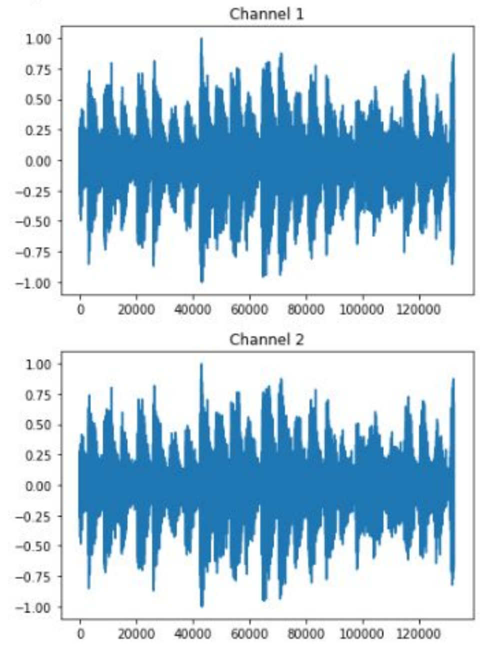
```
Sample waveform of acoustic guitar
Shape of waveform: torch.Size([2, 132299])
Sample rate of waveform: 44100
```



```
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

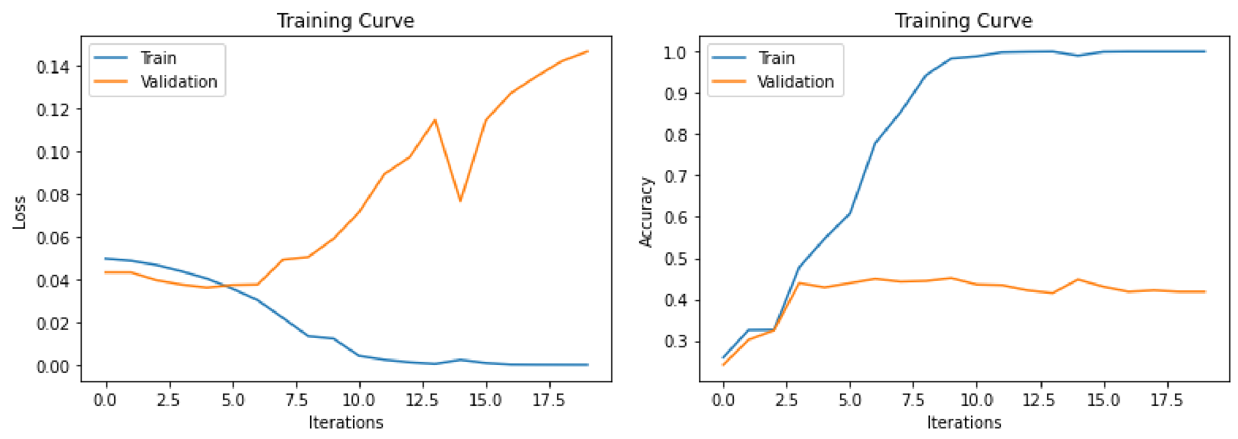


Figure B.1: Training curves for MyNet2, lr=0.001, batch_size=32, num_epochs=20

