Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Dyvik, Klaus
Waaler, Fredrik

Utilizing Audio Plugins for Automatic Music Transcription

How Production Tools Can Help Alleviate Dataset Deficiencies

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2022

NTNU
Norwegian University of Science and Technology

Dyvik, Klaus
Waaler, Fredrik

Utilizing Audio Plugins for Automatic Music Transcription

How Production Tools Can Help Alleviate Dataset Deficiencies

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU

Kunnskap for en bedre verden

Klaus Dyvik, Fredrik Waaler

Utilizing Audio Plugins for Automatic Music Transcription

How Production Tools Can Help Alleviate Dataset Deficiencies

Master's Thesis in Computer Science, June 2022
Supervisor: Björn Gambäck

Data and Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Abstract

Pertaining to the field of Automatic Music Transcription (AMT), this thesis investigates the effect that digital audio plugins has on transcription performance, when used to augment existing musical data. To best examine this, the thesis presents a complete system, capable of rendering both audio and Musical Instrument Digital Interface (MIDI) data through digital effects and digital instruments. Additionally, the system comes with a variety of optional settings, providing intricate control of the rendering process. The presented system is used to produce new datasets, based on popular datasets used in AMT. The produced data is then used together with a state-of-the-art music transcription model to get numerical quantification of how the rendering affects the prediction performance of the model. This is done over a variety of experiments to get a broad understanding of the weak and strong points of the rendering approach.

Generally, AMT refers to the process of using a computer to recreate the notes and performance style details from a piece of music. Its many possible use cases spans learning applications (e.g. presenting an instrument learner with the notation to any given song) and artistic work (e.g. sampling and remixes), to name a few. However, lack of large and well annotated datasets is among the hinders that so far have kept AMT models from reaching satisfactory performance.

Previously there have been several attempts to remedy the deficiency in musical data, using different kinds of augmentation approaches. These have spanned simple pitch-shifting to advanced Artificial Intelligence (AI) based vocoders. While some of these attempts have been successful, the augmentation process is usually quite uniform across the data, relying on one or a few techniques for all augmentations. This puts a limitation on how diverse the augmented data can become, and consequently on which weak-points can be addressed in the existing musical data. This thesis hypothesizes that by using audio plugins for musical augmentation, the rendering possibilities at hand are near infinite. In turn, this should give vast potential for remedying many deficiencies in current musical datasets.

The experiments conducted in this thesis show that validation performance has increased on several individual files, and even the overall precision score over entire datasets, when trained with the augmented data. On a general basis, however, the experiments show that plugin-rendered data tend to perform worse than their original counterparts. Together with these findings and a thorough review of the limitations and current capabilities of the system, this thesis serves as a starting point for using audio plugins to improve state-of-the-art AMT. Additionally, in light of the experimental results, a range of possibilities for future improvements are discussed.

Sammendrag

Denne masteroppgaven undersøker hvilken effekt digitale verktøy for musikkproduksjon kan ha på ytelsen til løsninger innenfor Automatisk Transkripsjon av Musikk (ATM). For å gi en detaljert og grundig undersøkelse av dette, presenterer denne rapporten et system som kan prosessere både lyd og MIDI med digitale effekter og instrumenter. Dette systemet benyttes til å generere nye datasett basert på flere datasett populært brukt i ATM-løsninger. De produserte datasettene blir videre brukt som treningsdata i et moderne ATM-system for å se til hvilken grad teknikkene brukt fungerer gjennom ulike eksperimenter.

ATM handler om å automatisk innhente informasjon om hvilke noter som blir spilt av i et musikalsk stykke, samt hvilken stil stykket er i. Dette er et fagfelt med mange mulige bruksområder, f.eks. i læring og produsering av musikk. Selv om det i de siste årene har blitt rapportert om løsninger i feltet med brukbar ytelse, er det fortsatt et stykke igjen å gå før disse løsningene oppnår perfekt transkripsjonsevne.

Den største bremsen for fremgang innenfor ATM antas å være mangel på store og godt annoterte datasett. For å kompensere for dette har flere løsninger tidligere blitt foreslått som forsøker å augmentere eksisterende musikk-data for å øke ytelsen til automatiske transkriberingsmodeller. Disse løsningene har brukt varierende fremgangsmåter som enkle endringer i toner, til mer avanserte teknikker som for eksempel endring av lyd ved hjelp av kunstig intelligens.

Det er dog til nå ingen som har undersøkt hvordan digitale verktøy for musikkproduksjon kan brukes til å automatisk endre musikalsk data, samt hva slags innvirkning dette har på automatisk transkribering. Dette er verktøy som er spesialutviklet for å endre musikalske signaler på nær uendelige måter. Gitt riktig bruk burde de derfor kunne benyttes til å generere data som kan supplementere eksisterende musikalske datasett for økt ytelse og resultater.

Eksperimentene gjort i sammenheng med denne rapporten viser at slike metoder kan ha positiv innvirkning på transkripsjonsytelse. Transkripsjonsevne økes for enkelte filer i valideringsdataen til flere datasett. I tillegg viser noen eksperimenter økt presisjon over hele datasett. Slike metoder har i sin helhet dog hatt negativ innvirkning på ytelse for ATM-systemet. I rapporten er det foretatt en grundig evaluering av hvorfor disse resultatene har forekommet. Videre diskuteres hvilke faktorer som må ligge til grunn for at digitale verktøy for musikkproduksjon kan øke ytelse for ATM-systemer i senere forsøk.

Preface

This thesis is written as a collaborative Masters Thesis project in Computer Science on the Norwegian University of Science and Technology (NTNU), during the spring semester of 2022. It is based on the preliminary thesis *Automatic Music Transcription with Deep Learning*, which was also written by the two of us.

The thesis has been written under supervision by a professor at NTNU, Björn Gamback. We are grateful for the feedback, discussion and reflections you have given during the work, and want to extend a special thank you.

Klaus Dyvik, Fredrik Waaler

June 9, 2022

Contents

List of Figures

List of Tables

1. Introduction

Automatic Music Transcription (AMT) is the task of transcribing music into a human-readable format, such as Musical Instrument Digital Interface (MIDI). It can be seen as reverse-engineering a musician's performance back to the sheet music he/she used to play it, in addition to the style the musician played it in. This technology can be impactful in music education, creation, and production. A music creator could use transcribed music to thoroughly study the styles of other musicians. A producer could use other transcribed scores for inspiration, learning, or sampling. AMT would also aid in other areas such as in the karaoke industry, where transcription can lead to an unlimited library of karaoke audio.

Current State-of-the-art (SOTA) solutions within AMT are performing the best in transcription of piano music, as will be uncovered in chapter 4. Transcription of other instruments is yet to achieve equal results to piano. One reason is that the availability of annotated datasets are larger for piano than for other instruments. This thesis investigates how audio plugins can be utilized to overcome data shortage issues within AMT. Audio plugins are digital tools for music production, for instance used to manipulate and generate audio with an near endless number of settings. They have become better over the years, and can even replicate real instruments digitally. Application of audio plugins to existing AMT datasets can therefore help increase their diversity, increasing performance for transcription of other instruments.

This introductory chapter will first describe the background and motivation for this thesis. Secondly, the goals and research questions the thesis will investigate are presented. Following is the research methodology used to investigate these research questions. Finally, a list of the contributions of this thesis is presented, before the general thesis structure is described.

1.1. Background and Motivation

Before this thesis, a preliminary report was made by the same authors as this thesis. The preliminary report had intention to investigate (1) the State-of-the-art (SOTA) of Automatic Music Transcription (AMT), (2) how synthesis techniques are utilized in current SOTA solutions to improve datasets used in AMT, and (3) how digital producer technology can be used for the synthesis of such datasets. The results from the preliminary report form the basis of this thesis, which is presented in this section.

While SOTA of AMT has shown promising results, this mainly applies to piano music. Results for many other instruments are still not close to the same performance. The preliminary report argues that this is mostly due to a lack in availability in datasets, which is fairly prevalent for piano. The reason why the availability for piano music is larger is due to its versatility in generating the datasets. First, recording of keystrokes is possible through a wide range of electronic piano devices. Doing so, the keystrokes are automatically annotated. Secondly, the piano has a distinct set of 88 keys (can vary), and annotation of a keystrokes must belong to one of the 88 keys. Therefore, piano recordings are simpler to transcribe by human professionals than many other instruments. This is compared to more dynamic instruments, such as the guitar, where strings are commonly tuned or bent to other pitches. The current SOTA for transcription of piano music is proposed by Kong et al. (2021). A recent attempt by Gardner et al. (2021) obtains a close to equal score for piano as Kong et al. (2021). In addition to transcribing piano, however, Gardner et al. (2021) also achieves new SOTA scores for other instruments using the same system.

The preliminary report shows that a wide selection of synthesis techniques recently have been used to successfully enlarge and improve musical datasets in a way that yields increased performance for AMT models, shown for instance in Southall et al. (2018) and McFee et al. (2015). These solutions are based on Machine Learning (ML) and/or created for the sole purpose of the AMT synthesis tasks. The preliminary report argues that scripting of modern music production technology may potentially serve as a more scalable and powerful tool for musical data synthesis. Modern music production technology referred to as audio plugins (as described in section 2.8) can be used to transform and modify audio in more or less infinite ways. Audio plugins are separated between effectors and generators. Effectors alter existing recorded music by applying what is known as effects. Generators generate music through instructions, either by generating new sounds or assembling sounds from libraries. These instructions are commonly stored in MIDI files (see section 2.4.4). Using both effectors and generators, it is possible to alter existing datasets in ways that may potentially help overcome the biggest hindrance in current AMT solutions, namely the lack of size and diversity in the musical datasets (Benetos et al., 2013).

By carefully selecting effects to apply to existing datasets through effectors, one can greatly enlarge not only the number of different sounds, but also the representativeness of the characteristics of the audio. Furthermore, the application of effects requires no further bookkeeping of metadata (e.g. instrument type) other than what is done in the original datasets. That being said, the application of effects does have some potential shortcomings. Effects are working on top of existing audio signals, setting a limit to how much these signals can be changed before they start sounding corrupted and unmusical. In addition, there is a possibility that the application of certain effects will make one instrument sound like another. For instance, by adding a lot of chorus to a piano, it will not be long until it starts sounding more like a pad, or a synth. This potentially breaks the system's ability to transcribe with respect to the source of each instrument.

The preliminary report also argues that generators should be used to supplement effectors in the task of expanding AMT datasets. Since generators work on MIDI rather than audio, they can be used to re-render existing MIDI files in the style of a new instrument. Then the larger availability of piano datasets can be utilized to expand datasets for other instruments. While requiring a little bit of extra bookkeeping (new instrument type), this approach still allows for a MIDI file to be rendered in an infinite amount of ways, ensuring that the data stays musical while still providing training data with a vast selection of traits.

1.2. Goals and Research Questions

To get an impression on whether automation of audio plugins can be used as a tool to enlarge datasets used in AMT or not, this thesis formulates the following goal:

Thesis Goal
Examine how musical data generated through audio plugins affect the performance of state-of-the-art automatic music transcription solutions.

This thesis goal is further split into three separate research questions that will be answered separately in an attempt to systematize the research and results. Each question is concerned with an approach for utilizing audio plugins to generate musical datasets.

Research Question 1 (RQ1) *Can effect-plugins be used to augment existing musical datasets in a way that increases performance for AMT models?*
RQ1 will be answered by testing the application of varying effects to the musical datasets and examining how this affects the prediction performance of an AMT model. Different setups of effects and rendered audio will be tested to thoroughly gauge the validity of applying effects, as well as to understand the merits and limitations of the approach.

Research Question 2 (RQ2) *Can generator-plugins be used to augment existing musical datasets in a way that increases performance for AMT models?*
RQ2 will examine re-rendering the MIDI of existing musical datasets, using a wide selection of different instruments from generators. The rendered audio will then be used as data for an AMT model to better understand the effect generator-plugins can have on expanding musical datasets.

Research Question 3 (RQ3) *Can generator-plugins be used to create an entirely new musical dataset? How can this complement existing AMT datasets?*
RQ3 will investigate how digital producer tools can be used to automatically create musical datasets, containing audio, MIDI, and metadata.

It is worth noting that both RQ1 and RQ2 are concerned with the application of audio plugins to existing musical datasets, which, as seen in for instance McFee et al. (2015) may help increase AMT performance. The impact of augmenting such datasets is of course limited by the quality of the original datasets. Thus, RQ3 supplements the other

two research questions by investigating how digital tools for music producers can be used to create entirely new datasets.

1.3. Research Method

The research questions introduced in section 1.2 together form the thesis goal, to examine how musical data generated through audio plugins affect the performance of SOTA AMT solutions. This thesis includes the description of an architecture (chapter 5), developed to answer the questions in the best way possible, first proposed in the preliminary report. This architecture must include a module for generating and modulating musical data through audio plugins, so it can create the new datasets as proposed in the research questions. These datasets are evaluated on a SOTA AMT model, so this is also integrated in the complete architecture. The architecture must be iteratively designed before performance is assessed through experiments. A design/experiment methodology is therefore suitable for researching the goals and the research questions.

The implementation of a design/experiment methodology used for this thesis begins with a planning phase where ideas for the final system are brainstormed and limited. This phase includes planning of what the system must contain in the scope of the thesis, and what potentially could be included if the results are positive. After the planning phase is completed, the next step is to work in two-week sprints, a term from the popular agile methodology SCRUM (Schwaber, 1997). A sprint is a set of development activities conducted over a pre-defined period. At the beginning of each sprint, a meeting is held for prioritizing tasks that are expected to be done over the following sprint. Having decided what is to be done within the next period, the design and experimental phases are continuously assessed and monitored throughout the time of the project.

1.4. Contributions

The overarching work of the thesis, spanning the architecture presented in chapter 5, the conducted experiments outlined in chapter 6 and the thesis itself include several individual contributions that can be of use to the general field of AMT. Briefly, these include:

1. A plug-and-play, modular system, capable of rendering both audio and MIDI through an extensive selection of digital audio plugins. Extending the augmentation possibilities for AMT to a near infinite amount.
2. A thorough assessment of how the use of audio plugins, both effectors and generators, affect the data quality for AMT datasets. Combined with an in-depth review of the current merits and limitations of the approach.
3. A reproduction of a SOTA AMT architecture, used for training and validation on AMT examples. Simultaneously, earlier work and results produced with the system

are validated.

4. A useful summary of the field of AMT and related background theory, necessary to truly understand the presented concepts. These are given in chapter 2 and 4, respectively.

5. Recommendations for further research, including new directions for use of audio plugins and generation of data, as well as suggestions for future improvements that can be added to the existing system.

1.5. Thesis Structure

In the remainder of the thesis, an extensive overview of the work towards the contributions will be presented. First, the necessary background theory will be presented in chapter 2. Secondly, the datasets used throughout the thesis and for the related experiments are presented in chapter 3. Next, the related work and research in the field of AMT are described in chapter 4. The architecture used to carry out experiments is then presented in chapter 5, before chapter 6 details the exact experimental plans, setups, and their results. Chapter 7 evaluates and discusses these results. Finally, a conclusion and suggestions for further work are given in chapter 8.

2. Background Theory

This chapter presents the background theory behind the concepts introduced in this thesis. First, the underlying music theory concepts will be introduced. Secondly, a brief overview of what Automatic Music Transcription (AMT) is and its various variations in the literature are presented. Afterwards, this chapter introduces time-frequency analysis techniques for extraction of frequency information from audio. The next section presents how audio can be represented, both digitally and for humans. A background to Machine Learning (ML) methods is provided, to form a basis of the techniques used in State-of-the-art (SOTA) in AMT. Then, the evaluation metrics used in general AMT and for this thesis are further highlighted. Finally, audio plugins are described - digital tools that can be used to augment data used in AMT - in further detail.

2.1. Music Theory

This section provides a brief introduction to music theory concepts. This theory is important, as it is essential for the following background theory of AMT. The section first describes what a tone and a pitch are. Secondly, what defines a note is described, including what the onset, offset, and Attack, Decay, Sustain and Release (ADSR) envelope is.

A pure tone is a sinusoidal playing at a certain frequency. For instance, a pure A_4 is a sinusoidal at a frequency of 440Hz. Tones from instruments generally have a Fundamental Frequency (F0), in addition to integral multiples of the F0 called *overtones*. Sounds from musical instruments are therefore time-evolving superpositions of several pure tones (Alm and Walker, 2002). It is however the F0 of a note that makes up the pitch. A sound has a certain pitch if it can be reliably matched by adjusting the frequency of a sine wave of arbitrary amplitude (Hartmann, 1996). Pitch is essentially what humans use to order notes from low to high.

When a pitch is increased from e.g. an A_4 to an A_5, the F0 is doubled to 880Hz. This is known as increasing the pitch by one octave. When lowering the pitch by one octave, the F0 is halved to 220Hz. Sound is therefore logarithmic. Because of this trait, it is normal to divide octaves into equal steps, forming what is known as the equal temperament scale. In western music, octaves are 12 equal temperament, meaning the scale is divided into 12 parts, known as semitones. Moreover, one semitone is further divided into 100 cents.

A note is the presence of a pitch. The time at which any pitch starts to play is the *onset*

of a note. For a piano, this would be when a tangent is pressed. Similarly, the *Offset* describes when the note stops. However, when a note stops is not the same time as when the tangent is released. When a tangent is released, the note takes a little while to stop completely, and even longer if the sustain pedal is pressed. Because the exact offset is hard to find, this can be defined arbitrarily. It could for instance be defined by a threshold relative to the maximum level of the note (Benetos et al., 2019).

Another way of perceiving a note is through Attack, Decay, Sustain and Release (ADSR) envelopes. ADSR envelopes describe how the amplitude of the note is changing over the four phases *attack*, *decay*, *sustain* and *release*. A model of these four phases is shown in figure 2.1. When a key is pressed, the *attack* is the time it takes for the note to reach its peak amplitude. After reaching the peak, the note will reach a *sustain* level through a *decay* phase. The sustain level lasts until the key is released. The *release* is the time it takes from the key is released to the sound being completely silent. Figure 2.1 shows linear transitions between the amplitude levels, but non-linear transitions are also possible. The start of the attack phase and the end of the release phase are the onsets and offsets of the note, respectively, also denoted in the figure.

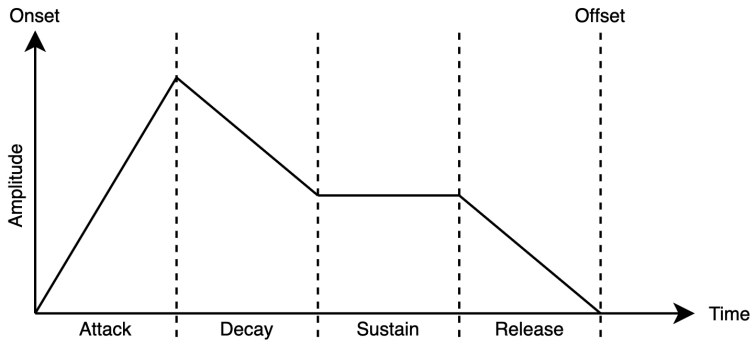

Figure 2.1.: ADSR envelope

2.2. Automatic Music Transcription

Automatic Music Transcription (AMT) is the task of transforming an acoustic signal into a symbolic representation (Klapuri, 2004). The symbolic representations range from staff notation to Musical Instrument Digital Interface (MIDI) notation, both described in section 2.4. This section presents what AMT is and how it varies within the literature. These variations are separated into three various groups: sub-tasks, levels of operation, and types of transcription. Sub-tasks are tasks that together form a complete AMT system. Levels of operation describe how the system operates in terms of prediction method and output. Types define how complex the input is in transcription systems.

A complete AMT system may get very complex. Therefore, AMT is divided into several sub-tasks in the literature. Benetos et al. (2013) specify that the core problems or

sub-tasks of AMT are multi-pitch estimation (determining what pitches are played) and note tracking (determining when a note starts and stops). This is what is required to transform an acoustic signal into a symbolic representation. The other sub-tasks involve finding information that if integrated could improve transcription performance: estimation of features relating to rhythm, melody, harmony, and instrument recognition. In addition, some methods include other niche tasks within AMT, such as the detection of when a piano pedal is pressed (Kong et al., 2021). Benetos et al. (2019) further separate AMT into four levels of operation:

- Frame-level transcription: Estimation of the pitches that are simultaneously active in a frame. A frame is a discretized piece of the audio set to a fixed length (e.g. 10ms). The pitches in each frame are analyzed separately. Contextual information between the frames could be used to filter out anomalies in the output. For instance, imagine the system predicts an E3, F3 and G3 in order, as shown in the three first notes of figure 2.9. If an A4 is predicted during these three notes, it does not follow any musical patterns, and would sound unmusical. This is contextual information that may be removed. Frame-level transcription is shown in figure 2.2a).

- Note-level transcription: Also called note tracking. A note is characterized by three elements: pitch, onset time and offset time. Compared to frame-level transcription, note-level transcription connects pitch estimates over time. Note-level transcription is shown in figure 2.2b).

- Stream-level transcription: Targets grouping of estimated pitches or notes into streams. Each stream typically corresponds to one musical source (e.g. instrument). They can be characterized by the timbre of each source. Stream-level transcription is shown in figure 2.2c).

- Notation-level transcription: Transcribing music into human-readable score, such as staff notation, further described in section 2.4.3. This requires retrieval of more information, such as rhythm and harmony, because such scores are structured with it. For instance Gerou and Lusk (1996) contain 160 pages of rules to follow when making staff notation. Staff notation is shown in figure 2.6.

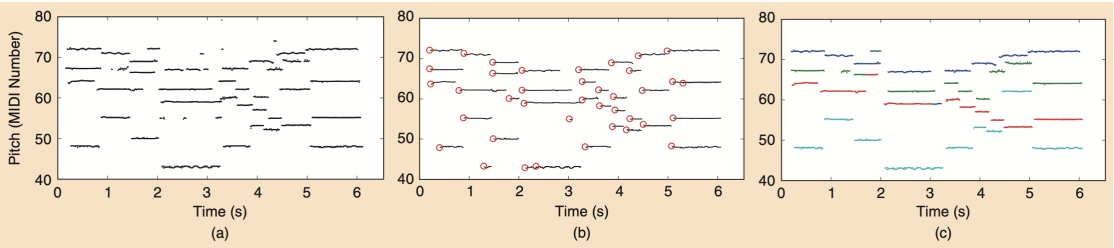

Figure 2.2.: Different levels of operation in AMT: a) frame-level, b) note-level, c) stream-level (edited version of figure from Benetos et al. (2019))

In addition to separating AMT into sub-tasks and levels of operation, there are various types of transcription. One may separate between single- and multi-instrument transcription. In single-instrument transcription, only one instrument is the target for the system, whilst in multi-instrument transcription, the system can transcribe more instruments. This is not to separate between the next type: transcription of *monophonic* and *polyphonic* music. Monophonic music is music with only one melody line, while polyphonic music has several.

2.3. Time-Frequency Analysis

Time-frequency analysis is about analyzing the content of the sounds produced by musical instruments. It involves for instance retrieval of information about the F0 and overtones of a note introduced in 2.1. As described, this information is what makes up the pitch. This section presents the Fourier Transform, a method to calculate the whole frequency spectra, from which the F0 and overtones can be retrieved. Following, the Fast Fourier Transform (FFT), a method used to calculate the Fourier Transform digitally, is described. Because these techniques reveal valuable information about the frequency spectra, their outputs are used as input in SOTA AMT methods (chapter 4). These outputs are known as spectrograms, further described in section 2.4.2.

2.3.1. Fourier Transform

The Fourier Transform converts a sound signal from the amplitude vs. time domain to the frequency domain. Given a sound signal $f(t)$ in an interval $[0, \Omega]$, the Fourier series of the signal is defined with equation 2.1. The equation is from Alm and Walker (2002).

$$c_0 + \sum_{n=1}^{\infty} \{c_n e^{i2\pi nt/\Omega} + c_{-n} e^{-i2\pi nt/\Omega}\} \tag{2.1}$$

Its corresponding complex Fourier coefficients in equation 2.2, also from Alm and Walker (2002).

$$c_n = \frac{1}{\Omega} + \int_0^{\Omega} f(t) c_n e^{-i2\pi nt/\Omega} dt, n = 0, \pm 1, \pm 2, ... \tag{2.2}$$

c_0 represents the constant background noise level, e.g. a constant air pressure level. $1/\Omega$ is the Fundamental Frequency (F0).

2.3.2. Fast Fourier Transform (FFT)

The method of digitally computing Fourier spectra is known as the FFT. The FFT provides an extremely efficient method for computing approximations of Fourier series

coefficients (Alm and Walker, 2002). These approximations are called Discrete Fourier Transforms (DFTs). The DFT of a sequence $\{f_k\}$ of N numbers is calculated with equation 2.3. The equation is from Alm and Walker (2002).

$$F[n] = \frac{1}{N} \sum_{k=0}^{N-1} * f_k * e^{-i2\pi nk/N} \tag{2.3}$$

How the FFT can be used as input to AMT methods is described in section 2.4.2.

2.4. Audio Representation Format

Audio may be represented in many ways, for instance analogously through an LP record, or in bytes in various digital sound formats. This section presents four ways audio can be represented, together with their traits. The waveform format is used for storing audio files digitally. A spectrogram is a two-dimensional representation of audio, commonly used as input representation for AMT systems (chapter 4). MIDI is a digital audio format for representing music in digital audio software. Lastly, the NoteSequence is a structure made for representing MIDI in AMT systems.

2.4.1. Waveform

This subsection presents the Waveform Audio File Format (WAVE) file format and its origin. Audio files in datasets used in AMT literature (presented in chapter 3) are generally stored in the WAVE format.

Pulse Code Modulation (PCM) is a scheme for converting an analog signal into digital form (Garg, 2007). Examples of such analog signals are video, voice, and music. The output of PCM is a series of binary numbers, called a PCM *waveform*. The scheme is fully reversible, making it possible to e.g. transmit sound through speakers. The scheme is also lossless, meaning there is no loss of quality from the original source.

One way of storing this waveform digitally is through the WAVE. This format was built jointly by IBM and Microsoft (1991), and is an extension of their generic Resource Interchange File Format (RIFF). The WAVE format implementation in Microsoft Windows is called the Microsoft Pulse Code Modulation (MPCM) format. The MPCM format contains a sample size field, used to calculate the sample rate of the audio. The sample rate specifies how many data points exist per second. The default sample rate on Microsoft Windows systems is 44.1kHz, using 5MB of space per minute of mono audio when storing 16 bits per sample. Figure 2.3 displays a WAVE file in its raw audio format.

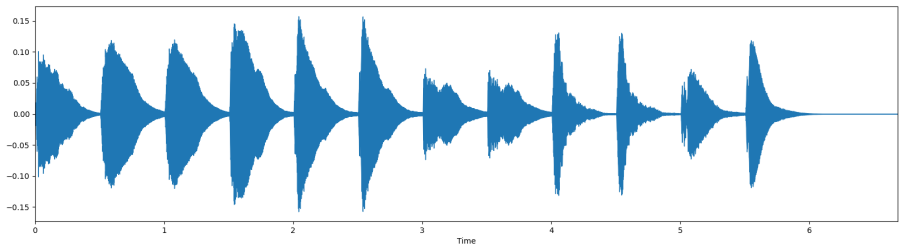

Figure 2.3.: Raw audio in WAVE format

2.4.2. Spectrograms

Spectrograms provide a time-frequency portrait of musical sounds (Alm and Walker, 2002). There are many variations of spectrograms. One of them is obtained by using the time-frequency analysis techniques presented in section 2.3, e.g. the FFT. Using the FFT on the whole sound signal captures the frequency magnitudes over the whole sound signal. To rather retrieve the frequencies at each point in time, the audio is split into a set of fixed-length windows. Then the FFT is calculated over each window. These windows are then joined together to form a spectrogram. An FFT spectrogram of the audio previously presented in figure 2.3, is shown in figure 2.4. Because of the traits discussed in section 2.3, spectrograms can be used as input for AMT ML systems.

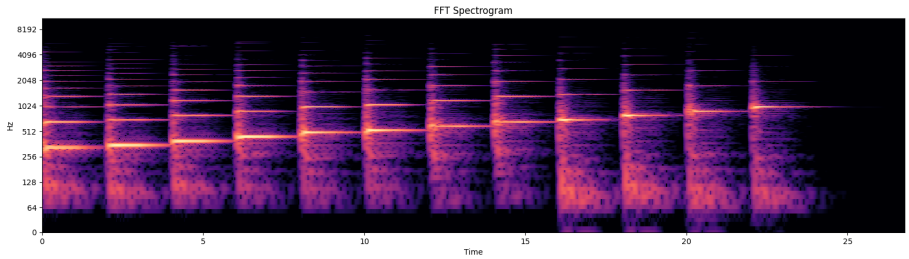

Figure 2.4.: FFT spectrogram of the audio in figure 2.3

As described in section 2.1, music is logarithmic. Spectrograms are therefore easier to interpret if portrayed on the logarithmic scale. If not, pitches in lower frequencies will be clustered together when depicted. Another conversion of the standard frequency space is the *mel scale* proposed by Stevens et al. (1937). The mel scale is a subjective scale for measuring the perceived pitch of humans to the frequency space. The mel scale is measured in mels, such that 1000 mels equals 1000Hz. Doubling the pitch in mels sounds subjectively twice as high. Stevens et al. do not provide the exact formula for conversion. The TensorFlow implementation, which is the implementation used in this thesis, use

the formula shown in equation 2.4 [1].

$$\text{mel}(f) = 2595 * \log_{10}(1 + \frac{f}{700}) \tag{2.4}$$

Figure 2.5 displays a spectrogram using the same audio file as used in figure 2.3 and 2.4

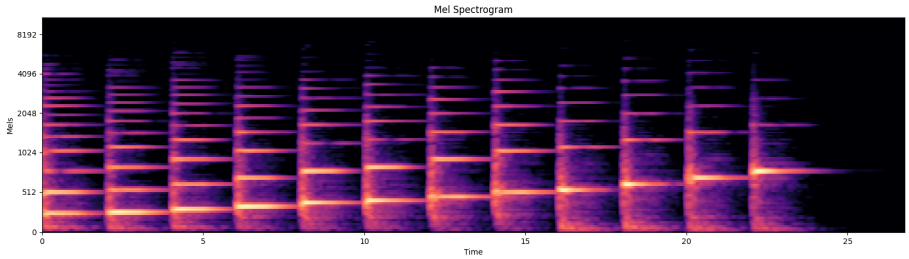

Figure 2.5.: Mel spectrogram of the audio in figure 2.3

The effect of using mel-scaled spectrograms in AMT systems were tested in Hawthorne et al. (2021), an AMT architecture further described in chapter 4. Mel-scaling had a positive impact on the results, and they suspect the mel scaling produces useful features that the model would otherwise have to use some of its own capacity to extract.

