Harmonics Bot: The Classical Classifier

Team 1: Christopher Watson, Nathan Edwards, Paul Thai

Shiley-Marcos School of Engineering, University of San Diego

AAI-511 - Neural Networks & Learning

Professor Mirsardar Esnaeilli, Ph.D.

August 11, 2023

Abstract

Music is a form of art that utilizes sound, rhythm, melody, instruments, and harmony to depict emotions, expressions and serves as a universal language that resonates and entangles those who listen. Deep Learning, a subset of machine learning, is a powerful tool that involves training artificial neural networks containing multiple layers to learn and recognize complex patterns in data. In recent years, the application of deep learning has extended itself and revolutionized the field of music analysis and classification. In this study, the team aims to explore the use of deep learning algorithms to predict, from a sample, which piece of music belongs to one of the nine composers. We are using Mido, a Python library that provides functionalities for working with musical instrument digital interface (MIDI) files and messages, to grab metadata from the MIDI files. We created our convolutional neural network (CNN) model to learn and extract patterns from the features to differentiate between the various composers and implemented a Long Short-Term Memory (LSTM) layer to capture sequential data and dependencies of musical compositions. As a result, we were able to achieve a final accuracy score of 72% from an initial start of 11%.

Keywords: CNN, LSTM, music, deep learning, Mido

Introduction

People would think that music and deep learning, two fields, on different realms of the spectrum are disparate entities; however, the collaboration between the two of them has led to a breakthrough in artistic inspiration and technological advances. As such, it is common in this day and age to see technology integrated with music to bring out new harmonies, rhythms, and beats that were previously unimaginable. In this paper, we tested the ability of our CNN to accurately predict which musical artists, out of nine composers, composed the song. If our CNN is successful in classifying the data, then we have succeeded in predicting the correct composer from a single sequence of the music files. This is important because it demonstrates the convergence of art and technology by illustrating how deep learning techniques can be utilized to decode and analyze complex patterns. This approach can be very useful in other applications such as the medical field where deep learning models can be developed to analyze certain underlying diseases from images of X-rays, MRIs, and CT scans for early detection, possibly saving a person's life.

PreProcessing

To preprocess the data, we first needed to understand how the data worked. It took four iterations of different data processors until we finally managed to settle on one that worked well for our model. The first data processor pulled out the notes but lost all metadata in reference to time or velocity. By the third attempt, we pulled out too many dimensions which ultimately lead to the data being too sparse to make good inferences in the model. Our fourth attempt finally created workable data. The notes were arranged into an array of size 128, this represents all the notes that a midi file can represent. If the note was on, the velocity would populate its position in the array. The channel dimension was flattened and discarded as it created too much dead space in the data. The array was populated over a set length of ‘ticks’ which is how the midi file records time. The highest velocity value during the sampling time remained. This went on for a sequence of 128 samples. The sampling time per sample varied; the first time the model was trained it used a sampling time of 200 ticks, and the second it used a sampling time of 180 ticks. This is because the sampling rate needed to change based on the speed of the music itself or it wouldn’t capture patterns in the data. Shown below is a picture of one of Mendelssohn’s songs sampled at 200 ticks with 128 samples, and the same song sampled at 128 ticks with 256 samples respectively:

