MONOPHONIC AUTOMATIC MUSIC TRANSCRIPTION WITH CONVOLUTIONAL NEURAL NETWORKS

Mina Lee

Turing Scholars Program
Plan II Honors Program
The University of Texas at Austin

May 2019

Philipp Krähenbühl, Ph.D.
Department of Computer Science
Supervising Professor

Gordon S. Novak Jr., Ph.D.
Department of Computer Science
Second Reader

ABSTRACT

Author: Mina Lee

Title: Monophonic Automatic Music Transcription with Convolutional Neural Networks

Supervising Professor: Philipp Krähenbühl, Ph.D.

This thesis utilizes convolutional neural networks for monophonic automatic music transcription of piano music. We present three different systems utilizing CNNs to perform onset, pitch, and offset detection to get a final output of sheet music. We compare these systems to the pYIN transcription system [8], and find that our systems perform a lot better than pYIN on piano music. Our TCN system based on Bai et al.'s TCN architecture [2] achieved the best results due to having the best offset detection, and we were able to get fairly accurate sheet music from this system.

ACKNOWLEDGEMENTS

I would like to first thank my advisor, Dr. Philipp Krähenbühl. He provided me with great feedback throughout the year, and I would not have been able to complete this thesis without him. Thank you so much Professor Krähenbühl!

I would also like to thank my friends and family, who gave me support throughout college. I would not have made it to this point without them. I would also like to thank my friend, Lydia Chen, in particular for her constant support and encouragement throughout this process as we both worked on our theses this year. Thank you so much!

TABLE OF CONTENTS

Chapter 1

Introduction

Music is an important part of the world, and an important part of our culture. Oral tradition was the primary method of preserving and sharing music throughout most of its history, but there is evidence that early societies like the ancient Egyptians used various forms of musical notation [11]. Western musical notation and sheet music as we know it today began its development in medieval Europe in the Church to preserve choral music. Sheet music was not used much outside of the Church until the invention of the printing press, which allowed printing on a large scale and enabled sheet music to spread. Today, sheet music has become the primary method of preserving and learning music for musicians, although oral tradition is still used sometimes to preserve traditional and folk music in particular. Unfortunately, writing sheet music can be a very tedious task, particularly when attempting to write sheet music from a recording. Automatic Music Transcription (AMT) is the process of using computers to automatically convert an audio file into sheet music, and it attempts to solve this challenge to help composers and musicians produce sheet music.

There are two main areas of AMT- monophonic and polyphonic AMT. Monophonic AMT deals with music consisting of a single melody without any accompaniment so that there is only one note at a given time. Polyphonic AMT deals with music consisting of multiple voices so that there may be a multiple notes playing at a given time. Polyphonic AMT is considered a lot harder than monophonic AMT due to the overlapping of harmonics, and there is still a lot of research being done to get polyphonic AMT to a satisfying result. Monophonic AMT has seen a lot more success, and has rarely been revisited by researchers as a result, leaving autocorrelation as the top method for monophonic AMT. We revisit the area of monophonic AMT in this thesis, and use neural network models to perform piano-roll estimation, where we perform pitch and note tracking. We do this by performing onset, offset, and pitch detection on an audio file. Taking the output from these models, we produce sheet music in the form of uncompressed MuseScore files, the type of file used by MuseScore, a free music notation software [10].

Chapter 2

Background

2.1 Sheet Music Components

Sheet music consists of various symbols to represent music. A musical line is read on a staff, which contains five lines and four spaces between the lines. Each line and space represents a different pitch, and the pitch is determined by which clef is at the start of the line. The treble clef represents higher pitches, while the bass clef represents lower pitches. In Figure 2.1 the treble clef is on the top staff and the bass clef is on the bottom staff. Notes are placed on the staff, and they are named A-G based on the clef and which line or space the note is on.

There are eight notes in an octave, so for example going from C to C, the notes are C D E F G A B C. However, sometimes notes have half steps between them, and these are called semitones. There are 12 semitones to an octave. When looking at a piano, you can think of semitones as the

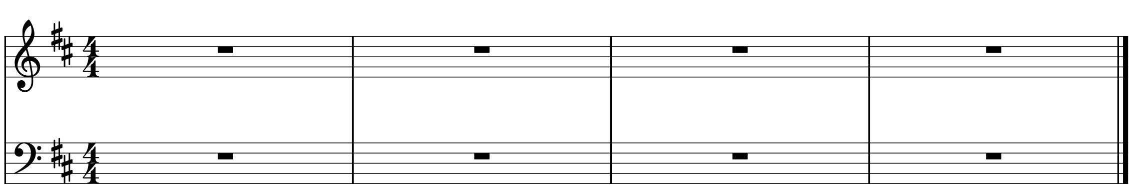


Figure 2.1. Musical Line.

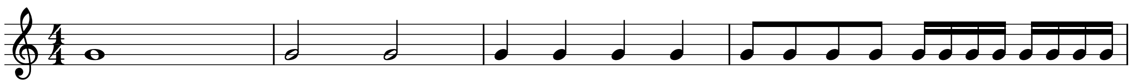


Figure 2.2. Note Values.

black keys and whole tones as the white keys. Semitones are denoted by sharps or flats, where sharps indicate the note is a half step higher, and flats indicate the note is a half step lower. Sharps and flats are shown next to notes, and are called accidentals. Another accidental is the natural sign, which tells you to play the whole tone of the note. Each semitone can be described with both sharps and flats, so for example Gb and F# are the same note. The key signature of a song tells you what sharps or flats to maintain throughout a song so you don't have to have an accidental every time the note comes up. The key signature can be seen by the clef as shown in Figure 2.1

While a note's pitch is determined by its position on the staff, its duration is indicated by the note symbol. The most common note values consist of a whole note, which represents 4 beats, a half note, which represents 2 beats, a quarter note, which represents 1 beat, an eighth note, which represents half a beat, and a sixteenth note, which represents a quarter of a beat as seen in Figure 2.2. For this program, we also include 32nd notes, which represents an eighth of a beat, and 64th notes, which represents a sixteenth of a beat. Rest values have a different appearance from note values, but they follow the same values as notes from whole rests to 16th rests and so on.

In order to create longer note durations, you can use dots and ties to string note values together. A dot next to a note adds half of the note's value to it, so for example a dotted half note would have a value of 3 beats $(2+\frac{1}{2}(2)=3)$. Ties connect notes, and they add the note values of each note together. For example, a whole note tied to a whole note would have a value of 8 beats

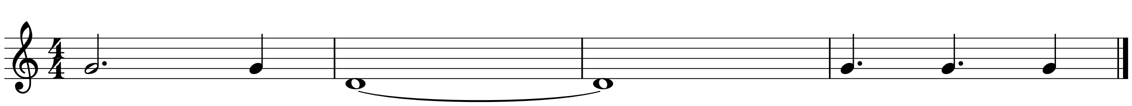


Figure 2.3. Dots and Ties.