2.4.3. Staff

Staff is a notation language that makes music readable to the human eye. It provides instruction on how to play a musical piece with an instrument based on a preset of rules. This section is similar to the section on staff notation in the preliminary report (section 1.1). An overview of such rules can be found in the *Essential Dictionary of Music Notation* (Gerou and Lusk, 1996). This dictionary covers 160 pages of rules for staff notation. Staff can therefore get very detailed, but can also be simplified as in figure 2.6, where the basic notation structure is shown. This figure consists of five lines with four spaces between the lines. Each of the lines represents a note in the order of (EFGABCDEF) as denoted by 1 in the figure. Notes can go beyond this scale with the use of ledger lines, denoted 2 (with notes in order of GAB).

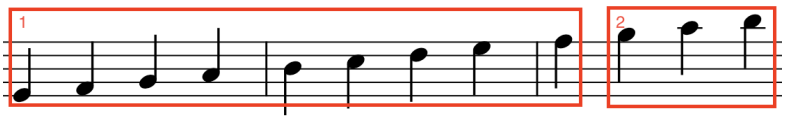

Figure 2.6.: Staff notation

[1] https://www.tensorflow.org/api_docs/python/tf/signal/linear_to_mel_weight_matrix

The first bar of *11 Bagatelles, Op. 119: 1. Allegretto*, written by Ludwig van Beethoven shown in figure 2.7 cover some of the rules listed in Gerou and Lusk (1996). Firstly, the $\frac{3}{4}$ fraction describes the pace of the musical piece. Furthermore ♭ and # are accidentals suggesting to lower and raise a note a half step respectively. The little dot over or under most of the notes denotes a staccato, meaning the note should be played with a quarter length of a regular note.

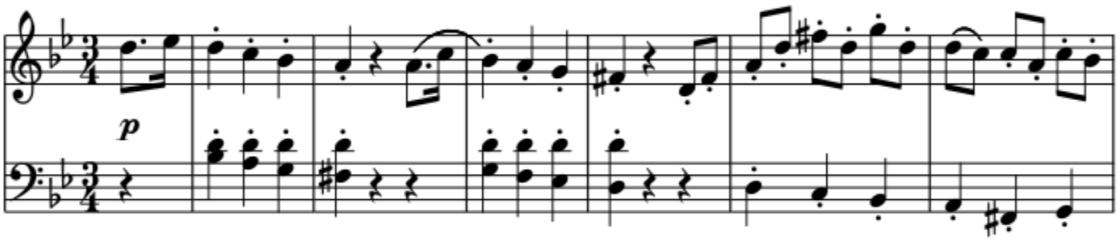

Figure 2.7.: First bar of 11 Bagatelles, Op. 119: 1. Allegretto, written by Ludwig van Beethoven

2.4.4. MIDI

Musical Instrument Digital Interface (MIDI) is a system that allows electronic instruments and computers to send instructions to each other (MIDI Manufacturers Association, 2009). These instructions are sent through MIDI messages, which can be used to produce audio. For instance, a digital keyboard tells the computer to play a C note, which the computer process to produce output. Typical use on a computer is to process the audio through a Digital Audio Workstation (DAW), further explained in section 2.8. In general, there are four basic MIDI messages: Note-on and note-off messages, to tell the computer to play/stop a note; velocity, which signals the force used to press a note; pitch bend, to instruct to raise or lower the played note's pitch; plus channel and polyphonic after-touch, symbolizing how much force is used to hold down a note.

Other types of MIDI messages include program changes to select sounds on another MIDI device, or control change messages which cover a wide range of specific behaviors related to the performance of controls like pedals, wheels, and other controllable devices.

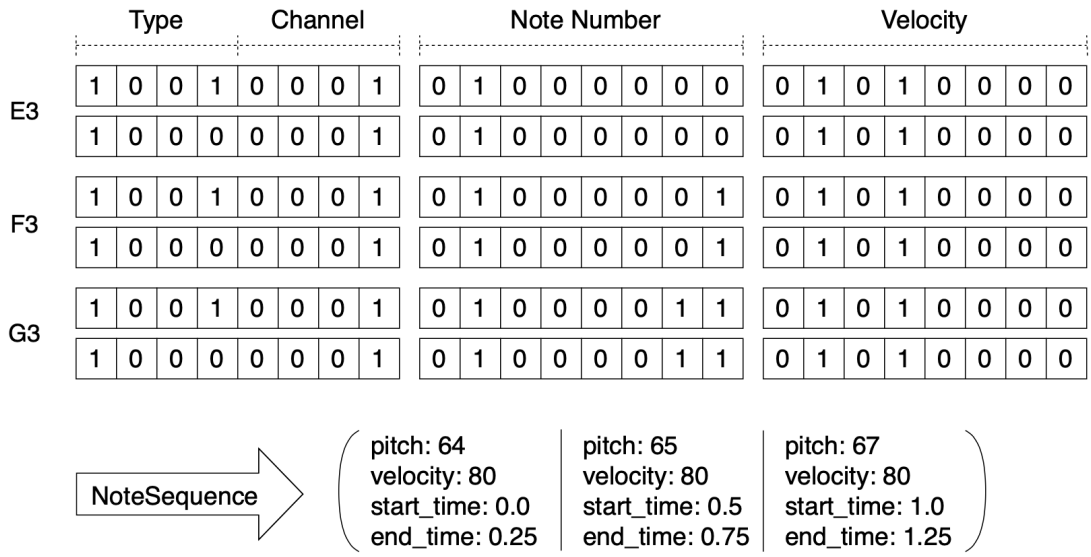

Figure 2.8.: MIDI messages and NoteSequence representation of the first three notes in the audio in figure 2.3

Typical MIDI messages are constructed as in figure 2.8. Here, the notes of E_3, F_3 and G_3 are instructed to play. In each of the messages, the first byte is a status byte representing the type of message, and the channel. The next two bytes are data bytes, containing values corresponding to the expected format of the status. For the first message, the status byte contains the type 1001, indicating a *note-on* event in channel 1. For a note-on event, the next byte represents the note number from 0 to 127, and the last byte represents the velocity of the note. The first message in the figure shows a E_3 (note number 64) with a velocity of 80. The next message contains the type 1000 for the same note, indicating a *note-off* event.

MIDI messages are not suitable for human interpretation in byte format. As a result, they are often visualized as a piano roll, as depicted in figure 2.9. The figure shows several MIDI messages together in what is called a MIDI file. This is the MIDI file for the spectrograms depicted earlier in figure 2.8.

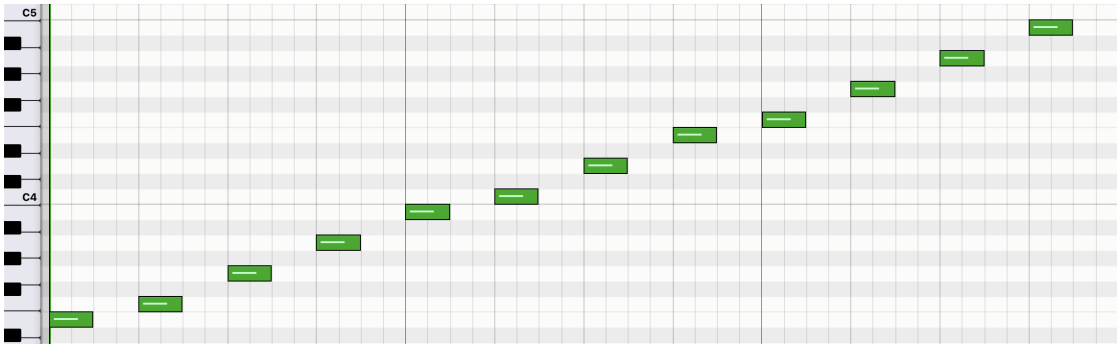

Figure 2.9.: Piano roll representation of MIDI

2.4.5. NoteSequence

Even though MIDI provides a detailed representation of how a musical performance is played, it may need further parsing to be interpreted by computers. One example is when using MIDI in an AMT system, the MIDI must first be converted to another representation. NoteSequence is a MIDI-like representation format for serializing MIDI for this purpose in Python. It was made by Google Magenta in the *note-seq* package[2]. Figure 2.8 shows the conversion of MIDI in bytes format to the NoteSequence representation. In addition to importing and exporting a NoteSequence into MIDI and other formats, the library has support for creating and manipulating them as well. Representing MIDI as a NoteSequence is used by Hawthorne et al. (2021) and Gardner et al. (2021) to represent note events in their deep learning models (see chapter 4).

2.5. Machine Learning

To properly comprehend the SOTA AMT techniques, it is helpful with a rough understanding of the methods and background theory this technology is built on. More or less all SOTA solutions in AMT are based on some form of Machine Learning (ML) (Benetos et al., 2013). This section will present an overview of the ML field, along with some of the most prominent architectures used for audio processing.

Without relying on any formal definition, ML at its most basic can be said to be any computer program that can increase its knowledge or performance by learning. This poses the question of what *learning* is, something that goes far beyond the scope of this thesis. Generally, however, the applied ML solutions that are seen today use models based on statistics to gather and abstract information from data (Wehle, 2017). The models can then be used to analyze and draw inferences from patterns of data, or to predict future values based on the underlying data.

[2]https://github.com/magenta/note-seq

While the ideas that underpin the majority of today's modern ML techniques are by no means new, it was not until the early 2010s that ML started seeing mainstream results. Increasingly cheap and powerful hardware like Graphical Processing Units (GPUs) and hard drives, plus new software allowed for quick processing of large data quantities through ML models. Over the last decade, this has led to huge leaps in computer vision, speech recognition and data analytics to name a few (Jordan and Mitchell, 2015). In the audio domain, ML has spawned solutions ranging from SOTA AMT to generative music models creating original pieces (Benetos et al. (2013); Chen et al. (2019)).

In general, ML can be roughly divided into three classes; supervised, unsupervised and reinforcement learning Mahesh (2020). In supervised learning, models learn to approximate a function that maps inputs to outputs based on labeled training data. A typical AMT use case would be learning audio characteristics (e.g. pitch and onset, see section 2.1) from labeled MIDI data. Contrary, unsupervised learning deals with unlabeled examples of data that it may for instance group into clusters of specific patterns or scan to find anomalies. Lastly, reinforcement learning is learning where the agent learns how to act based on feedback from the environment that it acts in, such as a program learning to play a video game by itself (Silver et al., 2016).

Widely used in all three classes of ML are Neural Networks (NNs) (Schmidhuber, 2015).

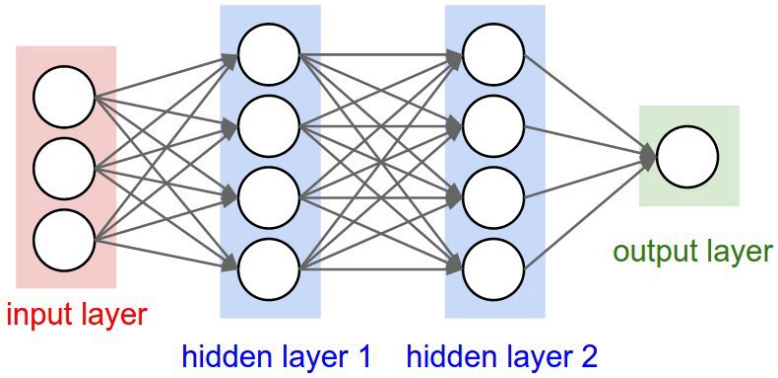

Figure 2.10.: The basic structure of a Neural Network

Figure 2.10 shows the basic structure of a NN. Generally, a NN is a set of connected nodes, called *neurons*, each taking some numbered input data and transforming it into some numbered output data through an activation function (Sharma et al., 2017). NNs have built-in layers so that the nodes in the first input layer passes their outputs to the second layer. The second layer passes their outputs to the third layer, and so on. The procedure continues until the output layer, which produces a numbered output for each of its nodes. Between all nodes in two adjacent layers, there are also assigned weights, determining how much the output from layer i should contribute as input in layer $i + 1$. NNs are then trained to approximate a function, mapping training input samples to training output samples. It does this by passing the input data through the

network and comparing the actual output to the desired one. Then, the weights of the networks are iteratively re-adjusted with a method called back-propagation. Simply put, back-propagation calculates the gradient of the loss between the actual output and the labeled output with respect to each weight in the network. This way, it can be determined how much each weight is to *blame* for the output error and then adjust accordingly. Running the back-propagation algorithm on enough examples of sufficient quality will let the NN converge to an optimal approximation (Schmidhuber, 2015).

Furthermore, it should be noted that while the description of a NN from the preceding paragraph can be viewed as a *general* NN, it is far from the only type. The general NN described above is often referred to as a Feed-Forward Neural Network (FFNN), while modern ML solutions more frequently use more specialized networks, like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). To familiarize the reader with the most important of these (relating to audio processing and AMT), a short description will follow below.

2.5.1. Recurrent Neural Networks

A Recurrent Neural Network (RNN) is an extension of NNs made to better handle time dependencies. As data is passed through a normal FFNN, information from earlier data will detrimentally vanish from the system. Thus, the model will have no way of *remembering* previous values it has seen, which often would be relevant for determining what to do with the current values.

To remedy this, RNNs are extended with recurrent loops from each node to itself on the layers in-between the input and output layers. This ensures that sequential information about the data is captured in the node itself because back-propagation now can regulate the importance of the data that the node passes recurrently (Rumelhart and McClelland, 1987). As a result, RNNs are most often used for time analysis. This lends itself nicely to audio, as this can be viewed as a time series of frequencies over time.

2.5.2. Conditional Neural Networks

Convolutional Neural Network (CNN) is a type of NN made for processing data that be viewed as a grid of numbers. The typical use case for this is images, which can be represented as a grid of pixel values. In AMT, these input images are typically spectograms as presented in section 2.4. CNNs differ from traditional FFNNs in that they take advantage of two additional types of layers; convolutional and pooling layers. These perform feature extraction on the input, while the proceeding layers map the extracted features into an output value, such as for FFNN (O'Shea and Nash, 2015).

The convolutional layers perform a filtering process on the input, which consists of placing a smaller grid over it. The values from the pixel in this grid, called the *feature extractor*, are then added together to form a single value. The feature extractor is applied

to every sub-image to produce the complete input to the convolutional layer. Next, a pooling layer is added to help reduce the dimensions of the data that is being output from the convolutional layer. This is done with a filtering process similar to that of the convolutional layer and helps the model reduce data complexity and model performance. Equally important, the pooling layers also summarize the features it was presented from the convolutional layer, allowing the model to operate on generalizable features rather than the precise position of pixels.

2.5.3. Transformer

The Transformer is an architecture for sequence transduction tasks presented by Vaswani et al. (2017). Sequence transduction tasks are any tasks where input sequences are converted into output sequences. SOTA AMT techniques are sequence transduction tasks because they convert a sequence of spectrograms into a sequence of note events, such as the NoteSequence structure described in section 2.4.5. The Transformer architecture from Vaswani et al. (2017) is shown in figure 2.11.

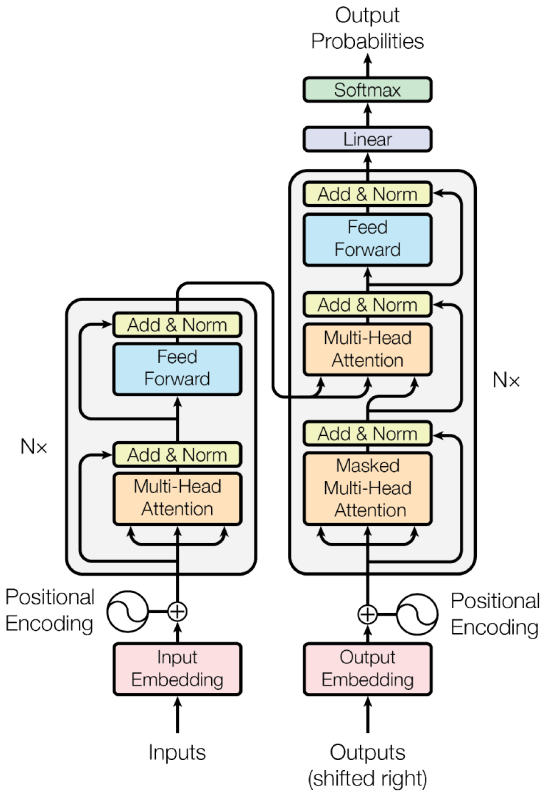

Figure 2.11.: The Transformer Architecture. Image from Vaswani et al. (2017)

The Transformer is solely based on attention mechanisms, which can be described as

mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The Transformer is based on an encoder-decoder architecture. At first, the encoder converts the input, represented as a sequence of symbol representations $(x_1, ..., x_n)$, to a sequence of continuous representations $(z_1, ..., z_n)$. Secondly, the decoder uses this output to convert it to an output sequence $(y_1, ...y_m)$, one element at a time. At the core of the encoder is a self-attention mechanism together with a standard feed-forward layer. The self-attention, introduced by Cheng et al. (2016), is a way to introduce undirected relations among tokens (symbol representation). Self-attention processes a sequence by replacing each element with a weighted average of the rest of the sequence (Raffel et al., 2019). Vaswani et al. (2017) rather use the multi-head attention, which to simplify stacks several singular self-attention modules, leading to better learning. The decoder follows the same structure as the encoder, however with the addition of multi-head attention over the output of the encoder stack to combine them. The Transformer is shown to achieve SOTA results for many sequence transduction tasks (Hawthorne et al., 2021).

2.6. Evaluation Metrics

Evaluation metrics are used to measure how well a model has managed to properly predict evaluation data by comparing the results to its ground truth. Related work in AMT (see chapter 4) generally evaluates performance based on precision, recall, and the F1 metric. Before assessing these metrics, this section investigates what a positive or a negative prediction means.

AMT is evaluated as a binary classification problem. Predictions in binary classification problems are distinguished between positive (*present*) and negative (*not present*) predictions. For instance, to simplify AMT, a note can be labeled as present (positive) or not present (negative) in a prediction. Moreover, a prediction can either be true (correctly predicted) or false (incorrectly predicted). A True Positive (TP) and True Negative (TN) prediction indicate that the prediction was correct compared to the ground truth label. Contrary, a False Positive (FP) and False Negative (FN) indicate that the prediction was wrong.

For AMT to become a binary classification problem, it is important to define what a positive prediction means. This thesis covers three various prediction types. The simplest type is to see if onsets over the course of the audio file are correctly predicted. The onset, as introduced in 2.1, is when a note starts. It is measured by dividing the audio file into small segments of a fixed length and is correctly predicted if the prediction occurs in the same time window as the ground truth. This window is often 50ms in length in AMT literature, as will be shown in chapter 4. Another type covered is measuring both the onset and the offset of a note. The offset is measured similarly to the onset, only for when a note stops. The final type is measuring what is known as the frame. It measures whether a piano-like representation of the predictions and targets match. In this type,

each second is divided into a fixed number of frames, where each frame represents a binary list of size 128 indicating if a note is present or not. The size of 128 indicates each of the 128 notes allowed in MIDI (section 2.4.4).

The three prediction types described form the categories of *Onset*, *Onset+Offset* and *Frame*. Counting the occurrences of TP, TN, FP, and FN alone of this does however not provide much information. Therefore, these occurrences can be combined into what is known as a metric. The metrics presented are precision, recall, and the F1 score.

Precision describes the number of TP predictions over the number of total positive annotations in the ground truth. Or stated more formally, the number of TPs divided by the number of TPs, plus the number of FPs. It is calculated with equation 2.5.

$$precision = \frac{TP}{TP + FP} \tag{2.5}$$

Recall gives the percentage of how many TP predictions there were in the number of predictions done by the predictor. It says something about how correct the model is at its predictions. Recall is calculated with equation 2.6.

$$recall = \frac{TP}{TP + FN} \tag{2.6}$$

The F-score, denoted F1 is defined as the harmonic mean of precision and recall, given in equation 2.7. The F1 score is widely used as a metric in AMT literature (chapter 4), shown in formula 2.7

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{2.7}$$

2.7. Audio Normalization

Audio volume depends on many factors: the recording device, storage formats, or the sample rate. Audio may even suffer from loss of data through compressing/decompressing of files and transmission over the internet, which again could affect the volume. More so, when audio is processed through software and other data operations (such as the software described in section 2.8), it is not uncommon that the volume of the audio is changed directly or indirectly (e.g. as a result of changed sample rate).

To have comparable results to each other when training an AMT model in two runs on two different datasets, it is important that the audio data has the same volume across all datasets. If not, one dataset may have either increased or decreased prediction score compared to the other. Depending on how you want to determine similar volume across tracks, there are different standardized ways to do so, called audio normalization. Two of the most common are presented below.

Peak Normalization

Traditionally, as music was processed by analog devices, the volume of an audio signal was measured as the amount of power sent through the electrical circuit that produced the signal. The higher the power of the signal (and thus its amplitude), the higher the volume. Since the devices used to measure the volume was purely physical, they had a built-in ceiling for how much power they could handle, called the *full scale*. A common normalization method, called peak normalization, uses such analog measuring devices (or a digital simulator of one) to normalize the audio relative to the full scale. This is done by measuring the distance from the highest audio peak (relative to full scale) across two audio samples. The two are then normalized by adding/subtracting gain to the sample that should be normalized, such that the distance from the highest peak to the full scale is the same for both samples.

Normalizing to 0 dBFS

A common variation of the type of peak normalization mentioned above, is to normalize the audio signal so that the highest peak hits 0dB. This is known as normalizing Decibels Relative to Full Scale (dBFS). Although this technique is applied to one signal at a time, not to normalizing with respect to another signal, it is still common in digital audio processing. The typical use case is to make an audio file as loud as possible without changing its dynamic range.

RMS Normalization

Root Mean Square (RMS) is a technique that is commonly applied to measure the average decibels in a signal over time. It is given by the formula:

$$RMS = \lim_{T \to \infty} \sqrt{\frac{1}{2T} \int_{-T}^{T} f(t)^2 dt} \tag{2.8}$$

Where $f(t)$ is a function mapping the time to decibel levels over an equal-spaced time-frame. To normalize over this RMS value, RMS normalization applies/subtract gain to one audio sample so that the average RMS of a signal becomes equal to a target signal. As with peak normalization, this average is measured relative to full scale.

2.8. Audio Plugins

Put simply, an audio plugin is a software that can be used to produce, analyze or alter audio. Typically, these plugins are written using standardized Application Programming Interfaces (APIs) so that they can easily be opened and used uniformly by the applications that use them. Generally, any program that can open and communicate with an audio plugin through its API is called a plugin host. This section will give a short overview

of the world of digital audio plugins and music production, relevant to this thesis. It will first cover shortly what a DAW is and how they relate to audio plugins. Then, an overview of the different plugin architectures in use today will be given, with a special focus on Virtual Studio Technology (VST). Lastly, the section will introduce how audio plugins are categorized and how they are commonly used to render and augment audio.

2.8.1. Digital Audio Workstation (DAW)

To paint a complete picture of the digital audio domain, this section gives a short introduction of a general DAW, the most common type of audio plugin host. A DAW is software made for musical audio editing, mostly with a focus on music production and/or recording. Some of the most popular DAWs include Ableton, Logic, and FL Studio (Sethi (2018)). While there naturally are some differences between these DAWs, their basic functionality remains the same; All allow audio recording, arrangement of musical samples into new pieces, and use of virtual instruments and effects (which is covered further in section 2.8.3). They all have support for MIDI, allowing direct recording of MIDI-supported instruments or controlling virtual instruments through a MIDI controller.

2.8.2. VST and Other Architectures

At the time of writing, the most used API for plugins is called VST. The VST architecture is written and maintained by Steinberg Media Technologies, who originally developed it to allow their own DAW, Cubase, to communicate with their own plugins. Yet, the architecture has seen widespread adoption and is supported by almost all the biggest DAWs and audio plugins (Steinberg Media, 2021). It is however important to be aware of the fact that there are two quite different versions of the VST architecture in use today: VST2 which was released in 1999 and VST3, released in 2008. VST3 was released to allow simplified communication and development. It introduced some important new features such as audio inputs for plugins and communication over multiple MIDI channels. However, VST3 has not yet reached the same adoption as VST2, and you will often see plugins distributed only as VST2 or as both VST2 and VST3 (however, more rarely only VST3).

Although VST is the most popular plugin API, there are also a lot of other architectures in circulation, such as Audio Unit (AU) and Avid Audio eXtension (AAX). When using a plugin in for instance a DAW, the DAW must also support the corresponding APIs. For instance, VSTs work in both FL Studio and Ableton on Mac, along with almost any DAW on Windows. Likewise, AU is specifically developed for Apple's own DAW, called Logic Pro, but is also compatible with FL Studio and Ableton if used on a Mac. Most plugins, however, are distributed as a single package, allowing you to choose the specific plugin architecture you want along with the installation. The specific architecture rarely has a lot to say about the performance of the plugin.

2.8.3. Effectors and Generators

As described by Goudard and Muller (2003), audio plugins can be roughly divided into three main categories of functionality; those transforming existing audio samples, those generating new audio through sound synthesis, and those used to analyze existing audio. For this thesis, we will disregard the latter, while referring to the first and second types as effectors and generators, respectively.

An **effector** is any audio plugin that is used to transform *existing* audio samples. This could be anything from a compressor to reverberation, saturation, and filtering. Common for all effectors is however the fact that they all use audio as input. Two examples of effectors include:

1. **RC-20**: This is an effector plugin that aims to emulate the feeling of old studio equipment, like cassettes and analog tubes. While this at first may not sound like an effector capable of drastic change, it is worth noting the plugin achieves this emulation by using modules as diverse as noise generation, saturation, distortion, reverberation, and fluttering, to name a few. These can all be controlled manually over a large parameter range, so naturally, the sound editing options are vast.
2. **SerumFX**: The built-in effect rack from Serum (see generator examples below), is packaged as a separate effector plugin. It comes with built-in equalizers, delay, distortion, chorus, reverb, flangers, envelopes, and many other tools for altering sounds.

A **generator** is any plugin used to generate new audio through sound synthesis. Thus, generators rely solely on MIDI as input for rendering the file to create an output audio signal. The rendering of audio can utilize a wide range of digital synthesis techniques, spanning everything from simple sine waves to complex wave-tables and high-quality samplers. Two examples of generators include:

1. **Serum** is an advanced synthesizer, relying on wave-table modulation to produce a huge variety of digital sounds. Serum also comes with a whole stack of built-in functionality for further sound-design options, like built-in effects, envelopes, and Low Frequency Oscillators (LFOs). Building on the large selection of wave-tables, this allows Serum to render digital sounds of high quality and variety.
2. **Kontakt**: This is a huge hardware library, relying on a detailed multitude of samples from actual instruments to transform these into digital instruments. This way, Kontakt can generate sounds likening everything from pianos and analog synthesizers to brass instruments and African percussion.

3. Datasets

This chapter will describe a subset of the datasets found in Automatic Music Transcription (AMT) literature, further described in chapter 4, and the datasets relevant for the experiments in this thesis, further described in chapter 6. An overview of the datasets is listed in table 3.1. The table shows the dataset metadata information with number of hours of audio, number of tracks, number of total instruments, and the number of instruments per track.

The dataset descriptions are based on the preliminary report (1.1), and are the same as those used in the experiments by Gardner et al. (2021), further detailed in section 4. These datasets are all used for AMT, spanning a variety of different genres, instruments, musical textures, and sizes.

Table 3.1.: Dataset Overview (data from Gardner et al. (2021))

Dataset	Num. Hours	Num. Tracks	Num. Instr.	Instr. per track
MAESTRO V3	199	1276	1	1
GuitarSet	3	360	1	1
MusicNet	34	330	11	1-8
URMP	1	44	14	2-5
SLAKH2100	969	1405	35	4-48
Cerebeus4	543	1327	4	4

3.1. MAESTRO

The MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) dataset is an audio collection of roughly 200 hours of piano performances, introduced by the Google Magenta team in conjunction with their Onset & Frames model for AMT (Hawthorne et al., 2018). The audio was captured by recording professional performers using a Yamaha Disklavier in an international piano competition. For each performance, associated Musical Instrument Digital Interface (MIDI) was captured simultaneously by a high-quality playback device.

This thesis is concerned with two different versions of the MAESTRO dataset: version one (V1) and version three (V3). Compared to V1, V3 had some extra data added from

new international piano competitions, while some wrongly included recordings of string instruments were removed. In addition, V3 also includes sostenuto and una corda pedal events, while V1 only includes the sustain pedal events.

3.2. GuitarSet

GuitarSet is a dataset curated by the Music and Audio Research Lab in New York (Xi et al., 2018). It is generated from guitar recordings, containing audio data together with metadata and annotations. An important distinction between GuitarSet and most other AMT datasets is that the annotations of GuitarSet come as JSON Annotated Music Specification (JAMS)-files, and not MIDI.

The recordings in the dataset are made with a hexaphonic pickup, which outputs one channel of audio signal per guitar string, as well as with custom annotation tools. The dataset consists of three hours of recordings from six different guitarists, playing 30 various excerpts ranging over three chord progressions and five genres.