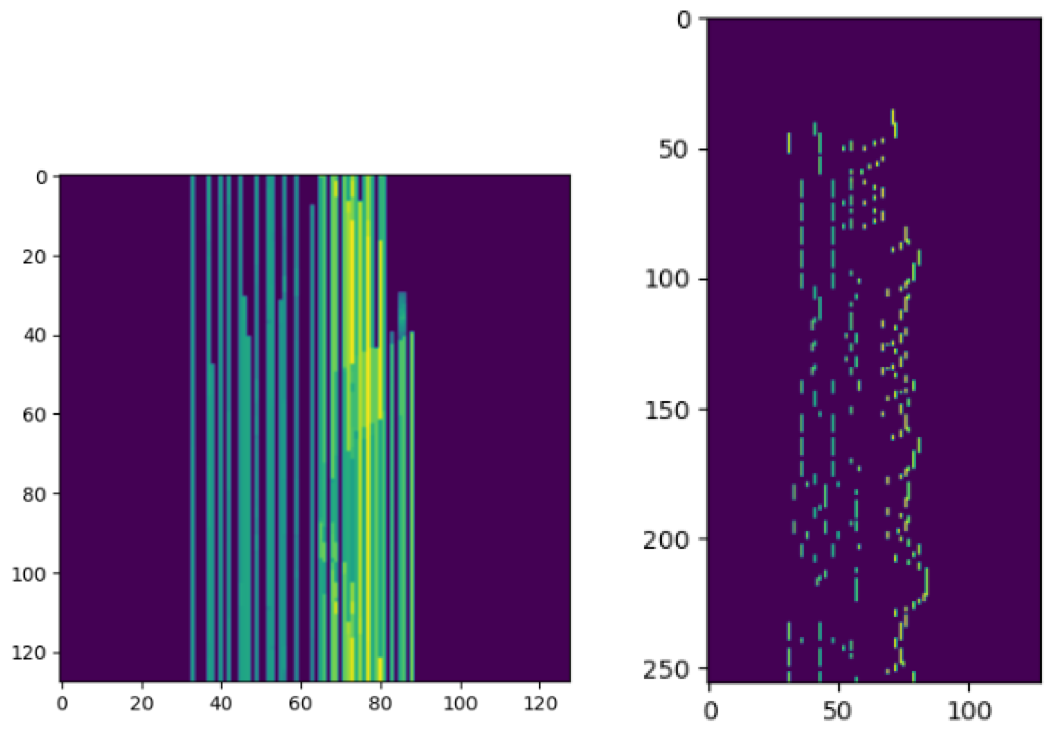


Figure 1: A Mendelssohn song sampled poorly(Left) and sampled correctly(Right).

The reason this happened is because the song is playing at a faster speed than we are sampling causing the notes to blur which makes it hard to recognize patterns in the data. Having a variable sample rate based on the speed of the song would vastly improve the quality of the input data for the model. Bach was our best composer in terms of accurate identification. Looking at his data set sampled at 200 which was our standard it was easy to see why:

