

Mats Jaer Nottveit
Håkon Anders Strømsodd

Automatic Music Transcription Using Self-Supervised Learning

Combining Simple Siamese Representation
Learning and Automatic Music Transcription

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2023



Norwegian University of
Science and Technology

Mats Jaer Nottveit
Håkon Anders Strømsodd

Automatic Music Transcription Using Self-Supervised Learning

Combining Simple Siamese Representation Learning
and Automatic Music Transcription

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2023

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



Mats Jaer Nottveit and Håkon Anders Strømsodd

Automatic Music Transcription Using Self-Supervised Learning

Combining Simple Siamese Representation Learning and Automatic Music Transcription

Master's Thesis in Computer Science, June 2023

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

Automatic Music Transcription (AMT) is a technique that involves utilising an algorithm to convert a musical recording into a musical notation format such as **Musical Instrument Digital Interface (MIDI)** or sheet music. **AMT** systems with good performance can provide value to musicians of all levels and can save considerable amounts of time when a transcription is needed. However, **AMT** is a difficult task, even for humans, and has only seen significant leaps in performance in the past five years (2018-2023). This recent interest in **AMT** and the accompanying increase in state-of-the-art performance is primarily attributed to the improved availability of training data and recent developments in deep learning.

Current state-of-the-art **AMT** systems perform very well in single-instrument settings. However, most of these models are limited to piano transcriptions since the available datasets primarily consist of piano performances. Training data containing multiple instruments is still severely limited, which negatively impacts the performance of multi-instrument **AMT** models. Current state-of-the-art multi-instrument transcription models perform around 20% worse on multi-instrument datasets than on single-instrument datasets. This supports the observation that the lack of multi-instrument training data is still the biggest roadblock for further improvement of multi-instrument **AMT** models.

To address this lack of data, this thesis aimed to investigate the use of self-supervised learning with **AMT** and observe how this affects the performance of the transcription model. In the experiments conducted in this thesis, the Simple Siamese representation learning algorithm was used to enable pre-training on unlabelled data. Unlike any previous related work, this method allows for the use of datasets containing only audio in addition to traditional **AMT** datasets, thereby expanding the range of usable training data.

The experiments carried out in this thesis investigate different ways of combining the Simple Siamese learning algorithm with the existing Onsets and Frames **AMT** model. The majority of these experiments performed on average worse when compared to traditional **AMT** models not utilising the Simple Siamese training algorithm. The best model from the experiments achieved multi-instrument performance metrics 2.5% lower than state-of-the-art models. While some experiments proved that the pre-trained model was able to learn some musical features, it was not enough to enhance transcription performance. These results point to the conclusion that combining the Simple Siamese training algorithm with the Onsets and Frames architecture does not contribute to a positive increase in performance.

All transcription models from the experiments were trained on the Slakh2100 dataset, while the **MTG-Jamendo (MTG-J)** dataset containing only audio recordings was utilised during pre-training. An open-source PyTorch data loader specifically for use with the **MTG-J** dataset was created, enabling fast and easy loading of audio. Additionally, this data loader provided parallelisation during loading and is capable of speeding up loading times during training by a factor of 10.

Sammendrag

Automatisk Musikktranskripsjon (AMT) er en teknikk som innebærer å bruke en algoritme til å konvertere et lydopptak til et musikknotasjonsformat som for eksempel MIDI eller noter. AMT-systemer med god ytelse kan være til nytte for musikere på alle nivåer og kan være svært tidsbesparende dersom en transkripsjon av et musikkstykke er nødvendig. AMT er imidlertid ansett som en svært vanskelig oppgave, selv for mennesker, og ordentlige fremskritt innen feltet har kun skjedd de siste fem årene (2018-2023). Denne nylig økte interessen for AMT og de tilhørende fremskrittene i ytelsen til de toppmoderne AMT-systemene skyldes hovedsakelig økt tilgjengelighet av treningsdata samt nylig utvikling innenfor dyp læring.