3.3. MusicNet

MusicNet is a dataset by Thickstun et al. (2016), which contains MIDI annotations aligned with audio. The audio consists of freely licensed classical music records from 10 composers performing 34 hours of musical compositions under various studio and microphone conditions. The audio is played with eleven different classical instruments, such as the piano and violin.

The MusicNet dataset is transcribed by manual annotation performed by experts to form ground-truth label data. Because of this, the labels ends up being less accurate than what you would have gotten from using a MIDI-compatible keyboard or a hexaphonic pickup to capture notes. As a result, transcription performance on MusicNet is consistently lower than on related datasets across a variety of different approaches, such as for Gardner et al. (2021).

3.4. URMP

The University of Rochester Multi-modal Music Performance (URMP) dataset consists of 44 simple multi-instrument classical pieces, accompanied by a musical score in MIDI format (Li et al., 2019). The pieces span duets, trios, quartets and quintets with a total of 14 different instruments, including strings, woodwinds, and brass.

The pieces are assembled from coordinated but separately recorded performances of individual tracks. As with the MusicNet dataset, it is worth noting that URMP is also

manually annotated to MIDI, putting a larger uncertainty margin on the ground-truth labels.

3.5. SLAKH2100

The Synthesized Lakh (SLAKH2100) dataset consists of high-quality renderings of instrumental mixtures and corresponding stems generated from the Lakh MIDI dataset (Manilow et al., 2019). The Lakh dataset is a collection of 176,581 MIDI files scraped from publicly available sources on the internet (Raffel, 2016). The renderings in SLAKH2100 are made using professional-grade sample-based virtual instruments (see section 2.8) on 2100 tracks from Lakh, resulting in 145 hours of audio stems. Every mixture contains stems from piano, guitar, bass, and drums, but some additionally include stems from other instruments such as brass, organ, and pipes. When Gardner et al. (2021) use SLAKH2100, they expand the dataset by taking 10 random subsets of at least 4 instruments from each of the files in the dataset into new combinations as a new track. This expands the number of training examples by a factor of 10, hence the large number of hours in table 3.1.

Cerebeus4

Cerberus4 is a dataset derived from SLAKH2100, obtained by mixing all combinations of the four instruments guitar, bass, drums, and piano, in tracks where those instruments are active (Gardner et al., 2021). Doing so results in 1327 tracks, with a total of 542.6 hours of audio.

4. Related Work

The technological era has opened up for more complex analysis of musical sounds, e.g. with the time-frequency analysis techniques presented in section 2.3. Furthermore, machine learning (section 2.5) has, as in many other fields, also shown promising results for Automatic Music Transcription (AMT). To understand the technology behind AMT, the first section of this chapter covers mathematical approaches based on signal processing and music theory. These are preliminary methods to the current State-of-the-art (SOTA) deep learning methods. The deep learning methods in question are presented in the following sections, separated into two types of architectures: Acoustic and Musical Language Models, and Transformers. Additionally, this chapter includes a section on Music Augmentation. The section further addresses previous methods, techniques, and technology that relate to synthetic augmentation of musical data, especially for AMT. More so, it also covers recent technological approaches that may help bridge the gap between AMT and music production technology, as briefly discussed in section 1.1.

4.1. Preliminary Approaches

This section covers two preliminary approaches to what the SOTA in AMT is. They are combinations of computing, mathematics, and music theory, and form the basis of the presented SOTA approaches in section 4.2 and 4.3. The first method is a method for transcribing audio, assigning pitches to the frequencies played in a sound segment by using the data produced by the Fast Fourier Transform (FFT) (section 2.4). The second method is based on assigning pitches to frequencies as well, however with the use of a method called Non-Negative Matrix Factorization (NMF), which can model frequency spectres over time.

Piszczalski and Galler (1977) aim at automatic transcription of musical sounds into the corresponding staff notation representation (section 2.4.3). This is done by converting sound signals into the frequency domain over short periods, creating a three-dimensional structure whose axes are time, frequency, and amplitude. This conversion is done with a slightly altered version of the FFT. The strongest frequency is then found in each of the time sections, which represent the pitches of the notes. The end of a note is defined as either going silent or as an abrupt change to another note. Knowing the beginning and end times of the notes, pitches are assigned to the average frequencies. These pitches can then later be converted to staff notation based on a predefined set of rules.

The work from Piszczalski and Galler is limited in its scope: It is only applicable to monophonic transcription, not capable of transcribing several pitches at once. In addition, they prompt for the time signature and number of beats before the first bar of the staff notation, making it less flexible for various playing styles. Piszczalski and Galler describe that the ability of the system to convert music into notation depends heavily on the complexity of the sounds presented to the analysis system. For instance, electronically-generated sinusoidals are the simplest as the Fundamental Frequency (F0) can easily be retrieved, and are easier to transcribe. Therefore Piszczalski and Galler aim to transcribe instruments with strong F0, such as the flute and recorder. In most musical instruments, however, the strongest-frequency line alone does not contain sufficient information for complete identification of the musical notes.

Smaragdis and Brown (2203) suggest that another way of modeling sound segments is with the use of Non-Negative Matrix Factorization (NMF). NMF was first introduced by Lee and Seung (1999) as a way to learn the most prominent parts of objects, such as faces or semantic features in text. As Smaragdis and Brown describe, the goal of NMF is to approximate a non-negative matrix $M \times N$ as a product of two non-negative matrices. The matrices produced are $W \in \mathbb{R}^{\geq 0, M \times R}$ and $H \in \mathbb{R}^{\geq 0, R \times N}$, where $R \leq M$, and sets the rank of approximation. This rank for approximation represents how many *bins* the object is approximated into.

An FFT spectrogram $X \in \mathbb{R}^{\geq 0, M \times N}$ is used as input when modeling sound segments with NMF. Smaragdis and Brown use a 4096-point FFT window. In addition, the rank of approximation R must be set. This parameter is ideally set to the number of unique notes played in the sound segment. If not, some notes are conjoined into one representation, and information is lost. The content of the produced matrices is as follows: Matrix W represents the frequency spectra for each of the unique notes. This is similar to taking the Fourier transform of each individual note present in the audio (if R is set correctly). Matrix H is a representation of when each of the note's frequency specters is active over time. This information can be mapped into for instance the Musical Instrument Digital Interface (MIDI) format (section 2.4.4) or the staff notation (section 2.4.3), if the corresponding pitches in W are assigned.

The methods from Piszczalski and Galler (1977) and Smaragdis and Brown (2203) are great for retrieving simple information about audio. The use of NMF can be seen as an improvement in comparison to the method of Piszczalski and Galler (1977), as it can transcribe polyphonic music, compared to monophonic music. Both methods do still suffer when more complex musical pieces are presented to the system, as the produced spectrograms are harder to interpret. Deep learning has shown to improve this, with the methods presented in the following sections 4.2 and 4.3. These methods are also based on the FFT, hence also based on learning the frequency specters on the notes, however in a black-box fashion.

4.2. Acoustic and Musical Language Model

This section presents a type of transcription system that is separated into an *acoustic model* and a *musical language model*. The acoustic model is a Neural Network (NN) used for estimating the probabilities of pitches in a frame of audio. The language model is a Recurrent Neural Network (RNN) (see section 2.5.1) that models the correlations between pitch combinations over time (Sigtia et al., 2016).

At the frontier of AMT solutions are two similar architectures from Hawthorne et al. (2017) and Kong et al. (2021). The latter is a refinement of the former. Both solutions take advantage of a *stack* model layout to predict sub-tasks within AMT.

In Hawthorne et al. (2017), the stacks are formed by a convolutional *acoustic model* architecture presented in Kelz et al. (2016), followed by a Bidirectional LSTM (BiLSTM) as the language model. Long Short-Term Memory (LSTM) is a type of RNN (Hochreiter and Schmidhuber, 1997), described in section 2.5.1. A bidirectional RNN is an RNN trained simultaneously in positive and negative time direction (Schuster and Paliwal, 1997). A BiLSTM is therefore an LSTM trained as such. The acoustic model described are Convolutional Neural Networks (CNNs) (see section 2.5.2) that take logarithmically filtered FFT spectrograms with 229 frequency bins as input. Hawthorne et al. calculate the spectrograms with an FFT window of 2048, a hop length of 512, and a 16kHz sample rate. The output from the acoustic model is then fed to the BiLSTM, forming a *stack*. For onset regression, this stack is followed to a fully connected sigmoid layer with 88 outputs predicting onsets for each of the 88 keys on a piano. Furthermore, they combine this onset stack with another stack trained with frame regression, with the same number of outputs as the prior. A stack with the same setup is used to predict velocities for the onsets, however, this is separated and does not affect the performance of the onset and frame prediction.

Kong et al. (2021) have similarly used the acoustic and language model for their experiments. The acoustic model is the same as in Hawthorne et al., but the language model consists of a Bidirectional GRU (BiGRU) rather than a BiLSTM. BiGRU are a another type of a bidirectional RNN, rather with a Gated Recurrent Unit (GRU) (Cho et al., 2014) at its core. The acoustic and language models in Kong et al. also form a stack to predict their AMT sub-tasks. In their work, their sub-tasks are velocity regression, onset regression, frame regression, and offset regression. The stacks are connected by another BiGRU where velocity connects to onset, and both onset and offset connect into frame regression.

One difference between the two papers other than their internal model configurations, is what is defined as an onset, as shown in figure 4.1. In Hawthorne et al. (2017), an onset is assigned to a predefined time window of 32ms. This is shown in the upper part of the figure. When transcribing audio, an onset will therefore belong to a predicted frame rather than the exact transcription point. The attempt from Kong et al. (2021) to solve this limitation is shown in the lower part of figure 4.1. In this model, each of

the frames rather gets assigned the distance to the nearest onset. This is done for a preassigned number of closest frames. This technique is thought to give more precise transcription since the exact predicted onset time is known by the model, shown by the results described in the next paragraph.

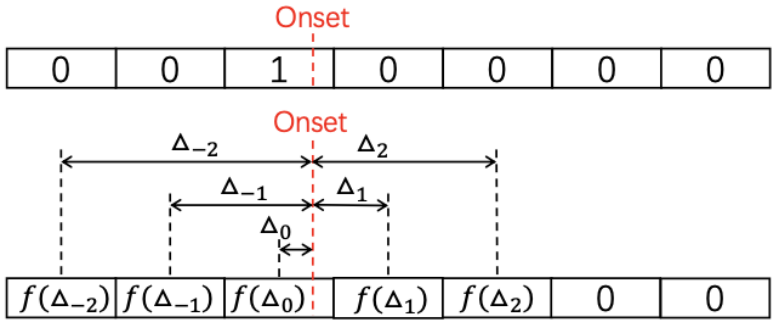

Figure 4.1.: Representation of onset in Hawthorne et al. (2017) (upper) and Kong et al. (2021) (lower). An edited version of fig. 1 in Kong et al. (2021)

Hawthorne et al. (2017) achieved a 78.30% Frame F1 score when trained and validated on the MAPS dataset. The MAPS dataset is a dataset consisting of approximately 65 hours of MIDI-aligned piano sounds. When trained on the MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) (Hawthorne et al., 2018) dataset, described in detail in section 3.1, the results for the MAPS validation dataset were updated to 84.91 for the same metric. MAESTRO is a larger dataset than MAPS, and more data is proven to perform better in AMT solutions (Hawthorne et al., 2018). The results when trained and validated with the MAESTRO dataset are shown in table 4.1. Then, the model's Frame F1 was as good as 89.19% (Kong et al., 2021). The proposed model from Kong et al. pushes the SOTA further to 89.62%. In addition, they improved the Onset F1 metric to 96.72% from 93.92% performed by Hawthorne et al. (2018). The Onset+Offset F1 metric was improved from 71.28% to 79.06%. All metrics were calculated with a tolerance window of 50ms, as introduced in section 2.6.

4.3. Transformer Methods

The Transformer architecture has shown SOTA results across many sequence transduction tasks, simply by varying the input and output representations of a shared, core architecture, as presented in section 2.5.3. AMT can also be treated as a sequence transduction task, for instance in Kong et al. (2021) and Hawthorne et al. (2017) where FFT spectrograms are used as input, and a MIDI-like structure is used as output. Therefore, attempts have been made to apply this architecture to AMT which is presented in this section.

Hawthorne et al. (2021) attempted to use the Transformer architecture for AMT. Their scope was transcription of polyphonic piano music. They define the input vocabulary as FFT spectrograms. The input spectrograms differ slightly from Hawthorne et al. (2017), presented in section 4.2. The similarities is that the sample rate is 16kHz, and the FFT length is of 2048 samples. The hop width, however, is 128 samples. The output is scaled to 512 mel bins (see section 2.4.2), rather than logarithmically-spaced frequency bins. The magnitude of the spectrograms is log-scaled, however. The output is represented as four tokens: **Note**, a note-on or note-off event for one of the 128 pitches; **Velocity**, indicating a velocity change for all subsequent events of 128 velocity values; **Time**, the absolute time location (a 10ms window) within a segment; and **EOS**, indicating the end of a sequence. When trained and validated on the MAESTRO V1 dataset, this architecture showed comparable results to Hawthorne et al. (2017) and Kong et al. (2021) in all metrics of Onset, Offset, and Velocity F1. The results from the MAESTRO V1 dataset are shown in the upper part of table 4.1.

Gardner et al. (2021) extend the idea from Hawthorne et al. (2021) to include transcription of polyphonic music with an arbitrary number of instruments. They do this by changing the output dictionary of their Transcriber architecture. First, they removed the **Velocity** token because it was not in their project scope. Secondly, they added an **Instrument** token, indicating which of 128 pre-defined instruments are playing. An **End Tie Section** token was also added, which ends a specific section at the beginning of the output that declares what note is playing at the start of a frame. They named the system the Multi-Task Multitrack Music Transcription (MT3), because an ideal AMT system should be capable of transcribing multiple instruments at once (Multitrack) for a diverse range of styles and combinations of musical instruments (Multi-Task) (Gardner et al., 2021).

The MT3 model is evaluated on all of the six datasets presented in chapter 3. These datasets are diverse in their types of instruments, meaning the multi-instrument performance of the model is possible to assess. The evaluation was done using two separate models: (1) An MT3 model trained on a mixed dataset of all of the above datasets, and (2) an MT3 model trained only on the dataset to evaluate. The models were evaluated with the metrics Frame F1, Onset F1, and Onset+Offset F1, described in section 2.6. The results from model (2) is shown later in table 6.2 for four of the datasets. Previous SOTA results on the evaluation of each of the individual datasets were outperformed by MT3 whether using the single or mixed dataset model. The only exception was compared to the work from Kong et al. (2021) with evaluation of the MAESTRO V3 dataset, which performed equally for all three metrics. The results for MAESTRO V3 are shown in the lower part table 4.1.

MT3 differentiates from the SOTA methods presented in section 4.2 through generalizability. MT3 takes advantage of existing off-the-shelf Transformer models, instead of having an acoustic and language model designed specifically for its purpose. In addition, MT3 performs equally to the SOTA, in addition to being able to transcribe all instrument classes.

Table 4.1.: Results on MAESTRO validation partition for related work. The table is separated between the V1 and V3 version of MAESTRO

	Model	Onset, Offset & Vel. F1	Onset & Offset F1	Onset F1
V1	Hawthorne et al. (2018)	0.78	0.81	0.95
	Kong et al. (2021)	0.81	0.82	**0.97**
	Hawthorne et al. (2021)	0.82	0.83	0.96
V3	Hawthorne et al. (2021)	**0.83**	**0.84**	0.96
	Gardner et al. (2021)	-	**0.84**	0.96

4.4. Music Augmentation and Related Technology

The idea of augmenting data for machine learning models to increase the size and/or complexity of a dataset is not new and has been applied to a variety of domains, ranging from images to analytical time series (Wen et al., 2020). In recent years, musical datasets have also been subject to various augmentation operations with the purpose of increasing the data quantity and quality. These augmentation strategies vary from simple pitch- and time-shifting operations, to the application of machine learning techniques to determine the position of new samples that are added to the music (McFee et al. 2015; Southall et al. 2018). In particular, this section will present previous augmentation approaches to musical data. It will also look at recent technological developments that help facilitate different augmentation operations through audio plugins (as described in section 2.8).

4.4.1. Previous Augmentation Approaches to AMT

One of the earliest augmentation approaches that was successfully applied to AMT, is the *Software Framework for Musical Data Augmentation* presented by McFee et al.. At the highest level, the framework uses what the authors have called deformation objects. Each such object implements one or more transformation methods, augmenting the audio in different ways. Each of these transformations takes an audio signal and its corresponding transcription as input before augmenting the data. The deformation objects implement four different types of augmentation strategies:

1. A pitch shift of the audio by a value of n $\in$ {-1, +1} semitones.
2. Time stretching the audio by a factor of r $\in \{2^{-\frac{1}{2}}, 2^{\frac{1}{2}}\}$.
3. Adding background noise to the audio. Noise chosen from three types (hall, subway, and random) is linearly mixed with the original audio signal.
4. Dynamic range compression (using two presets from the Dolby E compressor). A compressor limits the dynamic range (distance between maximum and minimum

volume) in the audio.

Using this software framework, McFee et al. created five different datasets from the MedleyDB dataset (Bittner et al., 2014) to validate their solution. The MedleyDB dataset consists of raw audio tracks with corresponding stems and MIDI annotations. Of the five datasets, one was without augmentations, and for the remaining four the deformation objects were added iteratively in the order listed above. While all augmented datasets outperformed the non-augmented dataset for AMT on F1 score, the authors reported that the dataset where only the pitch deformation was applied performed the best.

While simple, the augmentation techniques from McFee et al. prove that there is ground for believing that the data shortage in AMT can (at least, partially) be overcome using augmentation approaches.

Kim et al. (2018) went even further with data augmentation, developing a NN model for music synthesis that allows for flexible control of the timbre of an audio, called Mel2Mel. Timbre can be viewed as the particular tone or characteristic that separates one sound from another.

First, the initial MIDI representation of some audio is encoded and fed into the model, which further is encoded by an RNN. This RNN is conditioned on an embedded representation of the timbre space of an instrument, such that the RNN produces a mel spectrogram (see section 2.4.2) that corresponds to the input MIDI rendered with the given instrument. Next, this spectrogram is processed through a WaveNet vocoder to produce generated raw audio from the RNN-produced spectrogram. The WaveNet vocoder is a generative model that can generate raw speech signals from spectrograms, which Kim et al. (2018) extend for generation of music. Using this approach, Kim et al. is able to render any of the input MIDI data with any of the timbre encodings captured from a dataset, combining them into new musical data that can be used for AMT purposes.

As motivation for creating the system, Kim et al. argue that modern synthesizers and samplers (see section 2.8) sacrifice timbre control over synthetic sounds (which is also further discussed in chapter 7), and that when timbre control is offered, it is usually very limited. Yet, the system approach suffers from its own limitations. Training on datasets augmented through the model is indeed giving lower validation scores than their original counterparts. First and foremost, this is assumed (by Kim et al.) to be owed to degradation in audio quality. As the model goes through several stages of prediction, there incurs gradual degradation of the audio quality at each stage. Additionally, there is also some degradation when applying the WaveNet model, which is not able to process the audio in a lossless manner. In fact, in an ablation study performed by the authors, there was a drop from 4.3 to 3.1 (on a scale from 1-5) in the subjective musical quality assigned by participants to the original and rendered audio, respectively. Furthermore, even though the Mel2Mel approach is capable of producing advanced and flexible timbers, the timbers it can produce are still limited by the input datasets.

4.4.2. Augmentation Using Audio Plugins

Another way to augment existing AMT datasets would be to apply effects and/or to re-render MIDI using generators (as described in section 2.8). As thoroughly discussed in the preliminary report to this project (section 1.1), there is reason to believe that rendering data this way may help overcome some of the existing data deficiencies in AMT. However, these plugins are generally developed for manual use through Digital Audio Workstations (DAWs) and other audio processing software. Manual processing of datasets of the magnitude described in chapter 3 would not be feasible. Instead, it would be necessary to have a system in place that automates this process based on some input parameters (for instance what plugins and presets to use). Thus, the preliminary report also propose that a system could be made, using for instance the Application Programming Interface (API) to the Virtual Studio Technology (VST) architecture (see section 2.8), to automatically control the plugin operations. The proposed system would make use of plugins, similar to existing DAWs, except that the input and processing would have to be based on easily controllable parameters and lines of code, instead of a direct mouse and keyboard input (as is the normative case).

Between the writing of the preliminary report and this thesis, Braun (2021) published a report on how to *bridge the gap between DAWs and Python interfaces*, along with an application called DawDreamer. DawDreamer is a Python module capable of rendering audio and MIDI through VST effectors and generators (see section 2.8) by using a few lines of code. It supports stacking of VST instances that can be used to render both MIDI and audio, along with other parameters such as presets to use, audio duration, and sample rate. DawDreamer goes a long way in obsoleting the proposition in the preliminary report of making such a framework and has therefore been used for this exact purpose in the architecture (see section 5) that carries out the experiments related to this thesis (see section 6). It does however, comes with a few limitations of its own, which is further discussed in chapter 7.

5. Architecture

To conduct the experiments outlined in chapter 6, it is necessary to first have in place:

1. A system capable of automatically applying digital audio plugins to audio and Musical Instrument Digital Interface (MIDI), facilitating the rendering of new datasets.
2. A system that can convert these rendered datasets to a format that can be understood by a predictor model.
3. A State-of-the-art (SOTA) Automatic Music Transcription (AMT) model, that can be trained on the rendered data and used to infer notes from new audio examples.

This chapter will present these systems in the same order as listed above. Figure 5.1 shows an overview of the complete architecture of the final system that carries out the tasks. The architecture consists of three subsystems further described in this chapter: the AudioTransformer, the DatasetPreparer, and the PredictorModel

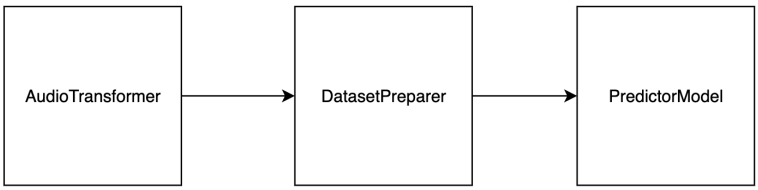

Figure 5.1.: An overview of the modules in the presented architecture

This chapter explains the architecture and inner workings of the system in further detail, highlighting the most important concepts and discussing the major architectural decisions. A section is dedicated to each of the systems that perform the respective tasks mentioned above, as they can be viewed as separate parts of the final solution. Each sub-section is dedicated to a separate module of its respective system, and these will be presented in an order that gradually comprises the whole solution.

5.1. AudioTransformer

The AudioTransformer is the system responsible for applying digital audio plugins to audio and MIDI, as described in section 2.8. It is built around the DawDreamer Python module,

earlier described in section 4.4.2, which makes it possible to render audio through audio plugins. It is therefore only applicable for audio plugins made with the Virtual Studio Technology (VST) architecture (see section 2.8). This section includes a description of the Pipelines module, responsible for generating a pipeline of operations for automation of DawDreamer, and the PipelineParser module, responsible for the automation itself based on the data from Pipelines. The various modules of the AudioTransformer is displayed in figure 5.2.

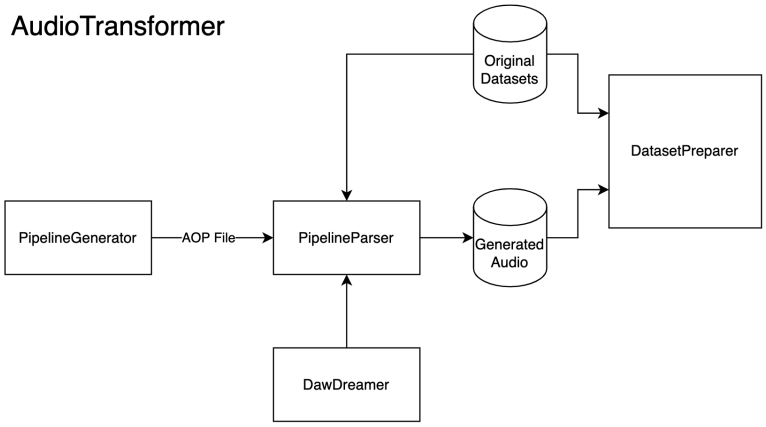

Figure 5.2.: Overview for the AudioTransformer

5.1.1. Pipelines

To first understand the AudioTransformer, it is important to understand the basis that it acts upon. This sub-section details operational pipelines used to represent and process audio and MIDI in the AudioTransformer.

Audio Operation Pipelines

To keep the operations of the AudioTransformer systematic and ordered, its essential that there is a uniform way of representing these operations in place. Hence, a standard for stating the effect- and/or generator-operations (see 2.8.3) to apply to a collection of audio and/or MIDI data, coined Audio Operation Pipelines (AOP), is presented. This is inspired by the deformation objects first introduced by McFee et al. (2015) (mentioned in section 4.4), which can be combined to create augmentation pipelines. While straightforward, AOP enables convenient operation storage, batch processing of audio, multi-plugin processing, easy combination of simultaneous operations, and more. Furthermore, when using AOP files, it is easy to go back and find out which audio was rendered using which effect and plugin.

The AOP file is a JSON document containing three main attributes: *id*, the identifier used to reference the AOP file; *pipelines*, which contains a list of pipelines for augmentation

of audio; and *org_folder*, which contains the path to the original dataset(s), such that original files can easily be opened and accessed during rendering. Each pipeline further specifies three main attributes:

- id: A unique identifier for the pipeline.
- bpm: The Beats Per Minute (BPM) to use when rendering the given pipeline.
- data: A list of operation objects.

An operation object represents the application of one or more generators and/or effects (see section 2.8) to an audio or MIDI file. The properties of an operation object are:

1. file: The file path of a MIDI or audio file, that will be used as input source for this operation.
2. generator: A JSON object with keys *plugin* and *preset*, and respective values pointing to the paths of the generator plugin and preset that should be used to render the input file. Naturally, applying a generator to an audio file makes no sense, so the generator value should be set to null unless the operation input is MIDI.
3. effects: A list containing JSON objects for every effector that should be applied to the input-source. These JSON objects should also have keys *plugin* and *preset*, with respective values pointing to the paths of the effector plugin and the preset that should be used. If no effector plugins should be applied, the effector value can be set to null.

Each operation in a pipeline represents one corresponding output audio file. If more operation objects are provided in a single pipeline, however, these are supposed to be merged together, resulting in a single output audio file from each pipeline. This responsibility is in practice up to the parser of the AOP file, which sends the instructions for generation, further discussed in section 5.1.2.

An example of a full AOP file is supplied in appendix G.

PipelineGenerator

PipelineGenerator is the module responsible for generating AOP files. It would be very cumbersome to manually specify what files to apply plugins to, not to mention manually assigning plugins and presets to each operation. Instead, the PipelineGenerator automates this. It is capable of generating AOP files in arbitrary sizes, in pretty much every way imaginable; combined audio and/or MIDI, with or without generators, and with any number of effects. To do this, the PipelineGenerator takes as input the path to the folder(s) containing MIDI and/or audio files to generate from. Along with configuration parameters, such as available plugins and the paths to the folder(s) containing presets. Additionally, the PipelineGenerator support optional parameters, for instance to decide the probability of a file from the input folder being included in the pipeline, the distribution

for use between the different plugins, and the number of different plugins to use for each operation.

Having the many options of the PipelineGenerator at hand, it is easy to quickly generate pipelines representing different renderings of a dataset. This is crucial when performing the experiments outlined in chapter 6, because different renderings must be analyzed in order to more closely understand what impact a set of operations have on a dataset.

5.1.2. PipelineParser

AOPs would be of no use if the operations they stated was never actually carried out. Thus, the AudioTransformer includes a module called PipelineParser, capable of parsing an AOP file, performing the operations, and saving the resulting output audio. Rendering of a pipeline is made possible by heavy utilization of the DawDreamer Python library, which is introduced in section 4.4.2. The PipelineParser packs the DawDreamer functionality by automatically creating the necessary processors and utilities to render audio and MIDI through audio plugins. These output audio files are saved in a chosen output directory, and given names matching their IDs, such that it is easy to backtrack the origin of each output file.

To gracefully handle parsing of an AOP file, PipelineParser comes with a few (optional) usage options and some built in safeguards to handle AOP syntax errors. Firstly, PipelineParser allows for specification of desired sample rate and duration of output audio. The sample rate defaults to 16kHz unless anything else is specified, and the longest duration of all input files (both MIDI and audio) is used as the standard length of each pipeline output. Furthermore, the PipelineParser will gracefully skip an operation if MIDI is supplied without a generator, or if a generator is assigned to an audio input. However, if any of the two aforementioned cases occur, a warning will be logged to the user as this likely is a result of an error in the mechanism used to create the pipeline. Additionally, the PipelineParser is also responsible for normalizing the volume (see section 2.7) of each rendered audio clip. For each rendered file, the volume is normalized according to the measured Root Mean Square (RMS) volume of the corresponding original file. This helps ensure that the original and rendered audio remain comparable, even after rendering.

5.2. DatasetPreparer

After a dataset has been rendered and saved by the PipelineGenerator, it needs to be parsed and presented to the PredictorModel (section 5.3) in a format that it can understand. To achieve this, the data is sent through a series of modules, collectively called the DatasetPreparer, making it ready to be used in the PredictorModel. This section presents the modules responsible for these transformations and how the data is transformed at each step in detail. The modules of the DatasetPreparer are displayed in figure 5.3.