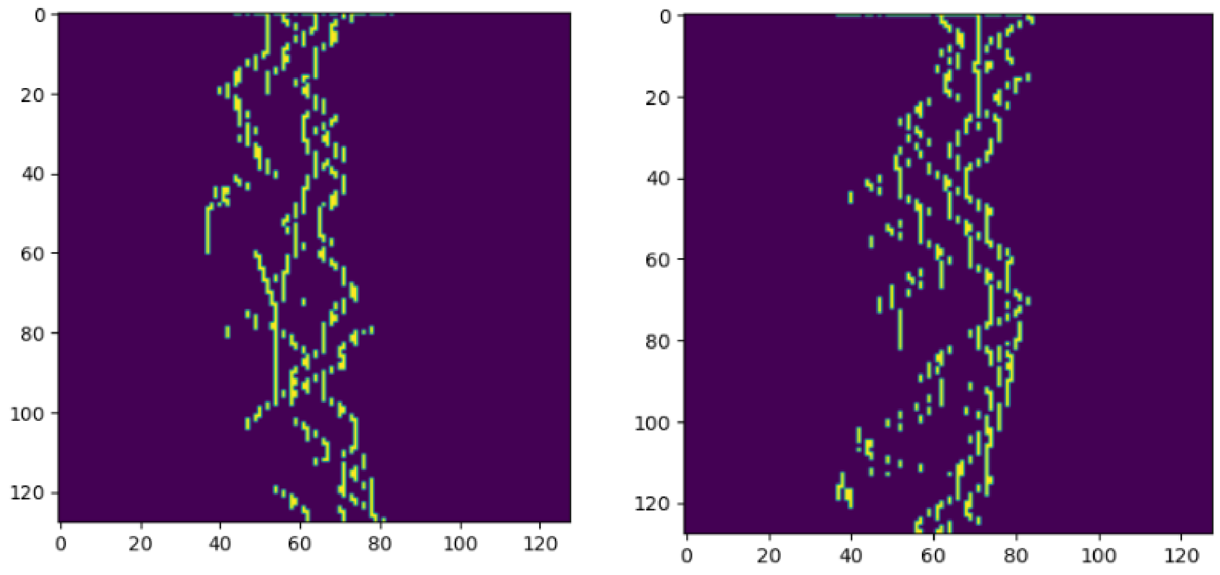
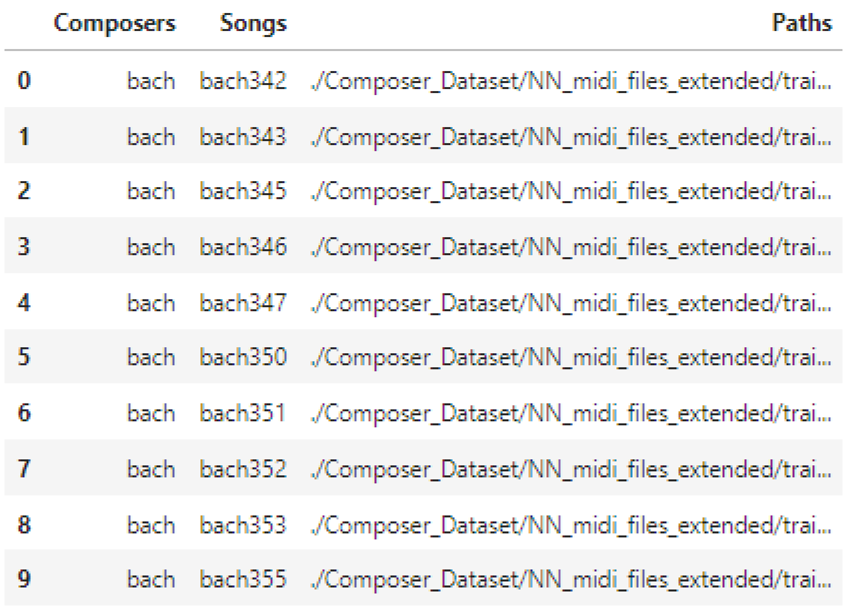


Figure 2: Similarities in Bach’s songs show with visual representation.

Even just visually looking at Bach's songs with the way our data was structured it was clear to see a resemblance between them. The better the sampling is done per composer the better the predictions for that composer became.

Dataset

The data set is a collection of midi files of different composers. To ensure the organization of the data set we created tables with training, and test sets for ease of use (pictured below).



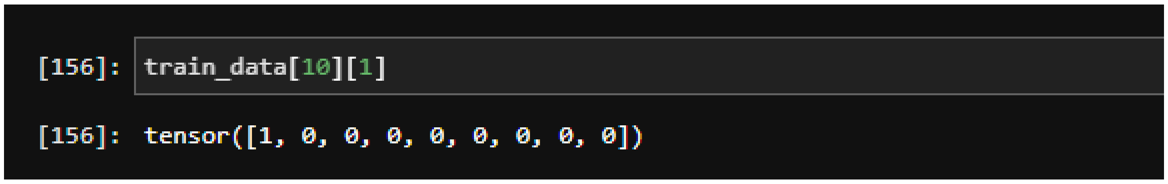
	Composers	Songs	Paths
0	bach	bach342	./Composer_Dataset/NN_midi_files_extended/trai...
1	bach	bach343	./Composer_Dataset/NN_midi_files_extended/trai...
2	bach	bach345	./Composer_Dataset/NN_midi_files_extended/trai...
3	bach	bach346	./Composer_Dataset/NN_midi_files_extended/trai...
4	bach	bach347	./Composer_Dataset/NN_midi_files_extended/trai...
5	bach	bach350	./Composer_Dataset/NN_midi_files_extended/trai...
6	bach	bach351	./Composer_Dataset/NN_midi_files_extended/trai...
7	bach	bach352	./Composer_Dataset/NN_midi_files_extended/trai...
8	bach	bach353	./Composer_Dataset/NN_midi_files_extended/trai...
9	bach	bach355	./Composer_Dataset/NN_midi_files_extended/trai...