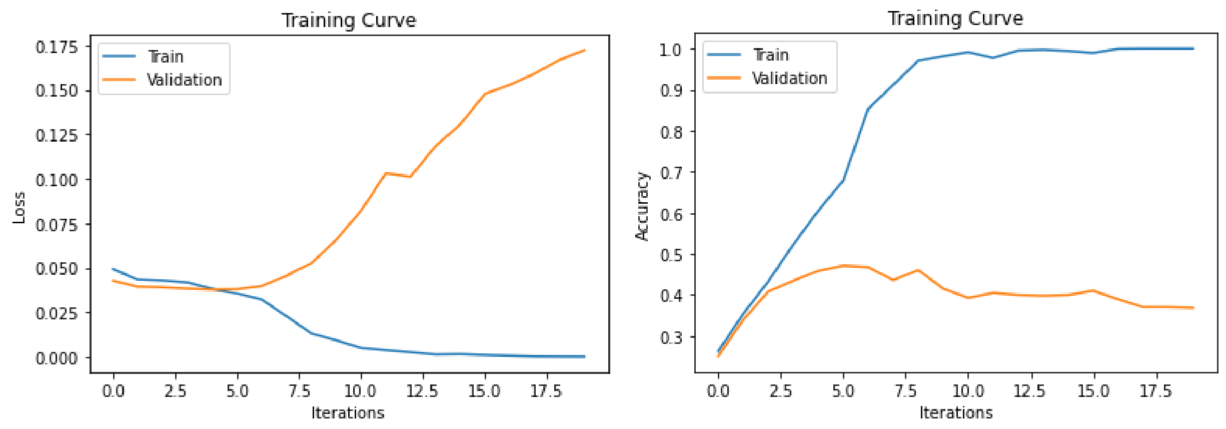


Figure B.2: Training curves for MyNet3, lr=0.001, batch_size=32, num_epochs=20

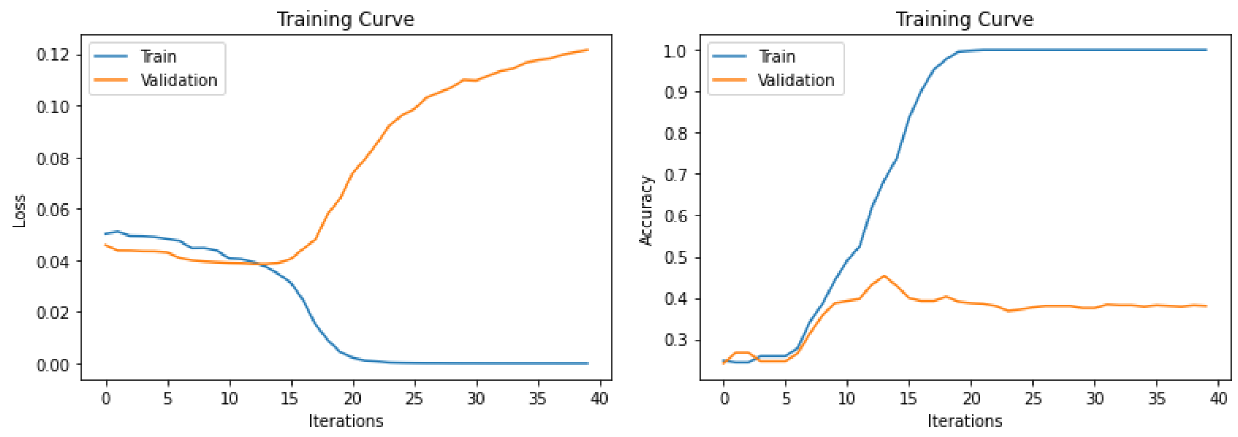


Figure B.3: Training curves for MyNet3, lr=0.0004, batch_size=32, num_epochs=40

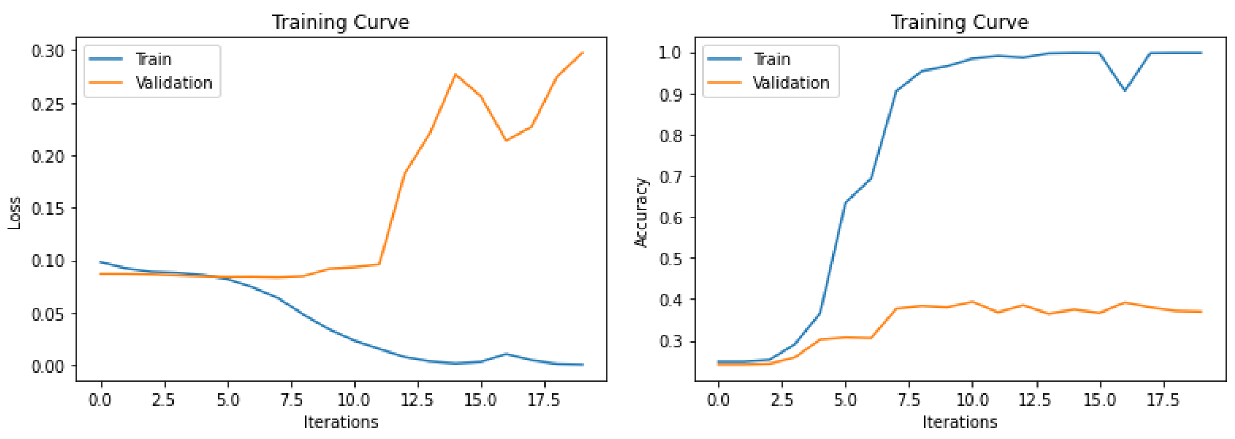


