An AI Approach to Automatic Natural Music Transcription

Michael Bereket
Stanford University
Stanford, CA
mbereket@stanford.edu

Karey Shi
Stanford Univeristy
Stanford, CA
kareyshi@stanford.edu

Abstract

Automatic music transcription (AMT) remains a fundamental and difficult problem in music information research, and current music transcription systems are still unable to match human performance. AMT aims to automatically generate a score representation given a polyphonic acoustical signal. In our project, we approach the AMT problem on two fronts: acoustic modeling to identify pitches from a frame of audio, and establishing a score generation model to convert exact piano roll representations of audio into more 'natural' sheet music. We build an end to end pipeline that aims to convert .wav classical piano audio files into a 'natural' score representation.

1. Introduction

Music transcription is still considered to be a difficult task even by human experts. For polyphonic audio, AMT faces a number of challenges: the acoustical signal of concurrent notes can have complex interactions, there can be large variations in audio signals between instruments, and the combinatorial output space is incredibly large. However, recent progress has been made with AMT through the use of neural networks. In the 2016 paper "An End-to-End Neural Network for Polyphonic Piano Music Transcription", Sigtia, Benetos, and Dixon describe a hybrid recurrent neural network (RNN) model for polyphonic AMT of piano music, which achieves state of the art performance [13]. Their model uses a Convolutional Neural Network (CNN) to tackle acoustic modeling (identifying pitches from a frame of audio) and an RNN based music language model (understanding the temporal structure of musical sequences for piano roll generation). In our project, we break down our music transcription system into two parts: acoustic modeling for pitch identification in polyphonic audio, and score generation for converting the resulting piano roll representation into 'natural' sheet music.

For acoustic modeling, we aim to transform polyphonic audio of classical piano music into a piano roll representation by predicting the presence of notes in each time frame. The input to our acoustic model is a time-frequency representation of audio frames. We then use a Convolutional Neural Network (CNN) to output a predicted set of notes that are present in the relevant audio frame. Our model closely follows the work of Sigtia et al. Thus, for each time slice in a given song, we can predict the corresponding notes, and by aggregating the outputs, we can construct our desired piano roll representation.

Music is often performed with an element of emotional expressiveness; as a result, the observed rhythms and tempos in the audio recordings of piano performances are often irregular. However, the outputs of our acoustic model provide us with exact representations of our original audio, corresponding precisely to the manner in which the piano piece was performed. If a score were to be exactly generated from this output without further transformation, unnatural and inconsistent patterns may arise, and it would not likely resemble a human experts transcription due to these irregularities. We tackle this issue by constructing a 'natural' score generation model consisting of two phases: tempo selection with rhythm bucketing, and smoothing. Tempo selection handles the matter of defining a tempo which we can visualize our score relative to (e.g. the length of quarter-note or dotted-eighth-note is only well-defined when a particular tempo is assumed). We encode our piano roll representation as note events, and we use a linear model which takes in a song segments note events as input and predicts the top k candidate tempos for that segment. Our smoothing step utilizes a Hidden Markov Model (HMM) to predict the original rhythm buckets (as written by a composer) given a series of observed rhythm buckets.

We completed this project for both CS221 and CS229 and have focused on the acoustic modeling for this class and the score generation task (tempo selection with bucketing and smoothing) for the CS221 project.

2. Related Work

3. Literature Review and Related Work

There has been substantial progress made in the field of automatic music transcription. AI techniques, and in particular neural networks, have met and surpassed the performance of traditional pitch recognition techniques on polyphonic audio, and we examined many different AI approaches to the AMT problem before deciding how we would model our project. For instance, LSTMs, which have been applied for sequence modeling in a variety of uses such as speech and text analysis, have been shown to be effective for modeling polyphonic musical sequences [3]. Many different neural networks have also been examined and compared for their effectiveness at framewise piano transcription [2]. Commercial systems, such as Anthem Score, offer AI-powered automatic transcription [1]. However, these services still face significant challenges in accurately predicting note occurrences and generating natural looking scores. Anthem Score approaches music transcription from the perspective of image recognition as well, and their utilization of a CNN as well inspired us to further explore CNNs for our acoustic model.

