Neural Music Transcription with Spatiotemporal Vision Models

Ansh Sharma, anshgs2@illinois.edu
Albert Xiao, anxiao2@illinois.edu
CS 543 - Fall 2022

Abstract

Music Transcription is a task that has often been left to experts in music theory in the past - requiring transcription of polyphonic audio into playable notes. However, with recent advances in deep learning, image and audio processing techniques alongside deep computer vision techniques have shown remarkable promise in automating this task.

In this project, we explore two computer vision-inspired models, Fully Convolutional Neural Networks and Recurrent Convolutional Neural Networks, and evaluate them on the MAESTRO dataset. We further explore different techniques to guide the model's training through designing a loss function with custom weightings for a cross-entropy-derived loss function. We also compare the performances of the two models with all factors held constant except the final head to directly compare results. Beyond this, we also consider the use of several transformer-based models, but report insufficient performance with a naive implementation as well as a few variations due to reasons explored in the appendix.

Full code is available here: https://github.com/AlbyYuggle/CS543Project/

1. Introduction

Music Transcription is a fairly difficult task without extensive training due to the difficulty in distinguishing rhythms and pitches between multiple overlapping notes in songs. As such, gaining access to transcribed sheet music for songs can often be prohibitively expensive or difficult for beginners who are unsure of whether they will even be able to play a given song. The field of Automatic Music Transcription provides a new option, aiming to convert audio files into a readable format by musicians. Prior research has explored both monophonic transcription, in which the audio file contains a single instrument that plays one note at a time, as well as polyphonic transcription[2] which can contain multiple notes at each time step, played by one or multiple instruments.

In our project, we want to focus on piano transcription in particular, making the problem at hand polyphonic in nature, but with only a single instrument, placing it somewhere toward the middle of the spectrum in terms of task difficulty. With this restriction, we only have two main subtasks to work towards: identifying note frequency and duration. From a very broad viewpoint, our approach will center around a spectrogram rather than raw audio data, allowing us to use computer vision-based techniques to extract notes from the spectrogram in image space. Prior works focusing on audio primarily utilize CRNNs, Convolutional Recurrent Neural Networks, applied to melspectrograms in order to capture the spatial and temporal natures of the problem[2, 3, 5]. For our contribution, we aim to explore and compare alternatives to this: in particular, the main alternatives we're considering at the moment include fully connected CNNs as a baseline as well as Transformer based architectures.

2. Methodology

2.1. Data Collection and Processing

Our dataset for this project was the MAESTRO dataset [4], which includes 120 GB of pairs of audio wav files and midi files for piano music. The original dataset contains 1276 wav files (around 200 hours of total recording) and corresponding MIDI files which contain information about note onset, offset, velocity, and pedal information. For the scope of this project, we only focus on the note onset and offset portions to decrease the complexity of the problem. In order to make this data usable for training, we need to standardize the formats of all of the audio recordings, by taking 10-second intervals from the audio files and converting them to log Mel Spectrograms (Figure 1), with the following parameters: n_fft=2048, win_length=2048, hop_length=160, n_mels=229, f_min=30, f_max=8000. This creates around 71,000 total input-output pairs.

In order to create a predictable output format, we convert the midi files to a binary NumPy image, with 1 representing the note being played/held, and 0 representing a note not being pressed (Figure 2). Since different midi files have different ticks per second or tempo, we standardize these values by bilinearly interpolating.

