Transcribing Solo Piano Performances from Audio to MIDI Using a Neural Network

by
Victoria Ebert

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented May 25, 2022
Commencement June 2023

AN ABSTRACT OF THE THESIS OF

Victoria Ebert for the degree of Honors Baccalaureate of Science in Computer Science presented on May 25, 2022. Title: Transcribing Solo Piano Performances from Audio to MIDI Using a Neural Network.

Abstract approved:_____

Patrick Donnelly

Automatic Music Transcription is a growing area of interest in Music Information Retrieval, and recent research has shown promise using onset detection on spectrograms. We propose a deep learning model that takes raw audio as input in order to transcribe a solo piano performance from audio to MIDI without complicated audio pre-processing and translation to spectrograms. This project focuses on the representation of piano audio as MIDI, but the contributions of this work will have applications in the area of piano sheet music creation. In this work we show that a bidirectional long short-term memory neural network can be used to transcribe a solo piano performance with high levels of both pitch and onset accuracy. Despite our unique approach, our model outperforms early models in this field and approaches competitiveness with top-performing models of the last two years. Our model is the only model in recent years to use raw audio as opposed to spectrograms or constant Q transforms. Because we use raw audio, we are able to capture resonance that exists within an audio recording that is not present in MIDI recordings. We also correctly model the behavior of the sustain pedal, despite having difficulty transcribing it correctly. These results show the promise of simpler models than are currently used for Automatic Music Transcription and bode well for the use of raw audio in music transcription.

Key Words: music, transcription, music information retrieval, MIDI, time-series

Corresponding e-mail address: victoriaebert00@gmail.com

©Copyright by Victoria Ebert
May 25, 2022

Transcribing Solo Piano Performances from Audio to MIDI Using a Neural Network

by
Victoria Ebert

A THESIS

submitted to

Oregon State University

Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science
(Honors Scholar)

Presented May 25, 2022
Commencement June 2023

Honors Baccalaureate of Science in Computer Science project of Victoria Ebert presented on May 25, 2022.

APPROVED:

_____

Patrick Donnelly, Mentor, representing School of Electrical Engineering and Computer Science

_____

Rebecca Hutchinson, Committee Member, representing School of Electrical Engineering and Computer Science

_____

Chet Udell, Committee Member, representing Department of Biological and Ecological Engineering

_____

Toni Doolen, Dean, Oregon State University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, Honors College. My signature below authorizes release of my project to any reader upon request.

_____

Victoria Ebert, Author

Table of Contents

Table of Contents **7**

1 Introduction **9**

2 Background **10**

2.1 Neural Networks 10

2.1.1 Recurrent Neural Networks 11

2.2 Music Transcription 12

2.3 MIDI 13

3 Related Works **14**

3.1 Music Generation 14

3.2 Classification 15

3.3 Alignment 15

3.4 Music Transcription 16

4 Dataset **17**

5 Preprocessing **19**

5.1 MIDI 19

5.1.1 MIDI to MIDI_CSV 19

5.1.2 MIDI_CSV to MIDI_TS 21

5.2 Audio 22

5.3 Audio and MIDI Alignment 22

5.3.1 Split 22

5.3.2 TensorFlow records 23

6 Experimental Design **24**

6.1 Model Design 24

6.2 Postprocessing 26

6.3 Evaluation Metrics 27

7 Analysis **28**

7.1 Full Testing Set 28
7.2 Individual Song Evaluations 30
7.2.1 Sonata in A Major, K. 208 30
7.2.2 Sonata in D Major, K. 118 36
8 Conclusion and Future Work **41**
8.1 Pedal 41
8.2 Velocity 42
8.3 Evaluation 43
8.4 Summary of Contributions 43
9 References **44**

1 Introduction

Imagine listening to a piece of music. As you listen, you observe the different notes, volumes, lengths, dynamics, and tempos. Now, imagine that you are asked to write down the music you just heard, and that your written copy needs to be precise enough that another musician could play it and reproduce the exact same sounds. Broadly, this describes the act of transcription. As you can hopefully imagine, it is not an easy task. When performed by humans, music transcription is time-consuming and takes expert knowledge. Transcription of music can preserve impromptu improvisations to be played again without requiring the musician to memorize what they did. It allows people access to sheet music that may not exist publicly, such as movie soundtracks. Transcription can also allow for the preservation of music that has been passed down orally.

In this project we investigate Automatic Music Transcription — an important field in the area of Music Information Retrieval (MIR). Formally, transcription is converting something, typically audio, to a symbolic format. The most basic application of transcription is a speech-to-text algorithm. Prior to the use of computers, all transcription was done by hand. In some fields, manual transcription is still used, such as stenography for court reporting. Transcription by hand or by computer is difficult because it typically needs to be exact, particularly in the aforementioned court reporting example. Humans' different accents, languages, dialects, and vocal tones all make speech transcription difficult, but a large amount of work has been put into speech-to-text transcription, and as early as 2004 a patent was granted that was able to translate speech upon request [1].

Nearly twenty years later, however, there is comparatively little research done in the area of music transcription. Similar to speech transcription, music transcription takes music from an auditory format to a visual one, but, unlike speech, music is often polyphonic, which means that many sounds happen at once. This overlap can make it more difficult to recognize what notes are being played at a specific time. When transcribed directly into sheet music by hand, the transcribers also need to focus on the key signature, tempo, time signature, and any accent marks in the music in addition to the notes and their timings. Music transcription with a one-to-one mapping — where a dictionary-like structure maps each auditory note to its corresponding symbolic counterpart — would be computationally infeasible, so research has focused on neural networks. Neural networks have been used in the areas of speech to text as early as 1995, where neural networks created transcripts for broadcast television [2]. Not long after, in 1999, researchers created a multi-layer-perceptron (MLP) that transcribed polyphonic piano music [3]. This initial approach performed well with simple chords, but not so well with actual songs.

In the years since the initial exploration of audio transcription, the field of music transcription has grown, with researchers attempting many approaches and variations. The method discussed in this paper is a Bidirectional Long Short-Term Memory (BLSTM) network (see Section 2.1.1),

that uses audio as the input and a time-series as the output. The time-series represents a MIDI file (see Section 2.3), used in most music transcriptions because it is a simple way to store musical information. MIDI is also easy to manipulate note by note. A time-series is needed to synchronize with our audio to train a neural network in order to label each output over time. Each output node represents a corresponding MIDI note, so when it is activated, that note is being played. Although other attempts have been made to transcribe piano performances, this attempt is novel in that it uses both a BLSTM and a raw audio file to transcribe a time-series representation of audio in a MIDI format.

Our goal with this project is to discover if using a BLSTM recurrent neural network with raw audio is a viable option for audio transcription. The contributions of this project include a new representation of MIDI files as a time-series dataframe, and training based on raw audio, without pre-processing, spectral analysis, or feature extraction. We envision a system in which pianists could compose music simply by playing their piano, rather than sitting down and writing sheet music. We investigate whether machine learning can be used to accurately transcribe piano music directly from raw audio, compare different neural architectures for music transcription, and determine whether performative expression can be captured with automatic music transcription.

2 Background

Music transcription tasks require background knowledge in neural networks, past music transcription attempts, and the symbolic representation used in the transcription, in our case MIDI. Because we use a recurrent neural network, we provide background on not only a basic neural network, but a recurrent neural network and a long short-term memory network. We briefly give background on music transcription, but leave most information on that subject to Section 3. We provide both a brief history of MIDI as well as describe its functionality.

2.1 Neural Networks

A neural network is a system of computing, abstractly inspired by the architecture of a human brain, thus, neural. A typical neural network is composed of many smaller processors, called neurons, that produce real-time activations [4]. The first layer of neurons is activated by the input data, and subsequent neurons are activated by weighted connections from the previous neurons. The simplest form of a neural network is a perceptron, which takes the form $\widehat{y} = A(b + \sum_{i=1}^{n} x_i w_i)$ where $\widehat{y}$ is the output, b is the bias, $[x_1, x_2, \dots x_i]$ are the inputs, $[w_1, w_2, \dots w_i]$ are the weights of each input, and $A()$ is a non-linear activation function such as a hyperbolic tangent function (tanh).

At a basic level, training a neural network is done by adjusting the weights and biases until the given output is as close as possible to the expected output. Neural networks have had success in the fields of game playing, pattern recognition, data classification, and modeling and prediction [5]–[8].

Neural networks in a recognizable form have been around since 1940, with deep neural networks existing since 1965. The ART model [5], introduced in 1988, recognizes patterns using recognition, reinforcement, and recall, and was theorized as a convenient method for recognizing and producing speech. Another early use of neural networks was in classification, in which a Naive Bayes Classification Algorithm helped detect breast cancer [6]. Recurrent neural networks (RNN) and long short-term memory (LSTM) networks, the basis for the neural network described in this paper, have existed since 1995 [4]. The specific type of network implemented by this model is a bidirectional LSTM (BLSTM) RNN. Large, deep LSTMs have been used for language translation, suggesting that LSTMs work well with complicated sequence problems [9].

Neural networks have had success in the area of speech transcription, especially backwards transcription — going from text to speech. WaveNet [10], which generated speech from text in both English and Mandarin, is a particular success. From speech-based audio neural networks, music-based audio neural networks have become increasingly explored in the literature, and become a wider field of interest in the research community. The meaning of a musical piece is often only understood in its auditory performance, unlike language where the meaning can often be derived simply from the text [11]. This fundamental difference causes both transcription and generation to require a different approach than standard speech networks.

2.1.1 Recurrent Neural Networks

The specific type of neural network implemented in this project is called a recurrent neural network (RNN). An RNN uses sequential data, and considers not only the current data, but prior data to determine the output. This consideration gives RNNs a rough working “memory” from which to collect past assumptions. In this paper, we create a bidirectional neural network, which means that not only does the network consider past information when determining an event, but it also considers future information. We use a special form of an RNN called a Long Short-Term Memory (LSTM) network, which keeps important information from much earlier in the model. In language modeling, this information may be parts of speech such as personal pronouns; if the word ‘she’ has been used for several sentences, chances are that we will refer to something as ‘hers’. This concept is likewise useful in music transcription as it allows us to consider context from both previous time steps and future time steps to determine the activated notes at the current time step. Figure 1 describes both an RNN and an LSTM, showing the increased internal complexity of an LSTM.

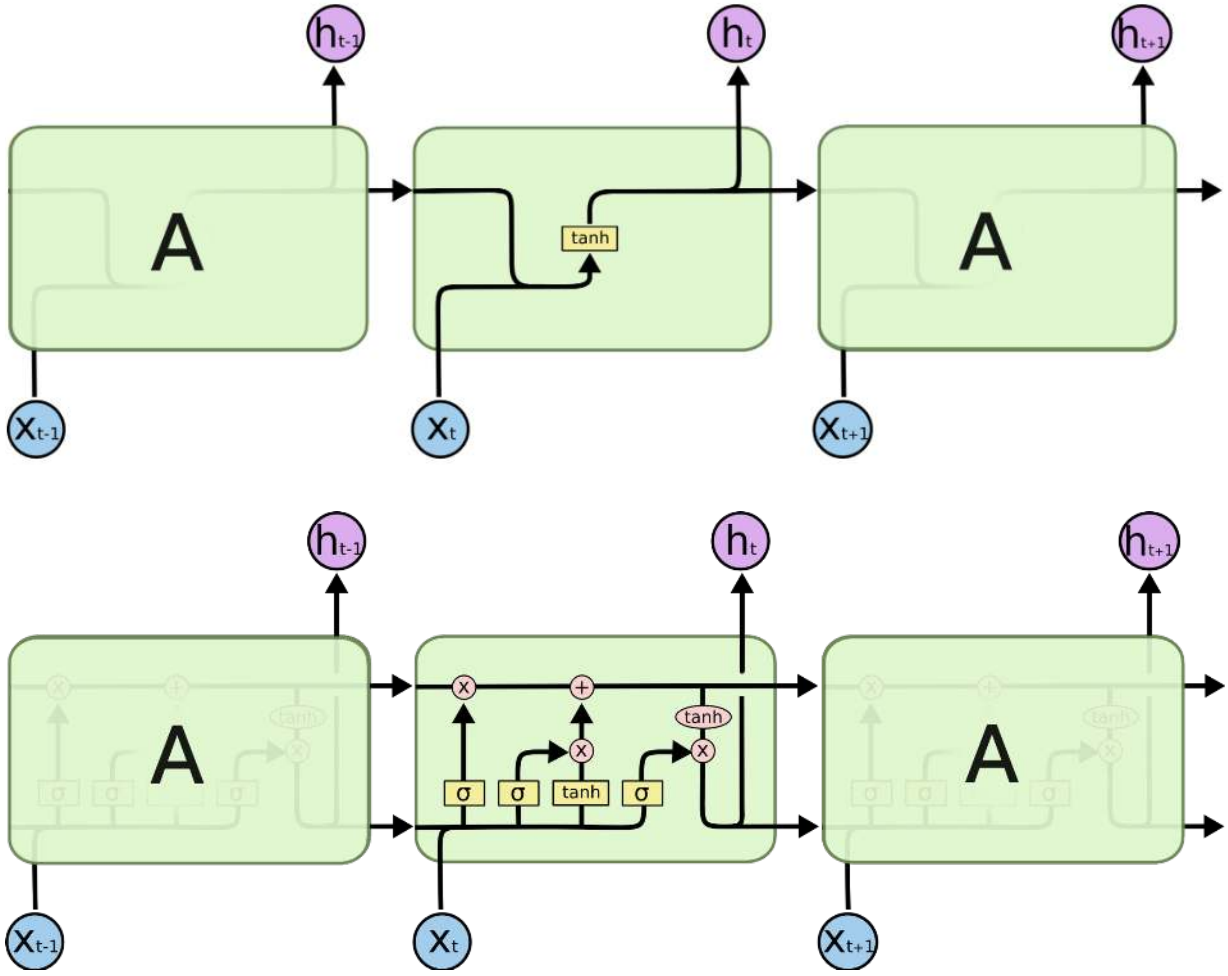


Figure 1: An example of a standard RNN (top) and an LSTM (bottom) from [12]. In each of these models, the blue circles represent the input, the purple circles represent the output, and a yellow rectangle represents a neural network layer. In the bottom model, a pink circle represents a point-wise operation.