CNNs have risen in popularity in recent years, especially with

tackling computer vision tasks [5]. The task of note detection for music transcription can be treated similarly to image recognition, as images of the time-frequency representations of audio can be created. However, some new challenges arise with note detection. Music notes are not localized to a single region in the same way that most objects in images are—a note at some fundamental frequency will be composed of harmonics at multiples of that frequency [1]. There is also interference among neighboring harmonics, which is analogous to an image classification task involving overlapping and transparent objects. Despite these challenges, CNNs have advantageous properties that can be applied to AMT. Previous experiments have suggested that aggregating information over several frames to inform a prediction can yield higher performance, and taking convolutions over input data allows our model to learn valuable features from polyphonic musical data.

The work of Sigtia et al. has also explored various models for pitch detection in the acoustic model. They explore Deep Neural Networks (DNNs) and Recurrent Neural Networks (RNNs) in addition to CNNs, and their results have shown that their CNN-based model outperform the others for the acoustic modeling task. We have used their CNN architecture as our guidance for constructing the acoustic model. In their paper, they further propose a Music Language Model (MLM) that utilizes RNNs in order to tackle polyphonic musical data, which generally poses a challenge for MLMs that utilize Hidden Markov Models (HMMs) [8].

4. Dataset and Features

4.1. Input Preprocessing

The dataset we used consists of 138 MIDI files of human performances of classical piano pieces [12]. To generate input for our acoustic model, we needed to first transform each MIDI file into raw audio data in .wav format. We downsampled the audio from 44.1 kHz to 16 kHz, and then we convert each .wav audio file into a time-frequency representation with the Constant Q Transform (CQT). An example result of the CQT is shown in Figure 1 (top). CQT represents amplitude against a logarithmic frequency scale, and this results in geometrically spaced center frequencies, thus maintaining linearity in pitch [9]. Furthermore, fewer frequency bins are needed, so we also have a reduction in input features. We compute CQT over 7 octaves with 36 filters per octave for a total of 252 filters (features per frame). We set our hop length to be 512, so the transform considers 512 samples per frame. This ultimately corresponds to a final frame rate of 16,000/512 = 31.25 frames per second. We normalize each of the 252 features across all the frames in our dataset by subtracting the mean and dividing by the standard deviation of each feature.

4.2. Ground Truth Labels

Each MIDI file encodes information about the audio by specifying note-on and note-off events. We generate our own ground truth labels by unraveling the note events in each MIDI file and creating a binary vector of size 88 for each time slice (using the same frame rate as before, 31.25 frames per second). This binary vector encodes whether the i^{th} note was present during the particular time slice. Concatenating the labels for a series of frames would result in a piano roll representation such as the one shown in Figure 1 (bottom).

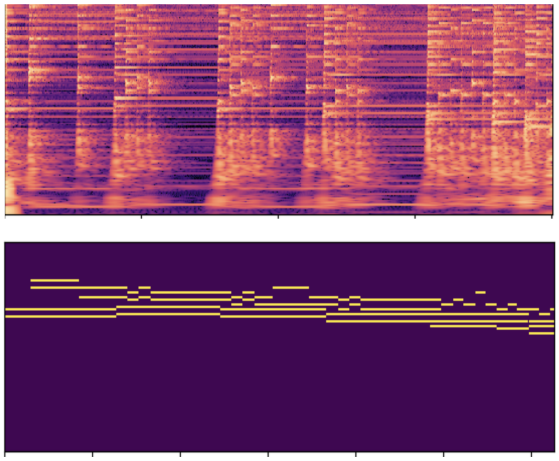


Figure 1: CQT representation of a .wav file over a 20 second time interval (top) and ground truth piano roll representation from MIDI file over the same time interval (bottom)

4.3. Experimental setup

After this stage of preprocessing, we are left with a dataset of 250850 frames in total, where our features are of dimension 252, and our labels are of dimension 88. We split our data into train and test sets, with 200680 frames (110 songs) for training and 50170 frames (28 songs) for testing (4:1 ratio). As input into our CNN for acoustic modeling, instead of passing in just a single frame as input, we pass in a context window of frames to the model. For the purpose of our initial experiments, we used a window size of 7 frames [13], where our task is to predict for the center frame in each window given as input. Thus, one single input would have dimension (252, 7), and the corresponding label is still of dimension 88.