$(4+4=8)$. These can be seen in Figure 2.3 with the dotted half note in measure 1 and the tied whole notes in measure 2 and 3. Thus, you can create a tremendous variety of note values with these discretized values.

Another important symbol in sheet music is the time signature, which tells you how to interpret note values. The time signature is present next to the clef at the very beginning of the song, and is shown again whenever it changes. It is composed of two numbers, one on top and one on bottom. The number on the top tells you how many beats are in a measure, which is denoted by vertical lines (or bars) on the staff. The number on the bottom tells you what note value counts as a beat. The common note values and their corresponding beats listed above were given in terms of when the quarter note counts as a beat, so this would be indicated with a 4 on the bottom. A common time signature is 4/4 time, where there are 4 beats to a measure and a quarter note counts as a beat. Another common time signature is 6/8 time, where there are 6 eighth note beats to a measure. The time signature can be seen by the clef in Figure 2.1.

Besides time signature, another thing that can impact a musician's perception of a beat is the tempo. The tempo of a song is given in beats per minute (bpm). It is present near the clef and tells you what note value to consider as a beat and the beats per minute of the song. For 4/4 time, the tempo is usually shown with the bpm for each quarter note, but sometimes composers give a bpm for each half note, which tells a musician to think of a half note as a beat for the given bpm. Another example is with 6/8 time, tempos are usually given with dotted quarter notes, telling musicians to think of dotted quarter notes as the beat instead of eighth notes.

While each component of sheet music contributes to capture a piece of music, the notes are the most essential part. We used onset, pitch, and offset detection to discern the notes from an audio recording. Onset and offset detection give us the timing of notes, and we can use the tempo of a song to calculate what beat the note occurs on and how many beats the note lasts for to actually figure out the notes to put on sheet music. We need a consistent tempo to do this accurately, and the only way to get a consistent tempo is to use a metronome. Thus, we chose not to include tempo detection in this thesis because having a metronome indicates people have already figured out the tempo for a piece, and people often do pick a tempo when composing a piece. We chose to not include time signature detection to this thesis as well because composers often pick a time signature when composing a piece, and having set time signatures reduces the problem space further to allow us to focus on getting accurate notes for the sheet music.

Having figured out the timing of notes with onset and offset detection and labeled tempos and time signatures, we use pitch detection to give us the pitch of each note, and essentially figure out the vertical position of each note in sheet music and whether a note has any accidentals. We chose to not include key signature detection in this thesis because composers often initially choose a key to write a piece in, and not including key signature detection allows us to isolate our problem and really just focus on figuring out the notes in a piece. Because we are not including key signature detection, we chose to show sharp accidentals next to any semitones to keep a consistent accidental when writing any semitones.

2.2 ADSR Envelope

We chose to use onset and offset detection to figure out the timing of notes based on the Attack Decay Sustain Release Envelope [3]. The ADSR Envelope describes the relationship between sound and volume over time when a note is played. It is the most common kind of envelope, as most instruments follow this model when a note is played. Per its name, the ADSR envelope has four stages: attack, decay, sustain, and release. We can discuss these stages in relation to a piano key being pressed.

The attack stage occurs when the piano key is pressed and there is an initial jump in volume as the beginning of the note sounds. The attack stage is associated with the onset of a note, and is a very distinctive stage for keyboard notes. Following the attack stage is the decay stage, where the sound decreases until it reaches the sustain level of the note. During the sustain stage, the note's volume holds steady as the key is continually held down. Following the sustain stage is the release stage when the sound decreases from the sustain level to zero after the key has been released. This stage is associated with the offset of a note.

Based on the ADSR Envelope, the attack stage and the release stage are two distinctive stages in the sound envelope for a note, and they can be used to detect the starts and ends of notes to figure out the timing of a note. In fact, a common form of onset detection for monophonic transcription is finding peaks in a signal's envelope. For example, the onset detection method by librosa [9] computes a spectral novelty function, which detects local changes in a signal's spectral content, and detects onsets as peaks in the envelope.

2.3 Monophonic AMT

In this thesis, we revisit the research area of monophonic automatic music transcription. One of the prominent methods in previous research for monophonic transcription is the collector system, which figures out onsets, pitches, and offsets based on autocorrelation pitch tracking. The estimate of autocorrelation of N length sequence x(k) is defined as

$$\frac{1}{N}\sum_{k=0}^{N-n-1} x(k)\cdot x(k+n) \tag{2.1}$$

where n is the lag or period length, and $x(n)$ is the time domain signal. Using autocorrelation pitch tracking, the collector detects moments in the sequence where the pitch is constant, and identifies a note onset as the first time step in the constant pitch sequence. A note offset is determined as the time step where the signal energy falls below a certain threshold. With detected onsets, pitches, and offsets, note durations can be calculated and a transcription can be made.

2.4 Deep Networks

In this thesis, we used convolutional neural networks (CNNs) for the task of monophonic automatic music transcription. CNNs are a special type of neural network that rose to popularity due to its success in image classification tasks, as well as audio synthesis with WaveNet [17]. CNNs contain convolutional layers, which essentially extract features from a region of the input through filters. With convolutional layers, the structure of the input is retained, allowing networks to recognize spatial and temporal dependencies. The idea of capturing temporal dependencies is particularly relevant for the task of automatic music transcription, since an audio file denotes sounds over a period of time.

A recurrent neural network (RNN) is another type of neural network that is able to capture temporal dependencies, and use its ”memory” from previous time steps to create an output for the current time step. We chose to use CNNs over RNNs for this task because CNNs are more powerful than RNNs since they can apply convolutions in parallel, they have more options for changing the receptive field of the network, they have more stable gradients than RNNs, and they also have a lower memory requirement for training [2].

Chapter 3

Preliminary Work

3.1 Sound Processing

In order to analyze music, we need to get it into a representation of frequency over time so that we can detect a note's pitch. One way to do this is through the Short-Time Fourier Transform (STFT) [1]. STFT essentially takes a signal and separates it into smaller time segments using a moving window and calculates the Discrete Fourier Transform (DFT) for it. It is defined as [1]

$$X_n(e^{j\omega_k}) = \sum_{m=-\infty}^{\infty} w(n-m)x(m)e^{-j\omega_k m} \tag{3.1}$$

where $X_n(e^{j\omega_k})$ is the STFT of input signal $x(n)$ and frequency ω_k with a window function of $w(n)$. This can be calculated with the Fast Fourier Transform (FFT) algorithm. Taking the STFT of a signal produces a frequency representation over time, where the frequency bins are spaced linearly. The length of the window determines the resolution of the time and frequency, with the tradeoff where longer windows have a higher frequency resolution but a lower time resolution.