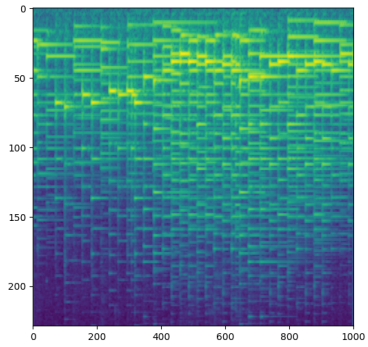

Figure 1. Input Spectrogram Visualization

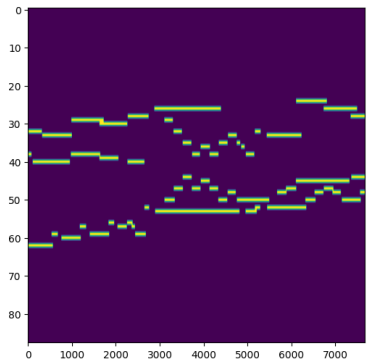

Figure 2. Output MIDI Visualization

Now that we have processed and standardized data, we can try different deep learning architectures to attempt to predict the MIDI representation from the log Mel Spectrograms. The three architectures we chose to investigate were Fully Convolutional Neural Networks, Recurrent Convolutional Neural Networks, and Sequence-to-sequence transformer models. Finally, after predicting the results, we convert the predicted NumPy array back into a midi file, which can be played or converted to sheet music by music production tools.

2.2. Fully Convolutional Neural Networks

Our inspiration for a Fully Convolutional Neural Network was taken from the U-NET[10], and the idea that our input and output are both images, so we can downsample the input spectrogram to learn the features, and upsample it back to decode the features into the desired output. Furthermore, we added skip connections to help identify more global features and help with gradient flow. We also took inspiration from ResNet[11], as we used residual connections, making it easier to learn weights for convolutions. Combining these key ideas from these notable papers, we arrive are our model, which is shown in figure 3. Another important factor is the Sigmoid last layer, which limits the output from 0 to 1, which is exactly what we are trying to predict.

2.3. Recurrent Convolutional Neural Networks

For the CRNN model, we drew inspiration from the Bytedance Piano Transcription Model [3], which uses a CRNN with many GRUs to solve a more complex problem that incorporates pedal, velocity, on/offset and frame as different learned components put together for the final transcription. Since we simplified the problem by neglecting components such as pedal and on/offset prediction, we would require a less complex model making it more feasible to train locally. As such, we took inspiration and used a GRU in place of our Conv1D layer in our FCNN model, and kept everything else the same. The motivation for this additional recurrent layer is that music is inherently time oriented, so we thought that the GRU would be able to capture some of these properties. The overall architecture is shown in figure 3.

2.4. Transformers

2.4.1 Seq2Seq Transformer Model

As mentioned previously, music is inherently time and order oriented, meaning the encoding of sequential/temporal relationships is expected to be key in producing accurate predictions. Because, of this, we thought that a sequence-to-sequence transformer model may be a good attempt at predicting an output. We began with a standard seq2seq model detailed in [9]. We added a linear layer and Sigmoid layer at the end of the model to reshape and bound the output to give us a binary prediction for each key at each time step. Overall, we found that the results for the transformer were extremely inadequate. The model tended to predict a constant image over all columns, as the model learned simply to predict the same response as the previous time step, as many notes are held for a beat or more. More details about this model's failure can be explored in Appendix A.

2.4.2 Transformer Encoder

Seeing the failures that occurred in the Seq2Seq model, we decided to try an encoder-only model, shown in figure 4. The main components of this model include the positional encoding, which helps the model learn the ordering properties of the input, the multiheaded attention layer, which helps the model find context between different features in the inputs, and the Conv1d+Sigmoid at the end which is specific to our problem. This is because the output of the base transformer has the same size as the input, and the Conv1d seeks to shift that to our output space, while the Sigmoid bounds the output. Overall, the model still predicted constant row outputs, which can be further explored in Appendix A.