5. Methods

5.1. Music Transcription as Image Interpretation

For our acoustic model, we have decided to use a Convolutional Neural Network (CNN). CNNs have risen in popularity in recent years, especially in the field of computer vision [5]. In computer vision, an input image is given to the CNN, and the image is passed through multiple layers, such as convolutional layers, pooling layers, and activation layers (where non-linearities can be applied). Convolutional layers consist of multiple filters, which can be interpreted as each learning some higher level feature of the image. In the AMT problem, note detection can be treated similarly to image recognition, as images of the time-frequency representations of audio can be created. Note identification in music is simpler than image classification in several ways: there are not many important textures to learn, and no rotation or scaling is involved [1]. However, note identification poses other new challenges. Music notes are not localized to a single region in the same way that most objects in images are—a note at some fundamental frequency will be composed of harmonics at multiples of that frequency [1]. Additionally, there is interference among neighboring harmonics, which is analogous to an image classification task involving overlapping and transparent objects. Despite these challenges, CNNs have advantageous properties that can be applied to AMT. Previous experiments have suggested

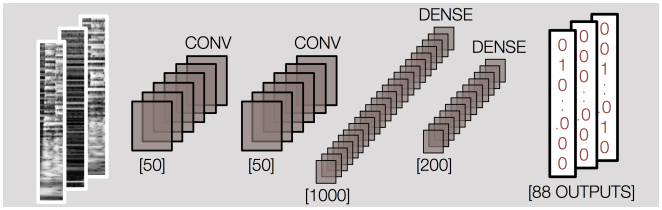


Figure 2: CNN architecture

that instead of simply classifying a single frame of input, better prediction accuracies can be achieved by aggregating information over several frames. Thus, given a context window of frames as input, a CNN model can perform convolutions across the frame axis and utilize neighboring information when producing a prediction for the center frame. Furthermore, along with the usage of the CQT for our input time-frequency representation, we can use CNNs to learn pitch-invariant features as we take convolutions along the frequency axis. Through the use of pooling layers and weight sharing (as opposed to all fully connected layers) in our CNN, we can also reduce the number of parameters in our model.

5.2. CNN Architecture

Our implementation is guided by architecture described in the previous work of Sigtia et al. We have started off with a network consisting of two convolutional layers and 2 fully connected layers, with an output of 88 neurons (one corresponding to each key on the piano). In the first convolutional layer, we have 50 filters, and a kernel size of (5,25), where 5 is along the frame axis and 25 is along the frequency axis. We follow this layer with a hyperbolic tangent activation layer, and we follow with a Max-Pooling layer using a pooling size of (1,3) (pooling only over the frequency axis) and use 0.3 Dropout. The second convolutional layer also has 50 filters, but instead with a kernel size of (3,5), where 3 is along the frame axis and 5 is along the frequency axis. We also use the hyperbolic tangent as the activation function for this layer, as well as the same Max-Pooling layer and 0.3 Dropout. For each convolutional layer, we use He normal initialization to randomly initialize the weights. We then follow immediately with 2 fully connected layers. The first layer consists 1000 hidden neurons, and the second layer consists of 200 hidden neurons, and each layer uses a sigmoid activation, with a dropout rate of 0.3 for each layer as well. Our goal was to begin with an architecture that was similar in structure to what Sigtia et al. had shown to be valid via a grid search over these hyperparameters, and then we would further adapt our model as necessary if we found that it wasnt as successful.

We finally create our output layer of 88 neurons, each fully connected from the previous layer. These also use a sigmoid activation function. Thus, for each neuron in the output layer, the prediction can be interpreted as the probability of the associated piano key being 'on' during the center time frame of the input. For each neuron, we use binary cross-entropy as our loss function. The architecture is summarized in Figure 2.

During training, we used a stochastic gradient descent optimizer with 0.9 momentum. We began with a constant learning rate of 0.01, and in our experiments, we iteratively refined a learning rate decay schedule, training over a total of up to 40 epochs.

6. Metrics

Our primary metrics for evaluating the performance of our acoustic model were accuracy and F1-score. For each time frame, we treat our CNN prediction as a composition of 88 binary targets. Thus, if we are evaluating our model over n frames, our accuracy is evaluated over 88n targets. Our F1-score would also be evaluated over 88n targets. The F1-score can be interpreted as a weighted average of recall and precision, and an ideal F1-score would be 1.0, while the worst would be 0.0 [11]. In particular, if we let TP denote the number of true positives, FN denote the number of false negatives, and FP denote the number of false positives:

$$\text{Recall } R = \frac{TP}{TP + FN}$$

$$\text{Prediction } P = \frac{TP}{TP + FP}$$

$$\text{F1-score } = \frac{2 \cdot R \cdot P}{R + P}$$

Intuitively, we can understand precision and recall as a means to understand and quantify the relevance of our predictions. More concretely, recall is the fraction of the present notes that are actually successfully identified, and precision is the fraction of identified notes that are actually truly present. Using F1-score allows us to handle the extreme class imbalance present in our data, since there are much more 0s than 1s in our targets. Measuring accuracy alone would not let us effectively evaluate our models performance (if we output all zeros, our accuracy would already achieve around 96%).