Figure 3: Sample of dataset training table.

Internal to the midi there are messages. Each message consists of a note, on or off, channel, note, velocity, and time. Some of the data is structured a little differently so depending on how it's processed it can end up with different results. The data itself is uneven so it needs to be sampled or padded for results. For our usage, we are doing sampling across the data. To artificially expand the data, as well as prevent overfitting to one song, we will 'jiggle' the data slightly. This just means sampling all the data twice but on the second round, we shift the data timewise just slightly. This cannot be done too many times or it will again begin to overfit.

Model Preparation

Since we will be using Pytorch for developing the model there are a few steps that need to be done before the model can be made. The first step is to get the data in the format that will work with the model. This requires figuring out what the model input would look like for a CNN-LSTM architecture for music classification. This leads to deciding to have the CNN layer first (Sainath et al., 2015). The idea of this design is to have the CNN do feature extraction before it is put into an LSTM layer. To get the sequence made inside preprocessing to be a valid input for a CNN we need to add an extra dimension to reflect the channel layer. There is one extra part that needs to be done for the labels that will involve encoding the composers since they are currently stored as strings. This is accomplished through one hot encoding.



```
[156]:  train_data[10][1]

[156]:  tensor([1, 0, 0, 0, 0, 0, 0, 0, 0])
```

Figure 4: Example Label data showing what model will be compared to.

Each column in the tensor will represent a different composer made from the one hot encoding process which means our model output will be outputting a similar tensor.

Model Implementation

For model creation, we started off with a basic CNN model with a few layers which were flattened into Full Connected (FC) layers. The first model had no LSTM layer and was done initially for testing. Below is an example of the initial mode.

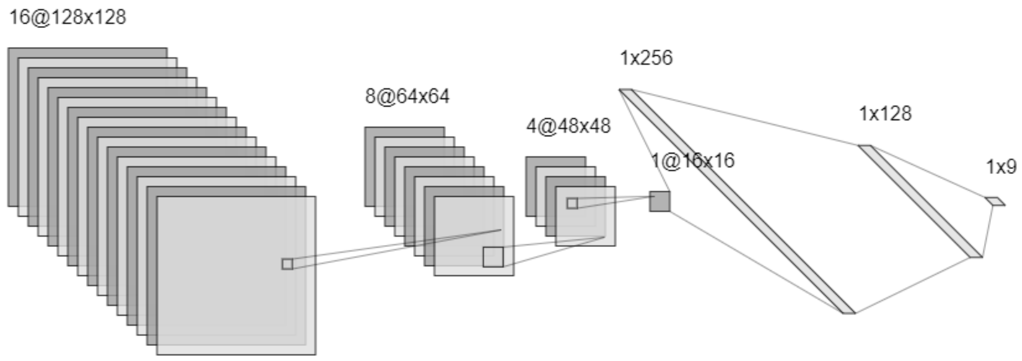


Figure 5: Initial Model Design

The initial model really did not perform to our expectations; it had an f1 score of 20%. Learning from our mistakes, we needed to use a denser model. We decided to incorporate an LSTM layer into our CNN model along with several other convolutional and linear layers to improve performance. This led to a new model shown below.