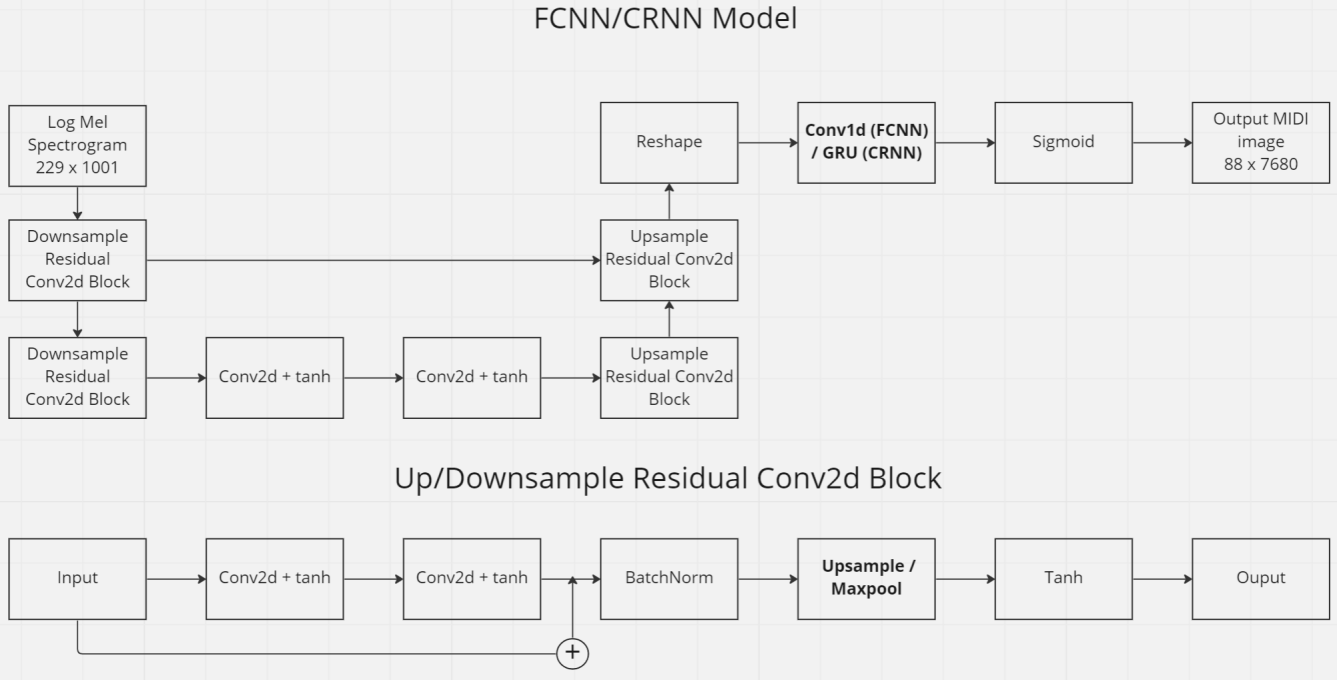

Figure 3. FCNN/RCNN Architecture

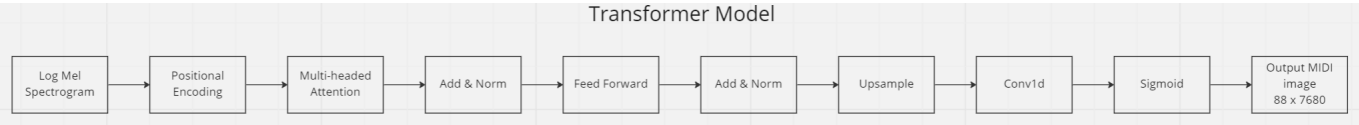

Figure 4. Transformer Encoder-Only Architecture

3. Experiments

We explore each of these models on the MAESTRO dataset. After subdividing the dataset into separate test and train splits, we standardize our training between CRNN's and FCNN's by making the remainder of the setup as identical as we can, consisting of using ADAM as our optimizer with learning rate 3e-4 and weighted variants of cross-entropy as our loss function. We evaluate models using a set of robust metrics as opposed to purely accuracy for reasons we will detail more in section 3.2 - more specifically, we provide precision, recall, and F1 scores for each run in addition to the accuracy. To standardize comparison, we evaluate each model after 1 epoch and then again after 2 epochs. We do note that most of the models would likely see significant improvements given even longer training time, as an observation of their raw predictions as opposed to rounded do seem to suggest that they are still in the process of learning - however, due to computing constraints for this project, we decided to constrain each model to just 1 epoch to gain a fair comparison.