7. Experiments and Results

We initially started with a very similar CNN architecture used in Sigtia et al. and we began our first training experiments locally on a smaller dataset of only 25 Mozart songs, as well as with reduced fully connected layers (200 hidden units in each layer). With a constant learning rate of 0.01 (learning rates of this magnitude were used in Sigtia et al.), we began running experiments with 0.5 Dropout. During these training runs, our model was inclined to output all zeros (i.e. predict that no notes appeared in any time frame). We knew that our model was definitely suffering from a lack of data, so as a result, our acoustic model did not have an opportunity to learn the features well enough. We decided to remedy this by acquiring more data and setting up our model to run with actual GPUs by using Google Cloud Machine Learning Engine resources. Before transitioning to running on Google Cloud, we decided to try and tune our dropout rate, and in particular, we wanted to verify that our network was even capable of learning any structure of our training data, and we were curious about whether reducing some regularization would combat underfitting and lead to any positive or interpretable results. We found that using a dropout rate of 0.3 instead of 0.5 would actually enable our model to recognize the presence of some notes, resulting in an F1-score of 32% (30% recall and 36% precision) and 99% accuracy on our validation set within 25 epochs.

Our next experiments involved our full dataset of 138 songs, and we ran these on Google Cloud using a Tesla K80 GPU. We focused on further tuning our dropout rate, as well as tuning the number of hidden units used in our fully connected layers. These were some of the main hyperparameters that differed from the

work of Sigtia et al., and we wanted to see if we could reproduce similar results using similar architecture for our CNN. After experimenting with dropout rates within the range 0.1 0.7 in a binary search manner, we ended up settling with a dropout rate of 0.3, which gave us our best performance on our validation set. We found that the best performance was achieved by using 1000 hidden units in the first fully connected layer, and this was our final configuration that we ended up using for the remainder of our experiments. With a constant learning rate of 0.01, we were able to achieve 44.34% F1-score and 98.15% accuracy on our validation set.

However, we found that our training performance was plateauing relatively earlyafter about 10 epochs, our models progress seemed to become stagnant as the F1-score continued to jump around 44% for the remaining epochs. We deduced that this plateau was due to training with too large of a learning rate, resulting in imprecise updates. We decided to tackle this problem by introducing a learning rate schedule. We tried 3 different step decays: 1) starting at a learning rate of 0.1 and then halving every 5 epochs, 2) starting at a learning rate of 0.05 and then halving every 10 epochs, and 3) using a learning rate of 0.05 for the first 5 epochs, dropping to 0.025 for the next 7, dropping to 0.0125 for the next 9, and dropping to 0.00625 for the remaining epochs. The secomd learning rate schedule ended up achieving the best performance and was effective at breaking the plateau. Using this step decay schedule, we were able to achieve our best results on our validation set, shown in Figure 3. This is likely because the initial larger learning rate is only needed for a few epochs in order to make more dramatic training progress, and for later epochs, the model can spend more time refining its performance with reduced learning rates.

An example prediction and its corresponding ground truth is shown in piano-roll representation in Figure 3. This example shows that our CNN can successfully learn to identify most pitches, even in polyphonic audio with overlapping notes played in the same time frame. There is still some noise picked up and not all notes are identified for their full duration. However, this is likely to be due to the fading volumes of some notes; while the MIDI file would show the full duration of the note, our audio input would not hear the end of some notes with the same amplitude, and thus, our CNN model would not be able to identify the trailing ends as easily. One potential way to tackle this would be to lower our threshold for classifying a note as present to allow less confident predictions of 'on' notes to still be counted, although this may pose problems with introducing more noise in other areas besides the fading ends of correct notes. Nevertheless, we believe that our model is able to learn significant features of the audio, as it sufficiently captures complex note patterns and is able to recognize multiple pitches even when they interact and overlap.

