Music Transcription using Deep Learning

Amandeep Singh
GaTech
asingh788@gatech.edu

Ben Bitzko
GaTech
bbitzko3@gatech.edu

Sharad Shashank
GaTech
sshashank3@gatech.edu

Siddharth Sen
GaTech
ssen79@gatech.edu

Abstract

Music transcription is the task of converting an audible musical signal into some form of music notation. It is an essential component in music signal processing that contributes to wide applications in accelerating the development of the music industry, aiding music education and musicology. Through this project, we aim to enable automatic music transcription via state-of-the-art deep learning techniques. We have implemented numerous techniques including CNNs, RNNs, and attention mechanisms, of which the best results were obtained from the Multi-Layer CNN.

1. Introduction/Background/Motivation

For this project, we have built a music transcription model using Deep Learning. Music transcription refers to the analysis of an acoustic musical signal so as to write down the pitch, onset time, duration, and source of each sound that occurs in it. These symbols communicate information about many musical elements, including pitch, duration, dynamics, or articulation of musical notes; tempo, metre, form, and details about specific playing techniques. In Western tradition, written music uses note symbols to indicate these parameters in a piece of music. Some examples of note symbols are shown in the table below.

Symbol	Meaning
♪	eighthnote
♩	quarternote
𝅗𝅥	halfnote
𝅝	fullnote

The traditional approach to music transcription is to transcribe music by ear manually, which can be an extremely cumbersome and time-consuming process, relying on years of knowledge and expertise. Through this project, we aim to accelerate the process to enable automatic music transcription via state-of-the-art deep learning techniques.
Music Transcription is an essential component in music signal processing that contributes to wide applications in accelerating the development of the music industry, aiding music education and musicology. It helps with musical information retrieval, music processing by changing the instrumentation/arrangement/loudness of different parts before resynthesizing a piece from its score, and has applications in Human Computer Interaction e.g. singing transcription. Additionally, it can enable musicological analysis of ethnic music for which musical notations do not exist, and assists amateur musicians play along with more complex musical compositions.
We've used a subset of the MusicNet dataset [3], which is a collection of 330 freely licensed classical music recordings, together with over 1 million annotated labels indicating the precise time of each note in every recording and the instrument that plays each note. This dataset is among the few which have been widely used for the purposes of music generation and transcription.

2. Approach

Our initial plan of action was to work on a couple of different approaches to build an automated musical transcription model. The idea was to learn low-level feature representations of music from raw audio data and predict the notes.

While attempting to leverage the MusicNet dataset for training our models, we encountered a few issues in the process of doing so. The first major issue was the size of the dataset. The MusicNet data obtained from the source expanded to more than 50GB after decompression. This was a major issue, not only for model training but also for data pre-processing and feature extraction. We attempted to resolve this issue by loading the dataset into a GCP bucket and linking the bucket to a Google Colab notebook for read/write access. After data pre-processing and training, we noted that even through the use of GPUs in Colab, our training time was far greater than what was realistically permissible within the scope of our project. We noted a training time of approximately 1 hour for 1 epoch for our initial models. This meant that we would not be left with any room for testing and hyperparameter tuning for improving our results.

Therefore, we decided to pivot our approach and use the mini-MusicNet dataset [2] instead, which is generated from a subset of the initially proposed MusicNet dataset. This dataset consists of n = 82,500 data points with d = 4,096 features and k = 128 binary labels per datapoint. Each data point is an approximately 0.093 second audio clip: these clips are sampled at regular intervals from the underlying MusicNet dataset. Each clip is normalized to amplitudes in [-1,1]. The label on a datapoint is a binary k-dimensional (multi-hot) vector that indicates the notes being performed at the center of the audio clip. The problem then is to predict which notes are being played in any given clip. We define train, validation, and test splits with n = 62,500, 10,000, and 10,000 data points respectively.

2.1. Data Preprocessing

All preprocessing necessary to get from audio files to audio samples and labels is handled in the construct.py file in the mini-MusicNet [2] GitHub repo. This code, which was run unchanged, handled the following steps:

1. A limited number of audio files were selected from MusicNet.
2. 250 equally separated audio samples were pulled from each audio file as a 1-D vector of amplitudes. A time window of 4096 was used, meaning that at a standard audio sampling rate of 44100Hz, this corresponds to about 0.093 seconds of audio.
3. From the corresponding midi file, it is determined what notes are being played at the midpoint of each audio sample. Treating these individual notes as independent classes, one-hot label vectors are generated.