3.1. Loss Function Design

When designing a loss function, we initially defaulted to a standard MSE loss in hopes that it would enforce visual similarity to the desired outputs. However, we quickly observed that this led to training collapse where the model would effectively learn to output a low probability for every square and predict all 0's when rounded, as each image has significant class imbalance with around 98% empty space and 2% notes, which makes sense given that a piano player only plays a couple notes out of the 88 provided at any given point. This led the model to easily learn that it can maximize MSE by simply prioritizing the 98% and ignoring the actual notes, which is far from the desired behavior.

We find that a more traditional loss for classification problems, Binary Cross Entropy, works far better for our settings due to penalizing the model for outputting incorrect answers with high confidence. Given that we are also enforcing the constraints of 1 epoch of train time, we sought to find out ways to modify the loss function that would result in quicker training. As such, we settled on testing out variants of BCE with weighting. More specifically, we took

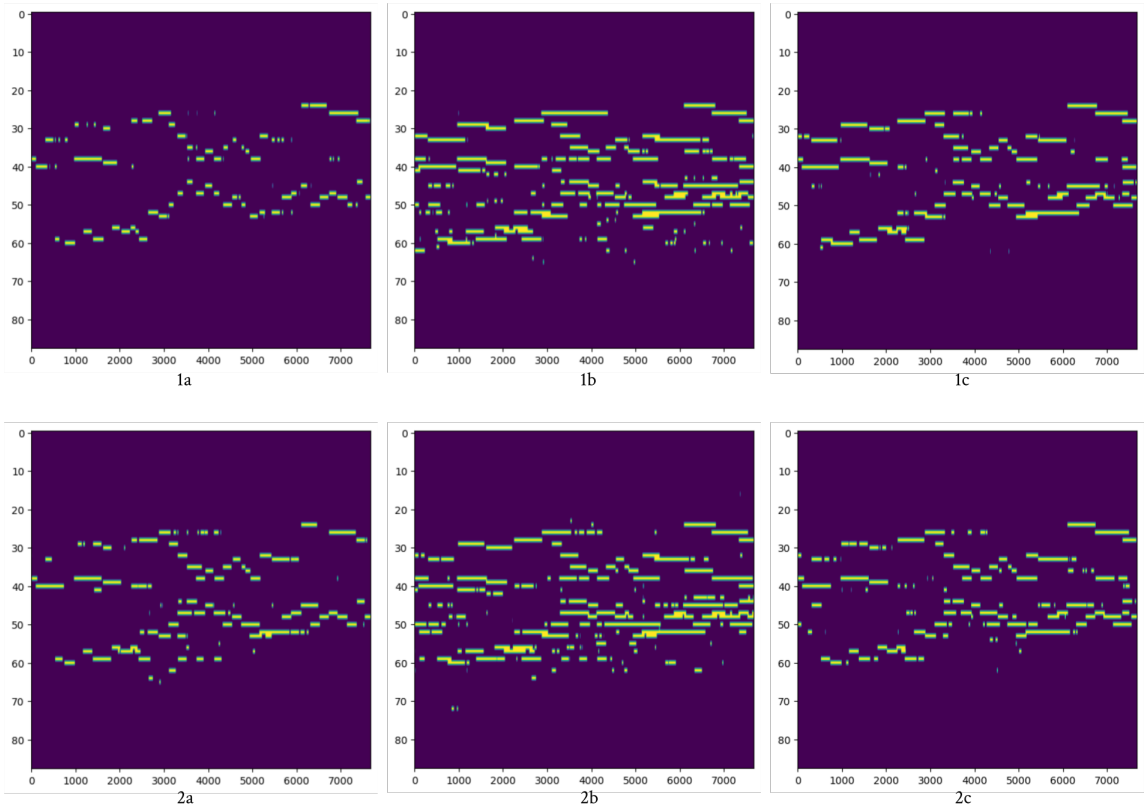

Figure 5. Visual Comparison of Results. 1a-1c contain outputs for the FCNN with weightings of $\beta = 0.05, 0.5, 0.2$. Similarly, 2a-2c contain the corresponding outputs for the CRNN model with the same set of weights. The target image for this audio sample is contained in Figure 2 within section 2 for reference. Observe the difference in note prediction density between the different weightings, but relative consistency between the two models.

Architecture	β	F1	Precision	Recall	Accuracy
CRNN	0.05	0.1903	**0.7435**	0.2563	**0.9811**
CRNN	0.5	0.2252	0.2997	**0.9102**	0.9495
CRNN	0.2	**0.2940**	0.4863	0.7453	0.9763
FCNN	0.05	0.2318	**0.6865**	0.3522	**0.9816**
FCNN	0.5	0.2326	0.3129	**0.9096**	0.9524
FCNN	0.2	**0.2970**	0.4785	0.7622	0.9756

Table 1. Model Performance after 1 Epoch of Training

the assumption that a pianist had around 5% of the notes being played at any given timestep, and chose a reweighting factor of β to revalue the importance of positive versus negative notes as follows (where y is the true value and $\hat{y}$ is our predicted probability):

$$\mathcal{L}(y, \hat{y}, \beta) = \frac{0.95}{1-\beta}(1-y)\log(1-\hat{y}) + \frac{0.05}{\beta}y\log(\hat{y})$$