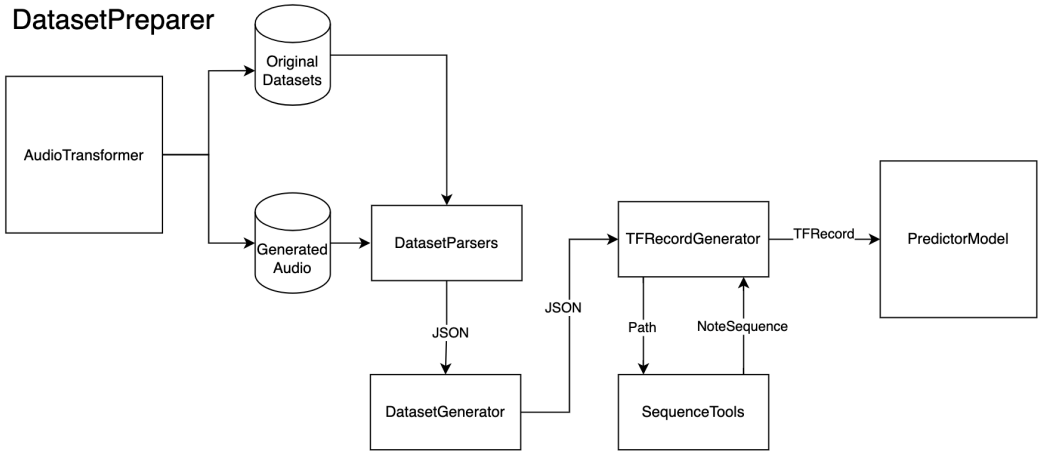

Figure 5.3.: Overview for the DatasetPreparer

5.2.1. DatasetParsers

After the data is rendered and new datasets are produced through the AudioTransformer, the DatasetParsers module is responsible for grouping the audio files produced by the AudioTransformer module in a sensible way, so they can be used further. Data is grouped such that it is easy to find affiliated MIDI and metadata (if any). DatasetParsers relies on regular expressions to extract the relevant data for each dataset. To avoid unnecessary duplication of the data related to a track, the DatasetParsers module is supplied with two parameters; the path to the rendered dataset and the path to the original dataset. Traversing these folders using regular expressions, the parser creates an object for each rendered track, and assigns with it related MIDI, instruments and other metadata from the original dataset.

5.2.2. SequenceTools

MIDI is more than just a representation of notes, as described in section 2.4.4. It is mainly a file format and cannot directly be interpreted by a computer unless instructed on what the data means. SequenceTools is a static class for converting MIDI files to an interpretable MIDI-*like* format and vice versa. The format in question is the NoteSequence format, presented in section 2.4.5.

5.2.3. TFRecordGenerator

When a dataset is fittingly grouped and parsed by the DatasetParsers module (see section 5.2.1), it needs to be converted into a format that the PredictorModel, described later in section 5.3, can understand. The PredictorModel takes the TFRecord file format as input. The TFRecord format, or TensorFlow Record, is a format for storing a sequence

of binary records (datasets) developed by Google[1]. A TFRecord file consists of several Examples, where one Example represents one data entry.

The datasets listed in chapter 3 all have great variation within the metadata in their original form. Some datasets include the various stems of the instruments, while some are single-instrument. Some datasets are stored in MIDI format, while others are annotated in JSON Annotated Music Specification (JAMS) or CSV files. Some data, such as for instance videos, are not used. Because of all of the variation, the TFRecordGenerator implements one unified structure for the PredictorModel to consume, with the following fields:

- **id**: Name of the entry
- **audio**: Byte string of audio
- **sequence**: Byte string of a NoteSequence object (see section 2.4.5 containing the note annotations the entry)

The TFRecordGenerator works by inserting a list of data to be processed represented as Python dictionaries. The input data contains paths to the belonging audio file, its MIDI, in addition the metadata needed to produce the output (name and program numbers). This is the output produced by the *DatasetParsers* module. After receiving the data, it is processed and stored in a TFRecord with the fields presented.

5.2.4. DatasetGenerator

DatasetGenerator is a module for utilizing the TFRecordGenerator module for generating datasets. It is essentially a script for sending the output from the DatasetParsers module to the TFRecordGenerator module. The DatasetGenerator reads a dataset configuration file, and maps the data to retrieve the correct information from the DatasetParsers module. The configuration file contains the name of the new TFRecord, the name of the original dataset and the path to the folder of the datasets. To generate a TFRecord for audio that is custom made with the AudioTransformer, the path to the folder of the newly generated audio is also provided.

5.3. PredictorModel

The presented AudioTransformer produce the content used to answer the research questions, however this section presents the system for validating how the produced custom datasets perform when used as input in AMT models. On the basis of the thesis goal and the introduction in 1, this system should be a SOTA solution within AMT. As uncovered in section 4.3, Gardner et al. (2021) and their Multi-Task Multitrack Music Transcription (MT3) architecture is recognized as the current SOTA within AMT,

[1] https://www.tensorflow.org/tutorials/load_data/tfrecord

because they yield better or equal results to previous SOTA solutions on all six datasets described in chapter 3. This architecture is therefore used as the system to answer the presented research questions. MT3 is an open-source project, and the code is available on GitHub[2]. It is built on top of the T5X Transformer, a specific implementation of the Transformer architecture presented in section 4.3 by Roberts et al. (2022).

The PredictorModel use the TFRecords (section 5.2.3) produced by the TFRecordGenerator as input. This data is then scanned by the DatasetTasks and the DatasetMixtureTasks modules, before chosen for training or evaluation. This way, MT3 can systematically process audio and find necessary metadata under training and validation. These two modules are described in this section. Additionally, this section includes a description of how evaluation of the PredictorModel is done.

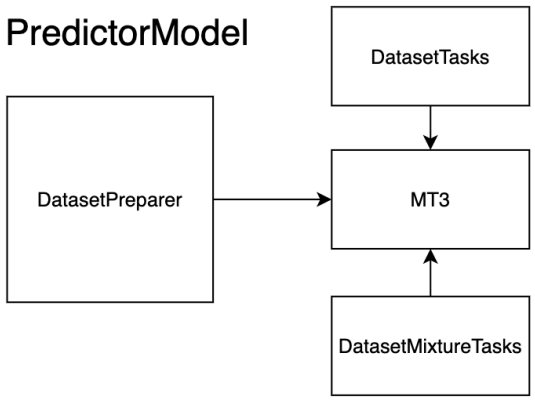

Figure 5.4.: Overview for the PredictorModel

5.3.1. DatasetTasks

The architecture takes advantage of the SeqIO library for handling data pipelines. SeqIO is a library for processing sequential data to be fed into downstream sequence models (Roberts et al., 2022), for instance a Recurrent Neural Network (RNN). A pipeline of processing is called a *task*. A task is defined by a data source, pre-processing steps, and a vocabulary to tokenize/de-tokenize each pre-processed feature for the model. The Task also contains post-processing steps to prepare the model output for evaluation, in addition to its evaluation metrics. New tasks must be registered in a directory called the TaskRegistry, and can later be retrieved when training the model.

Each experiment run must therefore be registered as a task in the TaskRegistry. This is done by the *DatasetTasks* module. It does so by scanning a local folder containing all TFRecords with data. Each TFRecord is then mapped with their respective train/validation label. The module is loaded upon run-time of the PredictorModel, before selected through configuration files when used.

[2]https://github.com/magenta/mt3

5.3.2. DatasetMixtureTasks

While some experiments listed in chapter 6 are solely based on single datasets, some experiments are based on mixtures of datasets. This is what the DatasetMixtureTasks is responsible for. These mixtures can for instance be mixtures between the original and a generated dataset from the DatasetGenerator, or mixtures between several of the generated datasets.

SeqIO (presented in section 5.3.1) has support for registering *mixtures* of the tasks registered from the DatasetTasks module, through a class they call the MixtureRegistry. The already defined tasks are registered to the MixtureRegistry with a name of the mixture task, together with the name of the tasks in which are going to be mixed. For each of these tasks, it is possible to define the ratio of samples to pick from each tasks. A ratio of 7:3 between task A and B would result in a mixture where 70% of the data is from task A, and 30% of the data is form task B. It is possible to mix a mixture with both another mixture or a task. The mixtures made by the DatasetMixtureTasks model, follow the same mixing strategy as in Gardner et al. (2021), to balance model performance on low- and high-resource datasets, which is: if dataset i has n_i examples, an example is sampled from that dataset with probability $(n_i/\sum_j n_j)^{0.3}$.

5.3.3. Evaluation

The evaluation of the PredictorModel is incorporated into the MT3 implementation. Evaluation is done with the *mir_eval* library. *mir_eval* is an open source software library which provides a transparent and easy-to-use implementation of the most common metrics used to measure the performance of Music Information Retrieval (MIR) algorithms (Raffel et al., 2014). MIR is a research field regarding information retrieval in music, where AMT is one of the core tasks (Hawthorne et al., 2021). Evaluation of AMT solutions is very varied within the literature. As described in 2.6, the window length of when an onset is correctly predicted for may for instance vary. In addition to the limitations in regards to definitions, various implementations of evaluation may produce different results. *mir_eval* therefore offers a unified way of measuring performance in AMT systems. As Raffel et al. points out, by encouraging MIR researchers to use the same easily understandable evaluation codebase, they can ensure that different systems are being compared fairly. The specific setup of the evaluation metrics and its parameters used in this architecture is further presented in section 6.2.4.

5.4. Summary

The architecture presented is a system containing three parts. First, an AudioTransformer which utilizes DawDreamer to application of generators and/or effectors to MIDI and/or audio. The AudioTransformer (figure 5.2) includes standardization of a pipeline format for processing and tracking of operations. Secondly, the DatasetPreparer (figure 5.3)

takes the processed audio and stores it in dataset files in the format of TFRecords. Finally, the PredictorModel (figure 5.4) contains the SOTA AMT called MT3, which uses the data stored the TFRecords to train a model and later evaluate how well the new datasets perform.

6. Experiments and Results

This chapter covers the experiments conducted in this thesis. The experiments build on the goal and the research questions first introduced in section 1.2. The goal for this thesis is as presented, to examine how musical data generated through digital audio plugins affect performance of State-of-the-art (SOTA) Automatic Music Transcription (AMT) solutions. The experiments are therefore configured to investigate how this can be done in the best way possible. This chapter first presents the experimental plan, where each experiment is described in detail. The chapter further discusses the experimental setup used in the experiments. This includes the experiment-specific setup of the architecture presented in chapter 5, the datasets used in the experiments, and the setup for the evaluation metrics used to evaluate the experiments. Finally, having described the plan and the setup for the experiments, the experiments are conducted and the results are presented and addressed. These results are further evaluated and discussed in chapter 7.

6.1. Experimental Plan

This section describes the experimental plan, where each of the planned experiments is described. The experiments are presented in a chronological order, ordered by when they are planned to be executed. The experimental plan contains one baseline experiment which acts as a reference point for the results in the other experiments. There are five other experiments, directed to provide as much information as possible to answer the thesis goal and research questions.

All experiments have the same structure. The experiments, except for the baseline experiment, will first produce a number of artefacts, or custom datasets, by augmenting four *original* datasets. The produced custom datasets can be mixtures of previously augmented datasets as well. These datasets are the MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO), GuitarSet, MusicNet and University of Rochester Multi-modal Music Performance (URMP) datasets presented in chapter 3. Their experiment-specific setup, including train/test/validation splits is further described in section 6.2.3. The custom datasets are generated using the Audio Transformer presented in chapter 5. Having produced the custom datasets, they will be used as training data for the Predictor Model, also introduced in chapter 5, and trained with the model setup further presented in section 6.2.2. Then, the trained model will be validated on the validation splits of the original datasets.

Experiment 0 - Baseline

This experiment acts as a baseline to the other experiments in this experimental plan. Therefore it does not aim to produce any novel results. The baseline experiment consists of training and validating the Predictor Model on the unedited version of the four original datasets. The baseline experiment will of course be helpful in validating that the model compiles and runs locally, but perhaps more importantly it will (hopefully) validate earlier research (Gardner et al., 2021) and thus provide a solid basis for comparison for the rest of the experiments.

Experiment 0: Train and validate all original versions of the four datasets.

Experiment 1 - Normalization of the input data

Before generating the artefacts for the remainder of the experiments, the data quality of the generation process has do be addressed. As discussed in section 2.7, it is important that the audio has the same volume across the experiments, and therefore has to be normalized. One reason is after investigating the dataset generated with audio plugins, the volume level differed from the original dataset, caused by rendering-issues with DawDreamer. Hence, the input from the generated dataset is not comparable to the clean dataset. Experiment 1 tests what normalization method to use for the remainder of the experiments. Three normalization methods were therefore tested as an initial step prior to the other experiments. The three normalization methods tested were Root Mean Square (RMS), Decibels Relative to Full Scale (dBFS) and Peak Amplitude (PA), all described in section 2.7. The methods are tested on a custom version of the GuitarSet dataset from experiment 1.2, augmented with the SerumFX effector. This experiment relates to Research Question 3 (RQ3), as it forms the basis of how to represent custom generated datasets as data.

Experiment 1: Test three normalization methods on the same custom generated dataset to see which is best.

Experiment 2 - Effectors

Experiment 2 is designed to answer Research Question 1 (RQ1), hence to investigate if effect plugins, or effectors (described in section 2.8.3), can be used to increase performance in AMT models. Experiment 1 therefore attempts to apply effects using effectors to the audio files of all of the datasets to create augmented custom datasets. Two different effectors, RC-20 and SerumFX, will be used to augment the audio files of the four presented datasets. These effectors have various traits, described in section 6.2.1. A handpicked selection of effects to apply to audio data has been chosen from these two effectors, based on the guidelines presented in section 6.2.1. The produced datasets are then trained and validated individually.

Experiment 2.1: Train and validate all four datasets modified with the RC-20 effector plugin.

Experiment 2.2: Train and validate all four datasets modified with the SerumFX effector plugin.

Experiment 3 - Generators

Experiment 3 is designed to answer Research Question 2 (RQ2), which questions if generator plugins, or generators (described in section 2.8.3) can be used to augment existing AMT datasets to increase performance. The experiment is therefore attempting to apply generators to the four datasets, and see how they impact transcription performance. That is, to re-render the Musical Instrument Digital Interface (MIDI) files from the original datasets in new styles and sounds. This experiment applies two different generators, Serum and Kontakt, to the datasets, both with various traits discussed in section 6.2.1. A number of presets are handpicked for the generators as well, with the guidelines listed in section 6.2.1. This experiment is therefore similar to experiment 1, but with the use of generators instead of effectors.

Experiment 3.1: Train and validate all four datasets re-rendered with the Serum generator plugin.

Experiment 3.2: Train and validate all four datasets re-rendered with the Kontakt generator plugin.

Experiment 4 - Mixing Generated Datasets With Each Other

Gardner et al. (2021) experienced a gain in performance when training their model using a mix of all six datasets their paper addresses. In experiment 4, this will be done the same way for the datasets generated through effectors, and for the datasets generated through generators, respectively. The mixing will follow the same sampling probability as in Gardner et al., described in section 5.3.2. As this thesis includes only four of the datasets, the formula makes sure that the probability of sampling from a low-resource dataset (e.g. URMP) is higher. The experiments will be done for all of the successful custom datasets from experiments 2 and 3. This experiment is therefore testing the ability of both the effector and the generator, hence directed to answer both RQ1 and RQ2. In addition, mixing of several datasets lay ground to what is important to answer RQ3.

Since it was uncovered in the results from experiment 3 (section 6.3), that Serum does not work, the experiments form here on forward does not consider Serum, although originally planned.

Experiment 4.0 (Baseline): Train a mixture of all original datasets validate on each of the four original datasets.

Experiment 4.1: Train a mixture of all datasets generated by the RC-20 effector from experiment 2.1, and validate on each of the four original datasets.

Experiment 4.2: Train a mixture of all datasets generated by the SerumFX effector from experiment 2.2, and validate on each of the four original datasets.

Experiment 4.3: Train a mixture of all datasets generated by Kontakt generator from experiment 3.2, and validate on each of the four original datasets.

Experiment 5 - Mixing Generated Datasets With Baseline

Although it is interesting to see the results of applying effectors to the whole dataset, data augmentation are thought to assist datasets to produce better results and not replace them. Therefore it would be interesting to see if the augmented datasets from experiment 2 (effector, for RQ1) and 3 (generator, for RQ2) can be used as a mixture with the original dataset, to act as a supplement rather than a replacement. It would also be interesting to see how a mixture of the custom and the original datasets in various mixing percentages performs.

Taking the URMP dataset as a base, this experiment will gradually mix the data from the best-performing custom datasets from experiment 2 (effector) and experiment 3 (generator). The mixture percentages to be experimented with are 25%, 50% and 75%. The 0% mixture is the result from the baseline experiment, and the 100% mixture is the result from experiment 2 and 3.

Experiment 5.1: Train and validate a mixture of a original URMP dataset with the best-performing effector dataset from experiment 2 in steps of 25%, 50%, and 75%.

Experiment 5.2: Train and validate a mixture of a original URMP dataset with the best-performing generator dataset from experiment 3 in steps of 25%, 50%, and 75%.

Experiment 6 - Using both the Generated Datasets and the Baseline Dataset

Experiment 6 is similar to experiment 5, where the generated datasets are supposed to be mixed together. However, instead of mixing the datasets, they are added to each other in its entirety. Hence, the number of dataset entries are doubled. Experiment 6 is also performed from the best-performing custom datasets from experiment 2 (effector) and experiment 3 (generator).

Experiment 6.1: Train and validate a dataset containing both the original URMP dataset and the best-performing effector dataset from experiment 2

Experiment 6.2: Train and validate a dataset containing both the original URMP dataset and the best-performing generator dataset from experiment 3

Experiment 7 - Generate Datasets With Easier Effects

Augmenting the datasets with the methods presented in experiment 2 and 3 creates more diverse data for the architecture to learn. However, without testing the various presets for the audio plugins individually, it is hard to know what preset works well with what settings, further addressed in chapter 7. For effectors, one possibility is that the current presets augment the signal too much. It would be possible to only send part of the original signal through the effector, a technique called parallel processing. While each track in the raw datasets are 100% of the original signal and 0% of the effector rendered signal, the datasets from experiment 2 are 0% of the original signal and 100% of the effector rendered signal. However, it is possible to create new audio by combining for instance 50% of the original signal, with 50% of the effector rendered signal by using a built-in mix knob in the effectors. This would be equivalent to parallel processing 50% of the signal through an effector.

In this experiment, various parallel processing strengths are tested, to see if application of easier effects performs better than the results from experiment 2. This experiment take use of the GuitarSet dataset augmented with the RC-20 effector. The experiment contributes to answer RQ1, as the technique is solely for effectors.

Experiment 7: Train and validate the model on the GuitarSet dataset augmented with the RC-20 effector in steps of 25%, 50%, and 75%.

6.2. Experimental Setup

This section presents the experimental setup used for running all of the experiments. The architecture for generating custom datasets for the experiments, the Audio Transformer, and the model for training and validating the system are both presented in chapter 5. This presentation does not include the exact configuration setup of the AudioTransformer, including used effectors, generators, presets and more. Thus, this is further detailed in this section. Nor does the presentation in chapter 5 include details on the internal configurations for the Multi-Task Multitrack Music Transcription (MT3) model, which is the core of what is used to train and validate the system. The configuration and setup of this model are thus described in this section. Furthermore, this section presents the setup for the datasets used in the experiments. This setup consists of describing the train/test/validation splits used, in addition to how these datasets are prepared for use in the proposed system. Finally, this section presents the evaluation metrics used to evaluate the models and its setup.

6.2.1. Audio Plugin Setup

As described in section 1.2, the main concern of this thesis is examining the effect of applying audio plugins to AMT datasets. The extent of success that this will have, greatly rely on the qualities and capabilities of the plugins that are used. The datasets

used for AMT, described in chapter 3, already contains high-quality musical renderings of varying texture, dynamics and complexity. Thus, it is important that the new renderings, using audio plugins, try to elevate the variation already found in the datasets. To best meet this requirement, plugins and presets (ready-made internal states of a plugin, see also 2.8) have been handpicked to best represent a musical variety of high degree.

Selected Effectors

As effectors, both SerumFX and RC-20 (see section 2.8) have been used for rendering audio through effectors. These two have primarily been chosen because while they both allow for varied effect application, they are juxtaposed the sounds the create. While RC-20 tries to emulate old analog studio equipment and helps add saturation and warmth to audio, SerumFX has a more digital feel and is more commonly used for completely digital instruments, like EDM synths or pads.

A few general guidelines have been adopted when selecting presets for effectors, in the hope of keeping resemblance between the MIDI and the rendered audio:

- No pitch altercation, as we want the output audio to still correlate with the original corresponding MIDI after an effect operation.
- No time altercation effects, as this could possible change the pitch of the notes (depending on how the effects are used) and their duration, making them not align with the original corresponding MIDI.
- No heavy echos or other effects that drastically change the characteristic of the sound, beyond that notated in the original corresponding MIDI.

For rendering, all stock presets for RC-20 have been chosen, except those violating the guidelines outlined above. Additionally, added presets have been retrieved from online sample libraries such as Splice [1]. This adds up to a total of 60 different presets.

For SerumFX, a total of roughly 60 presets have been chosen, spanning heavy digital distortion, to drone-like electronic effects and spacious reverb.

A full list of all the effector-presets used for RC-20 and SerumFX can be found in appendices C, and D, respectively.

Selected Generators

Kontakt and Serum (see section 2.8) have been used as generators to render MIDI for the experiments. Like with the selected effectors (see section 6.2.1), these two have been chosen because they represent two juxtaposed sides of the digital instrument domain. While SerumFX is a digital synthesizers, generating sounds through wave-tables and other

[1] https://www.splice.com

forms of digital synthesis, Kontakt works by rendering samples of real-world instruments, making them playable in the digital domain.

As with effectors, a few general guidelines have been adopted when selecting presets for generators:

- No arpeggiators have been used, as this would have spawned spontaneous notes that are nowhere to be found in the corresponding MIDI.
- No heavy echos, attacks or pitch-altercation that would change the audio drastically beyond the original MIDI has been used.
- Care have been taken to turn off special settings like *mono*, *glide* and voicing limitations (number of simultaneous notes) to ensure that the MIDI is rendered as it should.

For Serum, presets have been selected from the default library and from various packages downloaded from Splice, an online sample library [2]. The total selection adds up to over 400 digital synths, pads, leads, basses and more.

For Kontakt, presets have been selected from the Komplete Library ([3]), in total spanning over 200 basses, synthesizers, brass-instruments, guitars and more. As with effectors, a full list of the presets used for both Serum and Kontakt can be found in appendix E, and F, respectively.

6.2.2. MT3 Setup

All experiments (except for baseline) consist of creating and utilizing new, custom datasets in one way or another. In order to evaluate these new datasets, the experiments will as presented in chapter 5 take use of the MT3 architecture proposed by Gardner et al. (2021). The experiments build upon the same input and output representation as described in section 4.3. However, as opposed to MT3, all experiments in this thesis are ran without paying respect to what instruments are active at the time, hence removing the instrument token from the output vocabulary. This is described as the *flat* program granularity setting, as opposed to the *full* granularity, where each instrument class is assigned an instrument value in range of 1-128. Choosing the flat granularity is done because this thesis investigates how data augmentation as a whole can contribute to AMT datasets, and not how it can increase SOTA individual instrument transcription or the separation of audio. Although possible to toggle on, this setup removes the velocity token of the input dictionary, as MT3 has done, as velocity estimation is not a part of the scope of this thesis.

The input is represented by input spectrograms with the same parameters as MT3. The spectrograms are generated with an Fast Fourier Transform (FFT) length of 2048 samples

[2] https://www.splice.com
[3] https://www.native-instruments.com/en/products/komplete/bundles/komplete-13/

with a hop width of 128 (see section 2.3.2). These were calculated from audio of 16kHz sample rate. The spectrograms were also scaled to 512 mel bins on a log-scale magnitude (see section 2.4.2). The output vocabulary contains the following four tokens:

- **Note**: A note-on or note-off event for one of the 128 pitches.
- **Time**: The absolute time location (a 10ms window) within a segment.
- **EOS**: Indicating the end of a sequence
- **End Tie Section**: Declares which notes are already active at the start of a frame

To match the results presented in Gardner et al. (2021), each run is trained on 2^{19}, or 524288 epochs. Gardner et al. do however train all mixture models, models in which is a mixture of two or more datasets, with 1 million epochs. The mixture models in the experiments of this thesis retain training on 2^{19} epochs, to have a comparable results to the non-mixture models.

6.2.3. Dataset Setup

The experiments will be run on four of the datasets used in Gardner et al. (2021): MAESTRO V3, GuitarSet, MusicNet and URMP, all described in chapter 3. This section describes the setup done to use these datasets. These include the train and validation splits for all datasets. A summary of the number of songs and the total duration of each dataset is shown in table 6.1. Due to memory and space constraints, Synthesized Lakh (SLAKH2100) and Cerebeus4 were not used, which were the two left out datasets from Gardner et al. (2021).

Table 6.1.: Overview of the data used in the experiments

Dataset	Train		Validation	
	Songs	Hours	Songs	Hours
MAESTRO	161	25.1	137	19.5
GuitarSet	240	2.0	120	1.1
MusicNet	300	31.0	15	1.6
URMP	35	1.1	9	0.2

MAESTRO V3

The standard train/test/validation split for MAESTRO sorts 962, 177 and 137 performances into each split, respectively. The standard split for MAESTRO also ensures that no composition (which could be performed by multiple performers) appears in the same split. This warrants that the inference score from an AMT model trained on MAESTRO actually measures generalizability.

MAESTRO is a very large dataset, taking up 120GB of storage space. The training split contains 100GB of this data. Caching of this split is not possible on the hardware available, due to exceedance of the maximum memory (further discussed in section 7.2.3). Therefore, the experiments using the MAESTRO dataset in this thesis only take use of one sixth of the complete train split for training. This number is chosen solely because caching of one fifth also exceeded the maximum memory, while one sixth did not. The tracks included in the new train split is chosen at random, as this would create the most diverse dataset without analyzing it in detail. This resulted in 161 tracks, which is listed in its completeness of appendix B. The original validation split is unaltered.

GuitarSet

Similar to Gardner et al., the GuitarSet dataset (section 3.2) has the following train/validation split; for every genre across all performers, the first two progressions are used for training, while the last progression is used for validation.

The annotations of GuitarSet are in JSON Annotated Music Specification (JAMS) format. This is converted into MIDI by using a simple Python script, which is bundled together with the rest of the architecture presented in chapter 5, as a utility function.

MusicNet

The MusicNet dataset is bundled with MIDI files with the note annotations. Some of these MIDIs was corrupted. Thus, the MIDI annotations used were rather downloaded from a discussion board in a Google Magenta forum[4].

Even though MusicNet contains a standard train/test split, a new split is established by Gardner et al. to buff up the test set and to introduce a validation set. However, to ensure comparability to the solution from Gardner et al. (2021) and to get the new validation set, this thesis utilize the exact same split as them. The splits were retrieved from a post from one of the authors in the MT3 GitHub repository[5]. The split used is as listed below, where all other track IDs not mentioned is used as the train split (The test split is not used, but is included as the audio files are left out of the train split):

validation: 1733, 1765, 1790, 1818, 2160, 2198, 2289, 2300, 2308, 2315, 2336, 2466, 2477, 2504, 2611

test: 1729, 1776, 1813, 1893, 2118, 2186, 2296, 2431, 2432, 2487, 2497, 2501, 2507, 2537, 2621

[4]https://groups.google.com/a/tensorflow.org/g/magenta-discuss/c/J-bzp0QTk34
[5]https://github.com/magenta/mt3/issues/29

URMP

The MIDI annotations in the URMP dataset do not correspond to its original audio. At the beginning of each audio file, there generally is a small delay before the instruments start to play. The MIDI however starts immediately, and is therefore not aligned with the audio. This might come as a result of manual annotation of the notes. In an attempt to align the beginning of the MIDI to the beginning of the audio, the MIDI lasts longer than the audio files. This is because the temporal information is not correctly annotated either.

Gardner et al. (2021) has uploaded the dataset (TFRecord) they used in their experiments. After inspecting this data, they seem to have solved the issue with the original paper. The method of converting the dataset is not specified in their paper. Therefore, all the note annotations were retrieved from the TFRecord. These annotations were stored in the NoteSequence format (section 2.4.5), and had to be converted to MIDI first. This is done with the SequenceTools module, presented in section 5.2.2

A train/validation split for URMP is established by Gardner et al. and used in the related experiments of this thesis. Piece 1, 2, 12, 13, 24, 25, 31, 38, 39 constitute the validation set, while the remaining 35 pieces form the training data.

6.2.4. Evaluation Metrics

The performance of the experiments presented will be evaluated in comparison to the to the work in Gardner et al. (2021). Hence the evaluation of the experiments in this thesis will be done with the same metrics addressed in this paper. Evaluation is therefore performed with the three metrics from their paper, namely *Frame*, *Onset*, and *Onset+Offset* metrics, earlier presented in section 2.6. The results are mainly validated with the F1 score for these metrics, but some experiments contains the precision and recall scores in addition. As noted in section 2.6, all of these metrics require a tolerance window to decide when a prediction is counted as a hit. In most of the related work for AMT solutions listed in chapter 4, including the work from Gardner et al., this is set to 50ms. Therefore the evaluation metrics in this thesis is set accordingly.

6.3. Experimental Results

This section presents the results for the experiments presented earlier in this chapter. The results follow in the same order as the experimental plan in section 6.1.

6.3.1. Results from Experiment 0

Experiment 0 is meant to both act as a baseline for the rest of the experiments in the experimental plan, and to validate the results from the MT3 architecture presented in Gardner et al. (2021). In addition, it validates the datasets downloaded from the internet,

as some of them contain anomalies as listed in section 6.2.3. The results are displayed in table 6.2. The table shows the MT3 score and the score from the baseline experiment, denoted as E0, over the three metrics of Frame, Onset and Onset+Offset F1.