The Constant-Q Transform (CQT) is another way of representing frequencies over time. It is defined as [14]

$$X^{CQ}(k,n) = \sum_{j=n-\lfloor N_k/2 \rfloor}^{n+\lfloor N_k/2 \rfloor} x(j)a_k^*(j-n+N_k/2) \tag{3.2}$$

where $x(n)$ is the signal, $k = 1, 2, ..., K$ is the index for the frequency bins, and $a_k^*(n)$ is the complex conjugate of $a_k(n)$. $a_k(n)$ is defined as [14]

$$\frac{1}{N_k} w(\frac{n}{N_k}) exp[-i2\pi n \frac{f_k}{s}] \tag{3.3}$$

where f_k is the frequency center of bin k, s is the sampling rate, N_k is the window length, and $w(t)$ is a window function that's zero outside $t \in [0, 1]$. The CQT basically creates logarithmically-spaced frequency bins, and transforms a signal into the time-frequency domain. The CQT ensures that it has a constant Q-factor, which is essentially a ratio between frequency and resolution, for its bins. This means that CQT exhibits better a frequency resolution at lower frequencies and a better time resolution at higher frequencies.

We chose to use the CQT over the STFT to represent our audio because the CQT is better equipped to represent music with its logarithmically spaced frequency bins, which mirror the spacing of pitches. Since the CQT has fewer frequency bins to cover a wider range of frequencies, this also allows CQT to handle different octaves better than STFT. In addition, the frequency resolution from CQT mirrors how humans perceive frequencies, with a higher frequency resolution for lower frequencies and a higher time resolution for higher frequencies, whereas STFT has a fixed resolution determined by the width of its window function.

3.2 Probabilistic YIN algorithm (pYIN)

The probabilistic YIN algorithm proposed by Mauch and Dixon [8] performs fundamental frequency estimation. It is a variant of the YIN algorithm, which is often used for monophonic pitch estimation, and is based on autocorrelation. pYIN consists of finding multiple pitch candidates and their probabilities and then using a hidden Markov model to track the pitch candidates and output a monophonic pitch track. pYIN used the collector system described in the background section to create a transcription and detect notes. Tested on a database of synthesized singing, pYIN became the state of the art for monophonic pitch estimation. We compared the results of our model to the pYIN collector system.

3.3 Convolutional Neural Networks in Transcription

Convolutional Neural Networks (CNNs), which gained a lot of success in the area of image classification, have become popular in the field of Automatic Music Transcription (AMT) as well [7]. In general CNNs in AMT have been used for polyphonic AMT due to its increased complexity. They have been applied to many different tasks within polyphonic AMT, including onset detection methods [13] and piano-roll estimation methods [4, 5, 15].

CNNs have seen less popularity in the area of monophonic AMT, particularly because monophonic AMT has seen less activity recently and is considered an easy task compared to polyphonic AMT. However, CNNs have seen some recent activity in monophonic AMT in the task of end-to-end transcription, where sheet music is directly produced from models [12, 16]. The only use of CNNs in piano-roll estimation to our knowledge is the model CREPE (Convolutional Representation for Pitch Estimation) proposed by [6].

CREPE utilizes convolutional layers to estimate the pitch of each time frame independently, and it performed just as well or better than pYIN and SWIPE, the state of the art methods in monophonic pitch tracking. CREPE was shown to generally be more robust to noise, and to be very accurate even with a strict evaluation threshold. CREPE became the new state-of-the-art for monophonic pitch tracking, and demonstrated that CNNs can be successful in monophonic piano-roll estimation. In this thesis, we explore the use of CNNs in piano-roll estimation by using them in onset, offset, and pitch detection.

3.4 Temporal Convolutional Network (TCN)

The TCN is a generic architecture for the sequence modeling task proposed by [2]. The sequence modeling task has an input sequence, and the goal is to predict a corresponding output at each time step with the constraint that we can only use the inputs that have already been observed.

The TCN achieved this through the use of dilated convolutions, which are like normal convolutions but have set gaps, allowing one to get a larger receptive field. In order to get a large receptive field, but also consider each part of the input, the proposed TCN includes exponentially increasing dilation factors for the dilated convolutions.

The TCN is structured by residual blocks containing two layers of dilated convolutions with weight normalization applied and ReLU and dropout layers after them. Since the input and output of each residual block can have different widths, a 1x1 convolution is used to convert the tensors to the same shape so the input and output can be added to follow residual blocks. The TCN was applied to polyphonic music transcription, and was able to outperform many architectures, although not all. We used the TCN in one of our systems.

Chapter 4

Technical Section

The input for all of our models was the Constant-Q Transform of a wav file cropped down to 50 time steps. We used *librosa* [9] to load the wav files and calculate the CQT to generate 88 frequency bins that represents notes A0-C8, which are all the notes found on a piano. In training, we used the Adam optimizer with a learning rate of 1e-3 and weight decay of 1e-4 and the Binary Cross Entropy loss as our loss function for all of our models.

4.1 Onset Detection and Pitch Detection (OP)

The first system that we used included a model for onset detection and a separate model for pitch and note duration detection. We used convolutional layers

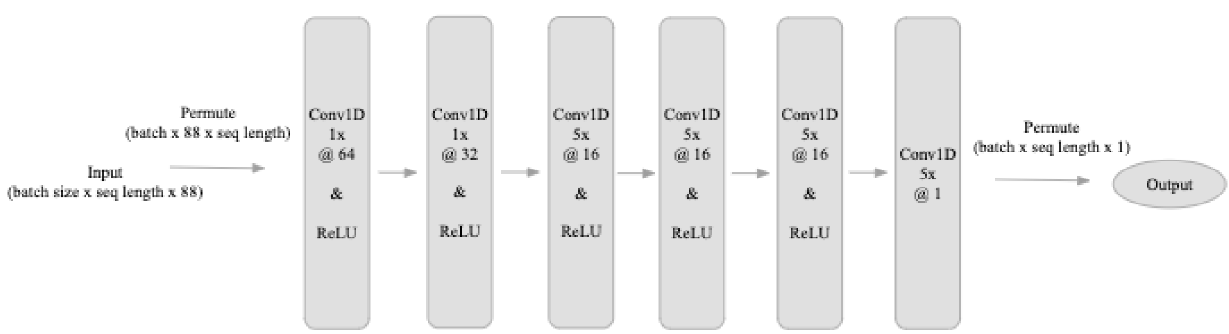


Figure 4.1. Onset Detection Model.