We took these audio samples, and transformed them into Fourier space (amplitude vs frequency) using a Fast Fourier Transform (FFT) on each audio sample.

After Fourier transform, each audio sample had 2049 features, corresponding to amplitudes at 2049 distinct frequencies between 0 Hz and 22050Hz, with linear spacing, meaning that 0 Hz corresponds to index 0, and 22050Hz corresponds to index 2048. Each index then constitutes an approximately 10.8Hz increase.

To further limit the scope of the problem, we focused on only the range of a piano, the highest note being C8 (4186Hz). This should then correspond to index 389 of the FFT. This means most relevant information in Fourier space should be contained within the first 389 indices of the processed data, and we can safely truncate most of this data. For good measure, we truncated at feature 512.

At this point, our processed audio samples were 1-D vectors with 512 features, corresponding to amplitudes every 10.8Hz from 0Hz to 5502Hz, except for the LSTM based models, where we clipped the input to the first 128 features only. Our processed labels were 1-D one-hot vectors of length 81, with each index corresponding to a note between A0 and F7 (F#7 through C8 were truncated because they did not occur in the dataset).

3. Experiments

All models in this section were of our own creation. No code was adapted from any sources.

3.1. CNN: 1-Layer

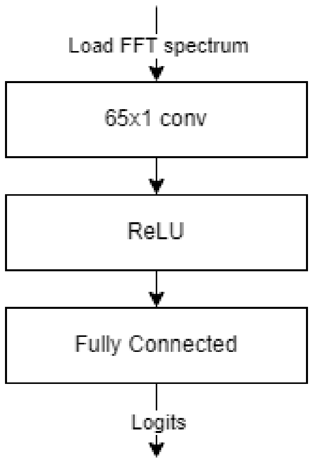
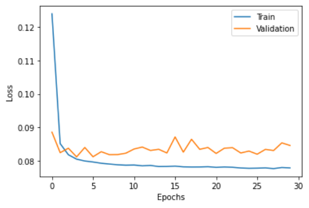


Figure 1: 1-Layer CNN

For our note label prediction task, one of the initial approaches we tried was using a Convolutional Neural Network (CNN) with just a single convolution layer. As this is a relatively simpler model with just one convolutional layer, we can treat this model as a reasonable baseline for future more-complex models.

Since our audio data has been pre-processed to 1-D vectors for both features and labels, we use 1 input channel and 1 output channel in our 1-D convolutional layer. Further, we set the kernel size as 65 while allowing for a padding of 32, stride of 1, along with a batch size of 32. We then pass the outputs of the convolutional layer through a ReLU non-linear transformation followed by a fully connected layer of size 512x81.

For training our model we used the Adam optimizer with a Binary Cross Entropy Loss (with Logits) and computed the average precision score for each training epoch. We trained our conv-net for 30 epochs with a learning rate of 0.001 and obtained a training and validation precision score of 0.61 and 0.56 respectively. The training and validation loss and precision curves can be seen in Figure 2.


(a) Loss Curve

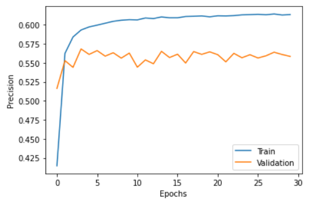

(b) Precision Curve

Figure 2: 1-Layer CNN Loss and Precision

3.2. CNN: Multi-Layer

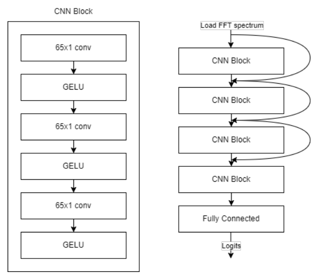
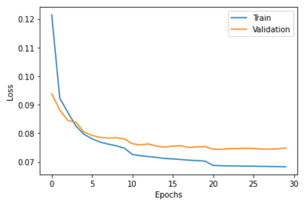


Figure 3: Multi-Layer CNN

We built a deeper 12-Layer CNN based heavily on the 1-Layer CNN. Skip connections were leveraged across every 3 layers to combat the vanishing gradient problem. Variations on this architecture, using different kernel sizes for the convolutions, different numbers of layers between skip connections, and different activation functions were tried, however none were able to outperform the most basic variation, in which the convolutions all kept the same parameters from the 1-Layer CNN.