De nåværende toppmoderne AMT-systemene viser svært gode resultater i enkeltinstrumentstranskripsjon. De fleste av disse modellene er imidlertid begrenset til pianotranskripsjon, ettersom store deler av den tilgjengelige treningsdataen kun inneholder piano. Treningsdata som inneholder flere enn ett instrument er fortsatt veldig begrenset, noe som negativt påvirker ytelsen til AMT-systemene for flerinstrumentstranskripsjon. De nåværende toppmoderne AMT-systemene for flerinstrumentstranskripsjon viser om lag 20% dårligere resultater på datasett med flere instrumenter enn på datasett med kun ett instrument. Dette støtter påstanden om at mangelen på treningsdata fortsatt er den største hindringen for ytterligere forbedring av flerinstrumentstranskripsjon.

For å takle denne mangelen på data, hadde denne masteroppgaven som mål å undersøke bruk av selvstyrte læringsalgoritmer sammen med AMT for å observere hvordan dette påvirker ytelsen til en transkripsjonsmodell. Under eksperimentene som ble gjennomført i denne masteroppgaven ble SimSiam-algoritmen brukt til å muliggjøre trening på umerket data. I motsetning til tidligere arbeid innenfor AMT tillater denne treningsmetoden bruk av umerket data (data som kun inneholder lydopptak) i tillegg til tradisjonelle AMT-datasett. Dette utvider dermed utvalget av tilgjengelig treningsdata.

Eksperimentene gjennomført i denne masteroppgaven undersøker ulike måter man kan kombinere SimSiam-algoritmen med den eksisterende AMT-modellen «Onsets and Frames». De fleste av disse eksperimentene resulterte i gjennomsnittlig dårligere ytelse enn tradisjonelle AMT-modeller som ikke benytter seg av SimSiam-algoritmen. Den beste modellen fra eksperimentene oppnådde om lag 2,5% dårligere resultater enn de toppmoderne systemene. Selv om noen av eksperimentene beviste at den forhåndstrengte modellen greide å lære visse musikalske trekk, var ikke dette tilstrekkelig til å øke ytelsen til modellen. Dette peker mot konklusjonen om å kombinere SimSiam-algoritmen med den eksisterende «Onsets and Frames»-modellen ikke bidrar til økt ytelse.

Alle transkripsjonsmodellene fra disse eksperimentene ble trent på Slakh2100-datasettet samt MTG-Jamendo-datasettet (MTG-J) der sistnevnte kun inneholder lydklipp. Under arbeidet ble en PyTorch-datainnlaster med åpen kildekode utviklet, spesifikt for lasting av MTG-J-datasettet. Denne datainnlasteren gjør at data kan hentes raskt og enkelt under trening og sørger for at parallellisering kan utnyttes. Sistnevnte medfører at innlastingstidene under trening reduseres med en faktor på 10.

Preface

This Master's Thesis was written as a collaborative project in Computer Science at the Norwegian University of Science and Technology (NTNU) during the spring semester of 2023. The work was conducted at the Department of Computer Science and was supervised by Professor Björn Gambäck. It is based on our preliminary report *Automatic Music Transcription Using Deep Learning* (Nottveit and Strømsodd, 2022), written during the fall semester of 2022.

First, special thanks go to our supervisor Professor Björn Gambäck for providing excellent guidance, support, and expertise throughout the work and for allowing us to explore the intersection of music and machine learning.

Additionally, we would like to thank Henrik Grønbech for initiating research into AMT at NTNU in his Master's thesis and for allowing us to reuse his implementation of the extended Onsets and Frames architecture. This allowed us to focus our attention on the self-supervised system and its incorporation into the extended Onsets and Frames architecture.

We are grateful to the IDUN HPC group at NTNU for providing us with the hardware necessary to investigate the combination of self-supervised learning and AMT (Själänder et al., 2019).

Finally, we would like to extend our thanks to our friends, family and fellow students. Their support, engagement, and insightful discussions have been invaluable.