2.2 Music Transcription

Several attempts have been made at music transcription using a variety of approaches. PerformanceNet [11] — which similar to WaveNet is a reverse transcription process — converts from a score to an audio file using a deep encoder and decoder. Another recent method proposes a hybrid recurrent neural network using a time-series as input and piano roll as output [13]. An attempt was made with a classification based approach training a deep belief network on spectrograms [14]. Another study attempted 76 different neural networks, each trained to recognize a different piano note, and then combined those to create a transcription network [15]. Other approaches include: outputting a tuple for each note, listing the start time, end time, note,

and volume [16], and detecting the onset of each note and calculating the rest of the data from there [17]. All of the listed attempts are discussed further in Section 3.

2.3 MIDI

Most music transcription approaches involve Musical Instrument Digital Interface (MIDI) files. MIDI is a symbolic representation of music, and was designed in the 1980s to be a simple, cheap way for manufacturers of digital devices to control their synthesizers independently of the device or vendor [18]. At the most basic level, MIDI is a communication system for digital music. The introduction of MIDI allowed more people to develop music; prior to digital music, most music composers were trained musicians. MIDI compatible instruments and software tools made it much easier to create music with no prior knowledge. Several MIDI music composition tools have such a well developed user interface that many people do not realize that they are using MIDI at all. A common digital audio workstation is Apple's GarageBand which can both record real instruments and use virtual synthesizers to create music [19].

Despite its usefulness and the typical practical applications of music creation, it is important to remember that MIDI is not a programming or markup language, and it is not a direct mapping of musical sounds [20]. MIDI is simply a way to communicate musical data. An example of the type of data that MIDI conveys is shown in Figure 2. For human musicians, sheet music describes how to recreate a piece of music; for piano, sheet music describes what keys to press and when, how long to hold down the keys and how loudly to play. MIDI specifies the electronic aspect — which circuit to turn on to create a pitch and at what time, how long to keep it open, and how loud or soft the pitch should be played [20].

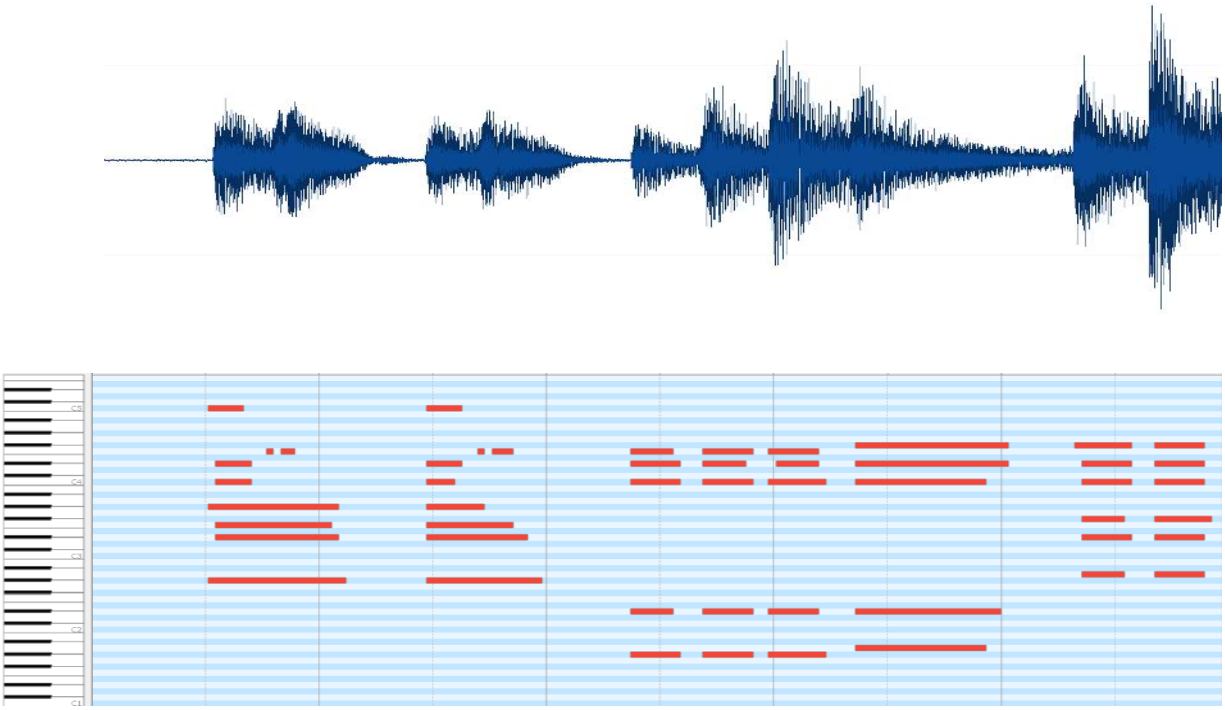


Figure 2: A comparison of MIDI to audio. Each blue line in the MIDI is a piano note, and each orange bar is a played note. The silence in the audio form is reflected in the absence of notes in the MIDI piano-roll, and peaks in the waveform correspond to onsets of notes in the MIDI. Not pictured here, each MIDI note played also has a corresponding velocity, indicating how hard to strike the theoretical piano key.

3 Related Works

Music Information Retrieval is a large field, encapsulating a rich set of subtasks including instrument recognition and source separation, music recommendation systems, and music categorization. Four focus areas of music information retrieval include generation, classification, alignment, and transcription. Music generation is composing a piece of music without a human composer. Classification is classifying music based on anything from genre, to time-period, to instrument, to composer. Music alignment is very similar to transcription, but uses an audio file to find the current location in a given MIDI file. Finally, music transcription, the topic of this thesis, is converting an audio file into symbolic notation.

3.1 Music Generation

One of the more common focus areas of neural networks in the musical field is that of music generation. Music generation is often referred to as computer music, or computer composition, as

it creates brand new music. PerformanceNet [11], is a generative neural network, analogous to a text-to-speech model where a model uses a score to generate audio. Performance Net is based on the speech generative WaveNet [10]. WaveNet was initially trained as a speech modeling network, but it can generate novel and realistic music as well. MelNet [21] is another generative neural network for sound, and uses Convolutional Neural Networks (CNN). CNNs are typically applied in image generation. Therefore, MelNet, uses spectrograms of wave files as the input, rather than an audio file. MelNet is better equipped for speech problems than for music problems, but was trained on both. A Music Transformer model is another generative neural network, using self-attention to generate new music [22]. This self-attention model was developed after researchers noted that music tends to have repetitive sections. Maintaining an internal state of produced audio, then, would be vital to producing realistic music.

One of the most cited generative neural networks for music is given in the paper that first introduced the MAESTRO Dataset [23], a now common dataset in music information retrieval tasks, and the dataset used in this project. This attempt was, again, based on WaveNet, as well as a previous transcription model by the same authors [17]. Another recent network for generation is based on a Transformer Autoencoder that uses the encodings across time to generate music, as well as a YouTube dataset of piano performances [24]. Most of the papers discussed in this section generate solo piano music, but researchers have explored other types of music. For example, Jukebox [25] is a generative neural network that generates vocal music as well as instrumental music.

3.2 Classification

Another common focus area of neural networks in music is in classification. One novel classification approach attempted to classify music into time-period using a convolutional neural network [26]. This attempt classified the first 30 seconds of classical piano solos, at a sampling rate of 8000 Hz. The authors decided to limit their model to only classical music to narrow down both the time period, and to keep a similar genre across classifications. Another classification problem is classifying different pitches as notes. A group of researchers attempted to classify pitches using fast Fourier transforms, and their model was robust enough to classify notes from multiple instruments [27].

3.3 Alignment

The third large music information retrieval focus area is music alignment. Alignment is similar to transcription in that in most attempts, it tries to find a MIDI file that matches an audio file. However, it differs in that it does not try to generate a MIDI file as the output. One attempt scraped the internet to extract 455,333 MIDI files [28]. These files were then used as training data to attempt to match a given MIDI file to an audio file from the Million Song Dataset. Another attempt worked in the opposite direction, attempting to find a MIDI file (from a

particular database) that corresponded with a given audio recording [29]. A slightly different approach to alignment is in the form of real-time score alignment [30]. In real-time score alignment, the neural network already has the MIDI file as it listens to the audio recording and attempts to find the correct location in the score.

3.4 Music Transcription

The music information retrieval focus area that is important to this paper is transcription. One of the earliest attempts at music transcription occurred in 2004, and used a combination of seventy-six different neural networks. Each network was trained to recognize a different note, and accounted for all but the lowest octave of piano keys [15]. A similar approach in 2011 combined eighty-seven independent support vector machine classifiers to predict the presence of the corresponding piano note, with a deep belief network and a frame-level note classification system [14]. Many attempts at transcription draw on methods from speech to text networks. One such attempt uses an end-to-end approach to jointly train both language and music models via convolutional neural networks and constant-Q transforms [13]. These same authors also attempted another method using an arbitrary frame-level classifier, where the output was the probability that a given pitch would be in a frame [31]. Another attempt, albeit backwards from what we are attempting, takes in a score, and attempts to generate the audio [11]. This attempt is based on the text-to-speech network WaveNet [10].

Another area of research uses the onset of notes to attempt transcription, including [9], [13], [14], [16], [17], [23], [31]–[37]. Of these papers, [17], [23], [33], [35] all use onset information for both training and evaluation, while the rest use onset information only for evaluation. One method of assessing onset information comes from outputting a quadruple of the form: `{start, end, note, volume}` [16]. One of the most famous transcription attempts predicts mainly on the onset of each note, and calculates the other data from there, including the velocity of the note [17]. The same authors later produced a paper combining the work in [10] and [17] to both transcribe and generate music [23]; this paper also introduced the MAESTRO dataset used in our attempt.

The attempt most similar to our approach presented here is a BLSTM model with an output reminiscent of piano-roll notation. This method, however, uses spectrograms of audio as the input, and does not consider velocity [34]. A fully different method uses sparse dictionary learning, with a dictionary of multiple atoms of different lengths for each pitch to jointly estimate pitch, onset, and duration in a convolutional sparse coding framework [35]. A prior convolutional sparse coding attempt pre-learned the dictionary by recording audio of each piano note and using the waveform to estimate activations [36]. Other attempts are similar to image classification networks, including one that uses convolutional and pooling layers [37]. One of the most recent attempts uses a Transformer model, a state of the art model with self-attention that has seen excellent results in many fields outside of music [38].

4 Dataset

Many different music datasets have been used for audio transcription tasks, including the Saarland dataset [39], used in [33], and more commonly the MAPS dataset [40], used in [13], [14], [16], [17], [31], [33]–[36], [41]. Until 2019, the MAPS dataset was state of the art, and was the dataset of choice for most music transcription tasks. In late 2018, however, the MAESTRO dataset was introduced [23]. A further breakdown of the aforementioned datasets is shown in Table 1.

Dataset Name	**Instrument**	**Year Published**	**Number of Recordings**
MAPS	Disklavier, Synthesizer	2009	60 (Disklavier) 210 (Synthesizer)
Saarland	Disklavier	2011	50
MAESTRO	Disklavier	2018	1276

Table 1: A comparison of music transcription datasets that use solo piano performances. Each of these datasets contains audio and MIDI collected using a Yamaha Disklavier piano. The MAPS dataset also includes synthesized audio recordings. The MAESTRO dataset has over a thousand more performances than both the MAPS and Saarland datasets, and is the most recently compiled.

For this project, we decided to use the MAESTRO dataset. The MAESTRO dataset was collected from over 10 years of performances at the international piano e-competition, where each piece is played on a Yamaha Disklavier piano. A Disklavier piano, shown in Figure 3, records precise MIDI information as it is played. Each time the pianist presses a key or pedal, the piano records which key/pedal, what time, and the precise velocity with which the object was pressed. The Disklavier also has the capability to play back MIDI files, all while still functioning and sounding like a classic grand piano. This functionality allows the judges to go back and listen to exactly the same performance played on the piano again. During this competition, an audio file of each performance was recorded, and in the dataset the MIDI and audio recordings are aligned to within 3 milliseconds of accuracy.