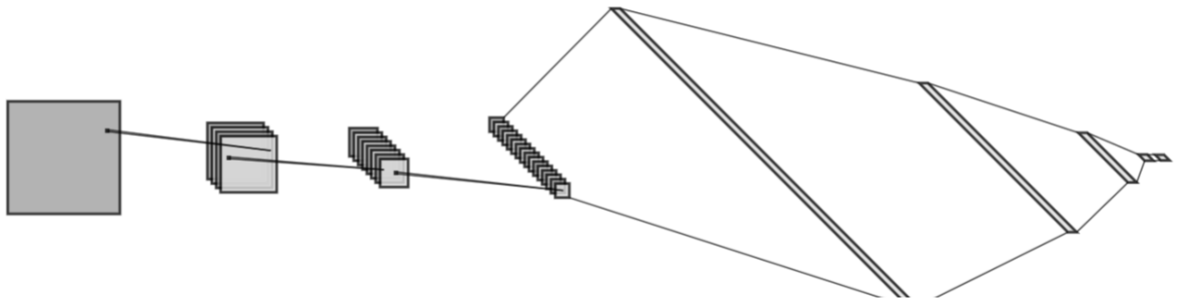


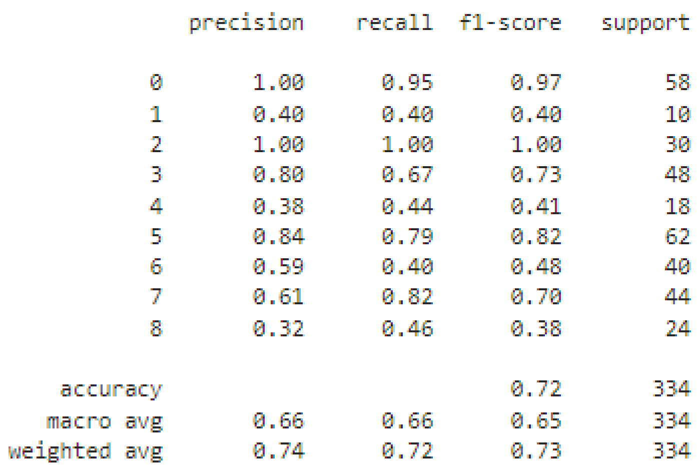
Figure 6: Second Model Design

From the model design shown in Figure 6, the 4th layer is an LSTM layer that flattens out the channel input from the LSTM. This model is beginning to look like a competent model to train our data on. We noticed through our experimentation that the model results perform well when the model is much larger and denser. When we added the extra CNN layer and made the LSTM deeper, we saw greater improvement and consistency whereas before, we had no consistency and our results were suboptimal. The caveat for a larger and denser model when training is the increase in the time it needs to train per epoch and the amount of time it takes to hit a local minimum. This has the possibility of overfitting the model in the train set. We can resolve this error by checking our performance on the test set at the end of every epoch to see if the model is improving. This was done to reduce overfitting and once we were satisfied with what we saw, it became our final model.

One of the methods that was done to assist the training process was transfer learning. This was done by adding an extra CNN or tweaking some layer parameters of the model. We would then load our previously trained model weights into their respective layers to speed up training and model testing.

Results

After countless hours of testing, and modification, followed by even more testing and modification to our dataset and model, we have achieved the results that we are satisfied with. Here are the F1 scores of the list of composers that we worked on. For Bach, we have an F1 score of 0.97, for Bartok, we have an f1 score of 0.43, for Bryd, we have an f1 score of 1.0, for Chopin, we have an f1 score of 0.73, for Handel, we have an F1 score of 0.41, for Hummel, we have an f1 score of 0.82, for Mendelssohn, we have an f1 score of 0.48, for Mozart, we have an f1 score of 0.70, and for Schumann, we have an f1 score of 0.38. For a total accuracy score of 72% from an initial start of 11%, we have greatly improved and elevated our model to new heights.



```
              precision    recall  f1-score   support

           0       1.00      0.95      0.97        58
           1       0.40      0.40      0.40        10
           2       1.00      1.00      1.00        30
           3       0.80      0.67      0.73        48
           4       0.38      0.44      0.41        18
           5       0.84      0.79      0.82        62
           6       0.59      0.40      0.48        40
           7       0.61      0.82      0.70        44
           8       0.32      0.46      0.38        24

    accuracy                           0.72       334
   macro avg       0.66      0.66      0.65       334
weighted avg       0.74      0.72      0.73       334
```