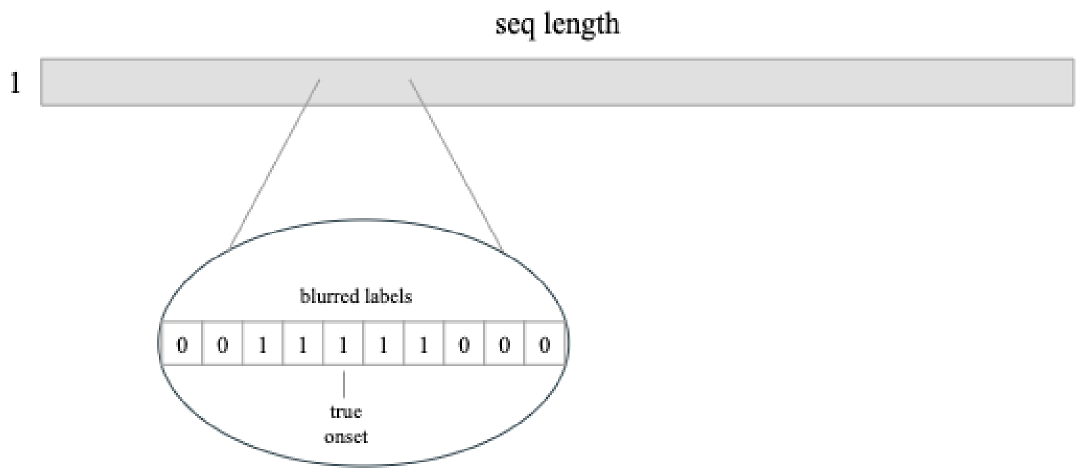


Figure 4.2. Labels for Onset Model.

for our onset detection model to classify each time step as an onset as shown in Figure 4.1. In our model, we applied convolutional layers over the sequence of the input to account for temporal dependencies. After convolving over time, we permute the output back to batch size x sequence length x 1 to predict whether or not there is an onset at each time step. During training, we have a batch size of 16 and a sequence length of 50. We had 88 frequency bins to represent all the notes on a piano.

For our labels for the onset detector, we marked each onset as a 1 and blurred it out over 5 time steps as shown in Figure 4.2. At first we viewed a certain time step to be an onset if the output of the model was over a certain threshold, but we found that to be too imprecise. We changed

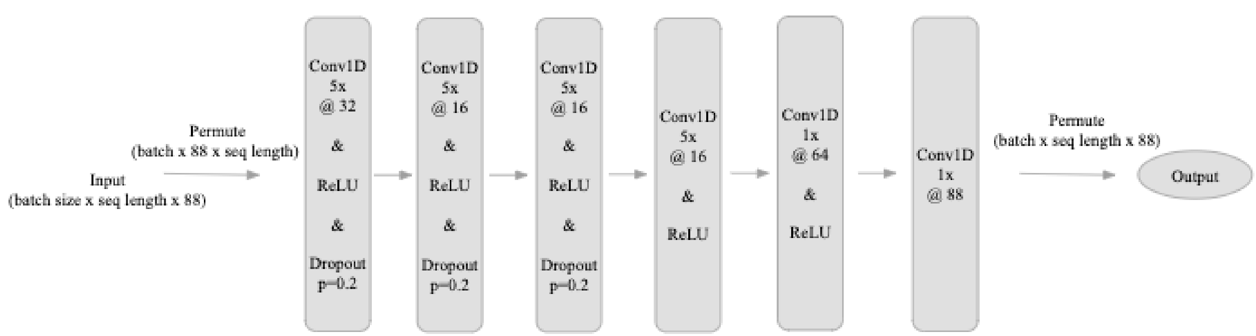


Figure 4.3. Pitch Detection Model.

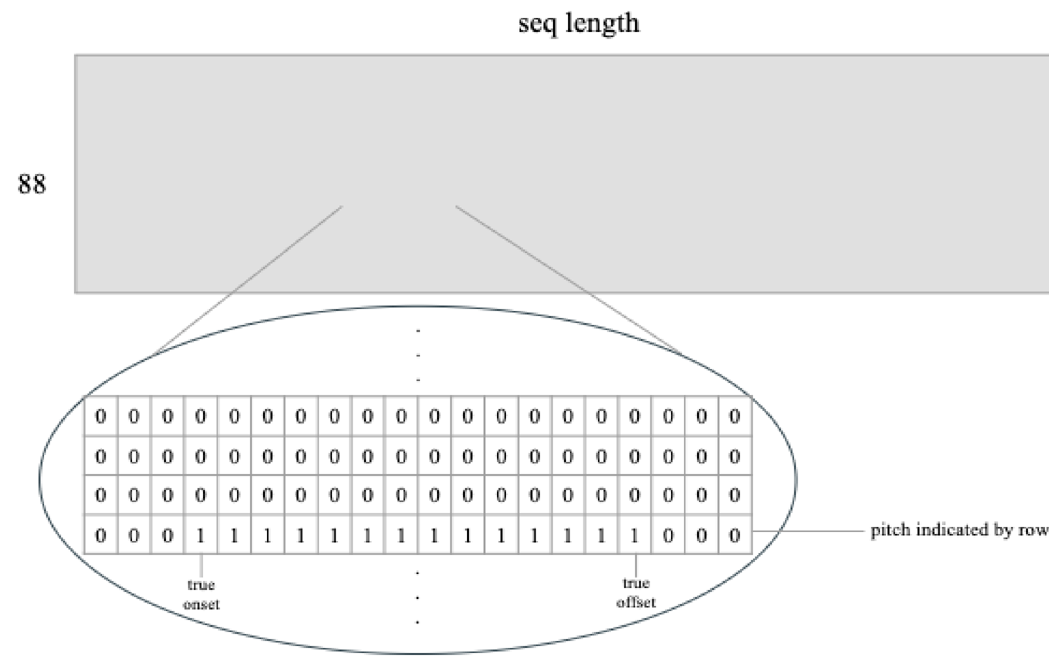


Figure 4.4. Labels for Pitch Model.

our approach to finding the peaks in the model output with a height of at least 1 and a minimum distance of 5 between the peaks since the onset labels were blurred over 5 time steps.

The pitch detection model that we created was similar to the onset model. As shown in Figure 4.3, we used convolutional layers over the sequence to capture the temporal dependencies in the audio and produced an 88 feature output to predict whether there is a pitch at each time step.

We used our pitch model to detect the pitch and duration of each note, so for the input of our model we only fed in the time slice between one detected onset and the next detected onset. From the output of our model, we took the max over the 88 pitch classes, and if it was over a threshold of 1, we counted that pitch. We stored all the detected pitches for a note, and chose the pitch that was detected the most to be the pitch of the current note. The offset then became the last time step where the determined pitch was found. Our labels for our pitch detection model were structured as shown in Figure 4.4. For each note, we marked 1s from the onset to the offset of the note on the row corresponding to the pitch of the note.