We explore 3 possible weightings - the first being an unweighted comparison with $\beta = 0.05$ as our estimate for the proportion of notes being played at any given time step. Additionally, we consider $\beta = 0.5$ and $\beta = 0.2$ for both the CRNN and the FCNN. Visualizations are provided for a sample on each of the 6 created models in Figure 5. For quantitative comparison, we evaluate our metrics after 1 and 2 epochs, detailed in tables 1 and 3.

We note in Table 1 that the $\beta = 0.2$ reweighting has the best F1 score in both cases by a sizeable margin, suggesting that there is likely an optimal weighting between a perfect balance and the original weightings that leads to bet-

ter results after initial training. It is likely that all of these methods may converge to similar F1 scores given sufficient time, but given our computational and temporal limitations for this project, we primarily observe the performance after the first few epochs.

Architecture	Parameter Count	Runtime per Epoch
CRNN	384552	7:27
FCNN	321544	5:21

Table 2. Model Complexities

3.2. Model Complexity Comparison

At a high level, we sought to make the two models as close as possible computation-wise in order to make the comparison fair. As such, the backbones of the models are identical, with the only difference being the predicting heads. Parameter counts between the two models are quite similar as well and are provided below alongside runtime in Table 2.

We can note that the runtime for the CRNN is reasonably more than the runtime for FCNN for a single epoch - there are multiple potential reasons as to why this may be the case. Due to the larger memory requirements for the CRNN, we were required to use a slightly smaller batch size (16 instead of 32), which may have increased runtime a bit due to reduced parallelization. Accounting for this and the slight increase in total trainable parameter count, the increased runtime seems about in line with what we'd expect.

3.3. Further Training Time

During training, we also observed that it seemed as if both model architectures could still reasonably gain performance within another epoch or so due to the raw predictions often having the right outputs but just not with enough confidence to round to a 1 as seen in Figure 6. As such, we decided to continue training each model with a second epoch as well to get a better understanding of if either model pulls further ahead of the other. Results for the second epoch of training for each of the same settings as in the first epoch results are provided in Table 3 alongside their relative improvements over the first epoch. Note that our models would likely continue to see improvement over multiple further iterations due to still underfitting the training data, but as stated before we are limited by computation and time for the scope of this project.