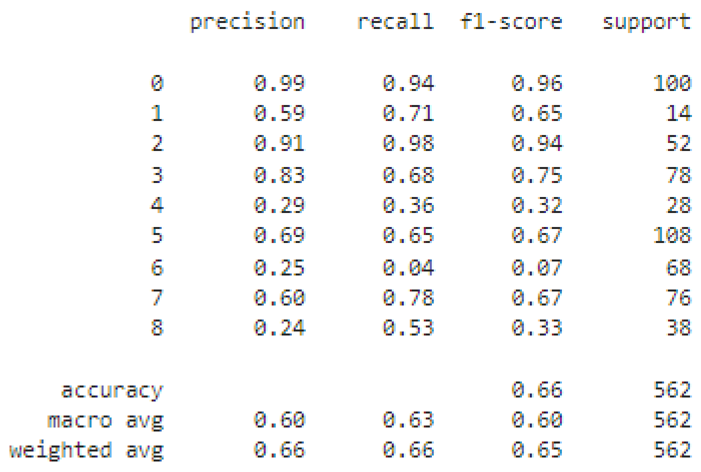
Figure 7: Final Results

Conclusion and Future Improvements

Overall, we achieved significant advancements and improvements in our experiment while exploring the different methodologies for creating our model. Satisfied as we were, we knew that there were a plethora of ways we can enhance and improve our model and results. Below are several elements that we can integrate into our project to enhance its scope.

Recurrent Neural Networks (RNN) thrive with sequential data as they utilize the hidden layers to store information from past inputs. The issue with this method is that it leads to two major problems: exploding and vanishing gradients. Exploding gradients occur when the gradients get too large during training which results in making the model unstable (Wong, 2020). Vanishing gradients refer to when the gradients get too small during training, preventing the weights from ever changing their values (Wong, 2020). Solutions to the exploding and vanishing gradients involve implementing gradient clipping or gradient scaling in order to decrease the likelihood of overflow and underflow. Gradient scaling is the process of normalizing the error gradient vector so that the vector norm equals a defined value while gradient clipping involves forcing the gradient values to a specific minimum or maximum if the range is breached (Brownlee, 2020). In our case, it would be better to utilize gradient clipping to ensure that if the gradients exceed the set threshold value, they would round to the appropriate scaled value.

Variable sequencing to match each song's tempo would be another implementation that we would add in order to accommodate certain sequence sizes. Upon checking the different sequence sizes, we noticed that the model produces better results for some composers than others. With a sequence size of 128, the model is exceptional at identifying Bach and Bryd, however, Mendelssohn doesn't have a very high reading as shown in Figure 8.