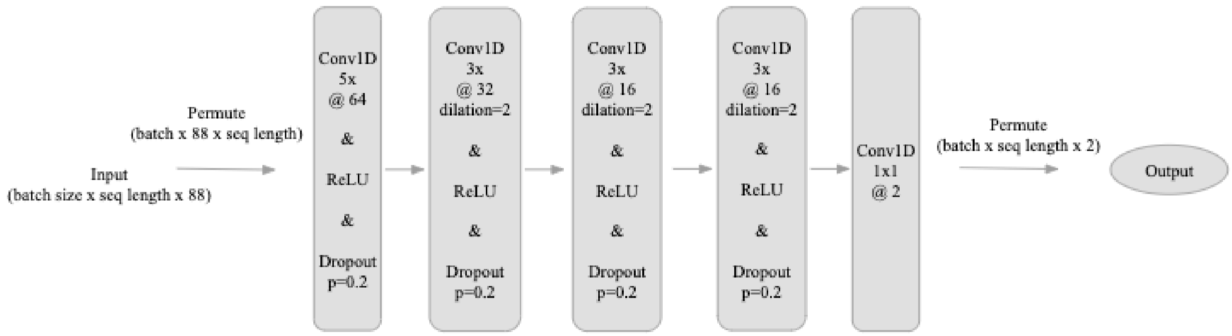


Figure 4.5. Onset/Offset Detection Model.

4.2 Onset/Offset Detection and Pitch Detection (OFP)

The second system that we created mainly only changed the onset detection model. We found offset detection to have the most unsatisfactory results from our first model, so we changed our system to have one model detect onsets and offsets and another model to detect pitch. Our new onset and offset detection model, as seen in Figure 4.5, has 5 one dimensional convolutional layers with ReLU activations and dropout layers with a probability of 0.2 between them. The onset/offset detection model predicts whether each time step is an onset or an offset.

We changed the labels to mark both onsets and offsets blurred over 5 time steps as seen in Figure 4.6. To determine the onsets and offsets, we followed the same method as before by finding the peaks in the output of our model. In the cases where an offset was not detected for an onset, we created the offset to be the next onset or 5 time steps after the onset for the last onset.

For pitch detection, we used the same model, but changed the input to be the time slice between the current onset and its offset. We stored all the detected pitches and chose the pitch to be the one present over the most time steps.

4.3 TCN Onset/Offset Detection (TCN)

The third system that we created only changed our onset and offset detection model. We wanted to further improve upon the offset detection of our model so we decided to incorporate the

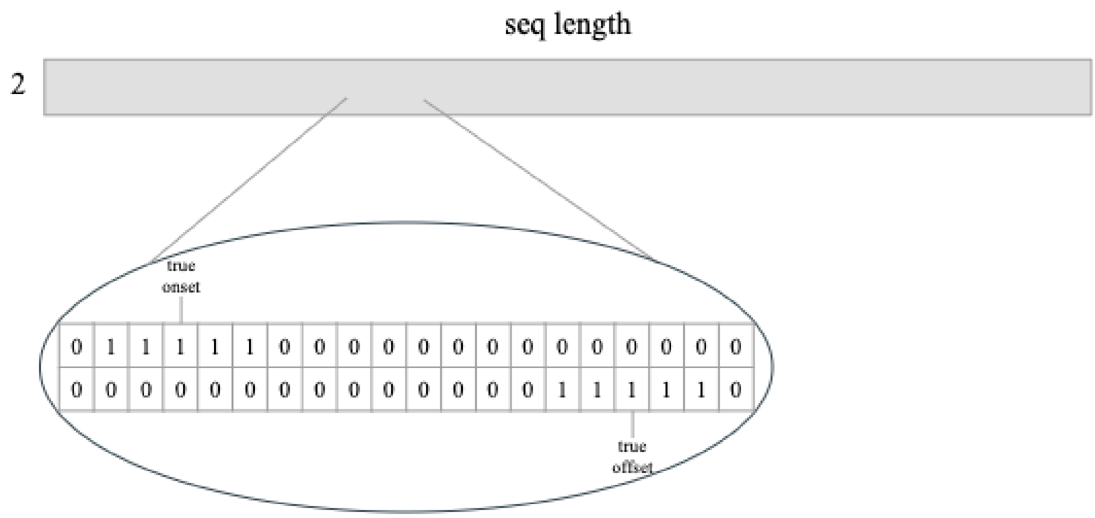


Figure 4.6. Labels for Onset/Offset Model.

temporal convolutional network discussed in the related works. This would allow us to have a more causal network through the use of dilated convolutions so that the network would use information from the previous steps in its prediction for the current step. We took the residual blocks found in [2]. For our network, we used a kernel size of 5 and a dropout probability of 0.2 for all of our blocks. For each block we had the dilation factor as 2^i at level i of the network. Since we wanted a receptive field of at least 50 due to the 50 sequence length during training, We had 3 residual

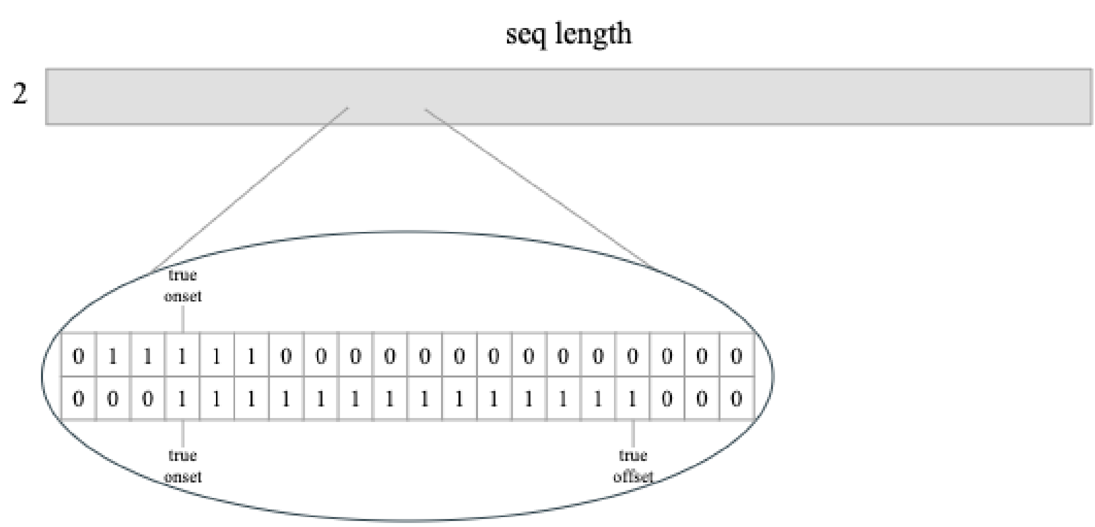


Figure 4.7. Labels for TCN Model.

blocks to get a receptive field of 65.