Figure 3: This promotional image for a Yamaha Disklavier piano is taken from [42]. It provides both an image of the piano, as well as gives examples of its use. The Disklavier is a computerized grand piano, so while it does have the same string and hammer mechanisms as a standard grand piano, there are sensors under each key that record the velocity of each strike. This piano can play pre-existing recordings by artists who have already recorded themselves, and it can be set up to mimic other Disklavier pianos and play live.

The MAESTRO dataset contains 1276 songs, and is a total of 198.7 hours long, containing 7.04 million individual notes [43]. The MAESTRO dataset has been used in recent papers, including [23], where it was introduced, as well as [37], [38]. In its introduction, a train/test/validation split, shown in Table 2, was determined to keep the same compositions separated across subsets, regardless of performer. Each of these papers produced compelling results using the MAESTRO dataset, and as it is the current state-of-the-art dataset, it was used in this paper.

Split	Performances	Duration (hours)	Size (GB)	Notes (millions)
Train	962	159.2	96.3	5.66
Validation	137	19.4	11.8	0.64
Test	177	20.0	12.1	0.74
Total	**1276**	**198.7**	**120.2**	**7.04**

Table 2: The authors of the paper that introduced the MAESTRO dataset provided recommended train, validation, and test splits. This table is a representation from the website where they published the MAESTRO dataset [43]. Across the 1276 performances, approximately 75% of the data was used for training, 11% for validation, and 14% for testing.

5 Preprocessing

Before we could train our neural network, we needed to pre-process the data and manipulate it into a representation compatible with machine learning experiments. Pre-processing meant different things for the audio and MIDI files, as they had different functions within the neural network. The audio file needed to be trimmed, downsampled, split, and converted to a tensor. The MIDI files needed to be converted to a time-series, split, and converted to a tensor. A full overview of the preprocessing steps is given by Figure 5

5.1 MIDI

The initial MIDI files were in a format that is readable by audiating software, but not in a format that we could manipulate. In order to extract timing information and convert our MIDI files into a time-series, we needed to first convert to MIDI_CSV files, and then convert from MIDI_CSV files to CSV files representing a time-series (MIDI_TS).

5.1.1 MIDI to MIDI_CSV

First, the MIDI files were transformed to MIDI_CSV files using the `py-midicsv` library [44]. MIDI_CSV files are written representations of the information stored in a MIDI file; this conversion provided us with a format that was easier to extract note and timing information from than a classic MIDI file. The first few lines consist of metainformation, which gives information on how the MIDI reader should read the file. After the metainformation, these files are formatted with the time, the message, the channel, the MIDI note number, and the velocity with which the

piano key was pressed. Because we only required information about notes, velocities, and timings, we discard the meta information in our conversion from MIDI_CSV to MIDI_TS. Figure 4 illustrates a comparison from [45] between sheet music, a representation of a MIDI_CSV file, and piano-roll notation (the typical method for displaying a MIDI file).

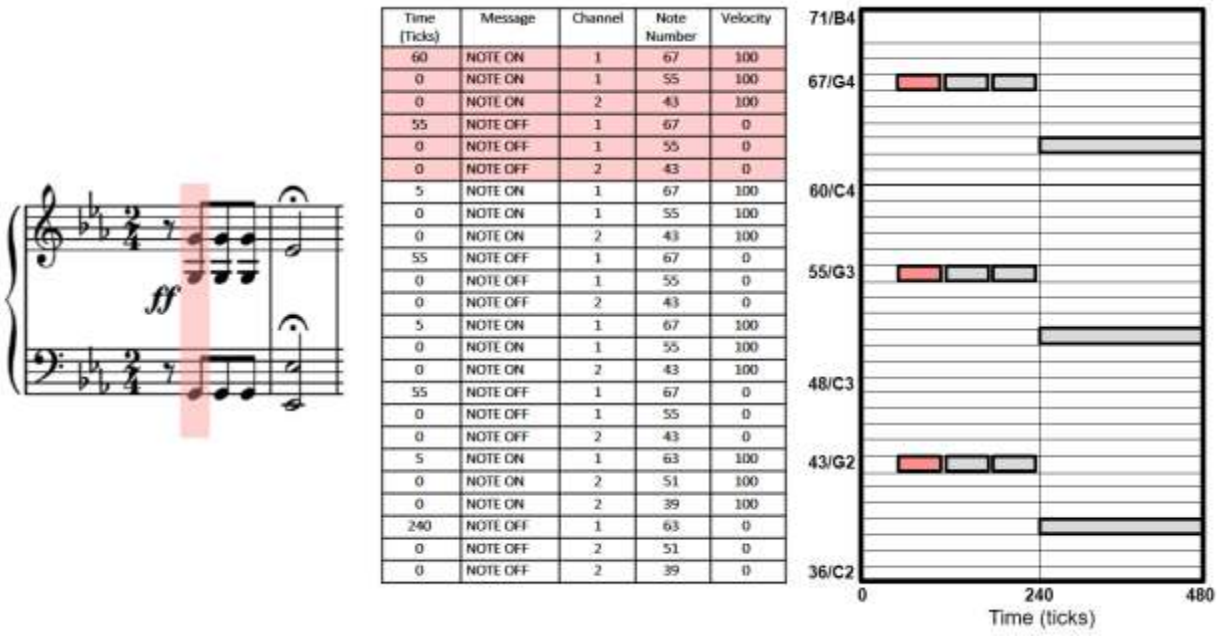


Figure 4: An explanation of a MIDI_CSV file, shown in conjunction with a piano score and standard MIDI notation (from [45]). The channel column is not used in our notation, as we are using a single instrument score. The time (ticks) column in this representation is the amount of time to wait between calls, whereas in ours, it is an absolute time, i.e., the first three lines would have a value of 60 in the ticks column, and the second three would have a value of 115.

The message header contains the information about what is about to happen. The value in the column is either `note_on_c`, `note_off_c`, or `control_c`, where `note_on_c` means the note begins at the given time, `note_off_c` means the note ends at the given time, and `control_c` means a pedal is being played at the given time. The note number is a number between 0 and 127, representing the corresponding MIDI note. If the call is `control_c`, the note represents a pedal, which can be either 64 (sustain), 66 (sostenuto), or 67 (soft). The velocity is a value between 0 and 127 representing how hard the note or pedal is pressed. A velocity of 0 with `note_on_c`, is the same as a call of any velocity to `note_off_c`.

5.1.2 MIDI_CSV to MIDI_TS

The MIDI_CSV file still needed to be altered to a format that was easy for a neural network to read. The raw audio is a time-based series, and so we needed to convert the MIDI_CSV into a time-based representation as well. In this way, we can focus on a single period of time, and correlate the audio time to the MIDI_CSV time, while keeping each time in sequence. The format chosen for this sequencing was a time-series. A time-series is a list of specific things that happen in specific places, at specific intervals of time. For this project, the interval chosen was 1024 samples, or 64 milliseconds in a 16000 Hz audio sample (see Section 5.2). We chose this time because 1024 is a common window size for audio manipulations, including a fast Fourier transform. Every 64 milliseconds, a new snapshot of everything that is occuring is recorded. An example of the header for the MIDI_TS files is shown in Table 3. Three extra columns were added to the end to account for MIDI pedals 64 (sustain), 67 (soft), and 66 (sostenuto), in order of frequency of appearances. The values in the time column are given in intervals of 64 milliseconds, 0.064, 0.128, 0.172, etc. The values in the MIDI_TS columns indicate the velocity with which the key or pedal was pressed. Velocity is typically given in a range from 0 to 127, and so in this format, the velocity is given as a percentage of the maximum, 127.

Time	MIDI note 0	MIDI note 1	...	MIDI note 126	MIDI note 127	MIDI pedal 64	MIDI pedal 67	MIDI pedal 66

Table 3: The MIDI_TS files are formatted with 131 columns. The first column contains the time, the next 128 columns contain the velocity of the corresponding MIDI note at the specific time, and the final three columns contain the velocity of the sustain, soft, and sostenuto pedals respectively, at the given time point.

Our MIDI_TS files require notes to start and end at exactly intervals of 64 milliseconds. True MIDI files do not adhere to this, as pianists have no need or want for that level of precision. To fit the MIDI file into the MIDI_TS, if a note was at all playing during a snapshot, it was counted as playing for the whole snapshot. If it started halfway through a snapshot, the MIDI_TS counted it as starting at the beginning, and if it ended at the very beginning of a snapshot, the MIDI_TS counted it as ending at the end. In a worst-case scenario, where the pianist played and released each note at 63 millisecond intervals, our entire song would be offset by 63 milliseconds from the pure audio. Each onset would be 63 milliseconds too early, and each offset would be 63 milliseconds too late. Although in studies, under ideal conditions, humans can hear differences of about 10-20 microseconds [46], we accept a maximum offset of 63 milliseconds. Our average

offset is closer to 32 milliseconds, and, when transformed back into a MIDI file, we believe the difference in timing to be negligible.

5.2 Audio

Compared to the MIDI files, very little preprocessing was required for the audio files. All audio files within the MAESTRO dataset were WAV files. Had the dataset included any MP3 files or any other audio file format, the files would need to be converted to WAV files. Because the files were already in the required format, the audio files only needed to be trimmed and downsampled.

To confirm that each audio file was the exact same length as the MIDI file, we trimmed all audio files to the exact length of the MIDI files. This trimming ensured that the neural network did not learn any extra "white space," which could be any post-performance silence or an audience response, and allowed the input and output to be the exact same length. Although most of the files within the MAESTRO dataset did not require trimming — a feature of the 3 millisecond alignment — it was done on all files to ensure consistency.

The typical sampling rate for audio files is 44100 Hz. This resolution is high, and a lot for our neural network to process. In order to reduce the input to the neural network, the audio was resampled to 16000 Hz. For the purposes of this study, we deemed 16000 Hz sufficient to cover the notes and overtones of a piano. The highest note on a standard piano is C8, which corresponds to a frequency of 4186.009 Hz. Using the harmonic series, we can determine that we capture the first and second overtones of the highest note, and cut off just before the third overtone at 16744 Hz [47]. Downsampling to 16000 Hz means that there is less information for the neural network to process by a factor of 2.5, while still retaining auditory fidelity.

5.3 Audio and MIDI Alignment

In order to ensure that the audio and MIDI_TS remained aligned when training the model, two transformations needed to be done on both files. These transformations aligned the data into a format that could be processed by a neural network. First, the songs needed to be split into chunks of the same size, and then they needed to be converted into TensorFlow records that could be stored and read as a TensorFlow dataset.

5.3.1 Split

For optimal processing by the neural network, we needed to ensure that each song was the same length. Because the songs ranged from 45 seconds to 45 minutes, we first split the songs into ten-second chunks. The final chunk in each song was zero-padded (silence was appended to the end until the duration of the chunk was ten seconds). This approach was chosen to process songs

of all lengths while minimizing the zero-padding necessary to standardize the song lengths. We performed this step after all other preprocessing steps and simultaneously on the two file types. This sequence of ten-second chunks was not only easier for our model to process, but also ensured that our model could learn songs of any length. After prediction, we can reconstruct any longer MIDI file from these ten-second splits as necessary. The MIDI_TS files had time-steps of 64 milliseconds, which means that our ten-second boundary was rounded to the nearest 64 milliseconds, 9.984 seconds.

5.3.2 TensorFlow records

We built our neural network models using the TensorFlow library [48]. In order to integrate our data with the neural network, we needed to create TensorFlow records (TFRecords). These records store the paired audio and MIDI_TS files as tensors, a TensorFlow specific data structure that stores data as multi-dimensional arrays. TensorFlow has a built-in module to encode and decode WAV files, as they are not naturally in a multidimensional array format. A time-series is inherently a multidimensional array, and, as such, our MIDI_TS remained in the same structure as they were in the CSV format. With both file formats represented as multidimensional arrays our audio and MIDI_TS files were aligned within the TFRecords. For an example of this alignment, see Figure 2 from Section 2.3 or Figure 8 from Section 7.2.1. Each TFRecord contains 16 songs, (the batch size). We bypassed creating TensorFlow records for the testing split as part of our eventual goal of real-time transcription. Instead, we converted each test audio file to a tensor as it was input to the neural network.

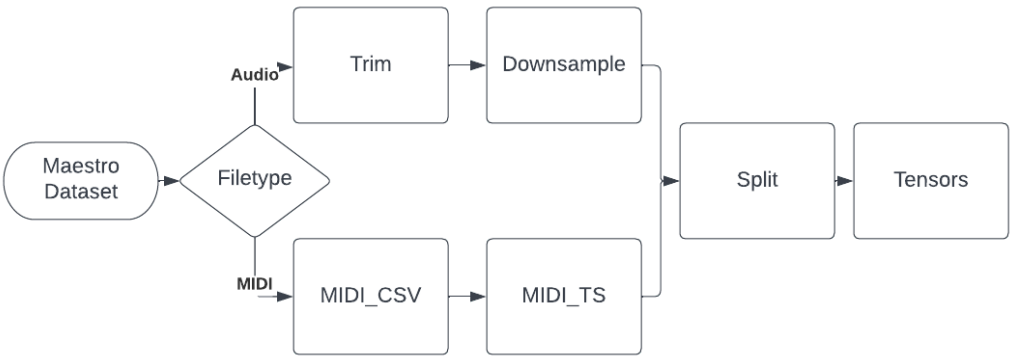


Figure 5: A flowchart showing the preprocessing steps for the data. The full dataset is split by the filetype, audio or MIDI, and then simultaneously preprocessed. For the audio file, preprocessing involved trimming “white space” and converting to 16000 Hz. For the MIDI files, we convert to MIDI_CSV and then to a MIDI_TS representation. Each performance is then split into ten-second chunks and then converted into tensors.