We note that both models demonstrate modest improvements on the scale of low single-digit percentage increases, though the outputs are not drastically different from a visual standpoint for the most part. We do see after the second epoch that the CRNN does outperform the baseline FCNN for the best F1 score by a comfortable margin, which is different from after just one epoch where they had around the same performance. This seems to suggest that the recurrent nature of the CRNN's final layer is able to generalize slightly better, but this is only really seen after a longer training span. It is likely that this trend may continue for further epochs, but we are unable to explore that at the moment due to computational restrictions.

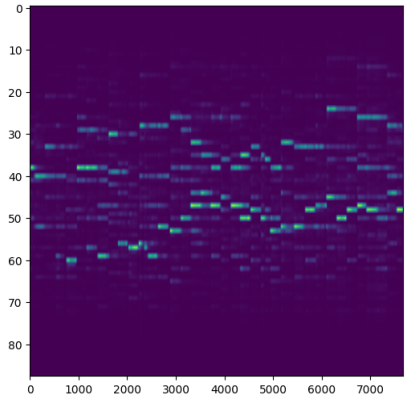

Figure 6. Raw prediction outputs for Unweighted CRNN Epoch 1

3.4. Learning Visualization

We additionally seek to understand the learning process for the models qualitatively to get a sense of how it learns the correct mappings. As such, we visualize early checkpoints for each model weighting in Figures 7, 8 and 9 - we specifically focus on the FCNN results as the CRNN results appear qualitatively similar, with the weightings making the key differences.

All three outputs are taken after 0.2 epochs of training, but the differences between them are quite interesting to note. We note that in the first visualization (Fig 7), which was trained with the default binary cross-entropy loss, the model has yet to learn anything meaningful about the exact structure of the music. However, what is interesting to note is that it seems to have gained a notion of beats as well as musical key, as there seems to be a very clear grid-like pattern present in the prediction even if the desired structure is far from visible. This seems to suggest that the model first gets an idea of when notes are likely to start/end as well as which notes are likely to be played given the input.

On the other hand, both $\beta = 0.5$ and $\beta = 0.2$ show remarkably more structure in their outputs after even just 0.2 epochs of training. The fully re-balanced dataset tends to be quite a lot noisier at this stage which we'd expect, while the intermediate balanced dataset provides fairly reasonable results qualitatively even at this early on in the training process, showing the impact of our reweighting

Architecture	β	F1	Precision	Recall
CRNN	0.05	0.2673 (+0.0770)	**0.7791 (+0.0356)**	0.4069 (+0.1506)
CRNN	0.5	0.2366 (+0.0114)	0.3176 (+0.0179)	**0.9319 (+0.0217)**
CRNN	0.2	**0.3101 (+0.0161)**	0.5100 (+0.0237)	0.7925 (+0.0472)
FCNN	0.05	0.2599 (+0.0281)	**0.7532 (+0.0667)**	0.3985 (+0.0463)
FCNN	0.5	0.2461 (+0.0135)	0.3371 (+0.0242)	**0.9134 (+0.0038)**
FCNN	0.2	**0.3043 (+0.0073)**	0.4998 (+0.0213)	0.7795 (+0.0173)