In addition to changing the structure of our model, we also changed our model to detect onsets and detect when a note was on or off and structured the labels as seen in Figure 4.7. We continued to blur the onset labels over 5 time steps, and also blurred the onset and offsets of our note labels. We found the onsets by detecting the peaks in our output again, and found the offsets by finding the last time step where a note was detected. We kept the pitch model the same.

4.4 Generating Sheet Music

We generated uncompressed Musescore files after getting a list of notes from our models. Since the onsets and offsets were labeled by time frames, we had to convert the frames back to ms with $frame * hop\_length / sample\_rate$ with a hop_length of 512 and sample rate of 22050. Taking the onset in ms, we then used the tempo and time signature of the song to calculate the measure and beat of the onset. We got the duration in terms of beats for each note by using the tempo and the onset and offset time in ms.

With this information for each note, the next step was to figure out the note value for each note. We processed one measure at a time to figure out the note values of the notes in the measure. This was the most challenging part of generating the sheet music because you can have so many different note values through the use of tied notes. We ended up using a recursive rounding scheme to figure out the best note value for a note starting with a round base b of 0.5. If this does not work because it violates the time signature or ends up overlapping with another note, we recurse with $b/2$ while $b >= 0.0625$. After getting note values for all the notes, we can then create a new Musescore file and use the song information labels including the time signature and tempos in our song to get the final product.

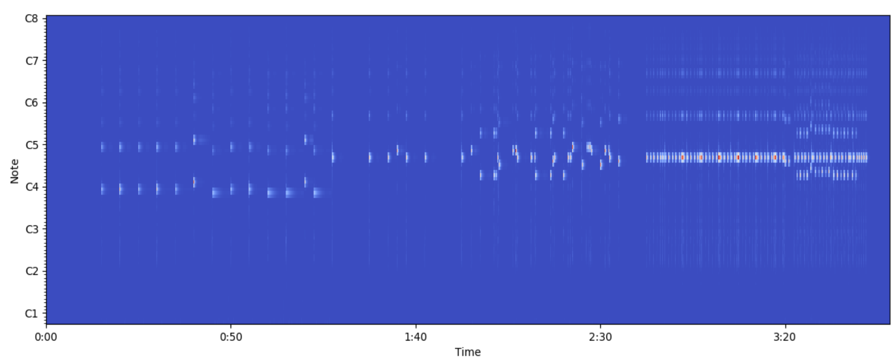
Chapter 5

Dataset

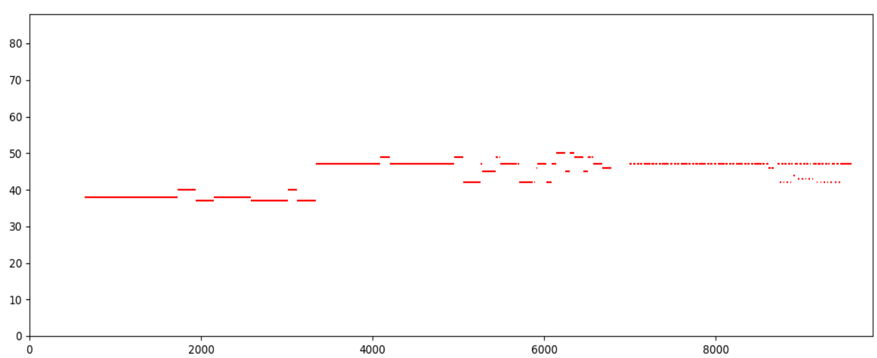
We used MuseScore [10], an open source notation software with a collection of public scores online, to generate our data. For our smaller dataset, we used 13 scores that all used the keyboard instrument. We isolated each part within a score and only used a part if it was monophonic and we were able to create labels for it. Unfortunately when writing a piano part on Musescore, you are unable to isolate the treble and bass clef from each other if both parts are written, so we were not able to use a lot of the parts in the larger dataset due to this problem. For our smaller dataset we occasionally had parts where there were multiple notes written at the same time, so that took away parts as well. In the end, we had 54 wav files for our smaller dataset and 65 for our larger dataset for piano instruments. For each wav file (isolated part), we stored song information labels and labels for each note. Our song info information included the time signature and tempo of the song and the time in ms and measure number that this information is at. For each note, we included the following labels:

start_time	the time in ms that the note onset occurs
end_time	the time in ms that the note offset occurs
pitch	the MIDI pitch of the note
measure	the measure number the note is in
beat	the beat within the measure the note onset occurs
note_val	a string denoting the value of the note
instrument	the instrument the part is on

We generated these labels by using ElementTree to read each Musescore file and get the needed information from the file. In Figure 5.1b, we show the start time, end time, and pitch labels for one of our audio files and compare it to the CQT of the audio. For the start time and end time, we converted the time in ms to the corresponding time frame.



(a) Constant Q Transform



(b) Note Labels

Figure 5.1: Comparison of CQT to Note Labels

Chapter 6

Results

We compared the result of our models using precision and recall. We calculate precision as

$$\frac{T_p}{T_p + F_p}$$

where T_p represents the number of true positives and F_p represents the number of false positives. We calculate recall as

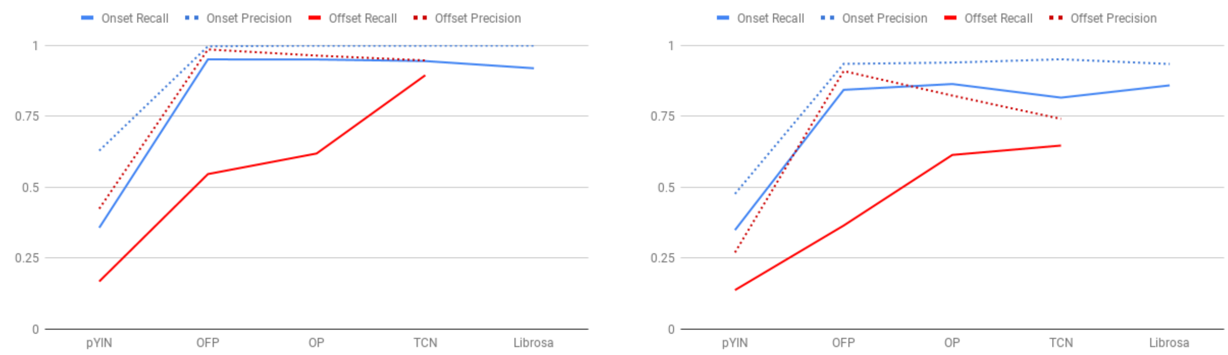
$$\frac{T_p}{T_p + F_n}$$

where T_p represents the number of true positives and F_n represents the number of false negatives. We conducted both a framewise and notewise evaluation of our systems.