6 Experimental Design

After selecting our dataset and preprocessing the data, we designed our experiment. To design our experiment, we first needed to formulate the design of the model, decide on post-processing steps for the model output, and understand the metrics that we will use to evaluate the performance of our model.

6.1 Model Design

The preprocessed audio files and MIDI_TS files were fed to the neural network as the input and expected output respectively. For this project, we used a bidirectional long short-term memory (BLSTM) version of a recurrent neural network (RNN) (see Section 2.1.1). An LSTM allows the network to remember the data that came previously. For our purposes, this memory is useful because it not only helped the network recognize what sound it was currently hearing, but it meant that it could classify what it was likely to be based on its past knowledge. A Bidirectional LSTM learns both forwards and backwards, and allows the network to base the current note prediction not only on past notes, but on future notes as well.

We ran multiple experiments to determine the optimal architecture for the neural network. The first attempt was modeled after an encoder/decoder model, and used dense layers in between BLSTM layers. This attempt, although working in the small scale, resulted in an exploding gradient problem when scaled up from a single song to all of the songs from the MAESTRO dataset. The most effective model architecture is shown in Figure 6. We used strictly BLSTM layers, decreasing in nodes by powers of two. The shape of the input data is `[156, 1024]`, and the shape of the output data is `[156, 131]`. This shape means that the network has two hidden layers, one with 512 nodes, and one with 256 nodes. The input and output layers had 1024 and 131 nodes respectively. These layers resulted in a total of 1923 parameters. Each layer had a dropout of 0.2, meaning that 20% of the nodes were discounted at each step. This dropout helps compensate for any overfitting of the model that may occur. The code for our model is shown in Figure 7.

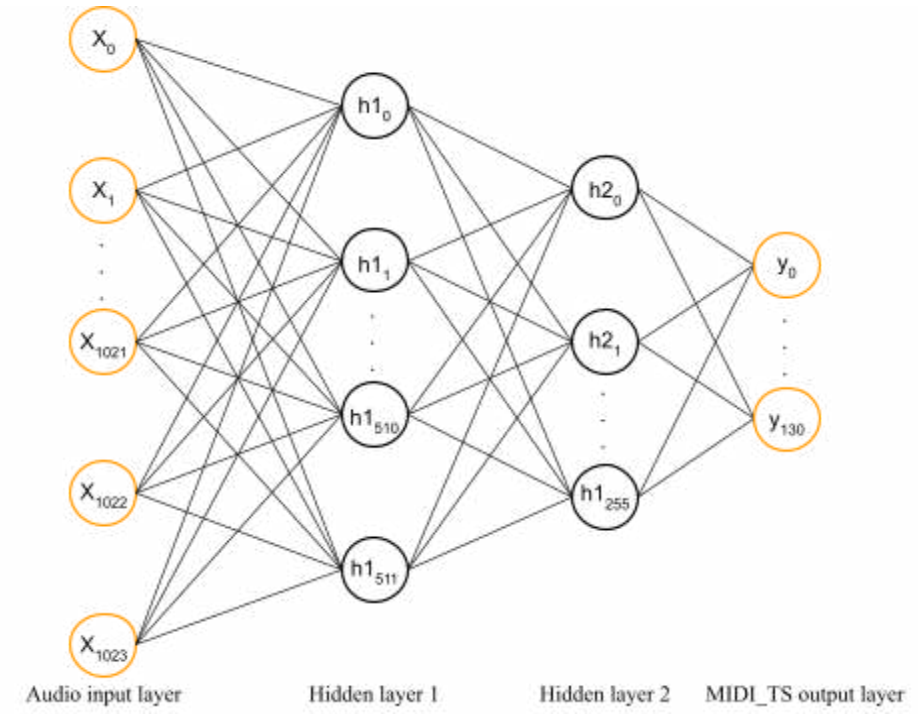


Figure 6: A visualization of the neural network. The network has an input layer that corresponds to the shape of the input, with hidden layers in decreasing powers of two. (The input layer is 1024 so our hidden layers have 512 and 255 nodes respectively). The size of the MIDI output layer is greater than the next power of two, due to the piano pedals, and so we end our model with a layer of 131 nodes.

After experimenting with both Adam and RMSProp optimizers, we chose to use the Adam optimizer — specifically the built-in Adam optimizer from the `tensorflow.keras` library. The Adam optimizer is a "first-order gradient-based optimization of stochastic objective functions" [49], that combines aspects of Adaptive Gradient Algorithm (ADAGrad) and Root Mean Square Propagation (RMSProp). The loss was calculated using a Mean Squared Error function. Figure 7 shows the basic model architecture and compilation metrics. Based on preliminary testing with larger epoch sizes causing exploding gradients, the model was fit with 250 epochs and a batch size of 96 (16*6 for each GPU used).

This model was trained on the DGX2 servers on the Oregon State University High Performance Computing Cluster[1]. The DGX2 cluster includes 6 Nvidia DGX2 nodes each with 48 CPU cores, 16 V100 GPUs, and 1.5 TB of RAM each. We requested the use of 6 GPUs, and 50MB of memory. After training the model on the full training set of 962 performances, each of the 177

[1] https://it.engineering.oregonstate.edu/hpc

test songs were split into ten-second chunks, transformed to tensors, fed to the trained model, and post-processed to produce MIDI files.

```
model = Sequential()

model.add(layers.Bidirectional(layers.LSTM(1024, return_sequences=True,
          dropout=.2), merge_mode='sum'))

model.add(layers.Bidirectional(layers.LSTM(512, return_sequences=True,
          dropout=.2), merge_mode='sum'))

model.add(layers.Bidirectional(layers.LSTM(256, return_sequences=True,
          dropout=.2), merge_mode='sum'))

model.add(layers.Bidirectional(layers.LSTM(131, return_sequences=True,
          dropout=.2), merge_mode='sum'))

model.compile(optimizer='adam', loss="mse", metrics=['accuracy'])
```

Figure 7: The code used to produce the neural network for this project. The model consisted of four BLSTM layers with decreasing nodes, and a dropout of 0.2 at each layer. We chose to use a summation algorithm to merge the forward and backward layers of the BLSTM. We compiled the model using the Adam optimizer and a mean squared error loss function.

6.2 Postprocessing

Without any post-processing, our model learned a lot of very quiet noise. The lowest value that our velocity can be is $\frac{1}{127}$ or ≈ 0.008. Some of the predicted velocities from our model were as low as $5 * 10^{-6}$. In order to combat this noise, we chose to set all values in the predicted output that were less than 0.1 to 0. According to [50], a Yamaha Disklavier II produced MIDI velocities in a range of approximately 15 to 120. This velocity corresponds to low-end values of $\frac{15}{127} \approx 0.118$. We chose to round down from 0.118 to 0.100 in order to adjust for variability within the model and improvements in Yamaha Disklaviers in the time since the publication of this paper, while still eliminating unintentional noise that may arise in our process. The model predicted across all 127 MIDI notes, despite the range of the piano being from MIDI note 21 to 108. We chose to predict across all 127 MIDI notes to allow our model the flexibility for future generalizations to other instruments that may have a different range than the piano. As expected,

after this post-processing the 40 unused MIDI notes on a piano were rightly predicted to be 0 in all cases.

In the true songs recorded by a Disklavier piano, each note maintains the same velocity throughout its duration. In the predicted songs returned by the neural network, each note had a different velocity at each of its timesteps, because we spliced the sequences based on our input size, and then reconstructed them. This change in velocity accounts for the inherent diminuendo of soundwaves as they resonate within a room. For a pure mathematical comparison, this fade-out is preferred, because it gives us more information about what our network was experiencing. If we translate our output MIDI_TS files to MIDI files without any further post-processing, however, we hear new note onsets every 64 milliseconds, as MIDI files are not designed to change the velocity of notes continuously. To counteract this feature in playable MIDI files, we implemented a "smoothing" function to the predicted MIDI_TS files that were converted to MIDI. Each instance of a note was replaced with the average velocity throughout its duration. In our full dataset model evaluation, the velocity value was ignored and simply treated as off — the velocity is 0 — or on — the velocity is other than 0.

In order to produce playable MIDI files, the MIDI_TS file was converted to a MIDI_CSV file by reversing the steps taken to convert the original MIDI_CSV file to a time-series. The MIDI_CSV file was then converted to a MIDI file, again using the `py-midicsv` library.

6.3 Evaluation Metrics

Initial evaluation of the model was done using the `mir_eval` metric toolkit library [51]. This library is designed to be a simple implementation of many different music information retrieval evaluation metrics. There were two modules from this library we determined to be useful for our analysis: `mir_eval.transcription`, and `mir_eval.multipitch`. The `mir_eval` documentation defines transcription evaluation as "given two sets of notes, we count how many estimated notes match the reference, and how many do not" where "a returned note is assumed correct if its onset is within +-50ms of a reference note and its F0 is within +- quarter tone of the corresponding reference note, ignoring the returned offset values." `mir_eval.multipitch` is defined as "identify[ing] all of the active fundamental frequencies in each time frame in a complex music signal," where "a frequency is 'correct' if it is within 0.5 semitones of a reference frequency." Although the `mir_eval.multiptich` method aligns with our project because we predicted the velocity of each note at a specific time frame, no other published papers used this evaluation metric, so we chose to save it for potential work in the future. In addition, the `mir_eval` library had no process for evaluating the piano pedals, so we dropped them from the evaluation to return to them in future work.

`mir_eval.transcription.evaluate` returns a dictionary of fourteen values, grouped into four categories:

1. Pitch, Onset, Offset (full data)
2. Pitch, Onset (full data no offset)
3. Onset
4. Offset

Each of these categories includes precision, recall, and f-measure metrics, and the full data and full data excluding the offsets also includes the average overlap ratio. In this case, **precision** is the proportion of correctly predicted notes to all predicted notes: $precision = \frac{true\ positives}{true\ positives + false\ positives}$. In the full data, a true positive is defined as a predicted onset within 50 milliseconds of the true onset, a pitch within 50 cents of the correct pitch, and the offset within the maximum of 50 milliseconds or 20% of the true note's duration. For the full data excluding offsets, the onsets only, and the offsets only, the true positive value is defined the same as the full data true positive, excluding the irrelevant data. **Recall** is the proportion of correctly predicted notes to all true notes: $recall = \frac{true\ positives}{true\ positives + false\ negatives}$, where a true positive is defined as above. The **F-measure** is the harmonic mean of precision and recall: $f - measure = 2 * \frac{precision * recall}{precision + recall}$. The **average overlap ratio** is the ratio between the length of time that two notes overlap, and the length of time that the notes span, combined: $overlap\ ratio = \frac{(min(true_offset, predicted_offset) - max(true_onset, predicted_onset))}{(max(true_offset, predicted_offset) - min(true_onset, predicted_onset)))}$.

7 Analysis

We evaluated the entire testing set using the `mir_eval` library, and compared our full data and full data no offset metrics to other models in the field. We also chose to examine multiple songs on an individual level, to attempt to determine the reasoning behind the `mir_eval` metric values. We present two of these individual evaluations below.

7.1 Full Testing Set

The `mir_eval.transcription` metrics for the testing split of our MAESTRO database were computed for each song, and then averaged to produce results for the dataset as a whole. Table 4 provides the `mir_eval.transcription` output for the testing data.

	Pitch, onset, and offset	**Pitch and onset**	Onset only	Offset Only
Precision	0.260	0.646	0.731	0.545
Recall	0.253	0.622	0.700	0.522
F-measure	0.255	0.630	0.710	0.530
Average overlap ratio	0.968	0.677	-	-

Table 4: mir_eval.transcription results across the whole test set. The precision, recall, and F-measures are provided for the dataset when evaluated using the pitch, onset and offset of all notes, excluding the offset, and the onset only and offset only.

While our onset only values are the highest, and indicate that we are correctly predicting the time at which notes begin, the main metrics that interest us are the pitch, onset, and offset precision, recall, and F-measure, and the pitch and onset precision, recall, and F-measure. Table 5 provides these metrics in comparison with other attempts in the field. Our preliminary model measures in the middle of attempts spanning from 2016 to 2021.

	Pitch, onset, and offset			**Pitch and onset**		
	Precision	**Recall**	**F_1**	**Precision**	**Recall**	**F_1**
Hawthorne et al., 2021 [38]	-	-	83.94	-	-	96.01
Hawthorne et al., 2018 [17]	51.32	49.31	50.22	84.24	80.76	82.29
Kelz et al., 2019 [16]	61.93	51.66	56.08	90.15	74.78	81.38
Ours, 2022	25.97	25.33	25.50	64.62	62.81	62.97
Kelz et al., 2016 [52]	20.13	27.80	23.14	44.27	61.29	50.94
Sigtia et al., 2016 [13]	17.64	19.71	18.38	44.97	49.55	46.58

Table 5: A comparison of our model with other models that were evaluated using the metrics from `mir_eval`. This table is sorted in descending order by the pitch and onset F_1 column.