Table 6.2.: Experiment 0 (Baseline): The obtained baseline results (E0) compared to Gardner et al. (2021) (MT3)

Dataset	Frame F1		Onset F1		Onset+Offset F1	
	MT3	E0	MT3	E0	MT3	E0
MAESTRO	0.88	0.78	0.96	0.88	0.84	0.63
GuitarSet	0.82	0.83	0.83	0.83	0.65	0.67
MusicNet	0.60	0.60	0.39	0.37	0.21	0.21
URMP	0.49	0.57	0.40	0.48	0.16	0.22

The results listed in the table are comparable to MT3. Although the MAESTRO dataset lost performance for all three metrics, it performed better than expected, due to the fact that only one sixth of the dataset was used for training. The Onset+Offset metrics suffered the most. The GuitarSet results was very similar to the MT3 results. It is notable that GuitarSet has higher performance than the baseline results for MAESTRO, when it is 12.6 times as small. Because both MAESTRO and GuitarSet are single-instrument datasets, the program granularity between flat and full should not matter (described in section 6.2.2), as the instrument token would remain constant during training. URMP and MusicNet are multi-instrument datasets, so the flat program granularity is expected to have an impact compared to MT3. From the results, MusicNet performs close to equal to its results in MT3. URMP on the other hand has an increase in performance for all metrics.

6.3.2. Results from Experiment 1

Experiment 1 tests what the best normalization method for the generated datasets are. The results from this is shown in table 6.3. From the leftmost column, three normalization methods are presented: RMS, dBFS and PA. From the table, the RMS normalization method performed the best, meaning this method is used before running all of the other experiments listed above. Note that the score for the RMS is the same as the score for experiment 1.2 for the GuitarSet dataset in table 6.4.

Table 6.3.: Experiment 1. Validation results for datasets generated with SerumFX normalized using three various methods

Type	Frame			Onset			Onset+Offset		
	P	R	F1	P	R	F1	P	R	F1
Baseline	0.83	0.82	0.83	0.83	0.83	0.83	0.67	0.67	0.67
RMS	0.83	0.79	0.81	0.82	0.80	0.81	0.64	0.63	0.63
DBFS	0.81	0.77	0.79	0.78	0.75	0.76	0.58	0.56	0.57
Peak	0.82	0.76	0.79	0.80	0.75	0.77	0.61	0.57	0.59

6.3.3. Results from experiment 2 and 3

Table 6.4 displays the results of experiment 1 and 2 separated into the metrics of Frame F1, Onset F1 and Onset+Offset F1, as set up in 6.2.4. It further maps the results from experiment 1 and 2 onto these metrics. These scores are separated by the columns MAESTRO, GuitarSet, MusicNet and URMP, which represent the dataset used for training and validation. The results are compared to the Baseline results previously presented in table 6.2.

The trend in the results is that all experiments fail to produce better results than in the baseline experiment. Of the various experiments, the effectors performed the best, where SerumFX outperformed RC-20. The results differed a maximum of 10 percent in all metrics except for in URMP, which was worse. Of the generators, Kontakt performed the best. Serum did unfortunately not work in the system at all, due to incompatibility with DawDreamer, as further described in section 7.2. Kontakt struggles, however, to keep up with the results from the effectors.

6.3.4. Results from Experiment 4

In experiment 4, mixtures of all custom dataset generated by individual audio plugins are used to evaluate each of the validation datasets. Table 6.5 displays the results from experiment 4. The table has the same metrics as displayed in table 6.4. In addition, each score contains a parenthesis containing the gain or the loss between the mixed model and its single-dataset counterpart from table 6.4.

The MAESTRO dataset performs a loss in all metrics, for all dataset mixtures. Most notably, it does so for the baseline mixture, in which all the other validation sets experienced an increase. While having a similar loss in Frame F1 for all mixtures in comparison to the baseline mixture, the Onset F1 and Onset+Offset F1 had a bigger decrease.

The GuitarSet dataset experienced similar transcription scores compared to the single-dataset experiments on the Frame F1 metric. On Onset and Onset+Offset F1, both

Table 6.4.: Results from experiment 2 and 3

Model	MAESTRO	GuitarSet	MusicNet	URMP
Frame F1				
E0 (Baseline)	0.78	0.83	0.60	0.57
E2.1 (RC-20)	0.74	0.79	0.57	0.48
E2.2 (SerumFX)	0.75	0.81	0.57	0.49
E3.1 (Serum)	0.03	0.01	0.05	0.06
E3.2 (Kontakt)	0.59	0.66	0.43	0.42
Onset F1				
E0 (Baseline)	0.88	0.83	0.37	0.48
E2.1 (RC-20)	0.85	0.76	0.32	0.21
E2.2 (SerumFX)	0.86	0.81	0.33	0.28
E3.1 (Serum)	0.02	0.01	0.02	0.02
E3.2 (Kontakt)	0.59	0.68	0.26	0.30
Onset+Offset F1				
E0 (Baseline)	0.63	0.67	0.21	0.22
E2.1 (RC-20)	0.55	0.58	0.17	0.05
E2.2 (SerumFX)	0.56	0.63	0.18	0.08
E3.1 (Serum)	0.00	0.01	0.01	0.01
E3.2 (Kontakt)	0.27	0.32	0.12	0.10

RC-20 and SerumFX became worse. Kontakt on the other hand had an increase for all three metrics. On the Onset F1, Kontakt performed 0.69 vs. 0.70 on the RC-20, while the scores were 0.68 in E3.2 (Kontakt) and 0.76 in E2.1. This differs from the pattern of MAESTRO, where all metrics sank similarly. Onset+Offset F1 with the Kontakt mixture had a larger increase than its baseline results.

MusicNet were slightly worse off on the Frame F1 metric for all mixtures. On Onset and Onset+Offset F1, it increased its performance in both the Baseline and the SerumFX mixture. RC-20 was slightly reduced in these metrics. Kontakt had a significant reduction in Onset+Offset F1.

URMP had an increase in all metrics, for all mixtures. The custom mixtures made with effectors and generators had all a larger increase than its baseline. However, none of the non-baseline mixture scores increased above the baseline scores. Over the mixtures, RC-20 increased the most, before SerumFX and Kontakt.

In addition to the results provided above, another table is given in table A.3 of appendix A. This table displays the same results as in table 6.5, however with the change from

Table 6.5.: Results from experiment 4. Each score contains the gain/loss from its single-dataset counterpart score shown in table 6.4

Model	MAESTRO	GuitarSet	MusicNet	URMP
		Frame F1		
E4.0 (Baseline)	0.69(-10.8%)	0.84(+1.5%)	0.59(-1.4%)	0.77(+33.7%)
E4.1 (RC-20)	0.63(-14.2%)	0.80(+1.0%)	0.55(-3.0%)	0.72(+50.0%)
E4.2 (SerumFX)	0.66(-11.9%)	0.80(-0.4%)	0.57(-1.1%)	0.72(+47.3%)
E4.3 (Kontakt)	0.53(-10.2%)	0.67(+1.3%)	0.41(-3.8%)	0.53(+28.0%)
		Onset F1		
E4.0 (Baseline)	0.78(-11.6%)	0.85(+2.5%)	0.39(+6.6%)	0.64(+34.0%)
E4.1 (RC-20)	0.56(-33.5%)	0.70(-7.4%)	0.31(-3.3%)	0.42(+98.5%)
E4.2 (SerumFX)	0.67(-22.6%)	0.77(-4.4%)	0.34(+3.9%)	0.46(+67.0%)
E4.3 (Kontakt)	0.42(-27.8%)	0.69(+2.3%)	0.25(-1.9%)	0.41(+37.4%)
		Onset+Offset F1		
E4.0 (Baseline)	0.45(-29.2%)	0.68(+2.5%)	0.23(+13.5%)	0.40(+79.6%)
E4.1 (RC-20)	0.27(-50.5%)	0.51(-11.5%)	0.17(-1.1%)	0.25(+353.7%)
E4.2 (SerumFX)	0.35(-38.0%)	0.56(-11.4%)	0.19(+7.5%)	0.26(+224.2%)
E4.3 (Kontakt)	0.19(-27.5%)	0.33(+4.0%)	0.10(-15.5%)	0.19(+89.9%)

the baseline E0 experiments, rather than the change from the corresponding E2 or E3 results. A loss in performance from the baseline is registered for MAESTRO, GuitarSet, and MusicNet. For URMP, however, the mixture performed better in the Frame F1 and the Onset+Offset F1 metric for both the RC-20- and SerumFX-rendered data.

6.3.5. Results from Experiment 5

Experiment 5 investigates if mixtures of the baseline dataset with the generated datasets, using the best-performing effector (5.1) and generator (5.2) from experiments 2 and 3.

The results from experiment 5.1 is shown in table 6.6. The table is divided into five runs containing mixtures of the URMP E0 (baseline) and URMP E1.2 (generated by the SerumFX effector) from 0%-100%. Hence the 0% result is the same result as E0 in table 6.4. Similarly, the 100% result is the same as the result for E2.2 in the same table. This table displays results from all five runs containing the precision, recall and F1 score for all three metrics of Frame, Onset and Onset+Offset.

Moreover, table 6.7 shows the results from the mixture between URMP E0 and URMP E3.2 (generated with the Kontakt generator). This table has the same fields as described

Table 6.6.: Experiment 5.1: Mixture of URMP dataset and the custom dataset from experiment 2.2 with various percentages of the custom dataset

Mixture	Frame			Onset			Onset+Offset		
	P	R	F1	P	R	F1	P	R	F1
0%	0.56	0.60	0.57	0.48	0.50	0.48	0.22	0.23	0.22
25%	0.55	0.56	0.55	0.46	0.46	0.45	0.23	0.22	0.22
50%	0.55	0.54	0.54	0.41	0.42	0.40	0.17	0.17	0.17
75%	0.54	0.52	0.53	0.36	0.36	0.35	0.14	0.14	0.14
100%	0.49	0.50	0.49	0.27	0.30	0.28	0.08	0.08	0.08

in the paragraph above for table 6.6.

Table 6.7.: Experiment 5.2: Mixture of URMP dataset and the custom dataset from experiment 3.2. with various percentages of the custom dataset

Mixture	Frame			Onset			Onset+Offset		
	P	R	F1	P	R	F1	P	R	F1
0%	0.56	0.60	0.57	0.48	0.50	0.48	0.22	0.23	0.22
25%	0.55	0.56	0.55	0.45	0.44	0.43	0.23	0.23	0.22
50%	0.55	0.54	0.54	0.42	0.44	0.42	0.20	0.21	0.20
75%	0.53	0.53	0.52	0.39	0.43	0.40	0.17	0.19	0.18
100%	0.41	0.43	0.42	0.29	0.34	0.30	0.10	0.11	0.10

A graph comparing the results between the effector (E5.1, SerumFX), and the generator(E5.2, Kontakt) is shown in figure 6.1. The graph shows the Frame, Onset and Onset+Offset F1 scores obtained after the gradually mixing the data from 0% to 100%. The effector scores are displayed in blue, while the generator are displayed in red.

Both the effector- and generator-generated datasets suffers a gradual loss in performance for all metrics between 0% and 100%. The loss follows a linear decrease for all steps until the 75% mixture, while 100% generated data drops drastically. The drops in results for the F1 score are between 4% and 7% for the effector, while the generator drops between 8%-10%. Overall, the effector performs better in the Frame metric, while the generator performs better in both the Onset and Onset+Offset metric, which is consistent with the results from experiment 2 and 3.

6.3.6. Results from Experiment 6

Experiment 6 tests how well addition of a custom dataset to the baseline dataset perform. This effectively doubles the dataset size. The baseline dataset used in this experiment was the URMP dataset. The custom datasets used was the datasets produced from with

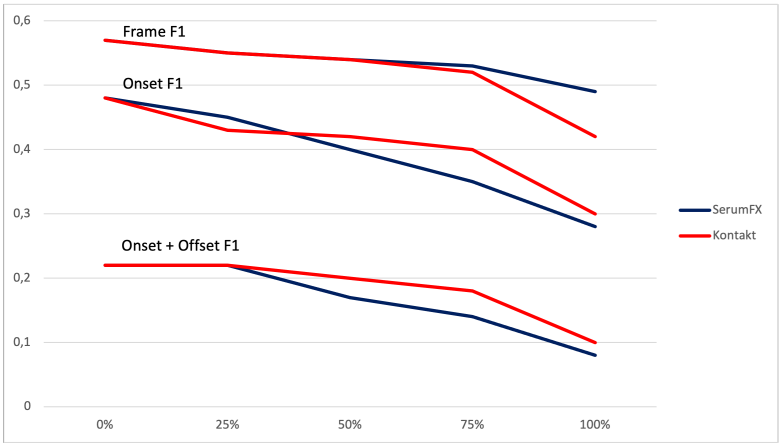

Figure 6.1.: Comparison between performance of the SerumFX effector (blue) and the Kontakt generator (red) when gradually mixing the original datasets into the augmented datasets

the best-performing effector in experiment 2, SerumFX (E3.2), and the best-performing generator in experiment 3, Kontakt (E3.1), for the URMP dataset. Runs with the mixtures of the respective datasets forms experiment 6.1 and 6.2.

Table 6.8 displays the results from experiment 6. The results are shown with Precision, Recall and F1 score for the Frame, Onset, and Onset+Offset metrics.

Table 6.8.: Experiment 6: Addition of a complete custom dataset to the baseline dataset

Type	Frame			Onset			Onset+Offset		
	P	R	F1	P	R	F1	P	R	F1
Baseline	0.56	0.60	0.57	0.48	0.50	0.48	0.22	0.23	0.22
E6.1 (SerumFX)	0.54	0.54	0.53	0.39	0.41	0.39	0.17	0.17	0.16
E6.2 (Kontakt)	0.54	0.53	0.53	0.42	0.43	0.42	0.20	0.20	0.20

The results here show that both the SerumFX (E6.1) and Kontakt (E6.2) additions performs quite similarly. Kontakt performs slightly better than SerumFX in Onset and Onset+Offset metrics. The correlation is very similar to the scores from experiment 5, in the experiments mixed with 50% of the data from the custom datasets. The score for SerumFX in E5.1 is similar to the scores in E6.1, and the score for MAESTRO in E5.2 is similar to the scores in E6.2. The addition of data does not seem to have an effect in comparison to mixing it.

6.3.7. Results from Experiment 7

In experiment 7, various strengths of the effector is applied to the GuitarSet dataset to see if easier augmentation can lead to better predictions. The results are displayed in table 6.9. The table contain results from 0% in strength to 100% in strength. The 0% result is the same as the baseline, while the 100% is the same as experiment 2.1 for GuitarSet. After first running the strength percentages from the experimental plan (25%, 50%, 75%), the results were best at 25%, and were promising. Therefore, to see if the results converged at an easier strength, a run with data augmented with 12.5% in strength was also done.

Table 6.9.: Experiment 7: Application of various strengths for the RC-20 effector on the GuitarSet dataset

Strength	Frame			Onset			Onset+Offset		
	P	R	F1	P	R	F1	P	R	F1
0%	0.83	0.82	0.83	0.83	0.83	0.83	0.67	0.67	0.67
12.5%	0.85	0.78	0.81	0.86	0.80	0.83	0.69	0.64	0.66
25%	0.85	0.79	0.81	0.86	0.79	0.82	0.69	0.63	0.66
50%	0.84	0.77	0.80	0.86	0.78	0.82	0.67	0.62	0.64
75%	0.83	0.77	0.80	0.84	0.77	0.80	0.66	0.60	0.63
100%	0.82	0.77	0.79	0.79	0.74	0.76	0.60	0.56	0.58

The results show that the application of effects decrease performance the system in terms of F1 score. Similar to experiment 5, this score decreases linearly to 75%, before dropping further at 100%. However, the step between 75% and 100% was not that drastic as in experiment 5. In scores between 12.5% and 50%, the precision has increased from the baseline experiment all three metrics, but comes at a cost of the recall. This means the system is better to leave out False Positives (FPs), but finds less of the True Positives (TPs). Although the results did not effectively increase performance, there is a takeaway that the system is not that worse with slight augmentation.

7. Evaluation and Discussion

This chapter will further examine the results presented in section 6.3 and assess how well the presented results underpin the thesis goal and research questions presented in section 1.2. A thorough evaluation will be given, analyzing what went wrong and what went right. Furthermore, the discussion section will highlight merits and limitations of the final system, relating these to the presented results and the fulfilment of the thesis goal as well.

7.1. Evaluation

Overall, the system presented in chapter 5 is very capable of rendering musical data (audio and MIDI) through audio plugins, both with generators and effectors (see section 2.8). Yet, as the results presented in section 6.3 show, when audio rendered through the system is used as input data for an AMT model, it usually performs worse than it would have done using the original data. This is with an exception for experiment 7, which will also be discussed further in this chapter. In light of the thesis goal, *examining how musical data generated through audio plugins affect performance of state-of-the-art automatic music transcription solutions*, this calls for further analysis. Section 7.2 outlines some of the limitations related to the system, and section 8.2 discusses how some of these can be alleviated to further enhance the system. Regardless, it is still critical to understand when (and why) the current system works/does not work in order to properly address any future directions. This section will particularly present analysis of the experiments, using statistics, visual tools and further inspection of the data as a ground for discussion.

7.1.1. Overall Prediction Ability

The graphs in figure 7.1 show the average prediction scores across experiment 2 and 3 for each of datasets used in the experiments. Prediction scores are shown across all plugins and measurements. As can be seen, the general prediction ability of the system tend to decrease when trained on rendered data, more so for generators than for effectors. An interesting feature of the graphs are the error bars, highlighting the difference between the validation data with the lowest and highest score. Evidently, the gap between the predictions are huge, even for small datasets like URMP and single-instrument datasets like GuitarSet.

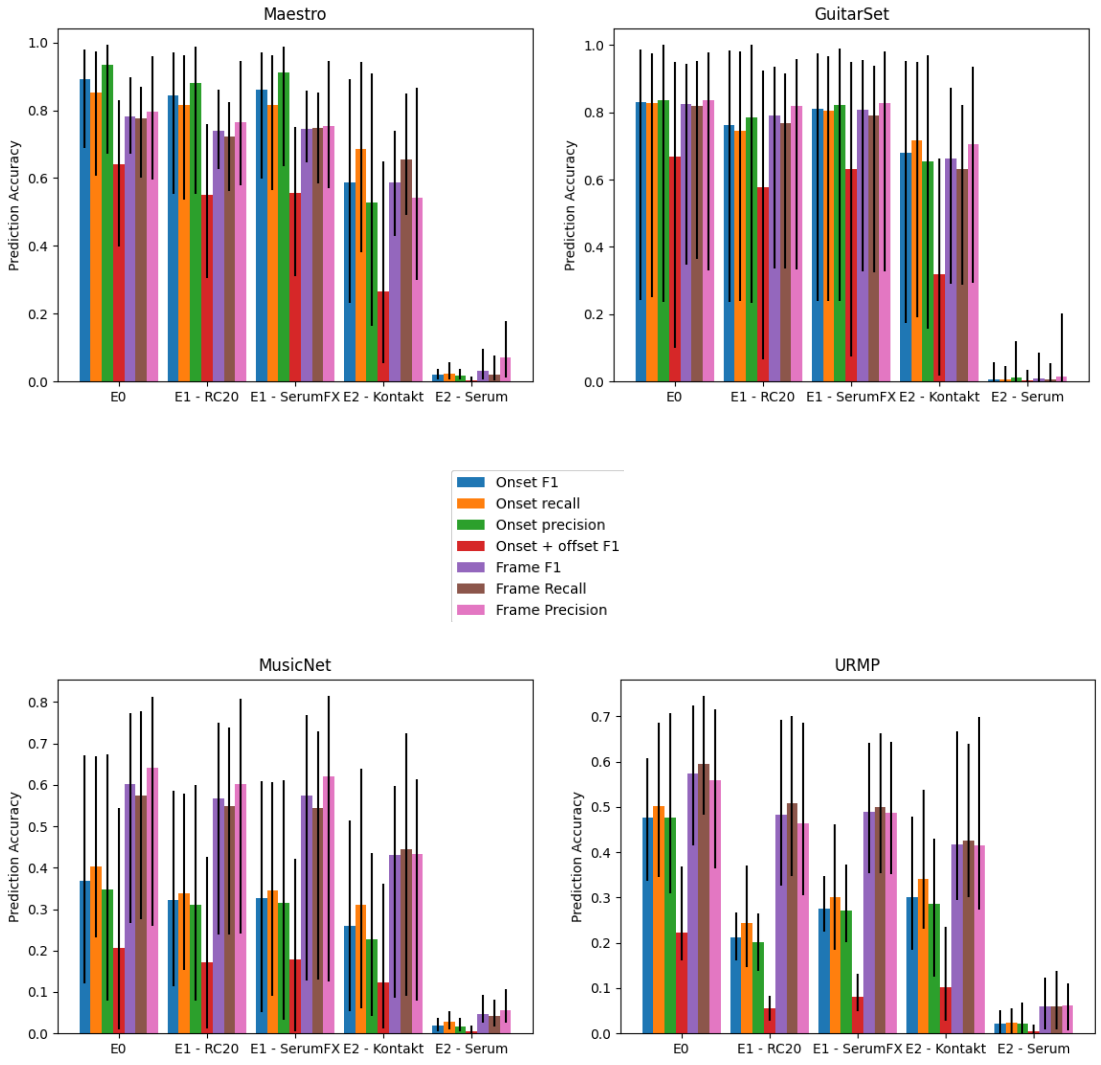

Figure 7.1.: Graphs showing the validation scores for experiment 2 and 3

Something to notice from the experiment results are the clear trend in correlation between prediction ability and instrument complexity for the datasets. The single-instrument datasets, MAESTRO and GuitarSet, spawn clearly superior prediction scores over the multi-instrument datasets, MusicNet and URMP. This applies to all tested plugins and all tested measures.

Table 7.1.: Improved/Worsened Count GuitarSet (E2 & E3). The table shows the number of files that improved/worsened their validation score in comparison to the baseline score for experiments 2 and 3.

	RC-20 (E2.1)	SerumFX (E2.2)	Kontakt (E3.1)
Onset F1	14/106	34/86	2/118
Onset Recall	11/109	42/78	13/107
Onset Precision	24/96	45/75	4/116
Onset + offset F1	5/115	30/90	0/120
Frame F1	13/107	29/91	0/120
Frame Recall	12/108	31/89	1/119
Frame Precision	43/73	50/70	7/113

While the rendered datasets generally perform worse than the baseline dataset, it should be noted that the model in fact leads to better prediction scores on some of the validation files. The table above shows the total number of positive/negative changes for GuitarSet validation files in experiment 2 and 3. GuitarSet was included because it most strongly illuminate the variability in how digital audio plugins worsen and improve prediction scores. For completeness, similar tables are included for the other datasets in appendix H.

As can be seen, the number of validation files that are worsened generally outnumber the number of improved ones, weighing down the overall prediction ability. Yet, in table 7.1, there are occurrences of several validation files with better prediction score when trained on the rendered data than on the original data. Given that the same operations that leads to these improvements do not also leave to bigger decreases elsewhere, this is a point that underpins the validity of the approach of rendering through plugins. The next question would be to investigate how exactly one ensures more operations that leads to improvement, and less that leads to decreasing validation scores, further described in section 7.1.2.

The mixture datasets in experiment 4, obtained by mixing all the custom datasets made by each of the plugins together, had varying results. The MAESTRO dataset generally decreased drastically (including baseline), while GuitarSet and MusicNet differed only slightly both positively and negatively. URMP performed a gain in score in all metrics, for all audio plugins used. The highest gain was +353.7% for Onset+Offset for the RC-20 effector. The reason that URMP performed well, is due to its small dataset size compared

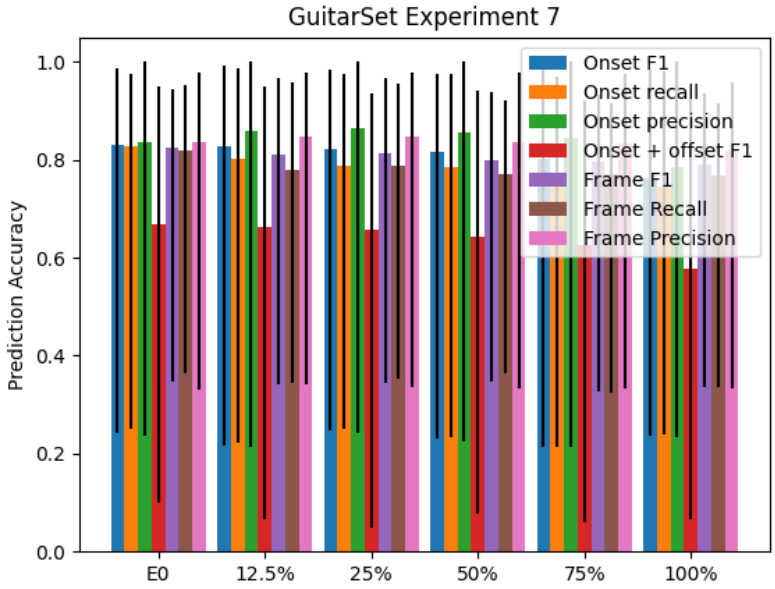

Figure 7.2.: Graph showing the validation scores across all measures for experiment 7

to the others, making it a lower-resource dataset. The results improved when using the data from the other datasets. In contrast, MAESTRO suffers, because it loose some of its performance when some of the higher-resource data gets replaced with lower-resource. There is similar behavior in MT3, where a mixture of all datasets increased performance in URMP. The performance of MAESTRO did not decrease in their model. However, the MAESTRO partition was six times as big, and also included the SLAKH2100 and Cerebeus4 datasets. This shows that low-resource datasets for one instrument class (e.g. classical instruments in URMP) can be supplied a higher-resource dataset of another instrument (piano in MAESTRO) to increase performance, even when using custom, augmented data.

Experiment 5 and 6 are very similar experiments to each other, and yields similar results. However, none of them improves beyond the raw baseline datasets. Experiment 5 covers mixing of custom datasets with the baseline datasets, by sampling various percentages from each dataset. Experiment 6 covers addition of the complete custom dataset to the baseline dataset, doubling its size. In the scores from experiment 6, the scores for the SerumFX (E6.1) is equal to the SerumFX scores at 50% mixture in experiment 5.1. Similarly, the Kontakt score (E6.2) was equal to the Kontakt scores at 50% mixture in experiment 5.2. Although these scores are achieved using a small dataset, this suggests that it does not matter if data is added to, or mixed into the baseline data to enhance transcription scores.

A graph highlighting the prediction-ability and variability from experiment 7 is shown in

figure 7.2. Contrary to the other experiments, we see that experiment 7 actually help increase the overall prediction capability for the system in terms of precision, spanning all measurements. More so, the reduction in prediction ability for the other prediction types (recall and F1) are significantly smaller when the effects are applied with a tad more carefulness. This is a big step in the right direction towards figuring out exactly how audio plugins can be optimized for use with AMT. The Mel2Mel approach mentioned earlier in this report (see section 4.4), argues that synthesizers sacrifice timbre control over synthetic timbre controls. This is used as an argument to assert that more Artificial Intelligence (AI) based approaches (like the vocoder presented) are better and more reasonable choices than synthesizers. Yet, Kim et al. put forward results that show clearly audible audio degradation and which do not add to the prediction capability of their baseline system. If nothing else, this report at least signs digital audio plugins up for the competition, and shows that they do have potential for meticulously augmenting audio for AMT purposes.

Relating to the research goals, it is possible to use effectors to increase prediction scores, as asked in RQ1. However, the increase is thus far not very high and only pertain to the precision measure and not recall and frame. For RQ2, the experiments has failed to apply generators to increase prediction scores, and tend to perform significantly worse than effectors. Generators perform better for single instrument datasets, however, which is assumed to be largely in part to the way instrument presets currently are used, replacing original instruments at random. This finding on generators is also relevant to RQ3, suggesting that live performance datasets would have an edge on datasets rendered completely using plugins. However, further evaluation and discussion will look into how this deficiency might be remediated.

7.1.2. Evaluation of Rendering Capabilities

While section 7.1.1 highlighted differences in prediction ability between the original datasets and the rendered datasets, this section further investigates and discuss *why* these occur in the first place. Analyzing and discussing these differences are crucial in order to understand where the model falls short and where it actually succeeds. In order to do this in a systematic manner, this section is further split into two subsections, *effects* and *generators*. These address use of the two types of plugins used in the project, respectively.

Figure 7.3 shows FFT spectrograms made from the original, SerumFX, and Kontakt versions of the first clip in the GuitarSet train partition, 00_BN1-129-Eb_comp. This is shown to compare how the audio visually look like.

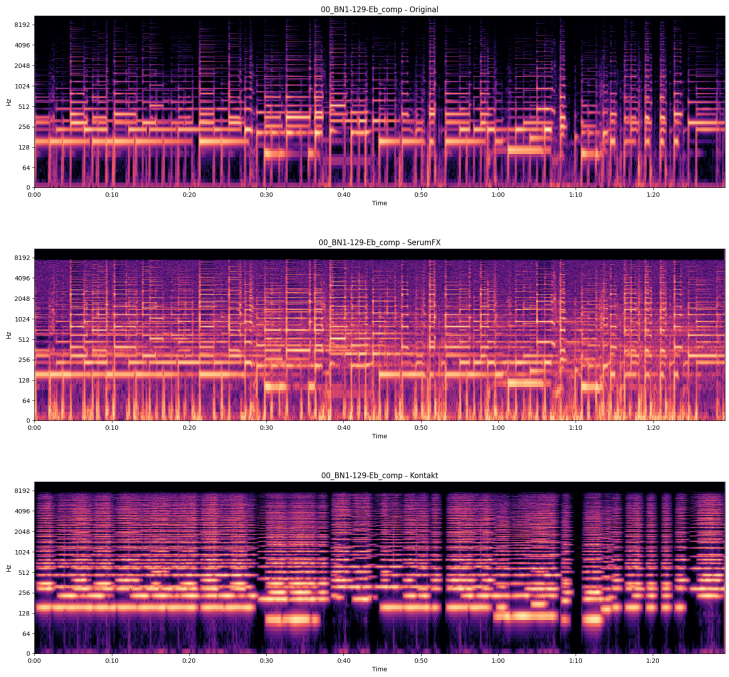

Figure 7.3.: FFT spectrograms of the original, SerumFX, and Kontakt versions of the 00_BN1-129-Eb_comp track from GuitarSet

The original and SerumFX FFTs seem similar, although SerumFX contain more powerful frequencies on the whole frequency spectrum. Effects alter the audio itself, also affecting the lower frequencies. The tracks in GuitarSet contain this kind of noise in the recordings. When rendered with Kontakt, however, these frequencies are gone, and only the frequencies belonging to the note played is present. Although this should give more crisp audio, it might loose some contextual information when for instance during evaluation. In addition, the notes from the generator recordings seems to be active longer, in comparison to their original counterpart. This can be seen by the way the spectrogram appears *streched*. The reason for this, is that MIDI annotations from GuitarSet wraps the whole note, not giving information about the decay phase in the Attack, Decay, Sustain and Release (ADSR) envelope (see section 2.1) of the note. Hence, audio made with generators using the proposed system might be active longer than its source audio, depending on the preset used and the quality of the MIDI.