The most interesting observation had to do with dropout. Every convolution layer had a dropout layer prior to its activation function. However, the best validation precision score was obtained with a dropout of 0. The presence of any level of dropout only served to decrease both the training *and* validation precision scores.

We used the same optimizer (Adam) and loss function (BCEWithLogitsLoss) as with the 1-Layer CNN. In 10 epochs with a learning rate of 0.001, this model achieved a training and validation precision score of 0.63 and 0.58 respectively. To assist in fine tuning, another 10 epochs were run with a learning rate of 0.0005, then another 10 with a learning rate of 0.0001. The fine tuned model achieved a training and validation precision score of 0.66 and 0.61 respectively. This model performed the best out of all our models. The training and validation loss and precision curves can be seen in Figure 4.


(a) Loss Curve

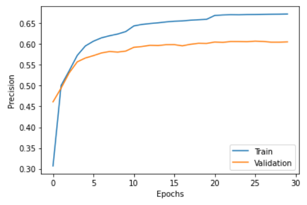

(b) Precision Curve

Figure 4: Multi-Layer CNN Loss and Precision

3.3. LSTM

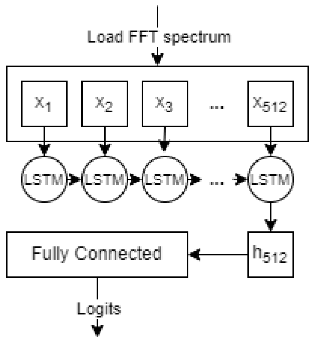
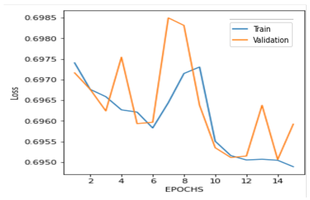


Figure 5: LSTM

To exploit recurrent neural networks for this sequence classification problem, we started off with a vanilla uni-directional LSTM layer followed by a fully connected linear layer. This model seemed to be performing poorly with the training and validation precision score being 0.23 and 0.19 respectively. The training and validation loss and precision curves can be seen in Figure 6.


(a) Loss Curve

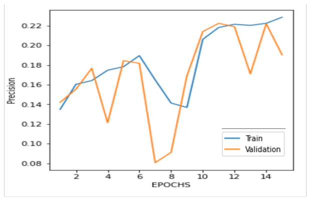

(b) Precision Curve

Figure 6: LSTM Loss and Precision

3.4. LSTM with Attention

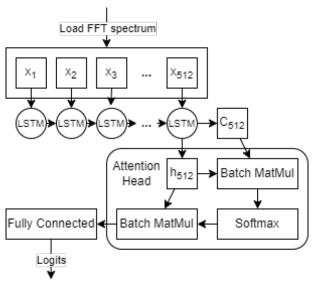
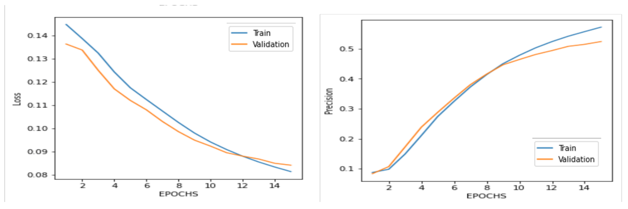


Figure 7: LSTM with Attention

Through this model we incorporate Attention mechanism in our LSTM model. We use attention to compute soft alignment score corresponding between each of the hidden state and the last hidden state of the LSTM. This model significantly improved the performance of the LSTM network. We witnessed the training and validation precision score going as high as 0.57 and 0.52 respectively. The training and validation loss and precision curves can be seen in Figure 8.