6.1 Framewise Evaluation

We conducted a framewise evaluation for onset and offset detection, where we based our evaluations off the time frames that onsets and offsets occurred. We defined a T_p as a label with a detected onset or offset within 2 time steps of the label, corresponding to the blurred labels. We defined a F_p as a detected onset or offset that is not within 2 time steps of a labeled onset or offset. Finally, we defined a F_n as a label that our model did not detect correctly.

For our OFP and TCN systems, we took the local maxima with a distance of at least 5



(a) Piano Dataset

(b) Larger Piano Dataset

Figure 6.1: Framewise Evaluations

between each peak and a height of at least 1 from our model output as our detected onsets and offsets for our systems. For our OP system, we did the same thing for our detected onsets, but a detected offset was defined as the first time step after time steps where the highest pitch value from the output was greater than 1. We did not include the rounding or discretization of notes for generating sheet music, since we were comparing the onsets and offsets based on the time frames. For our evaluation of pYIN, we examined the mono note output, which gave a pitch and a note state for each time step. A note state was defined as either an attack, stable, or silence note state. We counted the detected attack states as detected onsets from this output. We counted a detected offset as the first silence note state after a stable note state, or in the case where an attack note state occurred right after a stable note state, we counted the last stable note state in a sequence as the detected offset. We used a sample rate of 22050, a frame size of 2048, and a hop length of 512 for our pYIN evaluation. We did not do any post-processing for librosa's onset detection method [9], and used its output directly.

Figure 6.1 exhibits a framewise evaluation of our systems (OP, OFP, TCN), the pYIN transcription system [8], and librosa's onset detection system [9] on both of our datasets. We included the mean recall and precision for onset and offset detection for each system.

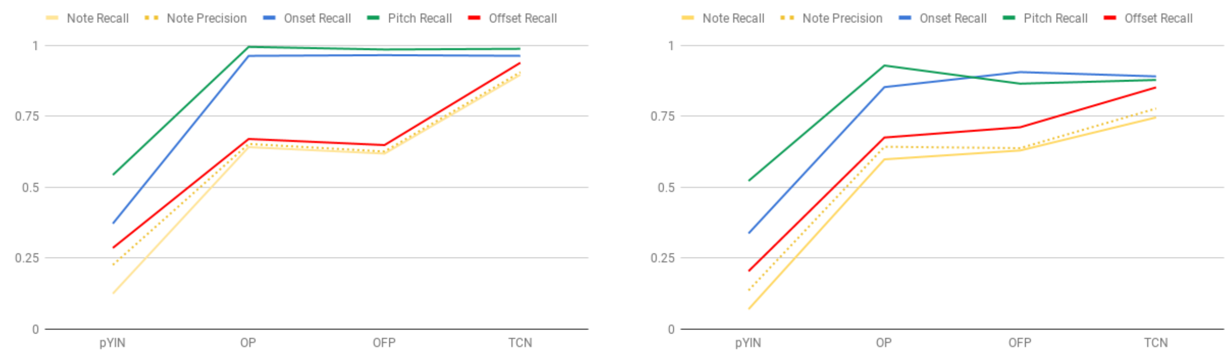
In Figure 6.1, we can see that the mean onset recall and precision for our systems were fairly similar, and they were fairly high. We saw a slightly lower onset recall and precision for our larger dataset compared to our smaller dataset, which we believe was due to overfitting. We were able to make a significant improvement from the pYIN transcription system for onset detection. Since the pYIN algorithm was primarily designed for pitch detection, we decided to include the onset detection method from librosa [9] to better evaluate our onset detection. We can see that our onset detection was comparable to librosa's onset detection. On our smaller dataset, all of our systems performed slightly better than librosa for the average onset recall. On our larger dataset, our OP system performed slightly better than librosa, and our OFP and TCN systems performed slightly worse than librosa, but they were within 0.05. Overall, we were able to achieve great results for our onset detection models.

For offset detection, we found that our OP, OFP, and TCN systems had varying results. One interesting result was that our OFP system had the worst offset recall, but the best offset precision, but we believe this was because our OFP system was not detecting very many offsets. Our TCN system had the best offset recall, but the worst offset precision. The TCN's offset recall was as expected since we thought that including more temporal dependencies with a model based on the TCN architecture [2] would improve upon offset detection. However, its offset precision was a bit unexpected, and we guess that it is worse because it may be more lenient as to whether a note has finished compared to the OP system, which is detecting an offset as whether a certain pitch stops and would often detect an offset a lot earlier than the TCN system would. Compared to the pYIN system, we found that all of our systems did better than the pYIN system. Unfortunately, librosa does not have an offset detection method, so we were not able to compare that as well.

6.2 Notewise Evaluation

We conducted a notewise evaluation of our systems, where we based our evaluation on the transcription that we created as the final output of our systems. We included note recall, note precision, onset recall, pitch recall, and offset recall for this evaluation as seen in Figure 6.2.

We defined a T_p for note recall and precision as a detected note where the onset, pitch, and duration all matched a labeled note exactly. We defined a F_p for note precision as a detected note



(a) Piano Dataset

(b) Larger Piano Dataset

Figure 6.2: Notewise Evaluations

that did not match any labeled notes. We defined a F_n for note recall as a labeled note that was not detected properly. For onset recall, we defined an onset as a T_p when the labeled onset and the detected onset occurred at the same time, so the measure and the beat of the detected note matched the measure and beat of the labeled note. We defined an onset as a F_n when we did not detect a note at the same measure and beat as a labeled note. We made our pitch recall dependent onset recall, and only calculated pitch recall for detected notes where the onset was a T_p. Thus, for pitch recall we defined the pitch of a note as a T_p when the MIDI pitch of a detected note with a T_p onset matched the corresponding MIDI pitch of the labeled note. We defined a F_n as a labeled note where we found the correct onset, but an incorrect pitch. We made our offset recall dependent on whether we had a T_p onset for a note as well. We defined an offset as a T_p when the duration value of a detected note with a T_p onset matched the corresponding labeled note. We defined a F_n as a labeled note where we found the correct onset, but an incorrect duration value, causing an incorrect offset.