At this time, our model performs better than Kelz et al. [46], and Sigtia et al. [13], which were both published in 2016. Although [16], [17], [38] outperform our current model, these results are encouraging. The F_1 score, when we exclude offset information from our preliminary model, is approaching competitiveness with recent models developed in 2021 by Google researchers, and our initial results outperform publications from only six years ago that are still relevant and cited today. However, of these published results, our approach is the only model to use raw audio as the input. Other approaches require preprocessing and feature exactration to produce spectrograms [16], [17], [38], [52] or constant Q transforms [13], [52]. We expect that with a few modifications, as discussed in Section 8, our model will be competitive with the newer models in the field, while retaining the advantage of minimal pre-processing on the input files. This minimal preprocessing has the potential to lead to next to real-time transcription of audio files.

Despite these encouraging results, there is much room for improvement in our first design. In order to discover how to best improve our model so that it can compete with Kelz et al. [16] as well as both studies by Hawthorne et al., [17], [38], we evaluated and scrutinized two individual ten-second songs within the dataset to help determine any areas of improvement.

7.2 Individual Song Evaluations

The test set consisted of 7123 ten-second song pieces, representing the 177 full-length performances. Although we cannot provide the results for each song individually, deeper examination on two songs with differing note lengths gives us the opportunity to examine the problems that arose, leading to our low `mir-eval` results.

7.2.1 Sonata in A Major, K. 208

As an example, let us walk through the first ten seconds of Sonata in A Major, K. 208, by Domenico Scarlatti, performed at the 2014 International Piano-e-Competition. This ten-second excerpt of the song has a waveform and an expected and predicted MIDI piano roll that are shown in Figure 8. Examining the expected MIDI output in the predicted MIDI output, we see that the overall shape is similar, but that notes are not always held for the correct amount of time. We explore these differences further using the heatmaps shown in Figures 9, 10 and 11.

Table 6 shows the values from the `mir_eval` library on the first ten seconds of Sonata in A Major, K. 208. The values in this table are weighted similarly to the values calculated for the full dataset.

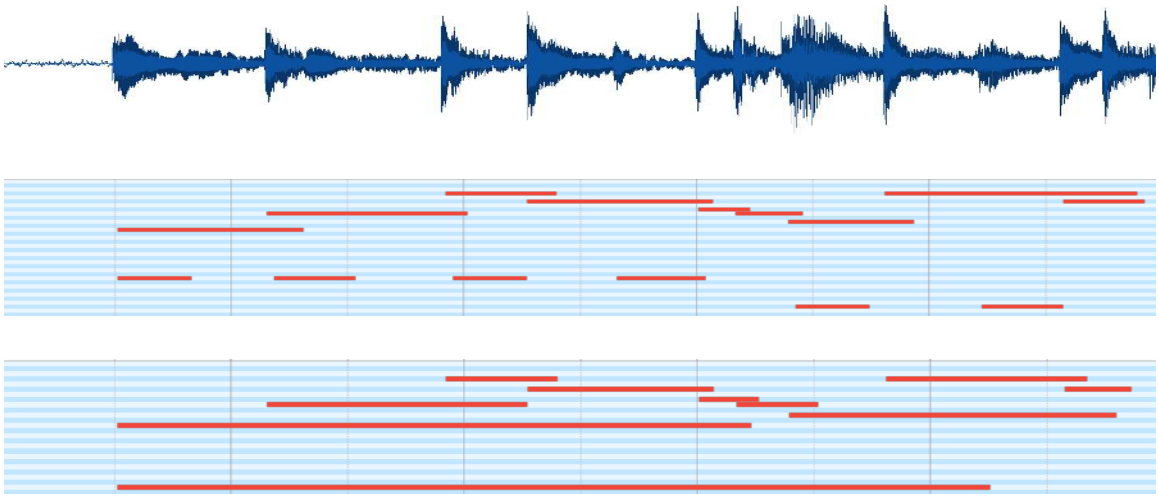


Figure 8: A comparison of the input (top), expected output (middle) and predicted output (bottom) for Sonata in A Major, K. 208.

	Pitch, onset, and offset	**Pitch and onset**	Onset only	Offset Only
Precision	0.347	0.669	0.823	0.734
Recall	0.267	0.516	0.634	0.556
F-measure	0.302	0.582	0.716	0.639
Average overlap ratio	0.993	0.760	-	-

Table 6: mir_eval.transcription results for the initial ten seconds of Sonata in A Major, K. 208.

These results give us insight into the notes and timing, but little information about the velocity. A heatmap is able to give us information about all three values. Figures 9 and 10 demonstrate the velocity and timing of all notes in the expected and predicted MIDI_TS files. For clarity, only the octaves in which notes are played are shown. In these heat maps, we can see precisely the velocity that the keys and pedals were pressed with, and what we predicted. We see that when the model predicted longer notes than were played, the velocity was lower than at the initial onset. We can quantify how close we were, for each note, at each timestep by subtracting the predicted values from the expected values, as shown in Figure 11.

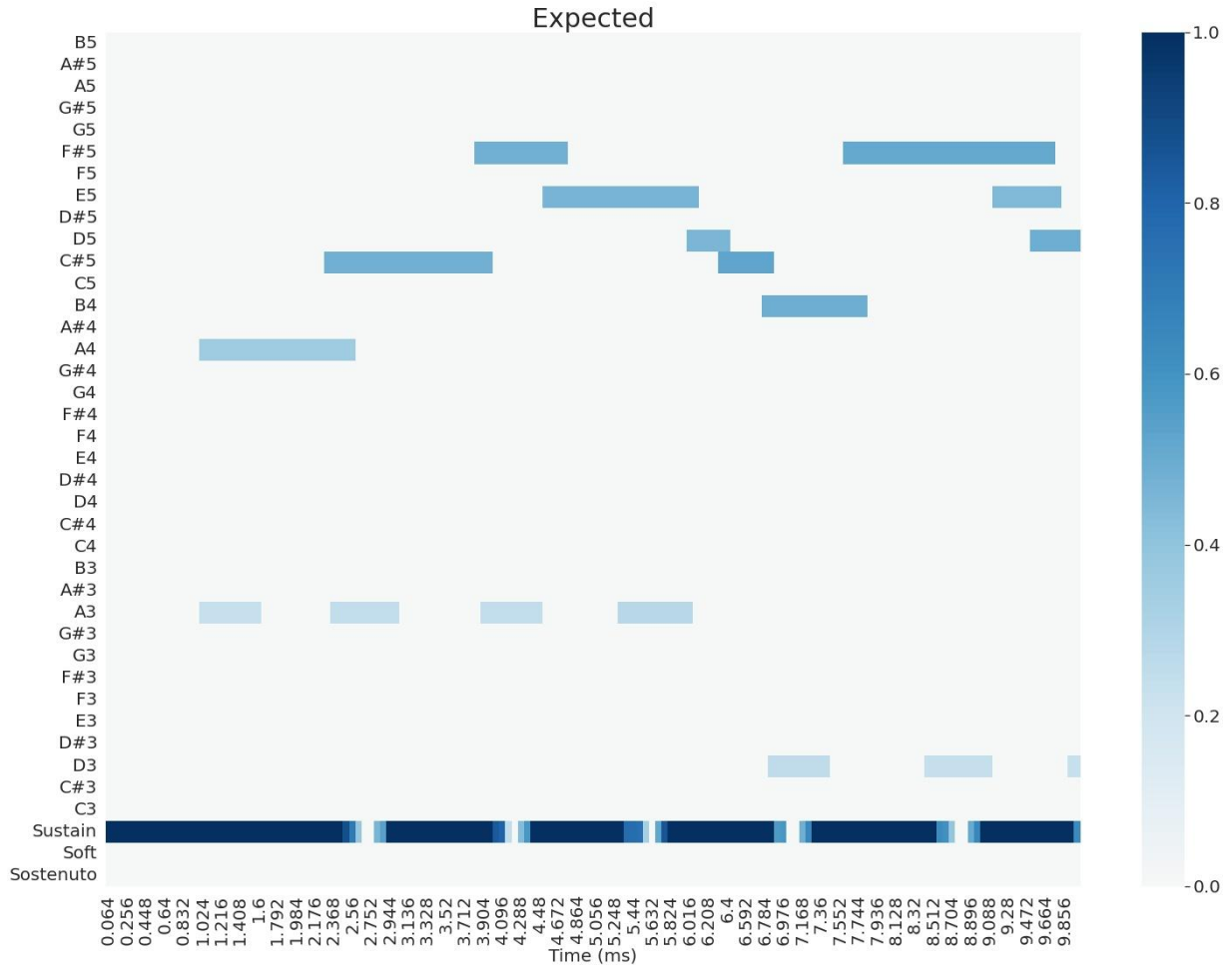


Figure 9: A heat map showing the notes, times, and velocities for the expected output of the model for the initial ten seconds of Sonata in A Major, K. 208.

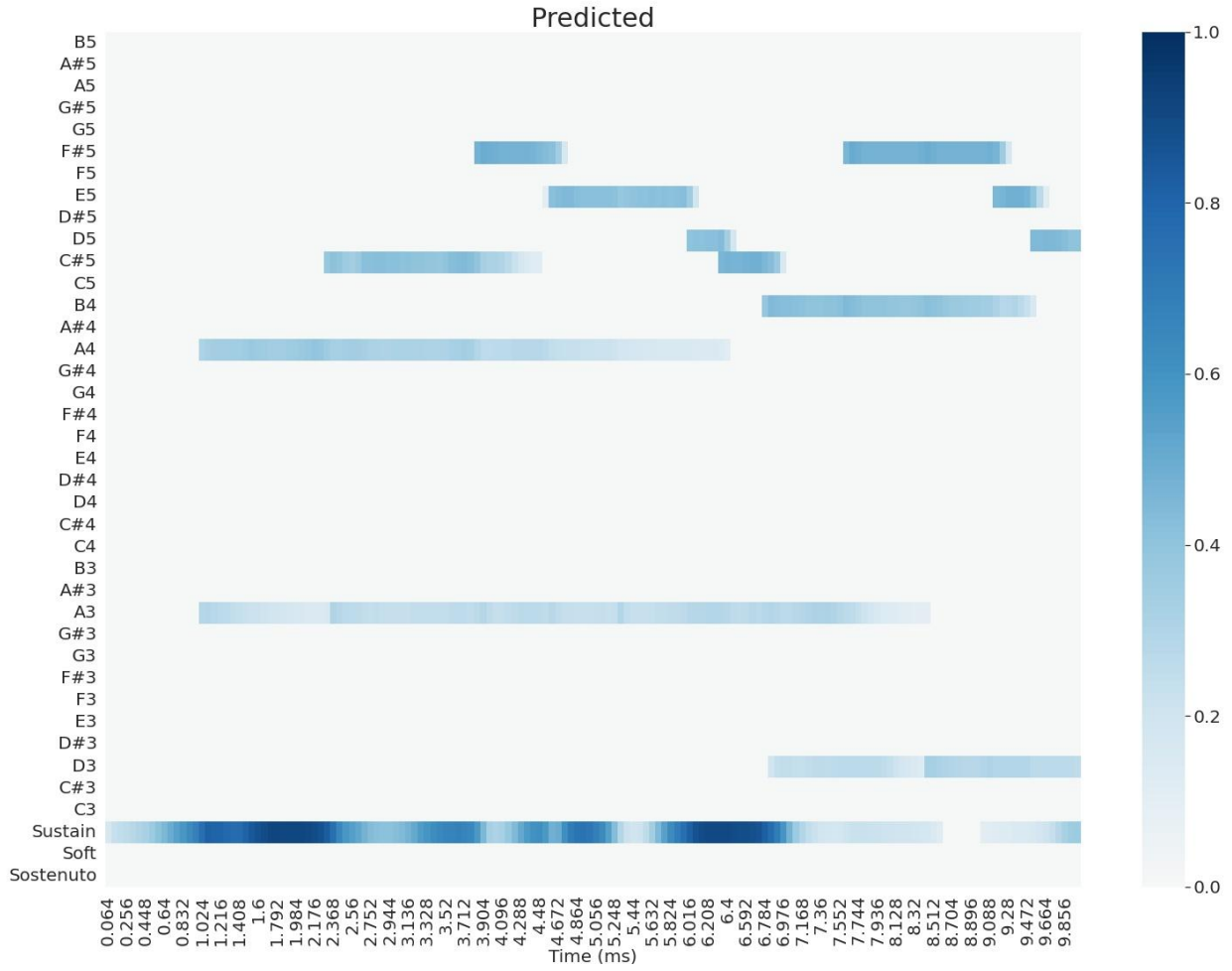


Figure 10: A heat map showing the notes, times, and velocities for the predicted output of the model for the initial ten seconds of Sonata in A Major, K. 208.