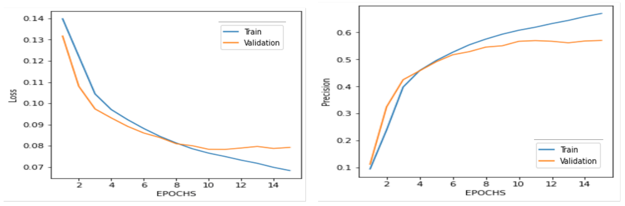


(a) Loss Curve (b) Precision Curve

Figure 8: LSTM+Attention Loss and Precision

3.5. LSTM with Self-Attention

This model is inspired from the work on structured self-attentive sentence embeddings [1]. It consists of two components, a bidirectional LSTM and a self-attention mechanism, which provides a set of summation weight vectors for the LSTM hidden states. These set of summation weight vectors are dotted with the LSTM hidden states, and the resulting weighted LSTM hidden states are considered as the vector representation for the discrete frequencies. The Self Attention model seemed to be performing quite well with the training and validation precision score going as high as 0.67 and 0.57 respectively. The training and validation loss and precision curves can be seen in Figure 9.



(a) Loss Curve (b) Precision Curve

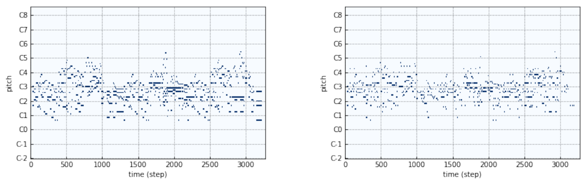
Figure 9: LSTM+Self Attention Loss and Precision

4. Results

The results and hyper-parameters from all models are summarized in Table 1. The Multi-Layer CNN model performed best out of all models, beating the performance of all of Thickstun's models [3] except for his CNN (which achieved a validation average precision of 0.678). This is particularly exciting because all models in Thickstun's paper were trained on the whole MusicNet database, which is more than 30 times larger than mini-MusicNet.

5. Final Musical Transcription

As previously mentioned, few audio segments were used from each recording in training to get samples from as many different musical contexts as possible, but the end goal of this project is to apply the model to a complete audio file, rather than just miscellaneous segments.



(a) True Transcription (b) Predicted Transcription

Figure 10: Transcriptions of '2575.wav'

'2575.wav' (Beethoven Piano Sonata 8 "Pathetique", 2nd movement) was processed as described in Data Preprocessing, except that it was divided into sequential time windows without any gaps. This was then passed into the Multi-Layer CNN, resulting in the transcription given in Figure 10. This transcription result is quite encouraging! It looks incredibly similar to the true transcription. It has the same clear structure and patterns, and seems to be missing some notes, but looks to have very few of the wrong notes.

Listening to the midi renderings is even more encouraging (the True Transcription and the Predicted Transcription can be heard on YouTube). It's worth noting that the True

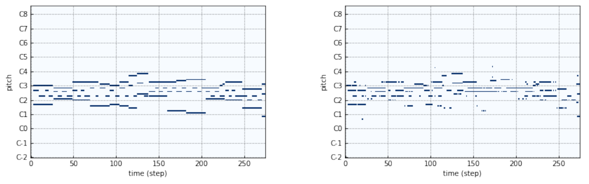
Model	Architecture/Hyper-Parameters	Train Loss	Val Loss	Train PrScore	Val PrScore
CNN 1-Layer	Optimizer: Adam, Loss: BCEwithLogits, Kernel Size: 65; Padding: 32, Epochs: 30, lr: 0.001, batch size: 32	0.078	0.084	0.61	0.56
CNN Multi-Layer	Optimizer: Adam, Loss: BCEwithLogits, Kernel Size: 65; Padding: 32, Epochs: 30, lr: 0.001, 0.0005, 0.0001, batch size: 32	0.070	0.076	0.66	0.61
LSTM	Optimizer: Adam, Loss: BCEwithLogits, 1 LSTM layer, dropout=0.8, Unidirectional, hidden size=1024, Epochs: 15, lr: 0.0001, 0.001, 0.01, batch size: 32	0.695	0.695924	0.23	0.19
LSTM+Attention	Optimizer: Adam, Loss: BCEwithLogits, 1 LSTM layer, dropout=0.5, Unidirectional, hidden size=1024, Epochs: 15, lr: 0.0001, 0.001, 0.01, batch size: 32	0.081	0.08416	0.57	0.52
LSTM+Self Attention	Optimizer: Adam, Loss: BCEwithLogits, 1 LSTM layer, dropout=0.5, Bi-directional, hidden size=1024, da = 350, Epochs: 15, lr: 0.0001, 0.001, 0.01, batch size: 32	0.068	0.0792	0.67	0.57

Table 1: Summary of Model Results

Transcription does sound a bit off with respect to the timing of notes. This is because of how the data was preprocessed. The ’correct notes’ in a given time window are taken at the middle of that time window. This means that when we go to reconstruct the original piece, whatever notes are being played at the middle of each 0.093 second clip will be projected to the entire clip, even if the actual note changed during that clip.