The detected notes from our systems that we used in this evaluation were the notes that appeared on the sheet music that we had as our final output. This means that for our final output, we took the output of our models, included post-processing for onset, pitch, and offset detection, and then rounded and discretized the notes to generate sheet music. The post-processing for onset and offset detection for onset and offset detection followed the post-processing that we discussed in the framewise evaluation section. The post-processing for pitch detection included taking our model output for the time slice that we had, finding the max value at each time step and seeing if it was greater than 1, and counting the row of that max value as the detected pitch number. We then took the pitch that appeared the most times in the time slice as the detected pitch. We created notes from these detected onsets, offsets, and pitches, by essentially creating a note at each detected onset and matching a detected offset to it and then using the detected pitch. For notes where we did not detect an offset, we counted the next onset as the note's offset. For notes where we did not detect a pitch, we set the pitch number to 0. After getting these notes, we rounded them to get our final notes that we use in this evaluation.

For our evaluation of pYIN, we also included the same post-processing for onset and offset detection that we discussed in the framewise evaluation. For pitch detection, we took the most common pitch that appeared in the stable state during our mono note output. After creating notes from the detected onsets, offsets, and pitches, we did the same rounding and discretization that we did for our systems to get our final notes to evaluate.

In Figure 6.2, we can see that the notewise recall and precision increased between our OP, OFP, and TCN systems. This was as expected because the notewise recall and precision were very strict in that a T_p only occurred when a detected note perfectly matched a labeled note. The offset detection was the weak link in all of our systems, and since the TCN system had the best offset detection, it was able to get the best recall.

One interesting result in our evaluation was that we found different pitch recalls for our three systems despite using the same model, and the OP system actually saw the best pitch recall. The input of our pitch model for the OP system was the time slice between two detected onsets, whereas the input of our pitch model for the OFP and TCN systems was the time slice between a detected onset and a detected offset. We belive the OP system performed the best on pitch recall because pitch detection was based on offset detection for the OFP and TCN systems, and if we detected an offset very early after a detected onset, we would be feeding in a very short time slice for the pitch model to detect a pitch from.

From Figure 6.2, we can see that the pYIN system performed very poorly compared to our systems. The pYIN system actually had a pitch recall of 0 for both of our datasets when we conducted our evaluation due to octave errors so the values that we show in Figure 6.2 for the pYIN system actually allow for octave errors and count a pitch as correct if it is the same pitch class, but a different octave. We were surprised that the pYIN system performed so poorly on pitch detection since that's what it was designed for, but we surmise that it performed badly because it was designed for pitch detection for singing instead of piano, and it may not be able to generalize well. We also surmise that the sample rate, frame size, and hop length that we used may have affected its poor behavior as well.

Overall, we found that we achieved good results for our systems, indicating that CNNs can have useful applications in monophonic transcription as well.

Chapter 7

Future Directions

In this thesis, we improve upon monophonic automatic music transcription. We attempted to test our systems on other instruments, but found that it did poorly because our onset detection models did not perform well on soft onsets. Thus, a future detection to take would be to explore monophonic transcription with other instruments with neural networks. Another direction to take is to attempt to add some noise to the synthesized recordings from MuseScore and test the systems on live recordings.

In this thesis we were able to create a database with public scores from Musescore [10]. One of the challenges in automatic music transcription is having larger databases, and we believe that utilizing public scores can make creating larger databases of synthesized music easier. Because we were trying to get monophonic piano recordings, we were not able to get as many songs in our database since Musescore does not let you mute the treble and bass clef separately for a piano instrument. Thus, we were only able to use scores that only had a treble or bass clef for a piano instrument part. However, using Musescore to create a database for other instruments in monophonic transcription or polyphonic transcription could be very helpful, and we hope that another direction people take will be to utilize websites like Musescore with public scores more to create larger databases.

Chapter 8

References

[1] J.B. Allen and L.R. Rabiner. "A unified approach to short-time Fourier analysis and synthesis". In: (1977). ISSN: 1558-2256. DOI: 10.1109/PROC.1977.10770.

[2] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". In: *arXiv e-prints*, arXiv:1803.01271 (2018).

[3] OB MOOGWAS EXCITED. "Trevor Pinch and Frank Trocco". In: *The Sound Studies Reader* (2012), p. 254.

[4] Curtis Hawthorne et al. "Onsets and Frames: Dual-Objective Piano Transcription". In: *arXiv e-prints*, arXiv:1710.11153 (2017).

[5] Rainer Kelz et al. "On the Potential of Simple Framewise Approaches to Piano Transcription". In: *arXiv e-prints*, arXiv:1612.05153 (2016).

[6] Jong Wook Kim et al. "CREPE: A Convolutional Representation for Pitch Estimation". In: *arXiv e-prints*, arXiv:1802.06182 (Feb. 2018).

[7] Luoqi Li, Isabella Ni, and Liang Yang. "Music Transcription Using Deep Learning". In: (2017).

[8] Matthias Mauch and Simon Dixon. “PYIN: A fundamental frequency estimator using probabilistic threshold distributions”. In: (2014). ISSN: 2379-190X. DOI: 10.1109/ICASSP.2014.6853678.

[9] Brian McFee et al. *librosa/librosa: 0.6.3*. Feb. 2019. DOI: 10.5281/zenodo.2564164. URL: https://doi.org/10.5281/zenodo.2564164.

[10] *Musescore*. https://musescore.org/en.

[11] Jim Paterson. *A Short History of Musical Notation*. URL: www.mfiles.co.uk/music-notation-history.htm.

[12] Mark A. Roman, Antonio Pertusa, and Jorge Calvo-Zaragoza. “An End-to-end Framework for Audio-to-Score Music Transcription on Monophonic Excerpts”. In: (2018).

[13] Jan Schluter and Sebastian Block. “Improved musical onset detection with Convolutional Neural Networks”. In: (2014). ISSN: 2379-190X. DOI: 10.1109/ICASSP.2014.6854953.

[14] Christian Schörkhuber and Anssi Klapuri. “Constant-Q transform toolbox for music processing”. In: *Proc. 7th Sound and Music Computing Conf.* (Jan. 2010).

[15] Jonathan Sleep. “Automatic Music Transcription with Convolutional Neural Networks using Intuitive Filter Shapes”. In: (2017).

[16] Karen Ullrich and Eeelco van der Wel. “Music Transcription with Convolutional Sequence-to-Sequence Models”. In: (2017).

[17] Aaron van den Oord et al. “WaveNet: A Generative Model for Raw Audio”. In: *arXiv e-prints*, arXiv:1609.03499 (Sept. 2016). eprint: 1609.03499.

Mina Lee was born in Cleveland, Ohio on September 6, 1997. She has lived in Seoul, South Korea, Singapore, and Austin, Texas. She enrolled in the Turing Scholars Honors and Plan II Honors programs at the University of Texas at Austin in 2015. In college, she joined Beauties and the Beat, the premiere all-female acappella group at UT Austin. As one of the arrangers for the group, she was inspired to write this thesis on automatic music transcription. Ms. Lee will start working as a Software Engineer for Google in fall of 2019.