For our train set, we achieve 74.09% F1-score and 99.81% accuracy using the third learning rate schedule (manual step decay). We believe that this indicates that we have overfit to our training set, and we believe that this can be remedied with more regularization and by exposing our model to more varied data. There is also an incredibly large space of possible notes sequences and combinations that could potentially appear in a song. Since our dataset only spans 6 composers and 138 songs, one of the issues that could be hindering our acoustic models performance would be the limited note patterns present in our data. We can remedy this by either acquiring more MIDI files of other songs across many more composers, or by generating synthetic data to train our model with. With synthetically generated data, we would then be able to ensure that our model is continuously exposed to a wide variety of acoustical signals, and thus we would increase the distribution of data that our model is trained on.

8. Score Generation

The second component of our music transcription pipeline is to take the output of our acoustic model and generate a corresponding 'natural' music score. We detail this step further in our report for CS221. We separate score generation into two problems: tempo selection with bucketing for initial score generation, and smoothing for refining. A score cannot be generated without a tempo to interpret rhythms relative to, and note durations are primarily expressed as part of a standard set of rhythmical values (for example, a quarter note, which is often 1 beat, or an eighth note, which is often half a beat). In order to generate our initial score, we need to select a tempo that aligns the observed durations as best as possible with the expected rhythmic buckets. We then apply a natural language model to attempt to smooth this initial output.

While we will not be going into much detail about the implementation and model structure of this component, we would like to provide an overview of how the data is further transformed from the output of our acoustic model.

8.1. Tempo Detection and Bucketing

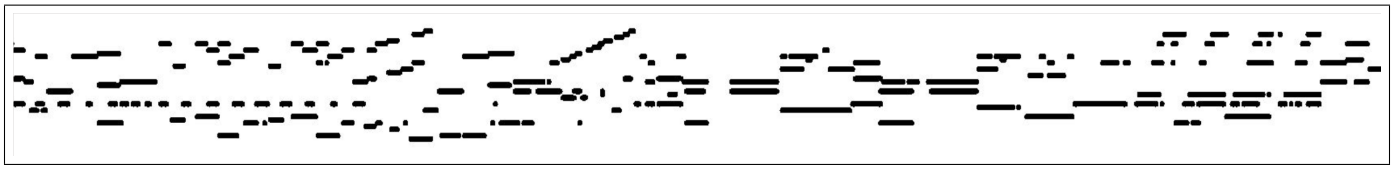
The tempo detection and bucketing module takes as input any observed features that are expected to occur 'on beat' and selects a tempo to minimize the observed distance between the durations of these features and ideal bucket durations. To do this, we defined a loss function to that measures how off beat a series of observations is and used stochastic gradient descent to optimize our tempo parameter to minimize this distance. Tested against ideal observed note and rest durations for songs without multiple tempos, our method predicted at least one tempo in the top four candidates sufficiently close to the approximate 'true' tempo to achieve less than 18% note by note error against the bucketed original composition. We also saw good performance when the technique was applied to not onset differences on acoustic model output.

8.2. Smoothing

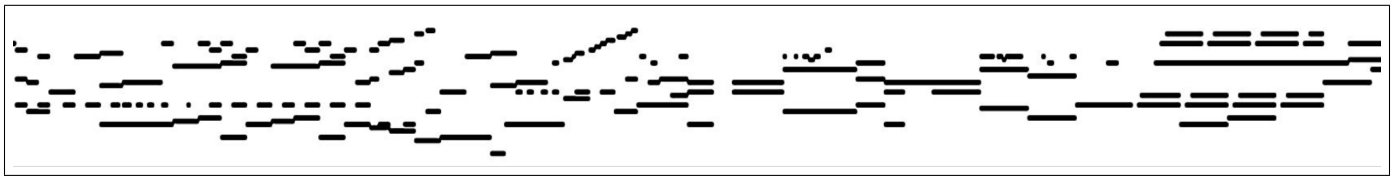
We used a Hidden Markov Model (HMM) as a natural language model to attempt smoothing on existing transcriptions. We modeled our hidden states as the bucketed originally composed rhythms, our emissions as the observed bucketed rhythms (given a good selection of tempo), our transition probabilities as n-gram probabilities over hidden states, our emission probabilities as multinomial, and our start probability as uniform over all states. We tested various forms of inference, maximizing likelihood conditioned on all emissions for both individual and the entire sequence of hidden states. We found that this model gives too much weight to expected transition probabilities and attempts to make all rhythms look similar (removing the uniqueness of songs).

LR Schedule	Accuracy	F1-score	Recall	Precision
Initial LR = 0.1, halving every 5 epochs	98.33%	52.85%	50.10%	59.74%
Initial LR = 0.05, halving every 10 epochs	**98.72%**	**55.07%**	**52.54%**	**60.51%**
Manual step decay	98.72%	54.45%	51.91%	59.49%

Table 1: Validation set results with step-decay learning rate schedules after 40 training epochs.