Table 3. Model Performance after 2 Epochs of Training

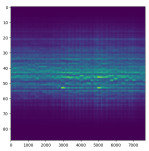

Figure 7. FCNN Partial - $\beta = 0.05$

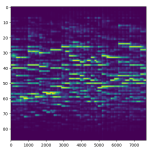

Figure 8. FCNN Partial - $\beta = 0.5$

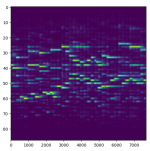

Figure 9. FCNN Partial - $\beta = 0.2$

3.5. Sample Audio Outputs

To further facilitate qualitative comparison between our results, we provide audio samples for each of our models and loss weightings generated for two 10 second segments. All outputs are available as MP3 files in our GitHub repository in the following folder https://github.com/AlbyYuggle/CS543Project/tree/main/outputmp3. From these two examples, we can immediately notice more audible results of our models. Although previously we could see that the MIDI image looked like it matched, it was more important to hear whether or not the transcription really sounded the same as the audio. From listening to the two examples provided, we can see that the general melody and harmony are well preserved in the transcription. However, the main difference between the different loss functions that we experimented with was the amount of noise added to the transcription. In our musical opinion, it is better to omit more subtle portions of the transcription than to add extra noise, as omission still allows for the clear acknowledgment of the main melody, harmony, counterpoints, and other music features, whereas noise begins to muddle these musical ideas. As such, we can hear that the results that we deemed best visually indeed did sound better, as they had significantly less noise, which corroborates our claims.

4. Discussion

In this project, we've explored neural transcription through three different spatiotemporal deep learning-based computer vision techniques. We were able to attain reasonable results qualitatively and quantitatively with two of the three models, though we observed that our third transformer-based architecture seemed to be a poor fit for this task when naively applied to the dataset. Nonetheless, our CRNN and FCNN methods both performed comparably, with CRNN gaining seeming improvements both quantitatively and qualitatively the longer we trained. This seems to suggest that the recurrent head of the CRNN does provide benefits to the models learning as opposed to the local temporal convolutions which only consider the temporally adjacent features when computing the final output. However, the FCNN architecture performs at least comparably within the first few epochs, only lagging slightly behind, suggesting that it still remains a fairly viable method.

We also explored the impact of loss-function reweighting and observed that it did make quite a significant difference with respect to how the model learns within the first few epochs. As opposed to a default cross-entropy or an artificially fully balanced dataset, we find that an intermediate weighting produces the best results with regards to F1 score and qualitatively.

Future work on this project would likely require further training for both the CRNN and FCNN, as we have seen that they both appear to benefit from further training and will likely continue to improve in performance for another few dozen epochs before overfitting - however, this would require more time and/or computational power than we have at hand at the moment for the scope of this project. We may also explore more nuanced modifications for the transformer-based architectures such as customized loss functions to avoid the model collapse that we observed in each of our experiments detailed in the appendix.

5. Statement of individual contribution

Both team members worked collaboratively in person for the majority of this project. In terms of the main responsibilities of each team member: Albert was primarily responsible for data acquisition and cleaning, designing scripts to extract and convert the data into a conducive format for the models, and converting the outputs back into midi files. Albert was also responsible for maintaining the codebase and repository. Ansh was primarily responsible for the model design and implementation, creating the architectures and training loops in PyTorch and running the experiments.

6. References

[1] Ruoyan Chen, Yiwen Liu, Automatic Music Transcription, http://cs230.stanford.edu/projects_fall_2020/reports/55773193.pdf

[2] Miguel A. Román, Antonio Pertusa, Jorge Calvo-Zaragoza, A holistic approach to polyphonic music transcription with neural networks, https://arxiv.org/pdf/1910.12086.pdf

[3] Qiuqiang Kong, Bochen Li, Xuchen Song, Yuan Wan, and Yuxuan Wang. "High-resolution Piano Transcription with Pedals by Regressing Onsets and Offsets Times." arXiv preprint arXiv:2010.01815 (2020). [pdf]

[4] Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset." In International Conference on Learning Representations, 2019.

[5] Gupta, G., Kshirsagar, M., Zhong, M. et al. Comparing recurrent convolutional neural networks for large scale bird species classification. Sci Rep 11, 17085 (2021). https://doi.org/10.1038/s41598-021-96446-w

[6] Datasets: https://paperswithcode.com/datasets?task=music-transcription

[7] Ming Liang and Xiaolin Hu, "Recurrent convolutional neural network for object recognition," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3367-3375, doi: 10.1109/CVPR.2015.7298958.

[8] Lea, Colin, et al. "Temporal convolutional networks for action segmentation and detection." proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

[9] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[10] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015.