Mats Jaer Nottveit and Håkon Anders Strømsodd
Trondheim, 1st June 2023

Contents

Abstract	i
Sammendrag	ii
Preface	iii
List of Figures	x
List of Tables	xi
Acronyms	xiii
1 Introduction	1
1.1 Background and Motivation	1
1.2 Goal and Research Questions	2
1.3 Research Method	3
1.4 Contributions	4
1.5 Thesis Structure	5
2 Background and Theory	7
2.1 Automatic Music Transcription	7
2.2 AI and Machine Learning	8
2.2.1 Self-Supervised Learning	8
2.2.2 Deep Learning	9
2.2.3 Transposed Convolution	11
2.2.4 Residual Neural Network	11
2.2.5 Siamese Neural Networks	12
2.2.6 Pre-Training	12
2.2.7 Image Augmentations	13
2.3 Music Theory	13
2.4 Fourier Transform	14
2.5 Audio Representation	14
2.5.1 Staff Notation	14
2.5.2 Waveform	15
2.5.3 Spectrograms	15

Contents

2.5.4	MIDI	16
2.6	Evaluation	17
2.6.1	Precision	17
2.6.2	Recall	18
2.6.3	F1 Score	18
3	Datasets	19
3.1	MAPS	19
3.2	MAESTRO	19
3.3	GiantMIDI-Piano	19
3.4	Million Song Dataset	20
3.5	MusicNet	20
3.6	Lakh	20
3.7	Slakh2100	20
3.8	Cerberus4	21
3.9	URMP	21
3.10	GuitarSet	21
3.11	MTG-Jamendo	21
3.12	MNIST	21
4	Related Work	23
4.1	Preliminary Approaches	23
4.2	Onsets and Frames	24
4.3	MT3	26
4.4	NoteEM	28
4.5	Comparison	30
4.6	Self-Supervised Pre-Training	31
4.6.1	Simple Siamese Representation Learning	31
4.6.2	BYOL Representation Learning	31
5	Architecture	33
5.1	Pre-Processing	34
5.2	Self-Supervised System	35
5.2.1	Data Augmentations	36
5.2.2	Encoder Architectures	37
5.3	Fully Supervised System	38
5.4	Post-Processing	41
6	Experiments and Results	43
6.1	Experimental Plan	43
6.1.1	Baseline Experiments	43
6.1.2	Experiments on Input Replacement	44
6.1.3	Experiments on Concatenation	45
6.1.4	Experiments on Augmentation Selection Mode	46

6.1.5	Experiments on Size of Unlabelled Dataset	46
6.2	Experimental Setup	47
6.2.1	Datasets	47
6.2.2	Third-Party Libraries	47
6.2.3	Spectrogram Generation	47
6.2.4	Network Parameters	48
6.2.5	Hardware	49
6.3	Experimental Results	49
6.3.1	Experiment 0: Simple Siamese Verification	50
6.3.2	Experiment 1: Onsets and Frames Baseline	50
6.3.3	Experiment 2: Input Replacement	51
6.3.4	Experiment 3: Early Concatenation	51
6.3.5	Experiment 4: Late Concatenation	52
6.3.6	Experiment 5: Augmentation Selection Mode	52
6.3.7	Experiment 6: More Unlabelled Data	52
7	Evaluation and Discussion	53
7.1	Evaluation	53
7.2	Discussion	59
7.2.1	Discussion of Results	59
7.2.2	Discussion of Architecture	61
7.2.3	Discussion of Data	63
8	Conclusion and Future Work	65
8.1	Conclusion	65
8.2	Future Work	67
	Bibliography	69
	A List of Third-Party Python Libraries	73
	B Detailed Projector Architectures	75
	C Individual Results from Repeated Runs	79
	D F1 Score Plots for Remaining Experiments	81
	E Example Transcriptions of Validation Data	83

List of Figures

2.1	Multilayer perceptron	10
2.2	LSTM unit	10
2.3	Transposed convolutional layer	11
2.4	Comparison of a CNN and a residual neural network	12
2.5	Staff notation	15
2.6	Example spectrogram	16
2.7	Piano roll	17
4.1	Original Onsets and Frames architecture	25
4.2	Extended Onsets and Frames architecture	27
4.3	U-Net architecture	28
4.4	Note _{EM} training algorithm	29
5.1	Overall architecture of the system	33
5.2	Simple Siamese training algorithm	35
5.3	Onsets and Frames architecture with sub-network	39
5.4	Sub-networks present in the modified Onsets and Frames model	40
5.5	Onsets and Frames architecture for the late concatenation approach	41
5.6	Model predictions and resulting MIDI after post-processing	42
7.1	Experiment 1b example predictions	54
7.2	F ₁ scores for experiment 1b and 1c	54
7.3	Experiment 2d and 2e example predictions	55
7.4	F ₁ scores for experiment 2c, 3a and 3b	56
7.5	F ₁ scores for experiment 4a and 4b	57
D.1	F ₁ scores for experiment 2b and 2d	81
D.2	F ₁ scores for experiment 5a and 5b	81
D.3	F ₁ scores for experiment 6a and 6b	82
E.1	Transcribed validation spectrograms for experiment 1b	83
E.2	Transcribed validation spectrograms for experiment 1c	84
E.3	Transcribed validation spectrograms for experiment 2a	85
E.4	Transcribed validation spectrograms for experiment 2b	85
E.5	Transcribed validation spectrograms for experiment 2c	86
E.6	Transcribed validation spectrograms for experiment 2d	86