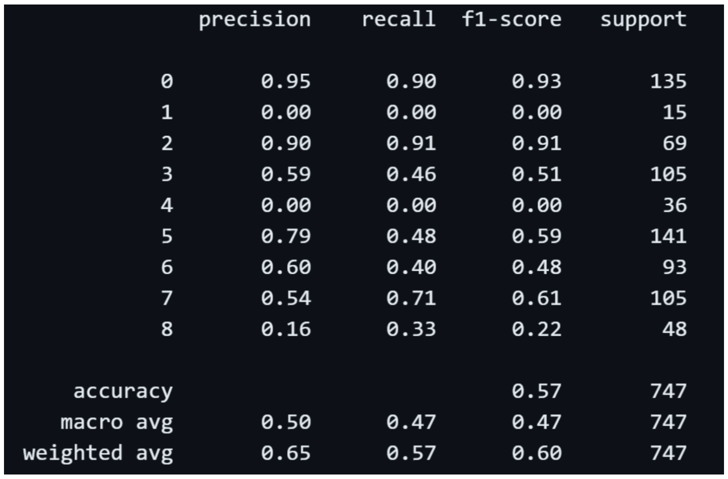
```
              precision    recall  f1-score   support

           0       0.99      0.94      0.96       100
           1       0.59      0.71      0.65        14
           2       0.91      0.98      0.94        52
           3       0.83      0.68      0.75        78
           4       0.29      0.36      0.32        28
           5       0.69      0.65      0.67       108
           6       0.25      0.04      0.07        68
           7       0.60      0.78      0.67        76
           8       0.24      0.53      0.33        38

    accuracy                           0.66       562
   macro avg       0.60      0.63      0.60       562
weighted avg       0.66      0.66      0.65       562
```

Figure 8: Results for the sequence of 128 for all songs

If we change the value of the sequences to 256, we can see Mendelssohn has a higher f1 score than previously shown, but then some other composers like Bartok and Handel take a hit from the changes.



```
              precision    recall  f1-score   support

           0       0.95      0.90      0.93       135
           1       0.00      0.00      0.00        15
           2       0.90      0.91      0.91        69
           3       0.59      0.46      0.51       105
           4       0.00      0.00      0.00        36
           5       0.79      0.48      0.59       141
           6       0.60      0.40      0.48        93
           7       0.54      0.71      0.61       105
           8       0.16      0.33      0.22        48

    accuracy                           0.57       747
   macro avg       0.50      0.47      0.47       747
weighted avg       0.65      0.57      0.60       747
```

Figure 9: Results for the sequence of 256 for all songs

Having a function that takes the appropriate sequence according to the tempo of the song would allow us to accommodate each composer's style of music by selecting the appropriate sequence value.

Our current model does wonders for us, however, we need something more. We need something bigger and larger than the one that we currently have. After testing the model with two convolutional layers and two linear layers, our results were atrocious. The model wasn't converging and our results were stale. However, after implementing an LSTM layer, 2 additional convolutional layers, and 2 linear layers, our results jumped. We were able to get better precision and recall values, better f1 scores, and better accuracy as well. By adding even more layers, we hope to enhance our model's capacity to capture intricate patterns and nuances within the data, potentially leading to improved performance and more accurate predictions.

Through this experiment of crafting sequences, constructing and refining the model, conducting rigorous testing, and iteratively enhancing its performance, the team gained a wealth of invaluable skills and insights. We honed our ability in utilizing Torch, learned how to make use of transfer learning, and even preserved the best model when it had the desired weights that we wanted. This project served as a testament to the synergy of our teamwork. By collaborating on our efforts, encouraging one another, and diligently working countless hours, we were able to amplify our success and finish a project we are truly proud of.

Appendix

A complete version of the code we have developed is available at the following link:

https://github.com/Lyfae/AAI-511-Final-Project

The work was divided equally among the participants of this project.

Tasks	Christopher	Nathan	Paul
Research	x	x	x
Data Processing	x	x	x
PreProcessing	x	x	x
Model Preparation	x	x	x
Training	x	x	x
Compare Accuracy and Results	x	x	x
Report	x	x	x

References

Sainath, T. N., Vinyals, O., Senior, A., & Sak, H. (2015). Convolutional, long short-term memory, fully connected deep neural networks. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. https://doi.org/10.1109/icassp.2015.7178838

Wong, W. (2021, December 13). What is Gradient Clipping? - Towards Data Science. Medium. https://towardsdatascience.com/what-is-gradient-clipping-b8e815cdfb48

Brownlee, J. (2020). How to avoid exploding gradients with gradient clipping. MachineLearningMastery.com. https://machinelearningmastery.com/how-to-avoid-exploding-gradients-in-neural-networks-with-gradient-clipping/