Figure B.4: Training curves for MyNet3, lr=0.0007, batch_size=16, num_epochs=20

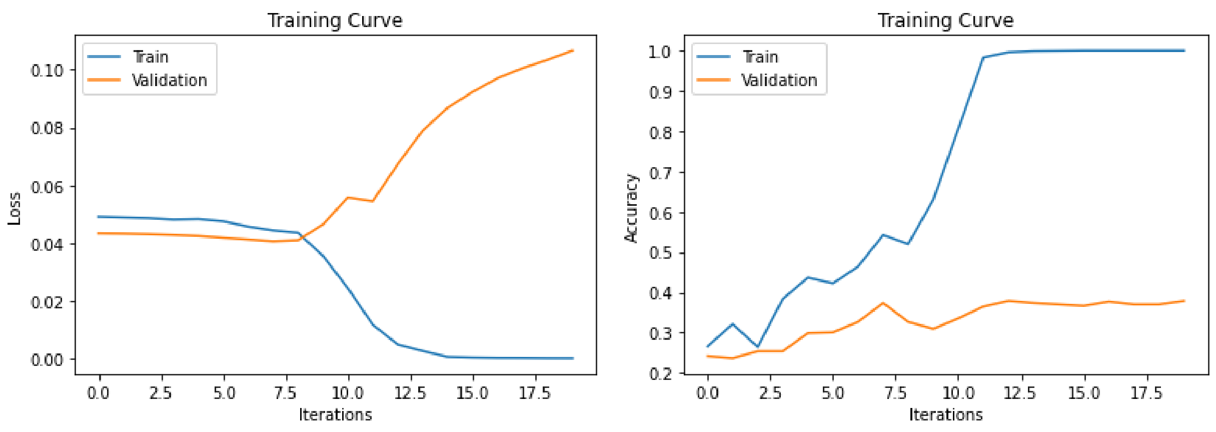


Figure B.5: Training curves for MyNet4, lr=0.0005, batch_size=32, num_epochs=20

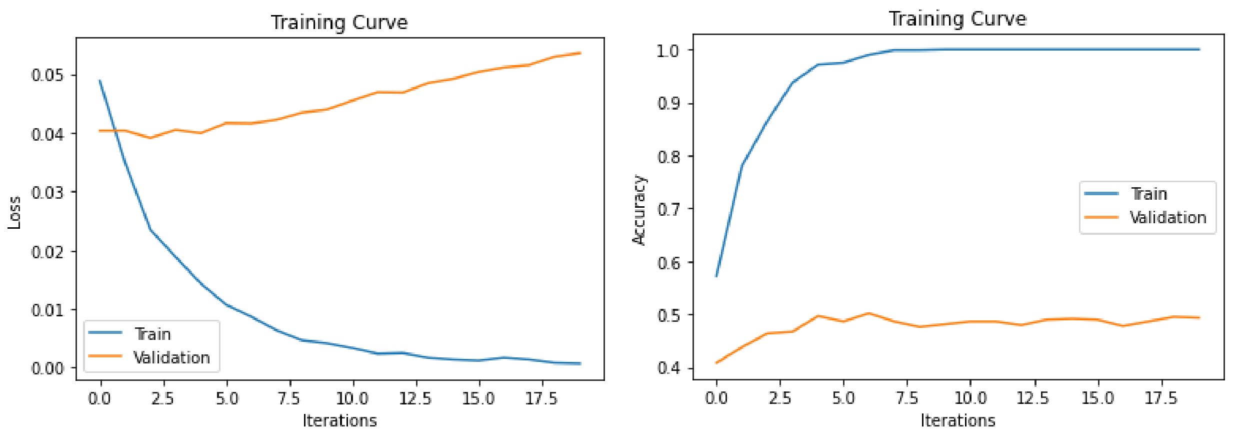


Figure B.6: Training curves for MyNet5, lr=0.0003, batch_size=32, num_epochs=20