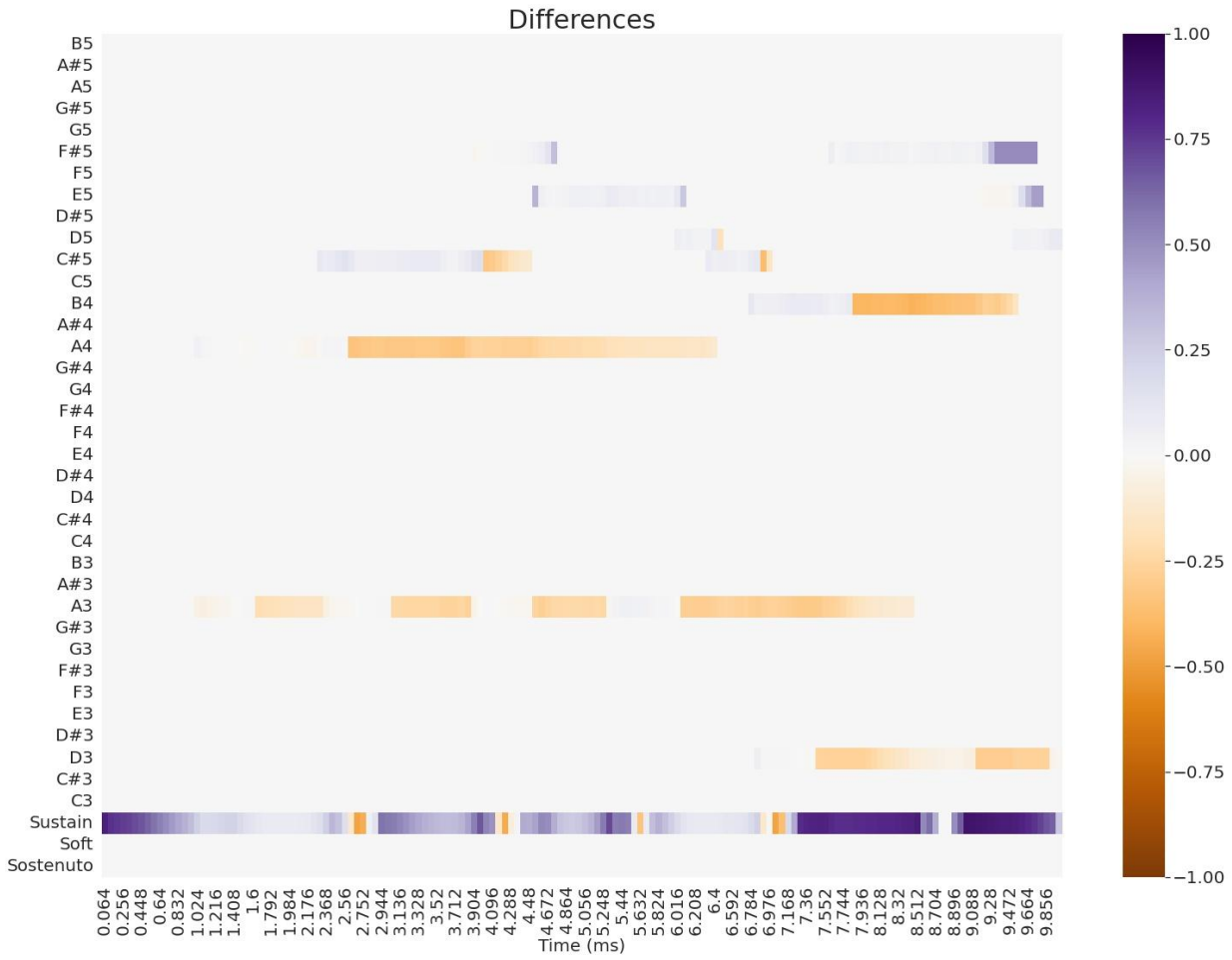


Figure 11: A heat map showing the notes, times, and difference between the velocities for the expected and predicted outputs of the model for the initial ten seconds of Sonata in A Major, K. 208.

From the scale on the left side of Figure 11, we can see that any orange values indicate that the value that we predicted was higher than the expected value, while purple represents predicted values that were lower than the expected value. Our mapping of MIDI velocity values from 0 to 1 means that our network is attempting to predict very precise numbers, without overfitting. This precision is a difficult task, demonstrated by the fact that in Figure 11 every note that is not a 0 is some shade of orange or purple on this graph. However, on this song, we see that when notes were correctly predicted to be playing, we have very very light shades of purple or orange, occasionally almost the neutral gray color. The prevalence of bright orange on the graph may, upon first glance, look like a demerit for our model, and is indicative of the problems with using `mir_eval` to evaluate our model. However, when we listen to the audio file for this song, and all songs, we hear notes held for much longer than recorded by the Disklavier piano. We infer that this longer duration is because individual piano notes resonate after the pianist has released the key; this feature is captured in the audio file, but is in conflict with the information presented in the MIDI file.

The purpose of the sustain pedal on a piano is to allow the strings within the piano body to reverberate further throughout the room. In the true heatmap, we observe that in the space between the offset and onset of the notes D3 and A3, the sustain pedal is being held (Figure 12). While our model did not do particularly well at predicting the pedal value, we observe that the offsets of notes do not occur when the sustain pedal is activated, indicating that we predict the effects of the pedal.

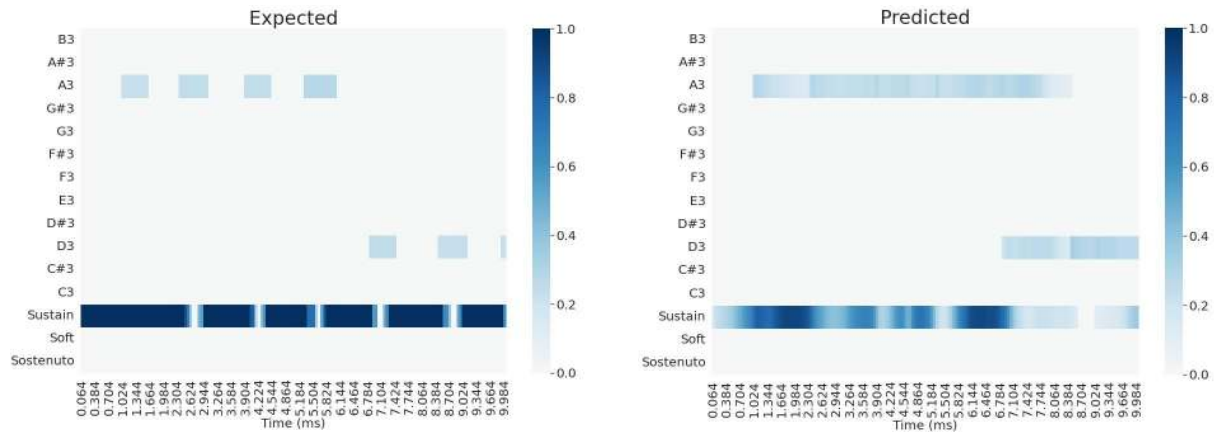


Figure 12: Comparative heatmaps for the third octave of the initial ten seconds of Sonata in A Major, K. 208. These heatmaps show that the pedal was activated between repetitions of A3 and D3 in the expected output, modeled by an increase in duration for the predicted output of A3 and D3.

Hawthorne et al. wrapped the pedal into their expected inputs to the model, and if the pedal was pressed while notes were active, the notes were extended for the length of the pedal [23], [38]. While we did not intentionally set out to model pedals this way, our model picked up on the fact

that that is the purpose of the pedals, and the note values were held. Integrating the pedal information is something that we plan to attempt as future work in order to increase the accuracy of our model when evaluated with `mir_eval`.

This pedal continuation, as well as the natural reverberation of a piano in a concert hall, made the sound of notes linger long after the physical key on the piano was released — something that our model was able to detect from the audio file, particularly when notes were repeated. This reverberation means, however, that our evaluation metrics are objectively low, partially because the audio and MIDI onsets and offsets do not align, and we consider notes played when they are not. Despite not using onset information as a feature of our model, we are still able to accurately predict onsets of most notes, particularly the onset of the first note in a repeating series.

7.2.2 Sonata in D Major, K. 118

The first example, in Section 7.2.1 has a majority of long notes. We also wanted to see how well our model did when the song consisted of shorter, staccato notes. This section assesses another ten seconds of a Domenico Scarlatti piece, again performed in 2014: Sonata in D Major, K. 118. Figure 13 shows the input, expected output and predicted output, and Table 7 shows that again, our `mir_eval` metrics are weighted similarly to the full dataset. Figure 13 and Table 10 both indicate that our model was better equipped to handle this song than our example from Section 7.2.1, as our `mir_eval` values are higher, and our piano roll images are more similar.

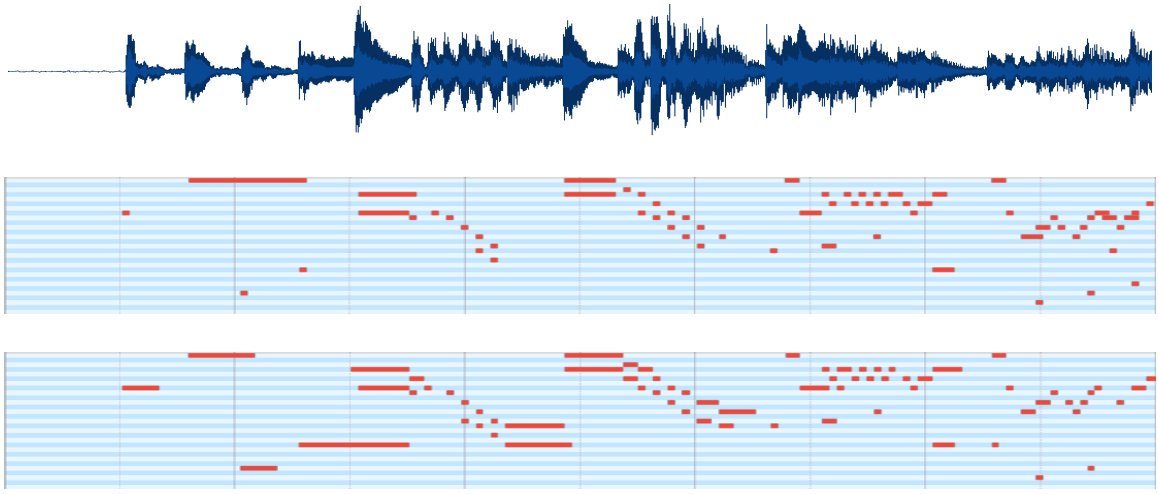


Figure 13: A comparison of the input (top), expected output (middle) and predicted output (bottom) for Sonata in A Major, K. 208.

	Pitch, onset, and offset	**Pitch and onset**	Onset only	Offset Only
Precision	0.624	0.894	0.894	0.765
Recall	0.589	0.844	0.844	0.722
F-measure	0.606	0.869	0.869	0.743
Average overlap ratio	0.989	0.838	-	-

Table 7: mir_eval.transcription results for the initial ten seconds of Sonata in D Major, K. 118

We again present the heatmaps for the expected values (Figure 14), the predicted values (Figure 15), and the differences (Figure 16) to observe any differences in velocity that may indicate a feature of our model.

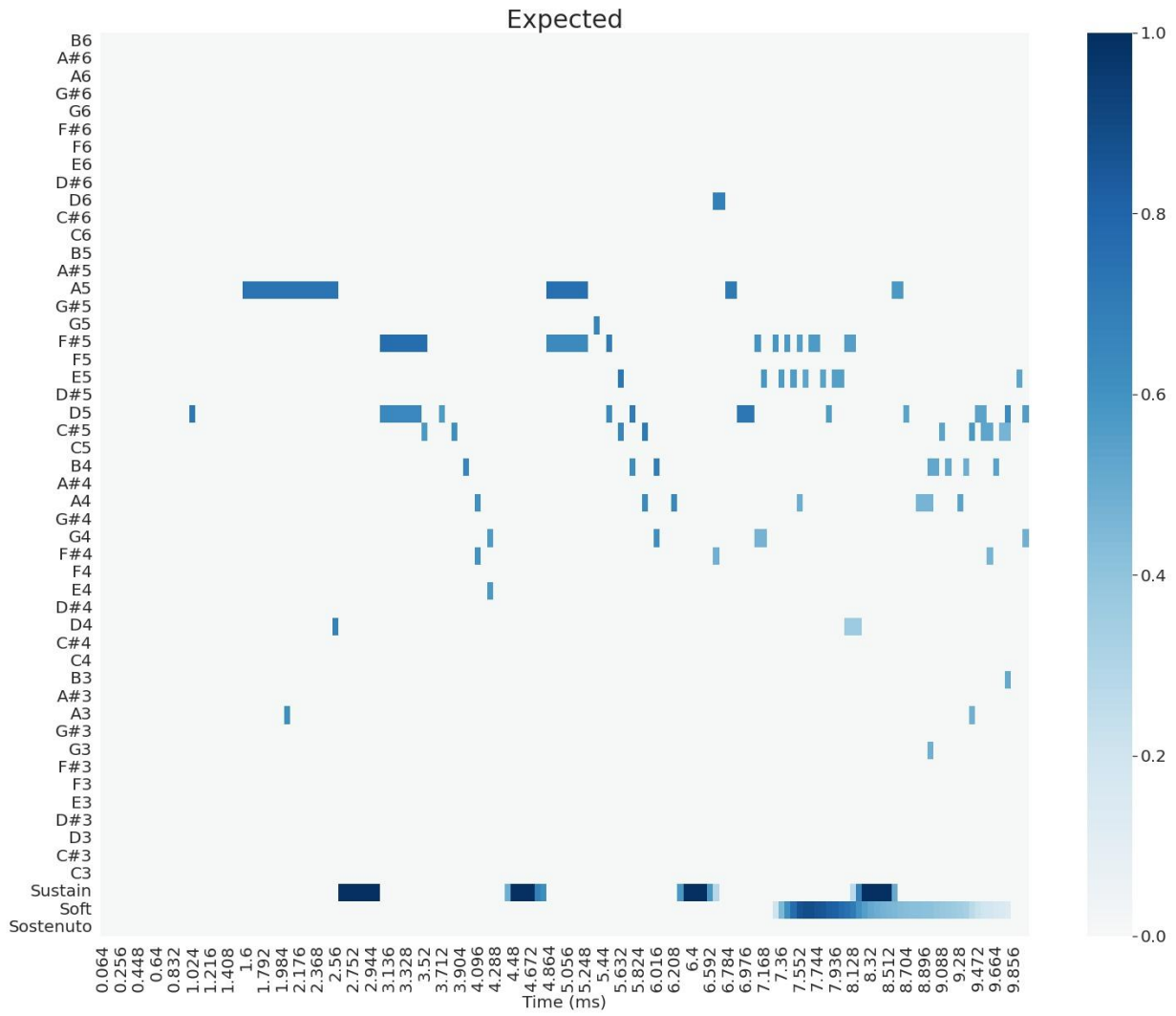


Figure 14: A heat map showing the notes, times, and velocities for the expected output of the model for the initial ten seconds of Sonata in D Major, K. 118