Effects

Experiment 2 clearly show that rendering datasets with blind application of default presets, no matter effector, do not compare well to the baseline datasets. Yet, as the prediction variability between datasets rendered with different plugins highlight, some rendering operations have significantly worse/better impact on AMT performance than

others. The validation score for GuitarSet rendered with SerumFX is for instance consistently higher than for GuitarSet rendered with RC-20. While these differences could be owed to a multitude of reasons, singling them out and scaling their significance would require more work and experimentation (see section 8.2). Could it for instance be that some of the presets used in RC-20 are destructive to the musical data? If so, why and how is this avoided in SerumFX? Is it the simple fact that Serum has more varied capabilities? Is it owed to particular abilities SerumFX has but not RC-20? Is so, which and why? The list goes on.

Furthermore, the next question to ask is why the plugins used in experiment 2 in fact lower the validation score of the dataset. As discussed in section 4.4, McFee et al. have for instance showed that application of noise and reverb can increase the validation score of a musical dataset for AMT, tasks that both RC-20 and SerumFX should be able to handle. Both effectors also apply other types of augmentation, so it is entirely possible that these are weighing down any performance gains from noise and reverb application. However, another possibility is that it is not the sheer application of the effects that are lowering the validation scores, but also the amount applied. As experiment 7 shows, by using the RC-20 effector through parallel processing (contrary to sequential, as experiment 2 and 3), the system is actually able to increase the precision performance of the system for both onsets and offsets. Given the results from experiment 7, it appears that the golden ratio of effect application lies somewhere in between 12.5 and 25%. This may vary from effector to effector and dataset to dataset, so section 8.2 presents some ideas on how this application could be calibrated automatically.

It is suspected there are two main reasons for why the reduction in effect application helps increase the overall system prediction ability:

1. Presets are doing too much/little: The type of effects that actually work may not be applied in the correct manner, and effects that have little or no effect might be overused.

2. Presets are doing something wrong: Additionally, it remains to weed out the type of effects that may be worsening the systems prediction capabilities.

Naturally, both these points boils down to the assumptions that were taken when storing presets for effectors (outlined in section 2.8). It could for instance be that the presets are modeling effects that are not well adjusted to the validation set. Furthermore, it could be that the assumptions made are only partially right, and processing with only 25% application helps mitigate these assumptions so that they are made useful. There are even a whole range of effect-types that have not been tested for application that could also lead to increased prediction performance, for instance pitch-modulation, chorus, filters and more.

A key take for answering RQ1 is that it is generally hard to map out how plugins can be used to increase validation scores. Even though it is possible (as demonstrated), it seems difficult to find a conjunction of use-settings and assumptions that works well.

However, because of the multitude of options, the experiments from this thesis should not be viewed as conclusive. In section 8.2, suggestions of several ways that one can try to enhance the system in order to further improve the use of effectors for data rendering is presented.

Generators

As with effectors, experiment 3 shows that blind application of the presets found in the Kontakt library produce datasets leading to lower validation scores when used for training, as compared to the original baseline. There could be a manifold of reasons for why this is the case: The sounds used for rendering are not realistic enough; Performance is not well enough encapsulated in MIDI to give audio of same quality as the originals; Too few, too many, or simply the wrong presets have been used. Given the space of the digital audio domain and the success of previously rendered datasets like SLAKH2100 (see chapter 3), we suspected that the answer (if it exists) is inclined towards the latter suggestion.

For the sake of future discussion, *wrong* presets is defined as presets lowering the validation score of a dataset when used for rendering. Thus, the *correct* presets are defined as an ideal set of presets used to maximize gain in validation score. As far as we can see, there are three main reasons why the presets used in experiment 2 are wrong. These are presented below, while section 8.2 mentions how to mitigate them.

1. Wrong assumptions: Outlined in section 2.8 are a list of rules that was used when saving presets for rendering. These are assumptions (like minimal delay, reverb, etc.) that were adopted in the hope of best creating realistic audio wile still conforming with the original MIDI. Without further conformation, it is entirely possible that this list of rules could benefit from closer examination, adding and/or adjusting/removing restrictions.

2. Wrong instruments: When rendering the audio for experiment 3, a consistent set of presets (see appendix F) has been used across all datasets. Thus, there has for instance been no care taken to liken the distribution of instruments in any of the original datasets. Furthermore, all MIDI have been rendered with a random generator presets. As the results slightly suggest (see table 6.2), this is especially a problem for single-instrument datasets like GuitarSet and MAESTRO. For GuitarSet for instance, the majority portion of guitars from the original dataset are swapped out with other instruments in the rendered data. Therefore, it is not that big of a mystery that the validation score becomes lower on the rendered data, since the validation data solely consists of guitar recordings. However, another factor that probably also affects the multi-instrument datasets, is the fact that MIDI may be rendered with a completely different instrument that what it was intended to play. If a MIDI that represents a high-pitched bell is suddenly rendered with a bass, it will produce training data that is (probably) never found in the validation data.

3. Wrong amount: Strongly connected to the notion of wrong instruments, is the fact that they amount of instruments that was used for rendering could be sub-optimal. For small datasets like URMP, the number of different instruments used are far larger than in the original dataset (and thus also the validation set). For larger multi-instrument datasets like MusicNet, the number of different instruments used are closer resembling the original dataset. Still no special care have been taken to liken the distributions of the instruments across the two.

While the possible reasons stated above are simply speculative, there are some supportive points that can be observed from the experiment results. Firstly, the results from experiment 3 (compared to baseline) are worse (in relative decrease) for single instrument datasets like GuitarSet and MAESTRO. However, this trend seem to decrease relative to the size of the datasets. Yet, it strongly lends some validity to the notion of *wrong instruments*. Another point is the observation that the results from E3 are not consistent with the results from E2. For instance, for the URMP dataset, both Onset and Offset scores are better for E3 than for E2. Still however, the Frame F1 score is lower for E3. While the reasons for this could be manifold, it suggests at least two tings: The effects used in E2 may smooth out the onsets and offsets too much, making them hard to learn during training; The mix of instruments used and they way they are used in E3 (assumptions about how they should be or the amount used) are making it harder for the model to learn when and/or where a note is playing.

As discussed above, there can be a variety of reasons for why the generator-rendered datasets does not yield good prediction results. The experiments with generators should thus not be viewed as conclusive for RQ2 nor RQ3, although they suggest that its hard to increase AMT prediction scores using generator plugins. As can be seen in the spectrograms in figure 7.3, the generator-rendered data lacks a lot of lower frequency material, probably contributing to the drop in validation scores. However, section 8.2 details how this can be remedied.

7.1.3. Evaluation of Datasets

To understand where the current system works and doesn't work, it is also important to take a look at the original data. By further analyzing the original data, it is easier to understand what the model is able and unable to learn. The rendering system can then be adapted to help bridge any knowledge gaps and even out the training base, in a sense of making it easier to extract information from it.

Table 7.2.: Best & Worst Frame F1 GuitarSet (E2 and E3). The three on the top are the files with the best Frame F1 scores for a given experiment, while the bottom three are the files with the worst Frame F1 scores. The score for a given file is given in parenthesis behind its name.

	Baseline (E0)	RC-20 (E2.1)	SerumFX (E2.2)	Kontakt (E3.1)
1.	05_Funk3-98-A_solo (0.9425)	05_Rock3-148-C_solo (0.9345)	05_Rock3-148-C_solo' (0.9553)	05_Funk3-112-C#_solo (0.8738)
2.	02_SS3-84-Bb_solo (0.9380)	03_Jazz3-137-Eb_solo (0.9254)	05_Funk3-112-C#_solo (0.9486)	05_Rock3-148-C_solo' (0.8267)
3.	05_Rock3-148-C_solo (0.9349)	05_Rock3-117-Bb_solo (0.9240)	05_Funk3-98-A_solo (0.9394)	05_Funk3-98-A_solo (0.8226)
-3.	00_SS3-98-C_comp (0.6598)	04_SS3-98-C_comp (0.6150)	00_SS3-84-Bb_comp (0.6350)	01_Jazz3-150-C_comp (0.4414)
-2.	02_Funk3-112-C#_solo (0.6597)	00_SS3-84-Bb_comp (0.5859)	01_Rock3-148-C_comp (0.6349)	05_Jazz3-150-C_comp (0.4325)
-1.	04_BN3-154-E_comp (0.3463)	04_BN3-154-E_comp (0.3351)	04_BN3-154-E_comp (0.3267)	04_BN3-154-E_comp (0.2904)

Table 7.2 show the three best and three worst predictions for each plugin on GuitarSet in experiment 2 and 3 using the Frame F1 measure. As can be seen, there are consistently large variations in the predictions, but some patterns can be found in relation to what validation files tend to get smaller and larger prediction scores.

For GuitarSet, for instance, we can identify that songs marked with *solo* tend to get the best prediction scores, while songs marked with *comp* tend to get the worst. For clarity, those marked with *solo* are simply solos, melodic passages of guitar. Those marked with *comp* are recordings where the performer also has layed down backing chords in addition to the solo. This is interesting feedback that can be used to further refine the way future datasets are rendered. For instance, if the model is struggling with identifying chords, it would be better to use instruments that lend themselves better to chords, like piano and guitar, instead of say a bass or a pad. Additionally, one could even begin experimenting with generators that automatically add chords to sequences, although this would require further manipulation of the MIDI (further discussed in section 8.2). While similar tables to 7.2 was created for the other datasets as well (MAESTRO, MusicNet, and URMP), no interesting patterns was identified. Yet, they are included in appendix I for completeness.

The original datasets themselves have some specific characteristics to them. The first thing to notice is the size and quality of the validation splits. MAESTRO contain 19.5 hours of validation data. GuitarSet contain only 1.1 hours, MusicNet 1.6 and URMP 0.2 hours. How diverse these validation sets can be questioned. Reading from the baseline experiment, MAESTRO scored 5 percent lower in both the Frame and the Onset+Offset F1 metric than GuitarSet, although MAESTRO has 12.6 times more train data. This quantity is after the splitting of the train split to one sixth of its original size. However when trained on the full dataset, as shown in Gardner et al. (2021), MAESTRO outperforms GuitarSet. The reason for this is possibly be that the GuitarSet validation split be less complex than the one in MAESTRO. The GuitarSet validation split does still contain 120 unique songs, making it more diverse as opposed to MusicNet and URMP, with 15 and 9 songs respectively. By having so little validation data, it is more difficult to see how new scores affect performance.

Although the takeaways relating to datasets outlined above are important for both RQ1 and RQ2, for instance they can help identify what effector and generator presets should be used, they are perhaps even more important for RQ3. Understanding the track- and instrument-distribution of existing datasets could be of great help when creating new ones. Measures could be taken to ensure that existing weak-points are not copied onto newly rendered datasets. For instance, given table 7.2, it is advisable that any new datasets have more chorded training tracks than what GuitarSet has. Moreover, considering the reflections on dataset sizes given above, it should be noted that any rendered dataset in relation to RQ3 should have a sizable and varied validation dataset. This would make it easier to see how new features affect performance, in addition to giving more precise pinpoints on weak and strong points from a prediction model.

7.2. Discussion

Following the experimental results from section 6.3 and the evaluations in 7.1, this section further discuss the merits and limitations of the work presented in this thesis. This includes reflections on internal parts of the architecture presented in chapter 5, and the system as a whole. Sequentially, these discussions are presented in three subsections; One relating to the *generation* of audio, one to the *prediction* quality of the system, and one to *other* limitations, such as constraints introduced by third-party libraries and hardware.

7.2.1. Audio Generation

The thesis goal presupposes a system that is capable of rendering large amounts of musical data through audio plugins. The AudioTransformer presented in section 5.1 is created to solve this task. It does so by utilizing the DawDreamer library presented in section 4.4.2 along with other concepts like pipelines, parsers and builders. In short, pipelines are used to effectively store, log and control the operations that the system should apply. This allows for processing of both single- and multi-stem (multi-instrument) tracks with various configurations. The pipelines are applicable to both generators and/or effect plugins. With regular expressions, the system can easily scan directories for files to use and order these in relation to each other (for instance MIDI with related audio), eliminating the need for any manual file handling during rendering. These files are then automatically handled to a pipeline builder, automating the creation of pipelines. This automation takes user specified options into account, like what plugins to use and their propensity for application. Moreover, options for rendering over SSH remedy the fact that most computing clusters will not be able to run the plugins used for rendering, due to software limitations, licensing, etc. This prevents unnecessary manual moving of files back and forth between computers between rendering and prediction. The resulting system is a intuitive, plug-and-play system that highly simplifies the process of rendering both MIDI and audio through both generators and effects.

While easy to use, the AudioTransformer do come with a couple of prerequisites and

limitations. A folder containing a set of audio presets for each of the audio plugins used needs to be fed to the AudioTransformer, before it can generate and process pipelines. Unfortunately, these presets must be chosen and saved manually by the user, something that demands both time and a certain level of competence. Although possible to automate, these presets must fulfill certain requirements (e.g. no delay, pitch-shifting, etc.), in order to make the rendered version of the track align with the original MIDI, as discussed in section 6.2. The controls of these settings do also vary widely from plugin to plugin, and for some plugins (like Kontakt, see section 2.8) even varies widely from instrument to instrument within the same plugin. Thus, any user of the system need to be at least somewhat knowledgeable on how to program the plugins that are used.

Furthermore, the AudioTransformer heavily utilizes a series of third-party libraries like DawDreamer. While this has been strictly necessary to complete the finished system, it do come with a couple of downsides. Any problems, reliability issues and functional shortcomings from these libraries are propagated and carried on to the final AudioTransformer system. For instance, a notable limitation is the fact that DawDreamer (and thus also the AudioTransformer) is only able to use the old VST2 plugins, contrary to the newer VST3 plugins. Moreover, DawDreamer only accepts plugin presets that are saved in the FXP format. Yet, many plugins only allow external processing of presets through for instance the FXB format amongst others. This significantly reduces the potential of the AudioTransformer, because fewer plugins may be used. Moreover, as the experimental results show, some plugins (like Serum) simply do not work, even if it meets the requirements of DawDreamer (VST2 and FXP). Upon inspection, it appears that data rendered with Serum through DawDreamer simply sound nothing like what it should in comparison to when rendered manually within a normal Digital Audio Workstation (DAW). Thus, it is not known if this uncovers some unknown bug in DawDreamer or if it is an internal configuration problem for Serum. Either way, it adds unnecessary complexity and usability-issued to the system.

In addition to the key points discussed above, the validity of the AudioTransformer is also connected to the validity of the audio generated with the AudioTransformer. This includes the points discussed in 7.1. First, the audio produced is able to be used as in the Predictor Model while it is still able to predict the validation sets of the original datasets. Secondly, the generated audio is aligned both with its original audio counterpart. Finally, a subjective listening exercise confirms that the audio is clear, the stems of the audio (if applicable) is clearly distinguishable and the audio starts and ends at the same time of its belonging MIDI.

7.2.2. Prediction Quality

The thesis goal not only entails the creation of a system for rendering musical data through audio plugins, but perhaps more importantly, promises to examine how this rendering process affect the performance of SOTA AMT solutions. Even though the AudioTransformer system is well-designed and is able to both apply effectors to existing

audio and re-render MIDI with generators, the thesis goal relies on the Predictor Model to see if the produced content improves or decreases prediction performance. The PredictorModel is as uncovered in chapter 5 an MT3 model proposed by Gardner et al. at its core, with miscellaneous utility and helper modules built around it.

A series of experiments was designed to best examine how the rendering through audio plugins using different settings affect the performance of the MT3 model over a series of datasets. The exact experimental plan was presented in section 6.1. As highlighted in section 6.3, the baseline results validates the previous results from the original paper by Gardner et al. (2021). This is a useful result in itself, as it shows the technique is reproducible and transparent.

The experiments done transcribes without respect to the various instruments (the flat program granularity), as described in the experimental setup in section 6.2. This is done by removing the instrument token from the output vocabulary. The decision was made on the basis that the thesis goal and research questions build on if data augmentation can assist AMT solutions to achieve better results, and not on how to increase transcription scores for individual instruments. Moreover, the MT3 system yielded better results when this was done in the work from Gardner et al.. This is because it generalizes the output from the instruments focusing on transcribing the pitches as a whole, instead of also remembering the various frequency profiles of the notes.

However, with regards to the thesis goal, the experiments shows that the current application of audio plugins have an overall bad effect on the experiments, or only slightly increased some of the prediction measurements. As discussed in the section 7.1, this is (if anything) likely owed to limited assumptions and simplifications when deciding on presets to use for these plugins. The worsened prediction scores are thus not a limitation of the system per se, but it shows that more work must be done in order to thoroughly map out how the plugins should be used.

7.2.3. Other Limitations

It should be noted that the work on the project has brought to light a few unforeseen limitations and challenges. Testing of different experimental setups has been heavily limited by time restrictions enforced by both hardware and software. Moreover, inconsistencies in datasets has, to some degree, limited the possibilities for experimentation.

As a first, the rendering of musical datasets through the AudioTransformer takes *relatively* long time. A rough estimation shows that it takes roughly 1 minute to render 10 minutes of audio. While some of this is a result of the time it takes to render the audio through DawDreamer, the majority of the processing time occurs as a result of other operations. In order for the rendered datasets to be comparable to the original datasets in terms of prediction performance, it is necessary that they match in both volume and tempo. Thus, the AudioTransformer must open each generated stem during processing, calculate its volume for normalization (see section 2.7) and adjust the volume accordingly. In

addition, when generator plugins are used (as in experiment 3), it is critical that the tempo of the generated track and the original match. While some datasets do already include accurate tempo metadata for each track, this is not the general case. Thus, the AudioTransformer needs to open each MIDI file to manually inspect the tempo, incurring more additional overhead.

While the extra operations for the AudioTransformer adds significant overhead, it is, however, still possible to render the majority of the datasets presented in chapter 3 within reasonable time. Naturally, this poses little to no trouble when rendering smaller datasets like URMP and GuitarSet (duration of 1 and 3 hours respectively). The problem arises when larger datasets like SLAKH2100 and Cerebeus4 are introduced. Following the estimation above, rendering SLAKH2100 would take approximately 94 hours. Clearly, this is doable for one-time purposes when validating the system etc. Nevertheless it poses a big limitation for the system in terms of scalability and use for commercial products, that may even require even larger training quantities.

Furthermore, significant extra operations is needed to process the MIDI of some of the datasets before they can be rendered further. Primarily this comes from a limitation within DawDreamer, enabling it to only render MIDI from the first track in a file. Additionally, for MIDI to be rendered through DawDreamer, it also specifically needs to be sent through channel 0. However, the MIDI of the guitars from the GuitarSet dataset (see chapter 3) is all created with one track and channel per string. Each of the GuitarSet MIDIs therefore needs to be merged to one track and channel before they can be processed. Secondly, the MIDI for URMP and MusicNet contains all the instruments in a song over separate tracks and channels in the same MIDI file. Therefore, before these MIDI files can be processed, they need to be split up and saved to individual files.

7.2.4. MT3 Implementation Limitations

When working with the MT3 implementation downloaded from GitHub, some trial and error was necessary to get the system up and running. The first thing to note is that as MT3 is a Google product, it depends on numerous Google dependencies. These dependencies include but is not limited to SeqIO (see section 5.3.1), TFRecords (see section 5.2.3) and T5X (section 4.3). Developers on these dependencies does not seem to use the latest stable version of their dependencies. The dependencies are continuously updated in their respective GitHub repositories either through bug fixes or feature extensions. The developers therefore rather use the latest version available on GitHub, meaning this is necessary when implementing one of their architectures as well, e.g. MT3. Therefore, when setting up MT3, some tweaking to which GitHub versions used were necessary. MT3 is quite slow to restart, so a lot of time were spent debug the code get the correct parameters to be set.

MT3 and T5X is built for training on Tensor Processing Units (TPUs), or Google Cloud TPUs, and not locally on Graphical Processing Units (GPUs), as the experiments presented in this thesis have done. It is therefore not heavily tested on use for GPUs. It

is worth noting that the experiences presented here might differ based on other setup. To use the T5X train and evaluate scripts on the experiments in chapter 6, both scripts had to configure *gpu_options.allow_growth = True* after importing TensorFlow to the scripts. This was because some processes take allocate all the GPU memory to itself. When set to true, the GPU memory is not pre-allocated and will be able to grow as you need [1]. Another issue was when training on multiple GPUs, the training freezes at random epochs. This problem generally occurred at any epoch between 5000 and 50000. The first solution to solve this were to restart the system after the system did not progress after a couple of minutes. Similarly to the debugging, the restarting of MT3 is quite slow, so training lost performance during this period. Another problem with this solution was that the system were only checkpoints after a number of preset epochs. For instance with a checkpoint period of 2^{14} (16384), some runs did not reach this number of epochs, and all the progress was discarded. Instead of utilizing the power of training with multiple GPUs, training was done on a single GPU, rather with more jobs running concurrently.

Pre-caching of dataset task datasets is possible with SeqIO (see section 5.3.1). Caching is done prior to all experiment runs, because it saves time as compared to preprocessing on-demand. However, the caching-script cannot pre-process large datasets. It was not able to cache the MAESTRO train split of approximately 100GB in size. That is, with a hardware memory limit of 128GB. The MAESTRO dataset was therefore had to be split to six equal-length parts, in order to be able to be cached. Hence only 16.67GB of the data was utilized. However, when caching data of this size, some would throw out of memory exceptions and failed, in about one fourth of the attempted caching runs of that size. Although the rendering would create larger overhead, as it has to store the input spectrograms in memory as well, it should still not reach the hardware limit. The main problem with this script was that it was very unpredictable. Moreover, there were various errors when the script failed due to caching. There were three errors: Slurm Aborted, which seem to occur when the code itself detected memory error; Slurm Killed, which occurred when the code did not produce an error; and MemoryError, which occurred when the code got out of memory during saving of the processed dataset. Caching of larger datasets could therefore take longer than expected, resulting in a delayed experiment schedule.

[1] https://stackoverflow.com/questions/39465503/cuda-error-out-of-memory-in-tensorflow

8. Conclusion and Future Work

This final chapter will precede to conclude the thesis as a whole, touching on its most important findings, as well as listing its contributions. The chapter will also present various ideas and considerations for future work, relating to capabilities, usability and more.

8.1. Conclusion and Contributions

To conclude, this thesis examines how the use of digital audio plugins for augmentation of musical data affects the performance of SOTA AMT solutions. In short, a complete system is created for rendering, training and predicting audio data (see chapter 5). The system is used to run a setup of various experiments (detailed in chapter 6) that helps map out the impact these audio plugins has.

It is shown, using a SOTA AMT system, that higher precision scores can be achieved for AMT datasets by applying effector plugins. This is as long as some care is taken to adjust the strength of the effects added. Performance gain is yet to be achieved in the other two measures used in this thesis; recall and frame. More so, decrease in performance is also displayed (even on the precision measure) when effects are applied blindly, i.e. using their default state without any editing. It is assumed the lack of improvement on these two measures is at least partially owed to wrong rules (or lack of rules) when selecting the effector presets to use for rendering. It is however also shown that there are significant differences in performance depending on what effector plugin is used, and why this is thus also needs to be further mapped out. All this is, in turn, is essentially the answer to RQ1, which was posed in the introduction to this thesis and promised to examine how effector plugins can be used to increase AMT performance.

The second main question of the report, RQ2, pertains to the use of generator plugins for musical data rendering and how this affects AMT performance. So far, no performance gains have been achieved using generators, and the majority of the experiments conducted yields significantly lower scores than for instance blind application of effectors (as described in the preceding paragraph). While the reasons for this may be manifold, work have been done to evaluate and find the most important ones. In summary, it is assumed that wrong generator presets may be chosen for the wrong MIDI files. By this, it is meant for instance that a MIDI file which originally represents a high pitched instrument is rendered with a bass, which typically is only used only for the lowest frequencies in a

song. This, in turn, would give the training dataset a lot of qualities that cannot be found in the validation data. Closely related, it could also be that the datasets rendered with generators should more strongly resemble the instrument generation from the original dataset they are modeled on. Moreover, it seems that a strong contributing factor to the drop in AMT performance on generator-rendered datasets is because of too clean signals. I.e. an audio signal without any low frequency content, which typically occur in live recordings because of room acoustics and recording setup. As with effectors, it is also possible that at least partially wrong assumptions on saving of generator-presets contribute to the fall in prediction performance. For instance, it could be that these generators should have other ADSR settings.

The last question that this report set out to answer, RQ3, concerns itself with how a completely new dataset could be rendered using MIDI from online sources and generator plugins. Although time constraints made it difficult to specifically create such a dataset (as detailed in section 7.2), the thesis still covers other relevant experiments and touches on a couple of points relating to how such a dataset should/could potentially be rendered. Many of the points mentioned in relation to generator-rendering in the preceding paragraph clearly pertains to creation of a new datasets as well, since such a dataset would also be rendered using generators. For instance, when creating such a new dataset, one should take care to assign MIDI to instruments that would be natural to use for the MIDI. Furthermore, steps should be taken to introduce some low frequency signal into generator-rendered audio, for instance using effectors. This would ensure that the new dataset is comparable to existing AMT in terms of frequency representation. Additionally, it is discussed in section 7.2 that lack of certain meta-data in AMT datasets entails a few extra complications when rendering data through audio-plugins. It is thus recommended that any new dataset includes detailed information on Beats Per Minute (BPM), audio volume and ADSR information.

In addition to the answers to the research questions stated above, which constitutes a broad investigation of how audio plugins can be used to augment and improve AMT datasets, this thesis also presents other contributions that can come of help to the general field of AMT. A complete plug-and-play open-source rendering solution is built and released for large quantities of musical data through digital audio plugins. Among features for the program are several options for detailed rendering control, including automatic pipeline building for both effectors- and or generator-plugins. In addition comes possible utilization of SSH to render remotely and capability for automatically scanning datasets and finding correlated files (e.g. audio and MIDI). Moreover, a SOTA AMT system, based on MT3, is integrated in the system. This is used to evaluate and the rendered datasets, but also verifies the previous results reported by Gardner et al. The thesis also gives a detailed description of background theory and work related to the field of AMT, especially pertaining to modern Machine Learning (ML) solutions and augmentation approaches. Comprehensive discussion and evaluation of the thesis results are also given, looking at where the conducted experiments fails and where they work. Additionally, ideas for future work and improvements will as stated, be given in section 8.2.

8.2. Future Work

Taking the experimental results (section 6.3), the evaluation (section 7.1) and the limitations presented in the discussion (section 7.2) into account, this section presents various ideas for further research to explore how data augmentation can be used for AMT purposes. This section first address possible improvements to the architecture itself, first presented in chapter 5. These are improvements that can not only enhance transcription scores, but also research usability. Secondly, this section will present recommendations and enhancements for the dataset augmentation, and how the current augmentation flow can be changed to provide even more scalable, varied and realistic rendition of audio.

8.2.1. Use of Audio Plugins for Automatic Music Transcription

In the evaluation and discussion of the results presented in this thesis, it is argued that even though the results from most experiments have not improved from the baseline results in their entirety, many entries in the validation sets are actually improved. As discussed, this is likely due to the variability of preset quality, of which some are usable while others should be discarded. A natural first step if one wanted to expand on the existing system would be to address manual saving of presets. As can be recalled from the discussion section, this requires both knowledge and comfortability with the plugins, upping the entry level for use.

One way to solve this is to make a script that altered the values of the audio plugin, before storing it in a preset file. The script could follow a set of rules chosen by the researcher, and generate presets based on random values within the boundaries. However, while this would alleviate both the time requirement and the need for knowing how to operate the plugins, the saved presets still rely exclusively on human assumptions about what these presets should contain in order to elevate the dataset quality. As discussed in section 7.1, while a lot of these assumptions may make sense (e.g. no delay or arpeggio), these audio plugins come with so many configurable options that it is impossible to say how each of them will affect prediction performance.

Instead, it would be perhaps be a better idea if a computer *learned* how to program these audio plugins with the goal of increasing prediction performance. One could imagine a quite rudimentary Generative Adversarial Network (GAN) setup, where one model handles AMT (like MT3) and another model outputs the parameter values for an audio plugin given a audio- or MIDI-file to render. The prediction would work as normal, while the preset programming model would work toward minimizing the loss of the predictor. This way, the system would (hopefully) be able to converge to an optimal solution for the presets. The model that programs the preset would thus be able to represent its own set of rules, which relies solely on feedback from the predictor, instead of human assumptions about how these presets should sound. This would require changes to the system, described in the next subsection.

Additionally, as discussed in the evaluation section, it may to be a problem (especially

for single instrument comparable to that of the corresponding validation set. It would be interesting to see what gains in prediction could be achieved if the system was expanded to take into account the original instrument distribution. This could for instance be done by storing the presets for generators in different sub-folders, representing instrument types (e.g. guitar, piano, bass, etc.). If the pipeline generator is then fed with numbers on the original distribution — which normally is fairly easy to retrieve from the dataset metadata — it could easily assign instruments for rendering that correlate with the ones used in the original data.