As for the Predicted Transcription, there are two immediately obvious observations to make:

1. It sounds a bit chaotic. While the vast majority of note pitches are correct, the notes are getting re-attacked seemingly at random. This makes sense though. The midi file format is event based. If a note is on in adjacent time steps, it will only be attacked in the first time step, and will sustain through the following time steps. If, by mistake, the note is off in one of the time steps through which it should be sustaining, midi will interpret this as a new occurrence of that note, and re-attack it. Our model seems to miss notes somewhat often, which would easily result in every sustained note getting re-attacked at random.
2. The few extra notes that seem to stick out in the Predicted Transcription piano roll do not obviously stick out in the audio. In fact, no incorrect notes at all seem to stick out in the audio. This implies that all notes that are incorrectly present are not dissonant with the notes that are supposed to be there. Perhaps these notes were predicted because higher frequencies in the harmonic series were ringing out unusually loudly above some note that was actually being played.



(a) True Transcription (b) Predicted Transcription

Figure 11: Truncated Transcriptions of ’2575.wav’

Ideally both issues above would be avoided, but these results are encouraging. They show that the model can learn pitch recognition very well. Seeing these issues is better than getting a bunch of incorrect and dissonant notes. Taking a closer look at a shorter section of the piano roll in Figure 11 though brings up an additional issue. While some notes aren’t being fully captured, we can clearly see in these piano rolls that some notes are being extended (particularly noticeable between time steps 150 and 200). If we listen to a real recording of this section, we will notice that the notes that our model extended are actually audible outside of when they are strictly defined in the True Transcription. This brings up a flaw in our dataset and approach.

6. Limitations and Future Work

Reverberation, whether due to the audio being recorded in an echoey room, a sustain pedal on a piano being active, or any other reason, causes notes in the recording to extend beyond their notated duration. In many cases (a piano sustain pedal in particular, since it disables the mechanism that dampens a note) it is impossible to tell from audio alone whether the musician stopped actively playing the note. This means that much of the data on which the model was trained may have had labels that seemed contradictory to the actual sound. As a result, the states of whether a note is on or off are not as separable on the basis of an audio clip as initially believed. This makes it unlikely for our approach to improve much more, and fundamentally puts a limit on the precision that such a model can achieve.

It's possible that a model might be able to learn how to predict whether a note is being sustained or not if it has context about when the note started, or what is happening before and after a given time window. The approach explored in this project did not incorporate any such context, and considering this context would require a fundamental shift in how the data is preprocessed.

Perhaps, rather than pulling single, isolated windows from each audio file, we should pull sequences of adjacent windows. We would be adding a dimension to both the data and labels corresponding to the index of that window in our ordered sequence of windows.

7. Work Division

The delegation of work among team members is shown in Table 2. All team members were active and helpful at all points in the project.

Student Name	Contributed Aspects	Details
Amandeep Singh	Literature Survey, Data Creation and Report	Surveyed existing methods, Scraped the data for this project, and contributed to the Final Report.
Ben Bitzko	Implementation and Visualization	Implemented the CNN architectures and experimented around with many variations. Modified the preprocessing code to process an entire audio file. Built the example piano roll visualizations and audio files.
Sharad Shashank	Implementation and Analysis	Implemented the LSTM architectures as well as the attention variants. Experimented with variations and analyzed the impact and success of these model variations.
Siddharth Sen	Environment Setup, Implementation and Report	Setup GCP pipeline with the goal of operating on entire MusicNet data. Implemented single-layer CNN and contributed to final report.

Table 2: Contributions of team members.

References

[1] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017. 4

[2] John Thickstun. mini-musicnet. `https://github.com/jthickstun/mini-musicnet`, 2022. 2

[3] John Thickstun, Zaid Harchaoui, and Sham Kakade. Learning features of music from scratch. *arXiv preprint arXiv:1611.09827*, 2016. 1, 4