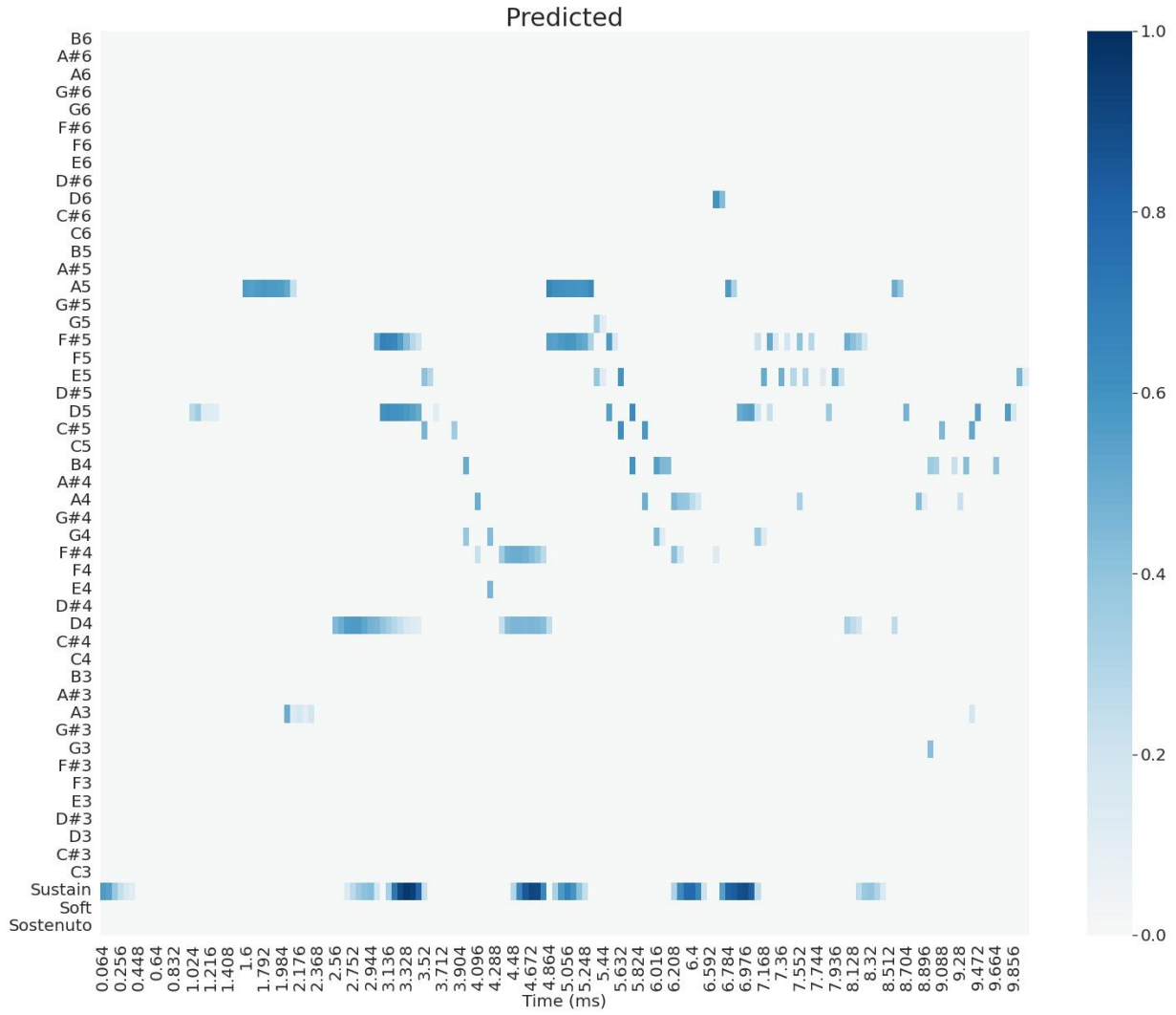


Figure 15: A heat map showing the notes, times, and velocities for the predicted output of the model for the initial ten seconds of Sonata in D Major, K. 118

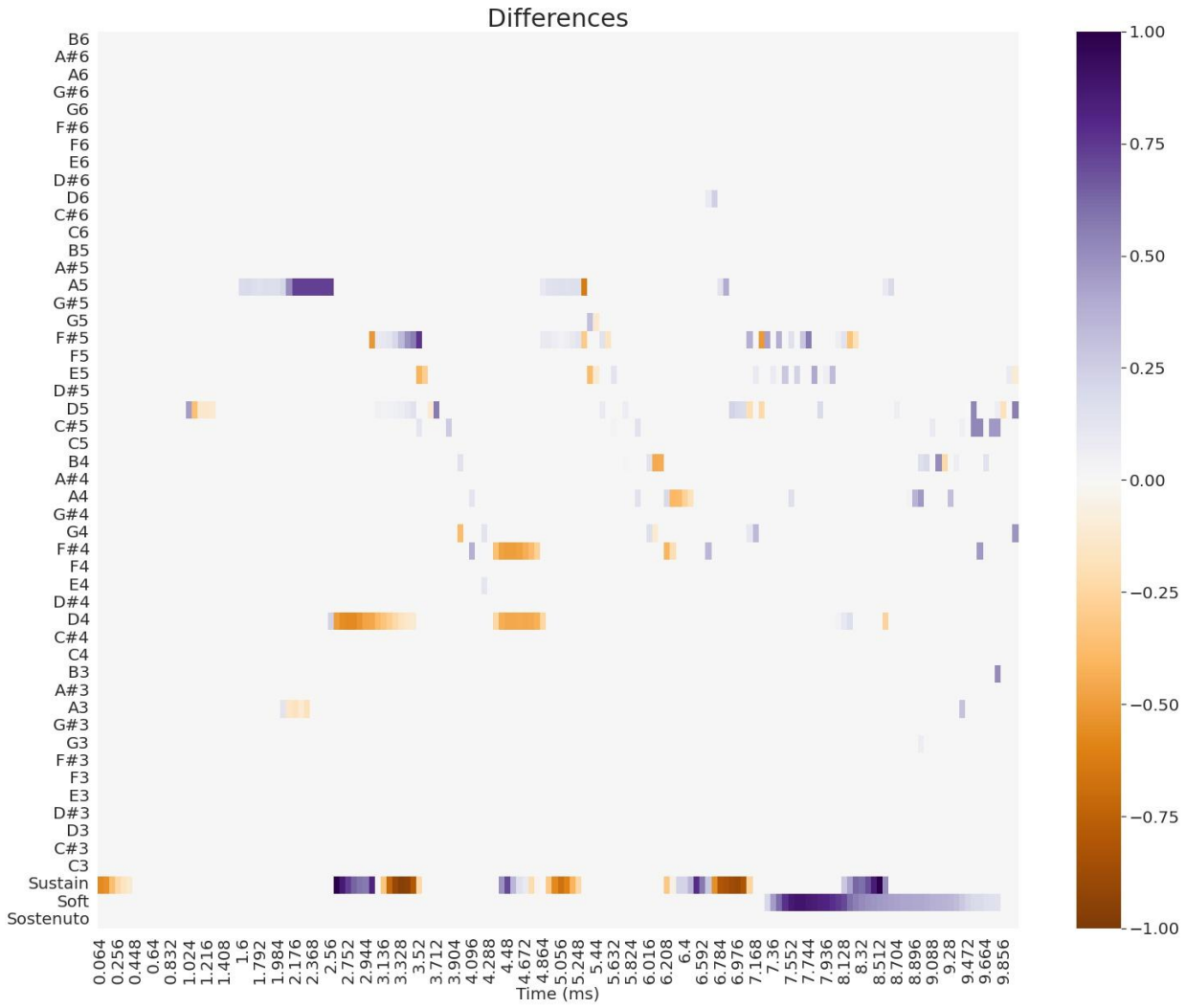


Figure 16: A heat map showing the notes, times, and difference between the velocities for the expected and predicted outputs of the model for the initial ten seconds of Sonata in D Major, K. 118.

In the previous example in Section 7.2.1, we illustrated issues that our model had with the sustain pedal. In this example, we note the same issues are present with the soft pedal. In fact, this time, we were unable to predict it at all. However, again, similar to the sustain pedal, our note predictions do reflect the behavior of the pedal. The soft pedal, as the name suggests, softens the sound of the notes. While the soft pedal is held down, our predicted notes tended to be of a much lower velocity, as seen by the prevalence of deeper purple towards the end of the differences heatmap in Figure 16. Just as with the example from Section 7.2.1 with longer notes, we tended to predict values longer than they were, particularly when the sustain pedal was held down. There are a few more instances where notes were not predicted at all, but in general, both examples show that our model did an excellent job at predicting when there would be sound coming from the piano, almost always predicted the correct notes in the correct octaves, and did relatively well at predicting the velocities of the notes. These successes are promising, and we consider them an excellent starting point for future research.

8 Conclusion and Future Work

We have shown that a BLSTM can be used to transcribe a solo piano performance directly from a raw audio file to a MIDI file. Although our results are not state-of-the-art, they demonstrate that intentional onset detection is not necessary to determine the onset of notes when using a BLSTM.

8.1 Pedal

One major drawback of our model was the treatment of the three piano pedals, particularly the sustain pedal. In a piano, the sustain pedal removes the dampers from the strings and allows strings to vibrate for a longer period of time, allowing the notes to reverberate for longer. This loss of dampers causes a legato effect, and makes the sound of notes last for longer. This effect is exacerbated in audio recordings by the acoustic features of the room that also cause sound to reverberate. We attempted to model this feature by including the pedals in our time-series, appending them after our notes columns. We hypothesized that this inclusion would allow the model to hear the sustained notes in the audio and correlate them with the presence and velocity of the pedal in the expected MIDI. Unfortunately, our model did not predict the pedals with any amount of accuracy. The pedals were modeled more after the piano keys, which have different strike methods than pedals. While the pedals themselves were not modeled, it appears that the pedal behavior was modeled, and notes were extended past their proper times at instances where the sustain pedal was pressed.

In the future, we propose a method similar to [17], [38] where the duration of a note is extended for as long as the pedal is held, or until the note is reiterated. This integration is done prior to training on the model, and the pedals are then excluded from the dataset. Integrating the pedal

should give us more accurate offsets, as well as more accurate onsets in instances where notes are reiterated under a pedal.

8.2 Velocity

The MAESTRO dataset includes information about the velocity of each note, which we chose to include in our time-series. The velocity ranges from 0 to 127, which we decided to model as a percentage. These numbers were over a wide range when normalized to between 0 and 1 and therefore difficult for our model to predict exactly. At times, this difficulty lead to a relatively large difference between the values of our true MIDI files, and our predicted MIDI files. In fact, when we converted to a playable MIDI format from a time-series, each time that had a separate velocity was treated as a new strike of a key. To combat the constant onsets, we took the average velocity across the duration of a note to create a playable file that sounded like real music. However, this averaging ignored the increase in velocity at the onset of repeated notes, when two notes were directly adjacent to each other in time.

Due to the loss of information from averaging the velocities, we chose not to include the velocity information in our full dataset evaluation, and instead to use it as supplementary data when evaluating specific performances. Future research is needed to determine the best way to smooth the velocities within the predicted output, whether by pre- or post-processing of the data, or updating the model to better predict the same velocity within a single note.

Velocity smoothing is, for the most part, only required when converting strictly to a MIDI file. If we were to instead attempt to convert from a time-series representation to sheet music, any decrease or increase in velocity would correspond to a crescendo or decrescendo. A suddenly higher velocity within a held note would indicate a repeated note under a slur or a pedal. In the future, we may be able to skip the MIDI step altogether and convert from a time-series directly to another musical representation, such as MusicXML or ABC notation.