8.2.2. System improvements

Although the architecture from this thesis (presented in chapter 5) has been designed to be flexible, the system still has room for improvements. The Audio Transformer (section 5.1) uses DawDreamer (section 4.4.2) at its core to render audio, which is a great tool, but not as flexible as expected. It is completely incompatible with some audio plugins, for instance Omnisphere [1], and experienced a lot of rendering issues with Serum (section 2.8). Rather than using DawDreamer, a tailored rendering software designed specifically for this system, as introduced in the preliminary report (section 1.1), could perhaps increase the flexibility of using audio plugins. As the preliminary report suggests, with the use of the Virtual Studio Technology (VST) Software Development Kit (SDK) provided by Steinberg Media (section 2.8) to create this internal software, altering of the specific parameters of the audio plugin would be possible during runtime.

The system would also greatly benefit from reductive measures on rendering time, allowing for more effective experimentation. As discussed in section 7.2.3, most of the rendering overhead occur as a result of extra operations relating to volume normalization. Additionally, some time is used to analyze associated MIDI files for tempo and instrument types. Since both of these operations are necessary to get a good foundation for comparison between the original and rendered data, there is little one could do in order to remedy this. Yet, this highlights the need for standardized ways to organize datasets in the field of AMT. Thus, it is suggested that future work looks at how this organization may be done best with regards to future use of the data, included rendering. Any such work should also include suggestions on how to handle fields such as tempo, instruments and volume. This information should, however, not be too hard to obtain during the organizing of the data. This serves as an alternative to current day practises, where these fields must essentially be retrieved individually by anyone who wishes to use the dataset with these values in mind.

Usability wise, a simple Graphical User Interface (GUI) would also impact the system positively. While the system is nicely packaged into modules, using it in its entirety still calls for manual running of several scripts, replacement of variables and settings and a rudimental understanding of how the whole thing is comprised. Introducing a GUI to the system would not only help unify the whole process, thus saving time, but could

[1] https://www.spectrasonics.net/products/omnisphere/

also help with keeping track of operations or visualize/listen to data and automatically analyze results. Perhaps even more important, the system would greatly lower the prerequisites necessary to use the system, making it more easier for anyone who would want to experiment further.

8.2.3. Future Generation of Datasets

While making the system more usable and removing bottlenecks with relation to time is great, further improvements and changes to the system would also need to be done in order to address the fact that rendering through plugins generally lower the model prediction score. Therefore this section give some thoughts on the generation of datasets in future research. As noted in section 7.1, further experimentation is needed to determine how audio rendering through plugins can be optimized for AMT. A good idea would probably be to start looking at more advanced requirements for the audio plugin presets.

As discussed in the evaluation section (7.1), the sheer size of even the smallest datasets spawn at least a hundred single-instrument tracks, while at the maximum numbering in the several thousands. Because of time- (saving the presets) and software limitations (the number of presets to choose from) however, these dataset sizes necessitate that some presets are used on more than one sub-track. While the occurrence of the same instrument across tracks is totally normal for the original datasets (see chapter 3), these will always have some variation, ranging from different performers, acoustic surroundings, tune and more. However, this is not the case for the tracks rendered through audio plugins. Tracks rendered with the same instrument, or more specifically preset, will have the exact same timbre no matter which MIDI file it is applied to. Future work should examine how subtle variations can be introduced to audio plugin presets, and even work to allow randomization of parameters such that the same preset can be used to render stems across tracks without incurring too much similarity.

Furthermore, the system could also be expanded in ways that let it augment data in new manners, while still using audio plugins. It would perhaps be interesting to look at how one could do realistic MIDI transformations, which can then be rendered by the system. As of now, the system is limited by the MIDI in the original datasets, and can only render these with different instruments. More so, mentioned in section 7.1, the rendering cannot measure up to the original performance. However, by augmenting the original MIDI, one could generate completely new datasets and render them through the system to obtain new accompanying audio. For instance, it would not take that much work to apply simple operations to MIDI, like changing the key, inverting chords, time-stretching and/or arpeggiating notes.

Augmentation of datasets has improved ML approaches within many areas, but has not yet been a key point within AMT research. Current AMT research are mostly focusing on a certain scope, making the systems less generic. The datasets from this thesis discuss train and validation sets specific to their scopes, making them not expand to other music types. For instance, MAESTRO is only for piano and GuitarSet is only for guitar.

Popular music contain more sounds made from electronic instruments such as synthesizers. Not only is data augmentation thought to help in these cases, but also on tracks with background noise, such as in live music. It would therefore be interesting to see how data augmentation works if a conjoined validation dataset made of a combination of several musical styles, instruments and background noises was available. As AMT results has converged for single-instrument piano music, and can possible do so for other instruments if enough annotated data is available, a dataset like that would call for improvements and other angles to tackle the problem within the literature. The conjoined validation set could possibly be filtered on musical instruments, such that single-instrument validation scores can be separated.

Bibliography

Jeremy F. Alm and James S. Walker. Time-frequency analysis of musical instruments. *SIAM Review*, 44(3):457–476, 2002. http://www.jstor.org/stable/4148384.

Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, and Anssi Klapuri. Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41(3):407–434, 2013. doi:https://doi.org/10.1007/s10844-013-0258-3.

Emmanouil Benetos, Simon Dixon, Zhiyao Duan, and Sebastian Ewert. Automatic music transcription: An overview. *IEEE Signal Processing Magazine*, 36(1):20–30, 2019. doi:https://doi.org/10.1109/MSP.2018.2869928.

Rachel Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Bello. Medleydb: A multitrack dataset for annotation-intensive mir research. In *Proceedings - 15th International Society for Music Information Retrieval Conference*, ISMIR, New York, NY, United States, 2014. ISMIR. http://www.terasoft.com.tw/conf/ismir2014/proceedings/T028_322_Paper.pdf.

David Braun. Dawdreamer: Bridging the gap between digital audio workstations and python interfaces. *CoRR*, abs/2111.09931, 2021. URL https://arxiv.org/abs/2111.09931.

Hongyu Chen, Qinyin Xiao, and Xueyuan Yin. Generating music algorithm with deep convolutional generative adversarial networks. In *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*, ICET, pages 576–580, Chengdu, China, 2019. IEEE. doi:https://doi.org/10.1109/ELTECH.2019.8839521.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *CoRR*, abs/1601.06733, 2016. URL http://arxiv.org/abs/1601.06733.

KyungHyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014. URL http://arxiv.org/abs/1409.1259.

Josh Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse H. Engel. MT3: multi-task multitrack music transcription. *CoRR*, abs/2111.03017, 2021. URL https://arxiv.org/abs/2111.03017.

Vijay K. Garg. Chapter 4 - an overview of digital communication and transmission. In *Wireless Communications & Networking*, The Morgan Kaufmann Series in Networking, pages 85–122, Burlington, VT, United States, 2007. Morgan Kaufmann. doi:https://doi.org/10.1016/B978-012373580-5/50038-7.

Tom Gerou and Linda Lusk. *Essential Dictionary of Music Notation*. Alfred Music, Los Angeles, CA, United States, 1996.

Vincent Goudard and Remy Muller. Real-time audio plugin architectures: a comparative study, 2003. http://recherche.ircam.fr/equipes/temps-reel/movement/muller/xspif/pluginarch.pdf.

William M. Hartmann. Pitch, periodicity, and auditory organization. *The Journal of the Acoustical Society of America*, 100(6):3491–3502, 1996. doi:https://doi.org/10.1121/1.417248.

Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse H. Engel, Sageev Oore, and Douglas Eck. Onsets and frames: Dual-objective piano transcription. *CoRR*, abs/1710.11153, 2017. URL http://arxiv.org/abs/1710.11153.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse H. Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the MAESTRO dataset. *CoRR*, abs/1810.12247, 2018. URL http://arxiv.org/abs/1810.12247.

Curtis Hawthorne, Ian Simon, Rigel Swavely, Ethan Manilow, and Jesse H. Engel. Sequence-to-sequence piano transcription with transformers. *CoRR*, abs/2107.09142, 2021. URL https://arxiv.org/abs/2107.09142.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–80, 1997. doi:https://doi.org/10.1162/neco.1997.9.8.1735.

Michael I Jordan and Tom M Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015. doi:https://doi.org/10.1126/science.aaa8415.

Rainer Kelz, Matthias Dorfer, Filip Korzeniowski, Sebastian Böck, Andreas Arzt, and Gerhard Widmer. On the potential of simple framewise approaches to piano transcription. *CoRR*, abs/1612.05153, 2016. URL http://arxiv.org/abs/1612.05153.

Jong Wook Kim, Rachel M. Bittner, Aparna Kumar, and Juan Pablo Bello. Neural music synthesis for flexible timbre control. *CoRR*, abs/1811.00223, 2018. URL http://arxiv.org/abs/1811.00223.

Anssi P. Klapuri. Automatic music transcription as we know it today. *Journal of New Music Research*, 33(3):269–282, 2004. doi:https://doi.org/10.1080/0929821042000317840.

Qiuqiang Kong, Bochen Li, Xuchen Song, Yuan Wan, and Yuxuan Wang. High-resolution piano transcription with pedals by regressing onset and offset times. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29:3707–3717, 2021. doi:https://doi.org/10.1109/TASLP.2021.3121991.

Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999. doi:https://doi.org/10.1038/44565.

Bochen Li, Xinzhao Liu, Karthik Dinesh, Zhiyao Duan, and Gaurav Sharma. Creating a multitrack classical music performance dataset for multimodal music analysis: Challenges, insights, and applications. *IEEE Transactions on Multimedia*, 21(2):522–535, 2019. doi:https://doi.org/10.1109/tmm.2018.2856090.

Batta Mahesh. Machine learning algorithms - a review. *International Journal of Science and Research (IJSR)*, 9:381–386, 2020. https://www.ijsr.net/archive/v9i1/ART20203995.pdf.

Ethan Manilow, Gordon Wichern, Prem Seetharaman, and Jonathan Le Roux. Cutting music source separation some slakh: A dataset to study the impact of training data quality and quantity. *CoRR*, abs/1909.08494, 2019. URL http://arxiv.org/abs/1909.08494.

Brian McFee, Eric J Humphrey, and Juan Pablo Bello. A software framework for musical data augmentation. In *Proceedings of the 16th ISMIR Conference*, ISMIR, pages 248–254, New York, NY, United States, 2015. ISMIR. https://brianmcfee.net/papers/ismir2015_augmentation.pdf.

IBM & Microsoft. Multimedia programming interface and data specifications 1.0, 1991. https://www.aelius.com/njh/wavemetatools/doc/riffmci.pdf.

MIDI Manufacturers Association. An Introduction to MIDI, 2009. https://www.midi.org/images/easyblog_articles/43/intromidi.pdf.

Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015. URL http://arxiv.org/abs/1511.08458.

Martin Piszczalski and Bernard A. Galler. Automatic music transcription. *Computer Music Journal*, 1(4):24–31, 1977. http://www.jstor.org/stable/40731297.

Colin Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD thesis, Columbia University, New York, NY, United States, 2016. https://colinraffel.com/publications/thesis.pdf.

Colin Raffel, Brian Mcfee, Eric J. Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, Daniel P. W. Ellis, and C Colin Raffel. mir_eval: a transparent implementation of common mir metrics. In *In Proceedings of the 15th International Society for Music Information Retrieval Conference*, ISMIR, New York, NY, United States, 2014. ISMIR. https://www.ee.columbia.edu/~dpwe/pubs/RaffMHS14-mireval.pdf.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019. URL http://arxiv.org/abs/1910.10683.

Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*, 2022. URL https://arxiv.org/abs/2203.17189.

David E. Rumelhart and James L. McClelland. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 318–362, Cambridge, MA, United States, 1987. MIT Press. https://ieeexplore.ieee.org/document/6302929.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi:https://doi.org/10.1016/j.neunet.2014.09.003.

M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997. doi:https://doi.org/10.1109/78.650093.

Ken Schwaber. Scrum development process. In *Business Object Design and Implementation*, pages 117–134, Burlington, MA, United States, 1997. Springer. doi:https://doi.org/10.1007/978-1-4471-0947-1_11.

Rounik Sethi. Top 12 most popular daws, 2018. https://ask.audio/articles/top-12-most-popular-daws-you-voted-for.

Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *Towards Data Science*, 6(12):310–316, 2017.

Siddharth Sigtia, Emmanouil Benetos, and Simon Dixon. An end-to-end neural network for polyphonic piano music transcription. *ACM Transactions on Audio, Speech and Language Processing*, 24(5):927–939, 2016. doi:https://doi.org/10.1109/TASLP.2016.2533858.

David Silver, Aja Huang, Christopher Maddison, Arthur Guez, Laurent Sifre, George Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel,

and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. doi:https://doi.org/10.1038/nature16961.

P. Smaragdis and J.C. Brown. Non-negative matrix factorization for polyphonic music transcription. In *2003 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (IEEE Cat. No.03TH8684)*, Workshop, pages 177–180, Cambridge, MA, United States, 2203. IEEE. doi:https://doi.org/10.1109/ASPAA.2003.1285860.

Carl Southall, Ryan Stables, and Jason Hockman. Player vs transcriber: A game approach to data manipulation for automatic drum transcription. In *Proceedings of the 19th ISMIR Conference,*, ISMIR, pages 58–65, Birmingham, United Kingdom, 2018. ISMIR. https://carlsouthall.files.wordpress.com/2018/08/player_vs_transcriber.pdf.

Steinberg Media. Our technologies, 2021. https://www.steinberg.net/technology/.

Stanley Smith Stevens, John Volkmann, and Edwin Broomell Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937. doi:https://doi.org/10.1121/1.1915893.

John Thickstun, Zaid Harchaoui, and Sham Kakade. Learning features of music from scratch. In -, ICLR, pages –, Seattle, WA, United States, 2016. ICLR. doi:https://doi.org/10.48550/arxiv.1611.09827.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Hans-Dieter Wehle. Machine learning, deep learning, and ai: What's the difference?, 2017. https://www.researchgate.net/publication/318900216_Machine_Learning_Deep_Learning_and_AI_What%27s_the_Difference.

Qingsong Wen, Liang Sun, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *CoRR*, abs/2002.12478, 2020. URL https://arxiv.org/abs/2002.12478.

Qingyang Xi, Rachel M Bittner, Johan Pauwels, Xuzhou Ye, and Juan P Bello. Guitarset: A dataset for guitar transcription. In *Proceedings of the 19th International Society for Music Information Retrieval Conference, ISMIR 2018*, ISMIR, pages 453–460, New York, NY, United States, 2018. ISMIR. https://guitarset.weebly.com/uploads/1/2/1/6/121620128/xi_ismir_2018.pdf.

A. Additional Results

The following tables display the results for experiment 0, 2, 3, and 4. Table A.1 and A.2 is an extension of table 6.2 (E0), table 6.4 (E2 and E3), and table 6.5 (E4). In addition to the F1 metric, as the mention tables include, the following tables include precision and recall as well. Furthermore, table A.3 displays the same information as table 6.5, however with the change from the experiment 0 (baseline), rather than the change from the corresponding experiment 2 or experiment 3 results.

Table A.1.: Full results for experiment 0, 2, 3 and 4 evaluated on the MAESTRO and Guitarset datasets

Type	Frame			Onset			Onset+Offset		
	P	R	F1	P	R	F1	P	R	F1
MAESTRO									
E0	0.79	0.76	0.78	0.93	0.85	0.88	0.66	0.60	0.63
E2.1	0.76	0.72	0.74	0.88	0.81	0.85	0.57	0.53	0.55
E2.2	0.75	0.75	0.75	0.91	0.82	0.86	0.59	0.53	0.56
E3.1	0.07	0.02	0.03	0.02	0.02	0.02	0.00	0.00	0.00
E3.2	0.54	0.66	0.59	0.53	0.69	0.59	0.24	0.30	0.27
E4.0	0.78	0.63	0.69	0.78	0.79	0.78	0.45	0.45	0.45
E4.1	0.70	0.58	0.63	0.50	0.66	0.56	0.25	0.32	0.27
E4.2	0.75	0.59	0.66	0.63	0.72	0.67	0.33	0.37	0.35
E4.3	0.55	0.52	0.53	0.37	0.54	0.42	0.17	0.24	0.19
GuitarSet									
E0	0.83	0.82	0.83	0.83	0.83	0.83	0.67	0.67	0.67
E2.1	0.82	0.77	0.79	0.79	0.74	0.76	0.60	0.56	0.58
E2.2	0.83	0.79	0.81	0.82	0.80	0.81	0.64	0.63	0.63
E3.1	0.02	0.01	0.01	0.02	0.01	0.01	0.01	0.00	0.01
E3.2	0.70	0.63	0.66	0.65	0.72	0.68	0.31	0.33	0.32
E4.0	0.85	0.82	0.84	0.86	0.84	0.85	0.69	0.68	0.68
E4.1	0.82	0.78	0.80	0.73	0.72	0.70	0.53	0.52	0.51
E4.2	0.82	0.79	0.80	0.78	0.77	0.77	0.56	0.56	0.56
E4.3	0.73	0.63	0.67	0.71	0.71	0.69	0.34	0.34	0.33

Table A.2.: Full results for experiment 0, 2, 3 and 4 evaluated on the MusicNet and URMP datasets

Type	Frame			Onset			Onset+Offset		
	P	R	F1	P	R	F1	P	R	F1
MusicNet									
E0	0.64	0.57	0.60	0.35	0.40	0.37	0.20	0.22	0.21
E2.1	0.60	0.55	0.57	0.31	0.34	0.32	0.17	0.18	0.17
E2.2	0.62	0.55	0.57	0.32	0.34	0.33	0.17	0.18	0.18
E3.1	0.06	0.04	0.05	0.02	0.03	0.02	0.00	0.01	0.01
E3.2	0.43	0.44	0.43	0.23	0.31	0.26	0.11	0.15	0.12
E4.0	0.63	0.57	0.59	0.38	0.43	0.39	0.23	0.25	0.23
E4.1	0.60	0.52	0.55	0.28	0.37	0.31	0.16	0.20	0.17
E4.2	0.63	0.53	0.57	0.32	0.39	0.34	0.18	0.21	0.19
E4.3	0.42	0.42	0.41	0.21	0.35	0.25	0.08	0.14	0.10
URMP									
E0	0.56	0.60	0.57	0.48	0.50	0.48	0.22	0.23	0.22
E2.1	0.46	0.51	0.48	0.20	0.24	0.21	0.05	0.06	0.05
E2.2	0.49	0.50	0.49	0.27	0.30	0.28	0.08	0.08	0.08
E3.1	0.06	0.06	0.06	0.02	0.02	0.02	0.01	0.01	0.01
E3.2	0.41	0.43	0.42	0.29	0.34	0.30	0.10	0.11	0.10
E4.0	0.80	0.74	0.77	0.65	0.64	0.64	0.41	0.40	0.40
E4.1	0.75	0.70	0.72	0.42	0.44	0.42	0.24	0.26	0.25
E4.2	0.78	0.67	0.72	0.46	0.48	0.46	0.26	0.27	0.26
E4.3	0.58	0.51	0.53	0.38	0.48	0.41	0.18	0.22	0.19

Table A.3.: Results from experiment 4. Each score contains the gain/loss from the baseline experiment counterpart score shown in table 6.4

Model	MAESTRO	GuitarSet	MusicNet	URMP
		Frame F1		
E3.0 (Baseline)	0.69(-10.8%)	0.84(+1.5%)	0.59(-1.4%)	0.77(+33.7%)
E3.1 (RC20)	0.63(-18.2%)	0.80(-3.4%)	0.55(-8.5%)	0.72(+26.4%)
E3.2 (SerumFX)	0.66(-15.3%)	0.80(-2.7%)	0.57(-5.7%)	0.72(+25.9%)
E3.3 (Kontakt)	0.53(-32.1%)	0.67(-18.7%)	0.41(-31.3%)	0.53(-7.0%)
		Onset F1		
E3.0 (Baseline)	0.78(-11.6%)	0.85(+2.5%)	0.39(+6.6%)	0.64(+34.0%)
E3.1 (RC20)	0.56(-36.4%)	0.70(-15.0%)	0.31(-15.7%)	0.42(-11.3%)
E3.2 (SerumFX)	0.67(-24.7%)	0.77(-6.6%)	0.34(-8.1%)	0.46(-3.6%)
E3.3 (Kontakt)	0.42(-52.0%)	0.69(-16.2%)	0.25(-30.8%)	0.41(-13.3%)
		Onset+Offset F1		
E3.0 (Baseline)	0.45(-29.2%)	0.68(+2.5%)	0.23(+13.5%)	0.40(+79.6%)
E3.1 (RC20)	0.27(-56.9%)	0.51(-23.6%)	0.17(-17.7%)	0.25(+11.1%)
E3.2 (SerumFX)	0.35(-45.2%)	0.56(-16.2%)	0.19(-7.4%)	0.26(+16.4%)
E3.3 (Kontakt)	0.19(-69.5%)	0.33(-50.5%)	0.10(-50.0%)	0.19(-12.9%)

B. MAESTRO Train Split

The following list shows the train split used for the MAESTRO dataset. As explained in chapter 6, it was necessary to create a new train split due to hardware limitations. The split is displayed in sorted order.

MIDI-UNPROCESSED_04-05_R1_2014_MID--AUDIO_04_R1_2014_wav--1
MIDI-UNPROCESSED_04-07-08-10-12-15-17_R2_2014_MID--AUDIO_12_R2_2014_wav
MIDI-UNPROCESSED_04-08-12_R3_2014_MID--AUDIO_04_R3_2014_wav--2
MIDI-UNPROCESSED_06-08_R1_2014_MID--AUDIO_06_R1_2014_wav--2
MIDI-UNPROCESSED_06-08_R1_2014_MID--AUDIO_07_R1_2014_wav--5
MIDI-UNPROCESSED_09-10_R1_2014_MID--AUDIO_09_R1_2014_wav--1
MIDI-UNPROCESSED_09-10_R1_2014_MID--AUDIO_09_R1_2014_wav--2
MIDI-UNPROCESSED_11-13_R1_2014_MID--AUDIO_11_R1_2014_wav--3
MIDI-UNPROCESSED_14-15_R1_2014_MID--AUDIO_15_R1_2014_wav--6
MIDI-UNPROCESSED_16-18_R1_2014_MID--AUDIO_18_R1_2014_wav--1
MIDI-UNPROCESSED_21-22_R1_2014_MID--AUDIO_21_R1_2014_wav--3
MIDI-UNPROCESSED_21-22_R1_2014_MID--AUDIO_21_R1_2014_wav--8
MIDI-UNPROCESSED_21-22_R1_2014_MID--AUDIO_22_R1_2014_wav--4
MIDI-Unprocessed_01_R1_2011_MID--AUDIO_R1-D1_02_Track02_wav
MIDI-Unprocessed_02_R1_2009_01-02_ORIG_MID--AUDIO_02_R1_2009_02_R1_2009_01_WAV
MIDI-Unprocessed_02_R2_2008_01-05_ORIG_MID--AUDIO_02_R2_2008_wav--4
MIDI-Unprocessed_02_R2_2008_01-05_ORIG_MID--AUDIO_02_R2_2008_wav--5
MIDI-Unprocessed_03_R1_2008_01-04_ORIG_MID--AUDIO_03_R1_2008_wav--2
MIDI-Unprocessed_03_R1_2009_03-08_ORIG_MID--AUDIO_03_R1_2009_03_R1_2009_05_WAV
MIDI-Unprocessed_03_R1_2009_03-08_ORIG_MID--AUDIO_03_R1_2009_03_R1_2009_06_WAV
MIDI-Unprocessed_043_PIANO043_MID--AUDIO-split_07-06-17_Piano-e_1-03_wav--3
MIDI-Unprocessed_044_PIANO044_MID--AUDIO-split_07-06-17_Piano-e_1-04_wav--3
MIDI-Unprocessed_047_PIANO047_MID--AUDIO-split_07-06-17_Piano-e_2-04_wav--1
MIDI-Unprocessed_04_R1_2011_MID--AUDIO_R1-D2_02_Track02_wav
MIDI-Unprocessed_04_R2_2008_01-04_ORIG_MID--AUDIO_04_R2_2008_wav--4
MIDI-Unprocessed_04_R3_2008_01-07_ORIG_MID--AUDIO_04_R3_2008_wav--5
MIDI-Unprocessed_051_PIANO051_MID--AUDIO-split_07-06-17_Piano-e_3-02_wav--4
MIDI-Unprocessed_051_PIANO051_MID--AUDIO-split_07-06-17_Piano-e_3-02_wav--5
MIDI-Unprocessed_052_PIANO052_MID--AUDIO-split_07-06-17_Piano-e_3-03_wav--2
MIDI-Unprocessed_052_PIANO052_MID--AUDIO-split_07-06-17_Piano-e_3-03_wav--5
MIDI-Unprocessed_055_PIANO055_MID--AUDIO-split_07-07-17_Piano-e_1-04_wav--1
MIDI-Unprocessed_059_PIANO059_MID--AUDIO-split_07-07-17_Piano-e_2-03_wav--3

MIDI-Unprocessed_05_R1_2009_01-02_ORIG_MID--AUDIO_05_R1_2009_05_R1_2009_01_WAV
MIDI-Unprocessed_05_R1_2011_MID--AUDIO_R1-D2_11_Track11_wav
MIDI-Unprocessed_066_PIANO066_MID--AUDIO-split_07-07-17_Piano-e_3-02_wav--1
MIDI-Unprocessed_067_PIANO067_MID--AUDIO-split_07-07-17_Piano-e_3-03_wav--1
MIDI-Unprocessed_06_R1_2009_03-07_ORIG_MID--AUDIO_06_R1_2009_06_R1_2009_03_WAV
MIDI-Unprocessed_06_R1_2009_03-07_ORIG_MID--AUDIO_06_R1_2009_06_R1_2009_07_WAV
MIDI-Unprocessed_06_R2_2008_01-05_ORIG_MID--AUDIO_06_R2_2008_wav--4
MIDI-Unprocessed_070_PIANO070_MID--AUDIO-split_07-08-17_Piano-e_1-02_wav--2
MIDI-Unprocessed_071_PIANO071_MID--AUDIO-split_07-08-17_Piano-e_1-04_wav--4
MIDI-Unprocessed_073_PIANO073_MID--AUDIO-split_07-08-17_Piano-e_2-02_wav--1
MIDI-Unprocessed_073_PIANO073_MID--AUDIO-split_07-08-17_Piano-e_2-02_wav--2
MIDI-Unprocessed_07_R1_2006_01-04_ORIG_MID--AUDIO_07_R1_2006_01_Track01_wav
MIDI-Unprocessed_07_R1_2008_01-04_ORIG_MID--AUDIO_07_R1_2008_wav--4
MIDI-Unprocessed_07_R3_2008_01-05_ORIG_MID--AUDIO_07_R3_2008_wav--1
MIDI-Unprocessed_080_PIANO080_MID--AUDIO-split_07-09-17_Piano-e_1-06_wav--2
MIDI-Unprocessed_08_R1_2008_01-05_ORIG_MID--AUDIO_08_R1_2008_wav--4
MIDI-Unprocessed_08_R1_2011_MID--AUDIO_R1-D3_08_Track08_wav
MIDI-Unprocessed_09_R1_2008_01-05_ORIG_MID--AUDIO_09_R1_2008_wav--1
MIDI-Unprocessed_09_R1_2008_01-05_ORIG_MID--AUDIO_09_R1_2008_wav--5
MIDI-Unprocessed_09_R1_2009_01-04_ORIG_MID--AUDIO_09_R1_2009_09_R1_2009_02_WAV
MIDI-Unprocessed_09_R2_2006_01_ORIG_MID--AUDIO_09_R2_2006_02_Track02_wav
MIDI-Unprocessed_09_R2_2009_01_ORIG_MID--AUDIO_09_R2_2009_09_R2_2009_02_WAV
MIDI-Unprocessed_09_R3_2008_01-07_ORIG_MID--AUDIO_09_R3_2008_wav--3
MIDI-Unprocessed_09_R3_2008_01-07_ORIG_MID--AUDIO_09_R3_2008_wav--4
MIDI-Unprocessed_10_R1_2006_01-04_ORIG_MID--AUDIO_10_R1_2006_02_Track02_wav
MIDI-Unprocessed_10_R1_2008_01-04_ORIG_MID--AUDIO_10_R1_2008_wav--1
MIDI-Unprocessed_10_R1_2009_01-02_ORIG_MID--AUDIO_10_R1_2009_10_R1_2009_02_WAV
MIDI-Unprocessed_10_R1_2009_03-05_ORIG_MID--AUDIO_10_R1_2009_10_R1_2009_04_WAV
MIDI-Unprocessed_10_R2_2009_01_ORIG_MID--AUDIO_10_R2_2009_10_R2_2009_03_WAV
MIDI-Unprocessed_11_R2_2008_01-05_ORIG_MID--AUDIO_11_R2_2008_wav--3
MIDI-Unprocessed_11_R3_2008_01-04_ORIG_MID--AUDIO_11_R3_2008_wav--4
MIDI-Unprocessed_12_R1_2009_01-02_ORIG_MID--AUDIO_12_R1_2009_12_R1_2009_01_WAV
MIDI-Unprocessed_12_R2_2008_01-04_ORIG_MID--AUDIO_12_R2_2008_wav--2
MIDI-Unprocessed_12_R3_2011_MID--AUDIO_R3-D4_09_Track09_wav
MIDI-Unprocessed_13_R1_2008_01-04_ORIG_MID--AUDIO_13_R1_2008_wav--2
MIDI-Unprocessed_13_R1_2011_MID--AUDIO_R1-D5_04_Track04_wav
MIDI-Unprocessed_14_R1_2009_01-05_ORIG_MID--AUDIO_14_R1_2009_14_R1_2009_04_WAV
MIDI-Unprocessed_15_R1_2006_01-05_ORIG_MID--AUDIO_15_R1_2006_01_Track01_wav
MIDI-Unprocessed_15_R1_2011_MID--AUDIO_R1-D6_09_Track09_wav
MIDI-Unprocessed_15_R2_2008_01-04_ORIG_MID--AUDIO_15_R2_2008_wav--3
MIDI-Unprocessed_16_R1_2008_01-04_ORIG_MID--AUDIO_16_R1_2008_wav--1
MIDI-Unprocessed_17_R1_2006_01-06_ORIG_MID--AUDIO_17_R1_2006_01_Track01_wav
MIDI-Unprocessed_17_R1_2009_01-03_ORIG_MID--AUDIO_17_R1_2009_17_R1_2009_03_WAV