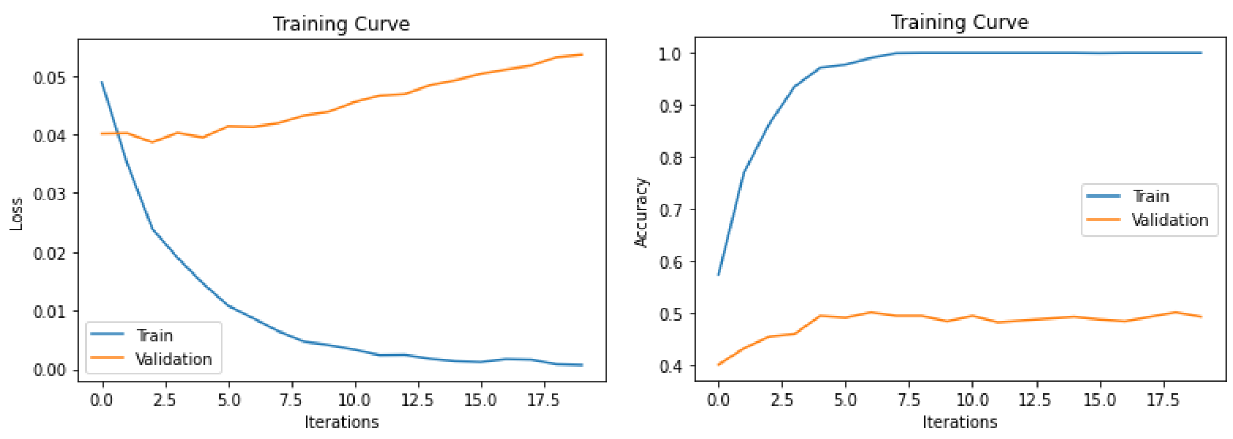


Figure B.7: Training curves for MyNet6, lr=0.0003, batch_size=32, num_epochs=20

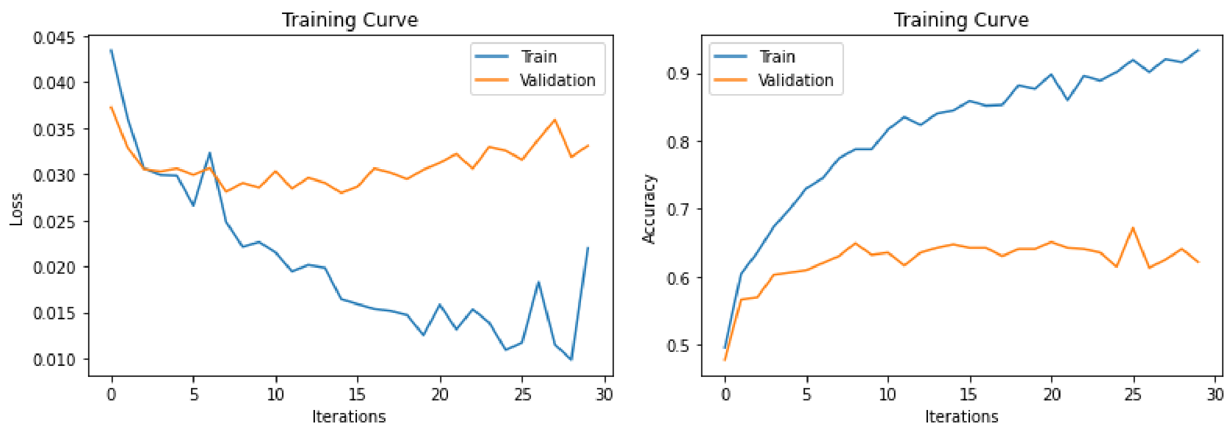


Figure B.8: Training curves for MyNet8, lr=0.0003, batch_size=32, num_epochs=30

Appendix C: Tabulated Results

Table C.1: 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