In addition, we chose to set all predicted velocities that were lower than 0.1 to 0. While this threshold removed a large amount of noise from the predicted data, it left in low values that accounted for resonance in a room, but that were not present in the true MIDI file. For further model improvement, we plan to select a higher number to use for thresholding. Different methods could be used to obtain this number, including finding the minimum velocity value (excluding 0) within the MAESTRO dataset, or finding a formal definition of the lowest MIDI velocity that is playable by a human on a Yamaha Disklavier piano. A further option is given by [50], which states that the MIDI velocities with which the Disklavier has a linear reproducing ability — that is the values with which the piano is able to play back the MIDI files with the same velocity — are between MIDI velocities 30 and 85. This threshold would give us a minimum velocity of $\frac{30}{127} \approx 0.236$. Following the same principle as before, rounding down to 0.2 would be a good starting point for thresholding.

8.3 Evaluation

The output of our model was a MIDI_TS file, which gave the velocities of all MIDI notes every 64 milliseconds. The `mir_eval` library required the onset and offset time for each note with a velocity greater than 0. In the future, we plan to take more time in determining the correct offset times for our notes when converting to the `mir_eval` format. Rather than just choosing the last time segment in a series of velocities, we propose adding an offset/onset at the boundary between two time segments with significantly different velocities. Future work will involve determining what signifies a significant difference in velocity.

As further work to improve this model, we need to manually evaluate the songs that the model performed well on, and the songs that the model did not perform well on, and use the information gathered to determine the direction that we need to pursue in adjusting the model. This evaluation will include comparing the full data F1 score across songs with long versus short notes, as shown in Section 7.2, songs with higher versus lower notes, songs with more versus less notes, and songs with the sustain pedal versus songs without the sustain pedal.

8.4 Summary of Contributions

Our preliminary model shows promise for the use of raw audio and MIDI_TS in the field of automatic music transcription. Our initial model is competitive with older models in the field that are still cited today, and is approaching competitiveness with models from the past two years. Our model also captures features from audio files that are not present in MIDI files, including the reverberation of notes throughout a room after the piano key is released.

While recent high-performing models require costly transformation to spectrograms, using the nLog(n) fast Fourier transform algorithm or other feature extraction approaches, our model only requires that the audio have a sampling rate of 16000 Hz. By providing a model that takes in raw audio, our approach has the potential to be adapted to work in real time.

Future applications of our model include allowing composition by improvisation, or the transcription of jazz piano improvisations for future reproduction. We can also foresee our model producing conversions of recordings by famous (long dead) musicians that could be played back on a Disklavier to simulate live concerts. This model could also be used pedagogically to assist piano students visualize their performances. Another way our model could be applied artistically is to provide increased accessibility to the Deaf community by visualizing concerts in real-time, or to provide pianists with artwork that represents their piano performance mere moments after they finish performing.

9 References

[1] R. J. Saindon, L. H. Estrin, D. A. Brand, and S. Brand, “Systems and methods for automated audio transcription, translation, and transfer with text display software for manipulating the text,” US6820055B2, Nov. 16, 2004 Accessed: Jun. 04, 2021. [Online]. Available: https://patents.google.com/patent/US6820055B2/en

[2] P. Woodl, M. J. F. Gales, D. Pye, and S. Young, “The Development Of The 1996 Htk Broadcast News Transcription System,” May 1997.

[3] M. Marolt, “A Comparison of Feed Forward Neural Network Architectures for Piano Music Transcriptions,” 1999.

[4] J. Schmidhuber, “Deep Learning in Neural Networks: An Overview,” *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015, doi: 10.1016/j.neunet.2014.09.003.

[5] G. A. Carpenter and S. Grossberg, “The ART of adaptive pattern recognition by a self-organizing neural network,” *Computer*, vol. 21, no. 3, pp. 77–88, Mar. 1988, doi: 10.1109/2.33.

[6] M. M. Saritas and A. Yasar, “Performance Analysis of ANN and Naive Bayes Classification Algorithm for Data Classification,” *Int. J. Intell. Syst. Appl. Eng.*, vol. 7, no. 2, Art. no. 2, Jun. 2019, doi: 10.18201//ijisae.2019252786.

[7] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[8] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, “State-of-the-art in artificial neural network applications: A survey,” *Heliyon*, vol. 4, no. 11, p. e00938, Nov. 2018, doi: 10.1016/j.heliyon.2018.e00938.

[9] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” *CoRR*, vol. abs/1409.3215, 2014, [Online]. Available: http://arxiv.org/abs/1409.3215

[10] A. van den Oord *et al.*, *WaveNet: A Generative Model for Raw Audio*. 2016.

[11] B. Wang and Y.-H. Yang, “PerformanceNet: Score-to-Audio Music Generation with Multi-Band Convolutional Residual Network,” *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 01, Art. no. 01, Jul. 2019, doi: 10.1609/aaai.v33i01.33011174.

[12] “Understanding LSTM Networks -- colah’s blog.” http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed Apr. 05, 2022).

[13] S. Sigtia, E. Benetos, and S. Dixon, “An End-to-End Neural Network for Polyphonic Music Transcription,” p. 14.

[14] J. Nam, J. Ngiam, H. Lee, and M. Slaney, “A Classification-Based Polyphonic Piano Transcription Approach Using Learned Feature Representations,” p. 7.

[15] M. Marolt, “A Connectionist Approach to Automatic Transcription of Polyphonic Piano Music,” *IEEE Trans. Multimed.*, vol. 6, no. 3, pp. 439–449, Jun. 2004, doi: 10.1109/TMM.2004.827507.

[16] R. Kelz, S. Böck, and G. Widmer, “Deep Polyphonic ADSR Piano Note Transcription,”

ICASSP 2019 - 2019 IEEE Int. Conf. Acoust. Speech Signal Process. ICASSP, pp. 246–250, May 2019, doi: 10.1109/ICASSP.2019.8683582.

[17] C. Hawthorne *et al.*, "Onsets and Frames: Dual-Objective Piano Transcription," *ArXiv171011153 Cs Eess Stat*, Jun. 2018, Accessed: Sep. 14, 2020. [Online]. Available: http://arxiv.org/abs/1710.11153

[18] G. Loy, "Musicians Make a Standard: The MIDI Phenomenon," *Comput. Music J.*, vol. 9, no. 4, pp. 8–26, 1985, doi: 10.2307/3679619.

[19] A. P. Bell, "Can We Afford These Affordances? GarageBand and the Double-Edged Sword of the Digital Audio Workstation," p. 23.

[20] J. Rothstein, *MIDI:A Comprehensive Introduction, 2nd ed.* A-R Editions, Inc., 1992.

[21] S. Vasquez and M. Lewis, "MelNet: A Generative Model for Audio in the Frequency Domain," *ArXiv190601083 Cs Eess Stat*, Jun. 2019, Accessed: Sep. 13, 2020. [Online]. Available: http://arxiv.org/abs/1906.01083

[22] C.-Z. A. Huang *et al.*, "Music Transformer: Generating Music with Long-Term Structure," 2019.

[23] C. Hawthorne *et al.*, "Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset," *ArXiv181012247 Cs Eess Stat*, Jan. 2019, Accessed: Jun. 28, 2020. [Online]. Available: http://arxiv.org/abs/1810.12247

[24] K. Choi, C. Hawthorne, I. Simon, M. Dinculescu, and J. Engel, "Encoding Musical Style with Transformer Autoencoders," *ArXiv191205537 Cs Eess Stat*, Jun. 2020, Accessed: Sep. 13, 2020. [Online]. Available: http://arxiv.org/abs/1912.05537

[25] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A Generative Model for Music," *ArXiv200500341 Cs Eess Stat*, Apr. 2020, Accessed: Sep. 13, 2020. [Online]. Available: http://arxiv.org/abs/2005.00341

[26] B. C. van 't Spijker, "Classifying Classical Piano Music Into Time Period Using Machine Learning," Feb. 29, 2020. https://essay.utwente.nl/80618/ (accessed Sep. 13, 2020).

[27] J. de J. Guerrero-Turrubiates, S. E. Gonzalez-Reyna, S. E. Ledesma-Orozco, and J. G. Avina-Cervantes, "Pitch estimation for musical note recognition using Artificial Neural Networks," in *2014 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, Cholula., Mexico, Feb. 2014, pp. 53–58. doi: 10.1109/CONIELECOMP.2014.6808567.

[28] C. Raffel and D. P. W. Ellis, "Large-Scale Content-Based Matching of MIDI and Audio Files," 2015.

[29] Ning Hu, R. B. Dannenberg, and G. Tzanetakis, "Polyphonic audio matching and alignment for music retrieval," in *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No.03TH8684)*, New Paltz, NY, USA, 2003, pp. 185–188. doi: 10.1109/ASPAA.2003.1285862.

[30] A. Cont, "A Coupled Duration-Focused Architecture for Real-Time Music-to-Score Alignment," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 6, pp. 974–987, Jun. 2010, doi: 10.1109/TPAMI.2009.106.

[31] S. Sigtia, E. Benetos, N. Boulanger-Lewandowski, T. Weyde, A. S. d'Avila Garcez, and S. Dixon, "A Hybrid Recurrent Neural Network For Music Transcription," *ArXiv14111623 Cs*, Nov. 2014, Accessed: Mar. 30, 2020. [Online]. Available: http://arxiv.org/abs/1411.1623

[32] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," *ArXiv14093215 Cs*, Dec. 2014, Accessed: Feb. 19, 2021. [Online]. Available: http://arxiv.org/abs/1409.3215

[33] J. J. Valero-Mas, E. Benetos, and J. M. Iñesta, "Assessing the Relevance of Onset Information for Note Tracking in Piano Music Transcription," p. 8, 2017.

[34] S. Bock and M. Schedl, "Polyphonic piano note transcription with recurrent neural networks," in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Kyoto, Japan, Mar. 2012, pp. 121–124. doi: 10.1109/ICASSP.2012.6287832.

[35] A. Cogliati, Z. Duan, and B. Wohlberg, "Piano Transcription With Convolutional Sparse Lateral Inhibition," *IEEE Signal Process. Lett.*, vol. 24, no. 4, pp. 392–396, Apr. 2017, doi: 10.1109/LSP.2017.2666183.

[36] A. Cogliati, Z. Duan, and B. Wohlberg, "Context-Dependent Piano Music Transcription With Convolutional Sparse Coding," *IEEEACM Trans. Audio Speech Lang. Process.*, vol. 24, pp. 1–1, Dec. 2016, doi: 10.1109/TASLP.2016.2598305.

[37] A. Grossman, "Automatic Music Transcription: Generating MIDI From Audio," p. 6.

[38] C. Hawthorne, I. Simon, R. Swavely, E. Manilow, and J. Engel, "Sequence-To-Sequence Piano Transcription With Transformers," p. 8, 2021.

[39] M. Müller, V. Konz, W. Bogler, and V. Arifi-Müller, "Saarland Music Data (SMD)," *Late-Break. Demo Sess. 12th Int. Conf. Music Inf. Retr. ISMIR*, 2011.

[40] V. Emiya, N. Bertin, B. David, and R. Badeau, *MAPS - A piano database for multipitch estimation and automatic transcription of music*. 2010.

[41] A. Cogliati, D. Temperley, and Z. Duan, "Transcribing Human Piano Performances into Music Notation."

[42] J. L. Harding, "Yamaha DCFX ENPRO," *Riverton Piano Company*. https://www.rivertonpiano.com/scottsdale/pianos/player/disklavier/dcfx-enpro.php (accessed Apr. 24, 2022).

[43] "The MAESTRO Dataset," *Magenta*. https://magenta.tensorflow.org/datasets/maestro (accessed Jul. 07, 2020).

[44] T. Wedde, *py-midicsv: A library for converting MIDI files from and to CSV format*. Accessed: Apr. 10, 2022. [Online]. Available: https://github.com/timwedde/py_midicsv

[45] M. Müller, "Music Representations," in *Fundamentals of Music Processing*, Cham: Springer International Publishing, 2015, pp. 1–37. doi: 10.1007/978-3-319-21945-5_1.

[46] K. Krumbholz, R. D. Patterson, A. Nobbe, and H. Fastl, "Microsecond temporal resolution in monaural hearing without spectral cues?," *J. Acoust. Soc. Am.*, vol. 113, no. 5, pp. 2790–2800, May 2003, doi: 10.1121/1.1547438.

[47] "Mathematics in World Music: The Overtone Series," *HUMANITIES+ Journal*.

https://www.hplusjournal.com/home/mathematics-in-world-music-the-overtone-series (accessed May 14, 2022).

[48] M. Abadi *et al.*, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” p. 19.

[49] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *ArXiv14126980 Cs*, Jan. 2017, Accessed: Jun. 03, 2021. [Online]. Available: http://arxiv.org/abs/1412.6980

[50] W. Goebl and R. Bresin, “Measurement and reproduction accuracy of computer-controlled grand pianos,” *J. Acoust. Soc. Am.*, vol. 114, no. 4, pp. 2273–2283, Oct. 2003, doi: 10.1121/1.1605387.

[51] C. Raffel *et al.*, “mir_eval: A Transparent Implementation of Common MIR Metrics,” Oct. 2014.

[52] R. Kelz, M. Dorfer, F. Korzeniowski, S. Böck, A. Arzt, and G. Widmer, “On the Potential of Simple Framewise Approaches to Piano Transcription,” *ArXiv161205153 Cs*, Dec. 2016, Accessed: May 06, 2022. [Online]. Available: http://arxiv.org/abs/1612.05153