List of Figures

E.7	Transcribed validation spectrograms for experiment 2e	87
E.8	Transcribed validation spectrograms for experiment 3a	87
E.9	Transcribed validation spectrograms for experiment 3b	88
E.10	Transcribed validation spectrograms for experiment 4a	88
E.11	Transcribed validation spectrograms for experiment 4b	89
E.12	Transcribed validation spectrograms for experiment 5a	90
E.13	Transcribed validation spectrograms for experiment 5b	91
E.14	Transcribed validation spectrograms for experiment 6a	92
E.15	Transcribed validation spectrograms for experiment 6b	93

List of Tables

4.1	Comparison between different state-of-the-art AMT models	30
5.1	Probabilities in Augmentation Selection Algorithm B	37
6.1	Parameters for mel-scaled spectrogram calculation	48
6.2	Parameters for the neural networks	49
6.3	Results from experiment 0	50
6.4	Results from experiment 1 and 2	50
6.5	Rerun experiments and their equivalents in Grønbech	51
6.6	Results from experiments 3, 4 and 5	51
6.7	Averaged results from repeated runs	52
A.1	List of third-party Python libraries	73
B.1	Projector in Encoder Architecture 1	75
B.2	Projector in Encoder Architecture 2	76
B.3	Projector in Encoder Architecture 3	77
C.1	Results from individual reruns of experiment 1c	79
C.2	Results from individual reruns of experiment 5b	79
C.3	Results from individual reruns of experiment 6a	80
C.4	Results from individual reruns of experiment 6b	80

Acronyms

ADSR Attack Decay Sustain Release.

AI Artificial Intelligence.

AMT Automatic Music Transcription.

BiLSTM Bidirectional Long Short-Term Memory.

BYOL Bootstrap Your Own Latent.

CNN Convolutional Neural Network.

DAW Digital Audio Workstation.

DFT Discrete Fourier Transform.

EM Expectation Maximisation.

F0 Fundamental Frequency.

FFT Fast Fourier Transform.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

LSTM Long Short-Term Memory.

MIDI Musical Instrument Digital Interface.

MIR Music Information Retrieval.

MLP Multilayer Perceptron.

MTG-J MTG-Jamendo.

NMF Non-Negative Matrix Factorisation.

NN Neural Network.

Acronyms

RNN Recurrent Neural Network.

SimSiam Simple Siamese Representation Learning.

SOTA state-of-the-art.

STFT Short-Time Fourier Transform.

1. Introduction

Automatic Music Transcription (AMT) is a technique that involves utilising an algorithm to create a symbolic representation of a musical composition through an audio recording of a performance. In essence, we want to create a system that is able to generate sheet music or some other musical notation format given an audio recording. **AMT** systems with good accuracy would greatly impact productivity for musicians worldwide, especially for composers and teachers.

The field of **AMT** has been researched for decades. Starting in the late 70s, the first **AMT** system was developed, which was a somewhat capable **AMT** system at the time. The model was, however, limited to monophonic recordings, meaning it could not transcribe multiple notes playing simultaneously. Developments in **Artificial Intelligence (AI)** and deep learning, as well as the release of **AMT**-applicable training data, have paved the way for the current **state-of-the-art (SOTA)** solutions within the field.