MIDI-Unprocessed_17_R1_2011_MID--AUDIO_R1-D7_02_Track02_wav
MIDI-Unprocessed_17_R1_2011_MID--AUDIO_R1-D7_04_Track04_wav
MIDI-Unprocessed_17_R1_2011_MID--AUDIO_R1-D7_05_Track05_wav
MIDI-Unprocessed_17_R2_2008_01-04_ORIG_MID--AUDIO_17_R2_2008_wav--3
MIDI-Unprocessed_17_R2_2009_01_ORIG_MID--AUDIO_17_R2_2009_17_R2_2009_03_WAV
MIDI-Unprocessed_17_R2_2011_MID--AUDIO_R2-D5_03_Track03_wav
MIDI-Unprocessed_18_R1_2006_01-05_ORIG_MID--AUDIO_18_R1_2006_02_Track02_wav
MIDI-Unprocessed_18_R1_2006_01-05_ORIG_MID--AUDIO_18_R1_2006_04_Track04_wav
MIDI-Unprocessed_18_R1_2011_MID--AUDIO_R1-D7_08_Track08_wav
MIDI-Unprocessed_20_R1_2006_01-04_ORIG_MID--AUDIO_20_R1_2006_04_Track04_wav
MIDI-Unprocessed_20_R1_2009_01-05_ORIG_MID--AUDIO_20_R1_2009_20_R1_2009_01_WAV
MIDI-Unprocessed_20_R1_2011_MID--AUDIO_R1-D8_02_Track02_wav
MIDI-Unprocessed_20_R1_2011_MID--AUDIO_R1-D8_04_Track04_wav
MIDI-Unprocessed_21_R1_2006_01-04_ORIG_MID--AUDIO_21_R1_2006_01_Track01_wav
MIDI-Unprocessed_21_R2_2009_01_ORIG_MID--AUDIO_21_R2_2009_21_R2_2009_02_WAV
MIDI-Unprocessed_21_R2_2009_01_ORIG_MID--AUDIO_21_R2_2009_21_R2_2009_03_WAV
MIDI-Unprocessed_22_R1_2006_01-04_ORIG_MID--AUDIO_22_R1_2006_01_Track01_wav
MIDI-Unprocessed_22_R1_2011_MID--AUDIO_R1-D8_13_Track13_wav
MIDI-Unprocessed_22_R2_2006_01_ORIG_MID--AUDIO_22_R2_2006_03_Track03_wav
MIDI-Unprocessed_22_R3_2011_MID--AUDIO_R3-D7_03_Track03_wav
MIDI-Unprocessed_23_R1_2011_MID--AUDIO_R1-D9_04_Track04_wav
MIDI-Unprocessed_23_R3_2011_MID--AUDIO_R3-D8_05_Track05_wav
MIDI-Unprocessed_24_R1_2011_MID--AUDIO_R1-D9_08_Track08_wav
MIDI-Unprocessed_24_R1_2011_MID--AUDIO_R1-D9_10_Track10_wav
MIDI-Unprocessed_25_R3_2011_MID--AUDIO_R3-D9_03_Track03_wav
MIDI-Unprocessed_Chamber5_MID--AUDIO_18_R3_2018_wav--1
MIDI-Unprocessed_R1_D1-1-8_mid--AUDIO-from_mp3_04_R1_2015_wav--4
MIDI-Unprocessed_R1_D1-1-8_mid--AUDIO-from_mp3_05_R1_2015_wav--2
MIDI-Unprocessed_R1_D1-1-8_mid--AUDIO-from_mp3_05_R1_2015_wav--5
MIDI-Unprocessed_R1_D1-1-8_mid--AUDIO-from_mp3_08_R1_2015_wav--1
MIDI-Unprocessed_R1_D2-13-20_mid--AUDIO-from_mp3_15_R1_2015_wav--1
MIDI-Unprocessed_R1_D2-13-20_mid--AUDIO-from_mp3_18_R1_2015_wav--2
MIDI-Unprocessed_R1_D2-13-20_mid--AUDIO-from_mp3_18_R1_2015_wav--4
MIDI-Unprocessed_R1_D2-21-22_mid--AUDIO-from_mp3_21_R1_2015_wav--4
MIDI-Unprocessed_R1_D2-21-22_mid--AUDIO-from_mp3_22_R1_2015_wav--5
MIDI-Unprocessed_R2_D1-2-3-6-7-8-11_mid--AUDIO-from_mp3_07_R2_2015_wav--1
MIDI-Unprocessed_R2_D1-2-3-6-7-8-11_mid--AUDIO-from_mp3_08_R2_2015_wav--1
MIDI-Unprocessed_R2_D2-12-13-15_mid--AUDIO-from_mp3_15_R2_2015_wav--1
MIDI-Unprocessed_R2_D2-12-13-15_mid--AUDIO-from_mp3_15_R2_2015_wav--2
MIDI-Unprocessed_R2_D2-19-21-22_mid--AUDIO-from_mp3_22_R2_2015_wav--4
MIDI-Unprocessed_Recital1-3_MID--AUDIO_01_R1_2018_wav--1
MIDI-Unprocessed_Recital1-3_MID--AUDIO_02_R1_2018_wav--4
MIDI-Unprocessed_Recital1-3_MID--AUDIO_03_R1_2018_wav--2

MIDI-Unprocessed_Recital1-3_MID--AUDIO_03_R1_2018_wav--5
MIDI-Unprocessed_Recital12_MID--AUDIO_12_R1_2018_wav--2
MIDI-Unprocessed_Recital13-15_MID--AUDIO_13_R1_2018_wav--2
MIDI-Unprocessed_Recital13-15_MID--AUDIO_14_R1_2018_wav--4
MIDI-Unprocessed_Recital16_MID--AUDIO_16_R1_2018_wav--2
MIDI-Unprocessed_Recital17-19_MID--AUDIO_17_R1_2018_wav--1
MIDI-Unprocessed_Recital17-19_MID--AUDIO_19_R1_2018_wav--6
MIDI-Unprocessed_Recital4_MID--AUDIO_04_R1_2018_wav--5
MIDI-Unprocessed_Recital5-7_MID--AUDIO_07_R1_2018_wav--1
MIDI-Unprocessed_Recital5-7_MID--AUDIO_07_R1_2018_wav--3
MIDI-Unprocessed_SMF_05_R1_2004_01_ORIG_MID--AUDIO_05_R1_2004_03_Track03_wav
MIDI-Unprocessed_SMF_07_R1_2004_01_ORIG_MID--AUDIO_07_R1_2004_06_Track06_wav
MIDI-Unprocessed_SMF_07_R1_2004_01_ORIG_MID--AUDIO_07_R1_2004_12_Track12_wav
MIDI-Unprocessed_SMF_12_01_2004_01-05_ORIG_MID--AUDIO_12_R1_2004_07_Track07_wav
MIDI-Unprocessed_SMF_13_01_2004_01-05_ORIG_MID--AUDIO_13_R1_2004_09_Track09_wav
MIDI-Unprocessed_SMF_16_R1_2004_01-08_ORIG_MID--AUDIO_16_R1_2004_13_Track13_wav
MIDI-Unprocessed_SMF_17_R1_2004_01-02_ORIG_MID--AUDIO_20_R2_2004_02_Track02_wav
MIDI-Unprocessed_SMF_17_R1_2004_04_ORIG_MID--AUDIO_17_R1_2004_09_Track09_wav
MIDI-Unprocessed_SMF_22_R1_2004_01-04_ORIG_MID--AUDIO_22_R1_2004_03_Track03_wav
MIDI-Unprocessed_Schubert7-9_MID--AUDIO_11_R2_2018_wav
MIDI-Unprocessed_XP_01_R1_2004_03_ORIG_MID--AUDIO_01_R1_2004_04_Track04_wav
MIDI-Unprocessed_XP_14_R1_2004_01-03_ORIG_MID--AUDIO_14_R1_2004_01_Track01_wav
MIDI-Unprocessed_XP_14_R2_2004_01_ORIG_MID--AUDIO_14_R2_2004_04_Track04_wav
MIDI-Unprocessed_XP_16_R2_2004_01_ORIG_MID--AUDIO_16_R2_2004_03_Track03_wav
MIDI-Unprocessed_XP_18_R1_2004_01-02_ORIG_MID--AUDIO_18_R1_2004_02_Track02_wav
MIDI-Unprocessed_XP_21_R1_2004_03_ORIG_MID--AUDIO_21_R1_2004_04_Track04_wav
MIDI-Unprocessed_XP_22_R2_2004_01_ORIG_MID--AUDIO_22_R2_2004_01_Track01_wav
ORIG-MIDI_01_7_10_13_Group_MID--AUDIO_02_R3_2013_wav--1
ORIG-MIDI_01_7_10_13_Group_MID--AUDIO_08_R3_2013_wav--3
ORIG-MIDI_01_7_7_13_Group__MID--AUDIO_12_R1_2013_wav--3
ORIG-MIDI_01_7_7_13_Group__MID--AUDIO_12_R1_2013_wav--5
ORIG-MIDI_01_7_7_13_Group__MID--AUDIO_13_R1_2013_wav--2
ORIG-MIDI_02_7_10_13_Group_MID--AUDIO_11_R3_2013_wav--1
ORIG-MIDI_02_7_10_13_Group_MID--AUDIO_12_R3_2013_wav--4
ORIG-MIDI_02_7_6_13_Group__MID--AUDIO_07_R1_2013_wav--1
ORIG-MIDI_02_7_6_13_Group__MID--AUDIO_07_R1_2013_wav--3
ORIG-MIDI_02_7_6_13_Group__MID--AUDIO_08_R1_2013_wav--1
ORIG-MIDI_02_7_7_13_Group__MID--AUDIO_15_R1_2013_wav--1
ORIG-MIDI_02_7_7_13_Group__MID--AUDIO_15_R1_2013_wav--2
ORIG-MIDI_02_7_7_13_Group__MID--AUDIO_16_R1_2013_wav--3
ORIG-MIDI_02_7_7_13_Group__MID--AUDIO_18_R1_2013_wav--3
ORIG-MIDI_02_7_8_13_Group__MID--AUDIO_11_R2_2013_wav--1
ORIG-MIDI_03_7_10_13_Group_MID--AUDIO_17_R3_2013_wav--2

C. RC20 Presets

This appendix list the presets that was used for the effector-plugin RC-20 when rendering musical data. The presets listed in C1 are default to RC-20 and come with the basic installation. The presets listed in C2 are downloaded from splice[1] and are all found in the *!LLMIND - M!ND WARP* pack.

C.1. Default Presets

808Crush.fxb
Casette1stGeneration.fxb
CasetteBadBatteries.fxb
CasetteGenerationsLater.fxb
DepthForPlucks.fxb
DistBrightTubes.fxb
DistFatIron.fxb
DoubleTracking1.fxb
DoubleTracking2.fxb
FlutterVobe.fxb
FluxChorus.fxb
FluxSpace.fxb
FluxTremolo.fxb
GazingGuitar.fxb
GhettoBlaster.fxb
GimmeSub.fxb
HotlinePhase.fxb
LoudnessButton.fxb
Lush&CrunchGuitar1.fxb
Lush&CrunchGuitar2.fxb
MagnitudeTransition1.fxb
MagnitudeTransition2.fxb
MagnitudeTransition3.fxb
NoMoreBoringPads.fxb
PadMagic.fxb
PhasingFun.fxb

[1] https://splice.com/home

RetiredAmp.fxb
SadPiano.fxb
Sampler12bit.fxb
Sampler8bit.fxb
SleeplikePoison.fxb
StrangeSpace.fxb
TheBassIsAlive.fxb
TooBright.fxb
TransmittingFromSpace.fxb
VHS.fxb
Vinyl1.fxb
Vinyl2.fxb
Vinyl3.fxb
Vinyl4.fxb
WarmSummerDay.fxb
Wow&Flutter.fxb

C.2. Splice Presets

IM_BitOfLuck.fxb
IM_Catastrophic.fxb
IM_Crackerjax.fxb
IM_GetItDone.fxb
IM_InTheBathroom.fxb
IM_InTheDen.fxb
IM_Juggernuts.fxb
IM_LilDirty.fxb
IM_MetalMarchers.fxb
IM_Pandora.fxb
IM_SayItInTheBack.fxb
IM_SimpleVintag.fxb
IM_Tangerine.fxb
IM_WeekendAtBernies.fxb

D. SerumFX Presets

The following are all the presets used for SerumFX in the experiments for this thesis. They can all be found in the default library for SerumFX.

8 bit.fxb
Acoustic.fxb
Alla Prima.fxb
Artistic Grace.fxb
Blessed.fxb
Bliss.fxb
Digital Bells.fxb
Distorgan.fxb
E Piano.fxb
Early Morning.fxb
Eighty Tree.fxb
Elementary.fxb
Elizabeth.fxb
Enchanted Blush.fxb
Ethereal Celesta.fxb
Fake Strings 1.fxb
Feng Shui.fxb
Furios.fxb
Inner Rythm.fxb
Julia.fxb
Kalimba.fxb
Karat.fxb
Kawaii.fxb
Lake of Dreams.fxb
Lead Back.fxb
Leviathan.fxb
Love You.fxb
Low Key.fxb
Low Life.fxb
Majestic Twelve.fxb
Mallet.fxb
Marble Floors.fxb
Melody Master".fxb

Mentality.fxb
MetalliPlucker.fxb
Nightcrawler.fxb
No Complaints.fxb
Noble.fxb
Organico.fxb
Passionate.fxb
Pizzicato.fxb
Pretty Lockett.fxb
Radiance.fxb
Red Bottoms.fxb
Riot.fxb
Shadow of The Beast.fxb
Shuffle.fxb
Smile.fxb
Sneakin.fxb
Stringer.fxb
Synthar.fxp
The Motto.fxb
The Projects.fxb
This Part.fxb
Traditional Rhodes.fxb
Trompette.fxb
Twisted Fantasy.fxb
Virus.fxb
Vision.fxb
Watching Movies.fxb
YellowTeeth.fxb

E. Serum Presets

The following are all the presets used for Serum in the experiments for this thesis. They can all be found in the default library for Serum. They are grouped by instrument type.

E.1. Bass

Ampology.fxp
Analog Pluck.fxp
Bandpass.fxp
Bass Chop 1.fxp
Classic but more.fxp
Deep Sync.fxp
Fm Wob.fxp
Funk Mogue.fxp
Jackson.fxp
Liquid Neuro.fxp
Mini Bob.fxp
Modern Fapping.fxp
Msaw fun.fxp
Mute Bass Gtr.fxp
Short Bass.fxp
Some Grit.fxp
Sqr Bass.fxp
Talk Wobz.fxp

E.2. Leads

8bit.fxp
A bit of luck.fxp
Araba.fxp
BottleBlower.fxp
Brave Heart.fxp
Brutality.fxp
C64 LEAD.fxp

Caliber.fxp
Dubrill.fxp
D_LEAD 2.fxp
Epic Lead.fxp
Festival Beez.fxp
In your face.fxp
Jacked.fxp
Jerk.fxp
Kidsos.fxp
Magic Pipe.fxp
Noor Santor.fxp
Peter Piper.fxp
Spookytastic.fxp
Theremax.fxp
Tokyo.fxp

E.3. Pads

arctic wind.fxp
BenjiPad.fxp
Big Minor Seventh Pad.fxp
Broken Bows.fxp
Celestial Light.fxp
Centipad.fxp
Emotional More.fxp
Epic Bandpass Pad.fxp
Event Horizon.fxp
Falling in Reverse.fxp
Gated Pad.fxp
Hold This Note.fxp
Holy Place.fxp
Hopeful.fxp
Horns of Fear.fxp
ISS Sunrise.fxp
Lost Souls.fxp
Lux.fxp
Menthol Strings.fxp
Mermaid.fxp
Monkchoir.fxp
Night Desert MW.fxp
Panic Run.fxp
Quasar.fxp

Raspy Strings.fxp
Serenity.fxp
Tibet Wind Atmos.fxp

E.4. Plucks

Big bells.fxp
Cliche Doot.fxp
Crushed Pluck.fxp
Dyk Pyk.fxp
FMmy.fxp
Mallety.fxp
Nothing Special.fxp
Plucked.fxp
Pluto.fxp
Power and progress.fxp
Snickers.fxp
wierd.fxp

E.5. Synths

Drawbar.fxp
Electric Light.fxp
Juno Bells.fxp
Monothone.fxp
Morricone.fxp
Mr Wiggles.fxp
Multicolor.fxp
Paving Rhodes.fxp
Ponderosa.fxp
Runtheharm.fxp
Sickly Keys.fxp
Vintage Bells.fxp
Vintage Space.fxp

F. Kontakt Presets

The list below present all presets used for Kontakt in the experiments related to this thesis. All presets are found in the *Komplete 13* bundle for Kontakt, and they are listed according to the package they are found in.

F.1. Basic

`Clavinet.fxp`
`E-Piano.fxp`
`Funk Bass.fxp`
`Jazz Guitar.fxp`
`Jazz Organ.fxp`
`Muted Trumpet.fxp`
`Ragtime Piano.fxp`
`Rock Guitar.fxp`
`Upright Bass.fxp`

F.2. Brass

`AltoSax.fxp`
`BaritoneSax.fxp`
`Bass Trombone.fxp`
`Flugelhorn.fxp`
`MuteTrumpet.fxp`
`TenorSax.fxp`
`TenorTrombone.fxp`
`Trumpet1.fxp`
`Trumpet2.fxp`
`Tuba.fxp`

F.3. Retro Machines

`Additive Piano.fxp`
`Catchy Filter Bass.fxp`

Chroma Brass 1.fxp
Chroma Brass 2.fxp
Chroma Clavinet.fxp
Chroma Flute.fxp
Chroma Keys 1.fxp
Chroma Keys 2.fxp
Chroma Pad 1.fxp
CP11 Clavi.fxp
CP11 Harpsi.fxp
Dirty Saw Lead.fxp
DX7 EP.fxp
Electric Grand.fxp
EP10 Piano.fxp
EP20 Harpsi.fxp
Forma Choir.fxp
Glassy Bell.fxp
Innocent Lead.fxp
Memory Bell.fxp
Memory Dark.fxp
Memory Sneaky.fxp
Memory Strings.fxp
Memory Syncstab.fxp
Metallic Lead.fxp
Miami DX.fxp
Mini Lead 01.fxp
Mini Lead 04.fxp
Mini Lead 07.fxp
Phuture Bell.fxp
Retro String 1.fxp
Retro Strings 1.fxp
RMI Lute.fxp
RMI Piano.fxp
Straight Pad.fxp

F.4. Scarbee Mark

Blue Ballad.fxp
Funky Rio.fxp
Pan Cake Kore.fxp
Pink Playboy.fxp
Steal A Dame.fxp
Stretch Tuned.fxp

Whats Up.fxp

F.5. Scarbee Rickenbauer Bass

Both Pure DI.fxp
British Setting.fxp
Double Date.fxp
Flame.fxp
Hammer Head.fxp
Neck Pure DI.fxp
Necklace.fxp
The Raven.fxp
Tight Rocker.fxp
Tweedman.fxp

F.6. Vintage Organs

Basic Gospel 1.fxp
Basic Gospel 3.fxp
Basic Jazz 1.fxp
Basic Jazz 3.fxp
Basic Rock 1.fxp
Basic Rock 3.fxp
Basoon 8.fxp
Blockfloete 4.fxp
Celestial Voices.fxp
Child In Time.fxp
Cornichons.fxp
Four On Six.fxp
Get Out of This Place.fxp
Gimme Some Lovin.fxp
Groove Holmes.fxp
Hunk O Funk.fxp
Im Down.fxp
Je taime.fxp
Late Night.fxp
Little Girl.fxp
No Woman No Cry.fxp
Oblighetto.fxp
One Organ.fxp
Praise The Lord.fxp
The End.fxp

```
ToneWheel.fxp
Trumpet.fxp
When The Musics Over.fxp
```

G. Example of an AOP file

Figure G.1 displays a short example of a Audio Operation Pipelines (AOP) file, as detailed in chapter 5.

```
{
    "id": "PipelinesId",
    "pipelines": [
        {
            "id": "Track1",
            "bpm": 120,
            "data": [
                {
                    "file": "Datasets/ExampleMidi.mid",
                    "generator": {
                        "plugin": "/Library/Audio/Plug-Ins/ExampleGenerator.vst",
                        "preset": "Presets/Generators/ExampleGenerator/ExampleGeneratorPreset.fxp"
                    },
                    "effects": null
                }
            ]
        },
        {
            "id": "Track2",
            "bpm": 100,
            "data": [
                {
                    "file": "Datasets/ExampleAudio1.wav",
                    "generator": null,
                    "effects": [
                        {
                            "plugin": "/Library/Audio/Plug-Ins/ExampleEffector.vst",
                            "preset": "Presets/Effectors/Example/Effector/ExampleEffectorPreset1.fxp"
                        },
                        {
                            "plugin": "/Library/Audio/Plug-Ins/ExampleEffector.vst",
                            "preset": "Presets/Effectors/Example/Effector/ExampleEffectorPreset2.fxp"
                        }
                    ]
                },
                {
                    "file": "Datasets/ExampleAudio2.wav",
                    "generator": null,
                    "effects": null
                }
            ]
        }
    ]
}
```


Figure G.1.: An AOP file, commanding the rendering of MIDI trough a generator and audio through an effector, combined with audio without further rendering, respectively.

H. Improved/Worsened Counts - Experiment 2 and 3

This appendix list tables giving counts of improved and worsened validation scores in comparison to the baseline for experiment 2 and 3. Tables are given for the MAESTRO (table H.1), MusicNet (table H.2), and URMP datasets (table H.3), while the similar table for GuitarSet is given in chapter 7.

Table H.1.: Improved/Worsened Count MAESTRO (E2 & E3)

	RC-20 (E2.1)	SerumFX (E2.2)	Kontakt (E3.1)
Onset F1	2/135	2/135	0/137
Onset Recall	0/137	0/137	0/137
Onset Precision	6/131	20/117	0/137
Onset + offset F1	0/137	0/137	0/137
Frame F1	0/137	1/136	0/137
Frame Recall	0/137	13/124	1/136
Frame Precision	0/137	5/132	0/137

Table H.2.: Improved/Worsened Count MusicNet (E2 & E3)

	RC-20 (E2.1)	SerumFX (E2.2)	Kontakt (E3.1)
Onset F1	2/13	1/14	0/15
Onset Recall	0/15	1/14	1/14
Onset Precision	3/12	4/11	1/14
Onset + offset F1	3/12	2/13	1/14
Frame F1	2/13	3/12	0/15
Frame Recall	4/11	5/10	0/15
Frame Precision	4/11	4/11	0/15

Table H.3.: Improved/Worsened Count URMP (E2 & E3)

	RC-20 (E2.1)	SerumFX (E2.2)	Kontakt (E3.1)
Onset F1	0/9	0/9	0/9
Onset Recall	0/9	0/9	0/9
Onset Precision	0/9	0/9	0/9
Onset + offset F1	0/9	0/9	0/9
Frame F1	1/8	0/9	0/9
Frame Recall	1/8	0/9	0/9
Frame Precision	1/8	0/9	1/8

I. Best/Worst Validation Scores - Experiment 2 and 3

This appendix shows tables with the best & worst Frame F1 for E2 and E3 with the MAESTRO (table I.1, and I.2), MusicNet (table I.3) and URMP (table I.4) datasets. The similar table for GuitarSet is given in chapter 7. For each table, the three files on the top are the ones with the best Frame F1 scores for a given plugin, while the bottom three are the files with the worst Frame F1 scores. The score for a given file is given in parenthesis behind its name.

Table I.1.: Best & Worst Frame F1 MAESTRO (Basline and RC-20)

	Baseline (E0)	RC-20 (E2.1)
1.	MIDI-Unprocessed_R1_D1-9-12_mid–AUDIO-from_mp3_09_R1_2015_wav–4 (0.8974)	MIDI-Unprocessed_R1_D1-9-12_mid–AUDIO-from_mp3_09_R1_2015_wav–4 (0.8617)
2.	MIDI-Unprocessed_R1_D1-1-8_mid–AUDIO-from_mp3_04_R1_2015_wav–3 (0.8832)	MIDI-Unprocessed_R1_D1-1-8_mid–AUDIO-from_mp3_04_R1_2015_wav–3 (0.8606)
3.	ORIG-MIDI_03_7_8_13_Group__MID–AUDIO_18_R2_2013_wav–3 (0.8635)	MIDI-Unprocessed_08_R1_2009_01-04_ORIG_MID–AUDIO_08_R1_2009_08_R1_2009_02_WAV (0.8341)
-3.	MIDI-Unprocessed_09_R3_2008_01-07_ORIG_MID–AUDIO_09_R3_2008_wav–7 (0.6925)	MIDI-Unprocessed_09_R1_2008_01-05_ORIG_MID–AUDIO_09_R1_2008_wav–4 (0.6361)
-2.	MIDI-Unprocessed_XP_19_R1_2004_01-02_ORIG_MID–AUDIO_19_R1_2004_03_Track03_wav (0.6765)	MIDI-Unprocessed_12_R1_2008_01-04_ORIG_MID–AUDIO_12_R1_2008_wav–3 (0.6329)
-1.	MIDI-UNPROCESSED_04-08-12_R3_2014_MID–AUDIO_12_R3_2014_wav–2 (0.6707)	MIDI-Unprocessed_09_R3_2008_01-07_ORIG_MID–AUDIO_09_R3_2008_wav–7 (0.6274)

Table I.2.: Best & Worst Frame F1 MAESTRO (SerumFX and Kontakt)

	SerumFX (E2.2)	Kontakt (E3.1)
1.	MIDI-Unprocessed_R1_D1-9-12_mid–AUDIO-from_mp3_09_R1_2015_wav–4 (0.8585)	ORIG-MIDI_03_7_8_13_Group__MID–AUDIO_18_R2_2013_wav–3 (0.7383)
2.	MIDI-Unprocessed_R1_D1-1-8_mid–AUDIO-from_mp3_04_R1_2015_wav–3 (0.8546)	MIDI-Unprocessed_12_R1_2006_01-08_ORIG_MID–AUDIO_12_R1_2006_03_Track03_wav (0.7377)
3.	MIDI-Unprocessed_08_R1_2009_01-04_ORIG_MID–AUDIO_08_R1_2009_08_R1_2009_02_WAV (0.8425)	MIDI-Unprocessed_08_R1_2009_01-04_ORIG_MID–AUDIO_08_R1_2009_08_R1_2009_02_WAV (0.7302)
-3.	MIDI-UNPROCESSED_04-08-12_R3_2014_MID–AUDIO_12_R3_2014_wav–2 (0.6542)	MIDI-Unprocessed_03_R3_2011_MID–AUDIO_R3-D1_03_Track03_wav (0.4362)
-2.	MIDI-Unprocessed_12_R1_2008_01-04_ORIG_MID–AUDIO_12_R1_2008_wav–3 (0.6500)	MIDI-Unprocessed_02_R1_2008_01-05_ORIG_MID–AUDIO_02_R1_2008_wav–3 (0.4331)
-1.	MIDI-Unprocessed_09_R1_2008_01-05_ORIG_MID–AUDIO_09_R1_2008_wav–4 (0.6458)	MIDI-Unprocessed_09_R1_2008_01-05_ORIG_MID–AUDIO_09_R1_2008_wav–4 (0.4289)

Table I.3.: Best & Worst Frame F1 MusicNet

	Baseline (E0)	RC-20 (E2.1)	SerumFX (E2.2)	Kontakt (E3.1)
1.	2289 (0.7740)	2611 (0.7497)	2611 (0.7693)	2611 (0.5970)
2.	2611 (0.7569)	2289 (0.7243)	2308 (0.7212)	2300 (0.5786)
3.	2198 (0.7059)	2198 (0.6954)	2289 (0.7113)	2198 (0.5556)
-3.	2477 (0.5400)	2160 (0.4150)	2477 (0.5031)	2160 (0.2935)
-2.	2160 (0.4145)	2477 (0.2620)	2160 (0.3842)	2477 (0.0867)
-1.	2466 (0.2671)	2466 (0.2396)	2466 (0.1275)	2466 (0.0856)

Table I.4.: Best & Worst Frame F1 URMP

	Baseline (E0)	RC-20 (E2.1)	SerumFX (E2.2)	Kontakt (E3.1)
1.	13_Hark_vn_vn_va (0.7237)	02_Sonata_vn_vn (0.6931)	02_Sonata_vn_vn (0.6422)	02_Sonata_vn_vn (0.6669)
2.	02_Sonata_vn_vn (0.6693)	13_Hark_vn_vn_va (0.6319)	13_Hark_vn_vn_va (0.6051)	13_Hark_vn_vn_va (0.5499)
3.	01_Jupiter_vn_vc (0.6205)	01_Jupiter_vn_vc (0.5136)	01_Jupiter_vn_vc (0.5493)	39_Jerusalem_vn_vn_va_sax_db (0.4195)
-3.	38_Jerusalem_vn_vn_va_vc_db (0.5430)	25_Pirates_vn_vn_va_sax (0.4265)	31_Slavonic_tpt_tpt_hn_tbn (0.4274)	01_Jupiter_vn_vc (0.3418)
-2.	12_Spring_vn_vn_vc (0.5158)	24_Pirates_vn_vn_va_vc (0.3742)	12_Spring_vn_vn_vc (0.3777)	12_Spring_vn_vn_vc (0.3043)
-1.	24_Pirates_vn_vn_va_vc (0.4147)	12_Spring_vn_vn_vc (0.3253)	24_Pirates_vn_vn_va_vc (0.3526)	24_Pirates_vn_vn_va_vc (0.2942)

Dyvik, Klaus & Waaler, Fredrik

Utilizing Audio Plugins for Automatic Music Transcription

NTNU
Norwegian University of
Science and Technology