[11] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[12] Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

7. Appendix A: Transformer Failures

As detailed in section 2, we designed multiple transformer-based models in hopes of utilizing its successes in sequential modeling for NLP towards this task as well. However, we immediately ran into several issues, mostly centering around the highly dense nature of our input and output which made transformers quite difficult.

7.1. Seq2Seq Failure

Our initial approach utilized the standard sequence 2 sequence implementation of a transformer as detailed in [9]. We utilized a transformer architecture with 2 encoder and 2 decoder blocks, alongside an internal feed-forward network of dimension 200.

Once training, we quickly noted however that this model was highly problematic. Our initial problem came due to the quadratic nature of the self-attention mechanism which made our space requirements a lot higher for this model and forced us to reduce batch size massively. We compromised on this by reducing our training inputs to 1-second intervals instead of 10 to allow for more reasonable batch sizes.

Even so, this didn't help our other major problem - the model effectively came to learn that the most likely output for any time step would be the previous time step, leading it to give reasonable results in training as the mask only hid the current stage and afterward while proving abysmal during evaluation as it would nearly always output constant notes for the entire span. We attempted to shift back the input that was provided even further, but the model seemed to always collapse to the same state of predicting constant values as shown in Figure 10.

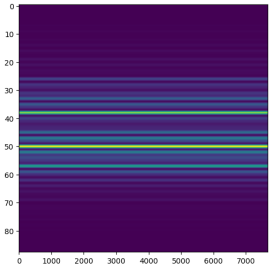

Figure 10. Constant output prediction for transformer architecture

7.2. Iterative Generation Failures

As a secondary attempt, we realized that rewarding the model during training for only predicting the next stage was an extremely limiting task, as the obvious solution for it would be to duplicate the previous input resulting in constant note streams. Instead, we sought to iteratively update only the next timestep's prediction and force the model to use it's own predictions for subsequent predictions, making it much more difficult to predict the same output each time.

This may have worked in theory, but we ran into significant issues in implementing this in practice. To begin with, this increased the computation time immensely, as we would now have to run the model 768 times per input (or 7680 if we were using the full 10-second inputs), which slowed down training significantly. Furthermore, this extremely large computation graph proved far too large to keep track of on our hardware for gradient calculations, forcing us to attempt to zero out the gradient every 16 time steps or so. At this scale, however, we no longer were free of our initial issue, as the model could once again learn that it was optimal to predict the same thing for 16 time steps (which evaluates to around 1/50th of a second), placing us effectively back at our original issue as there often isn't too much change even over that span.

7.3. Blind Seq2Seq Failure

In hopes of circumventing this issue entirely, we attempted to feed in fully blank inputs to the decoder, only containing positional information. Yet even this model seemed to collapse to the same state, for reasons we are unsure of why. We suspect that this may just be due to the dense nature of our inputs and the model being unable to learn significant distinguishing features from the positional encoding that we are feeding in. Potentially experimenting with the design for the positional encoders may help, but at this point, we decided that it would make more sense to fully scrap the decoder component of the model as we were anyways barely feeding in any meaningful information.

7.4. Encoder Only Failures

As a final attempt, we decided to simplify the architecture to be encoder only, utilizing a linear layer on the output, upsampling, and then utilizing a Conv1d layer as in our FCNN to create our desired output shape. In this case, we finally got a slightly different output - it predicted an additional note at one point in the middle that wasn't there earlier! Unfortunately, that was the only improvement it provided – for the most part, the output was constant and there was effectively no visually discernible understanding of the inputs. We are still unsure as to why this is the case, but it seems like the model is simply unable to lower the loss beyond this constant input - perhaps longer training and more experimentation with the loss function and optimizers may be able to alleviate this issue, but we decided at this point that it would be better to focus our time on the other two models given their better results.

It is possible that we missed out on common solutions to utilizing transformers for dense time series due to our lack of experience with the model, but we surmise after exploring these potential fixes that finding a transformer-based solution is non-trivial with our current background.