Though recent years have seen the release of labelled **AMT** training data, the main roadblock preventing further advancements is the lack of high-quality labelled training data for multi-instrument **AMT**. The main goal of this report is to explore the use of self-supervised pre-training for **AMT** in order to mitigate this data shortage. Self-supervised learning is a relatively new field of machine learning, seeing advancements, especially in later years with the release of Siamese representation learning methods. These methods allow the use of unlabelled data during pre-training. In an **AMT** setting, this means datasets containing only audio, which are more abundant and easier to come by than traditional labelled **AMT** datasets.

This section will first present the motivation for the research carried out in this thesis before formulating a main research goal and accompanying research questions. The research questions are designed to structure the work such that answering each research question ultimately contributes to fulfilling the main goal. Then, the research method is presented, followed by the noteworthy contributions provided by this thesis. Finally, the general structure of the thesis is presented.

1.1. Background and Motivation

Leading up to the work done in this thesis, a preliminary report was written by the same authors. The report investigated the related work in **SOTA AMT** and tried to identify the most critical obstacles preventing further advancements. It also investigated how self-supervised learning can be used with **AMT** to enable the use of training data not directly applicable to current **SOTA** methods. The results identified in the report formed the foundation for the motivation of this thesis, presented below.

1. Introduction

While general-purpose multi-instrument [AMT](#) systems have not yet reached the level required for public use, their eventual benefits are countless, and use cases span a wide range of applications. Such a system would be capable of generating transcriptions for musical pieces without available transcriptions and could even transcribe improvised performances. Additionally, when rearranging or reharmonising a piece, a transcription is often needed. In this case, an automatically generated transcription could save hours or even days of work for a musician, depending on their skill level and the length and complexity of the composition.

Even though research on [AMT](#) has been carried out for decades already, it remains a difficult task for computers. The field has seen great advancements in single-instrument transcription due to developments in deep learning as well as the release of large datasets containing piano performances and transcriptions. However, current [SOTA](#) solutions still struggle with generalising to multi-instrument music transcription. One major reason for the lack of progress in this branch of [AMT](#) is the low amount of training data available. As all modern approaches rely on some kind of machine learning algorithm, large amounts of training data are needed to train an effective [AMT](#) system. However, there is a lack of datasets for [AMT](#), especially datasets containing recordings of multiple instruments. Therefore, current [SOTA](#) models are largely limited to piano transcription rather than multi-instrument transcription.

Some types of self-supervised learning allow for additional training, usually pre-training, on typically unusable training data. This could, for instance, be methods taking advantage of datasets containing only [MIDI](#) or only audio to improve the performance of the final [AMT](#) model. The preliminary report identified that although some [SOTA](#) solutions have utilised self-supervised learning, they have not yet utilised it in a way that enables the use of unlabelled data. Investigations into such methods open up the possibility of using datasets containing only audio or [MIDI](#), which are far more accessible than current multi-instrument [AMT](#) datasets.

1.2. Goal and Research Questions

Due to the massive lack of high-quality labelled training data available for multi-instrument [AMT](#), this thesis aims to investigate if self-supervised learning can be employed in [AMT](#) to mitigate this data shortage. To clearly describe the purpose of this investigation, the following goal is defined:

Goal *Examine if self-supervised pre-training can be used to increase the performance of [AMT](#) models.*

To facilitate the research and experimentation used to reach this goal, several research questions are formulated to divide and structure the work contributing to the overall goal:

Research Question 1 *How does the use of a pre-trained encoder, used to replace the input to the transcriber network affect the performance of an **AMT** model?*

This research question will be answered by replacing the transcriber’s input with an encoding and observing the effect this has on the performance of the model. Different encoder sizes are also tested to understand the possible advantages and limitations of this method.

Research Question 2 *How does the use of a pre-trained encoder, used to supplement the input to the transcriber network affect the performance of an **AMT** model?*

This research question is closely related to research question 1, but instead of examining the effect of replacing the input to the transcriber model, we will examine the effect of injecting an encoded spectrogram into the transcriber model. Tests for injecting the encoded spectrograms at different locations in the model are also performed to discover how this affects transcription performance.

Research Question 3 *How do different modes of selecting data augmentations affect the performance of an **AMT** model?*

This research question investigates how changing the augmentation selection algorithm affects the performance of the model. To test this, an encoder is pre-trained using a different augmentation selection scheme, after which a transcriber is trained using this encoder.