(a) CNN prediction



(b) Ground Truth

Figure 3: a) A 20 second segment of example predictions visualized as a piano roll.
b) The corresponding ground truth piano roll of the same 20 second segment.

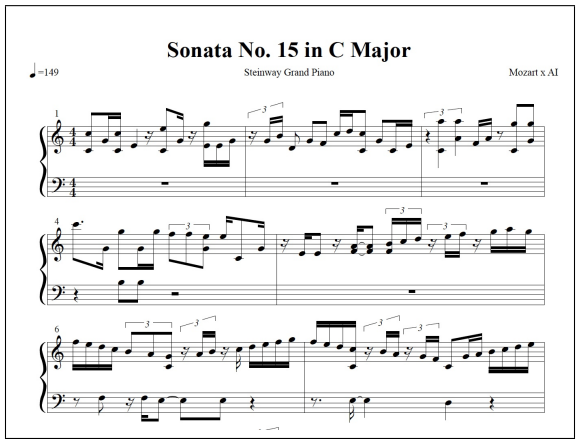


Figure 4: Our full pipeline transcription (acoustic model and tempo detection) of Mozart's Sonata No. 15 in C Major

9. Conclusion and Future Work

In this project, we have implemented an end-to-end pipeline to convert .wav piano audio files into a 'natural' score-like representation by breaking down the AMT problem into two main components: acoustic modeling for pitch detection, and score generation. Here, we have presented our acoustic model, a Convolutional Neural Network that can identify the presence of notes in a given frame of audio. We have found that our model can successfully identify pitches in substantially complex polyphonic audio, and in conjunction with the tempo selection and smoothing of our score generation model, we are able to generate corresponding scores for our raw audio. Our experiments support the advantages and effectiveness of CNNs for acoustic modeling that Sigtia et al. explored in their work.

We expect that we can achieve higher performance by acquiring more polyphonic piano audio and generating synthetic data, and random noise can be incorporated into our training examples as well to make our model more robust to to a wider range of input audio. Furthermore, we would consider exploring other input representations for our acoustic model. Instead of using the CQT representation, we could utilize the Mel-scaled short-time Fourier transform [10], or other representations with higher temporal resolution such as the variable-Q transform [13]. Other loss functions could also be more appropriate—a weighted loss function for each node would be able to assign more consequence to false negatives, incorrectly classified examples where the particular note was indeed present. Additionally, while the architecture proposed in Sigtia et al. does seem to be effective for our acoustic model, the opportunity still remains for further hyperparameter tuning. For example, we could potentially insert more convolutional layers, experiment with more complex architectures, adjust how dropout is applied, and continue to explore different configurations for our fully connected layers. Furthermore, our current model is structured with each of the 88 output nodes optimizing individually, and common relationships and correlations among multiple notes are not necessarily being captured. Overall, there are many potential paths to explore in terms of restructuring our acoustic model, and we expect that further tuning and experimentation can help refine our model and improve performance.

10. Code

Our code for both the acoustic model and score generation implementations can be found in our Github repository: https://github.com/mbereket/music-transcription. [14], [7], [4], [6]

References

[1] Music transcription with convolutional neural networks. https://www.lunaverus.com/cnn.

[2] E. B. A. Ycart. On the potential of simple framewise approaches to piano transcription. 2016.

[3] E. B. A. Ycart. A study on lstm networks for polyphonic music sequence modelling. 2017.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[5] O. Abdel-Hamid, A. Mohamed, H. Jiang, and G. Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. 2012.

[6] D. L. D. E. M. M. E. B. O. N. B. McFee, C. Raffel. librosa: Audio and music signal analysis in python. 2015.

[7] F. Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[8] S. D. E. Benetos. A shift-invariant latent variable model for automatic music transcription. 2013.

[9] H. Fugal. Optimizing the constant-q transform in octave. 2009.

[10] M. Huzaifah. Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. 2017.

[11] R. Joshi. Accuracy, precision, recall and f1 score: Interpretation of performance measures. 2016.

[12] B. Krueger. Classical piano midi page. http://www.piano-midi.de/.

[13] S. Sigtia, E. Benetos, and S. Dixon. An end-to-end neural network for polyphonic piano music transcription, 2015.

[14] vishnubob. Python midi. https://github.com/vishnubob/python-midi.