Research Question 4 *How does the amount of unlabelled data affect the results of the **AMT** model?*

This research question will be answered by pre-training two encoder models on different amounts of unlabelled data and comparing the results between two transcriber models utilising these encoders.

1.3. Research Method

Initially, a structured literature review was conducted. To gather as much information as possible on **AMT** and the current **SOTA** models, a *snowballing*-approach was utilised. Here, related work cited in each paper is investigated further, which in turn leads to additional potentially relevant articles. This literature review ultimately resulted in the related work presented in Chapter 4 and the datasets presented in Chapter 3.

Before implementation of the model could commence, a substantial amount of planning was conducted to decide the specific topic of research and what type of data was available for use, depending on the final architecture. This investigation indicated it would be beneficial to use self-supervised learning to solve the limitations of the amount of labelled data. Additionally, a decision had to be made on the specific type of self-supervised

1. Introduction

learning algorithm to utilise. To help make a decision on which algorithm to choose, additional research, specifically in the field of self-supervised learning, was conducted in order to find a method that was possible to implement during the limited time available but still had the potential to provide valuable results.

In order to make good use of the time available, a week-by-week plan for the entire thesis was first created. This schedule contained information about when specific tasks were the main focus and at what point they were planned for completion. The plan was also continuously updated to account for unforeseen factors during initial planning.

1.4. Contributions

Through the experiments conducted in this thesis and the theory and architecture presented in Chapters 2 and 5, several contributions that could provide value to further research on [AMT](#) are identified and summarised as follows:

- A thorough investigation into different ways of pre-training audio encoders using the [Simple Siamese Representation Learning \(SimSiam\)](#) training algorithm and the combination of these methods with an existing [SOTA AMT](#) architecture to discover how this affects the performance of the [AMT](#) model.
- An open-source implementation of the [SimSiam](#) training algorithm and a pre-training system with spectrogram augmentations able to train on unlabelled audio data. This system is integrated into an existing [AMT](#) implementation, providing a complete pipeline containing both training steps.
- An open-source PyTorch data loader designed specifically for the [MTG-Jamendo \(MTG-J\)](#) dataset. This data loader automatically generates spectrograms from the audio in the dataset and is capable of parallelising this task during training, resulting in loading times between batches being reduced by an order of magnitude.
- Recommendations for future work within the field, both specifically using self-supervised learning for [AMT](#) and for the field of [AMT](#) as a whole.
- A comprehensive overview of the current and previous [SOTA](#) solutions in [AMT](#) as well as a presentation of the necessary background theory needed to understand the concepts referenced throughout this thesis.
- A presentation of related work on self-supervised Siamese representation learning and augmentation techniques commonly utilised by these training algorithms.

1.5. Thesis Structure

1. *Chapter 2* presents the necessary background theory for the concepts described in the following sections. It covers the fields of [AMT](#) and [AI](#) and gives a brief introduction to music theory and different ways of representing audio.
2. *Chapter 3* presents the different datasets referenced throughout this report and what kinds of data they consist of.
3. *Chapter 4* covers the related work in the field, including both preliminary approaches to [AMT](#) and modern [SOTA](#) solutions relying on deep learning. It also provides an overview of [SOTA AMT](#) models and how they differ from one another in both architecture and performance. Lastly, this chapter also presents related work on Siamese representation learning.
4. *Chapter 5* presents the architecture utilised in this thesis. It covers the four main parts of the system, the pre-processing step, the self-supervised system, the fully supervised system and the post-processor. This chapter also details how these four systems are combined and describe the different encoder architectures utilised in the experiments.
5. *Chapter 6* presents the experimental plan, detailing each experiment and their purpose, the experimental setup, covering details about third-party libraries, system hyperparameters and hardware, and finally, the results from running these experiments.
6. *Chapter 7* evaluates and discusses the results presented in [Chapter 6](#), comparing the results with the baseline experiments and the related work.
7. *Chapter 8* gives some final thoughts on the results and contributions provided by this thesis and presents recommendations for future work on [AMT](#).

2. Background and Theory

This section will present the background and theory necessary for understanding concepts covered in the later parts of the thesis. First, [AMT](#) concepts are covered before presenting the field of [AI](#) and music theory. Finally, different audio representations and the evaluation criteria utilised in the experiments are presented. Sections 2.1 through 2.6 are based on [Nottveit and Strømsodd \(2022\)](#) with several additions and minor modifications.

2.1. Automatic Music Transcription

[AMT](#) is the process of automatically transforming acoustic signals into some form of musical notation ([Benetos et al., 2013](#)). Musical notation can range from [Musical Instrument Digital Interface \(MIDI\)](#) to staff notation. [AMT](#) can be used by musicians to make scores over improvised recordings to reproduce the performances. Another use is to get scores for pieces that do not have one or one that is not easily accessible.

[AMT](#) systems can get quite complex and are therefore divided into sub-tasks. The main sub-tasks are multi-pitch detection and note tracking. Multi-pitch detection determines which pitches are being played, and note tracking determines the start and end of a note. Other sub-tasks include recognising different instruments and which instruments play which notes, as well as determining the rhythm of a song. [Benetos et al. \(2019\)](#) separate operations in [AMT](#) into four levels:

- **Frame-Level Transcription** estimates the pitches within a frame. A frame is a specific slice of an audio recording spanning a small time frame (usually around 10ms), meaning one second of audio could, for instance, be divided into 100 frames. Each frame is usually analysed separately. However, in some cases, contextual information, such as neighbouring frames, is also included in the analysis.
- **Note-Level Transcription** tracks which notes are being played and when they are being played. The note-level transcription connects pitch estimations over time. In addition to the pitch, it also tracks a note's start and end. However, it is often difficult to track the end of a note, as a clear endpoint is often not present or can be ambiguous.
- **Stream-Level Transcription** groups notes into streams, where each stream is typically one instrument. The most common method of distinguishing which notes belong to each instrument is to take advantage of the fact that different instruments produce different sounds even when playing the same pitch. This concept is known as timbre.

2. Background and Theory

- **Notation-Level Transcription** is the conversion of music into a human-readable format, typically staff notation. As concepts like the tempo, beat, time signature and key are unknown, notation-level transcription aims to quantify these unknown variables. Knowing these variables, it is possible to create human-readable music notation, such as staff notation.

2.2. AI and Machine Learning

All **AMT** models make use of **AI** methods to function. Today, most solutions utilise machine learning, a sub-field of **AI**. Machine learning algorithms train **AI** models to solve specific tasks. There are numerous machine learning models, such as decision trees, simple regression analysis or **Neural Networks (NNs)**, each capable of solving different problems. Machine learning algorithms are usually placed into one of the following categories depending on how the model is trained.

- **Supervised Learning** algorithms rely on labelled data for training. The data used for these approaches are usually organised as pairs of observations and targets, i.e., the model's input and the desired output. **AMT** models are typically created using supervised learning algorithms, as most datasets provide both musical recordings (model input) and **MIDI** transcription (desired model output).
- **Unsupervised Learning** algorithms, in contrast to supervised learning algorithms, do not need target values in their training data. The models produced by these algorithms are typically used to find structure in data, for example, by grouping data on similarities. Clustering algorithms are typical examples of models that use unsupervised learning.
- **Reinforcement Learning** algorithms are learning algorithms that learn through exploring an environment by making actions. Training then occurs when the model is given feedback on which actions were good or bad, depending on the system's overall goal. Reinforcement learning algorithms are typically used to create autonomous agents such as robots or vehicles.

2.2.1. Self-Supervised Learning

In addition to the three main paradigms of machine learning algorithms, some additional learning methods exist. Self-supervised learning uses a combination of unsupervised and supervised learning. This could, for instance, be achieved by performing unsupervised pre-training on a model and then later training it through a supervised training algorithm. This can sometimes help alleviate problems with low amounts of labelled data for classification tasks.

2.2.2. Deep Learning

Neural networks constitute the entire sub-field of Deep Learning and have seen a significant increase in popularity during the last ten years, mainly due to the use and increased availability of **Graphics Processing Units (GPUs)** for training neural networks. The architecture of neural networks is based on the neurons and synapses in real biological brains and how these cells communicate. Countless deep learning architectures exist, each suited for different kinds of tasks. The most basic yet widespread type of deep learning architecture is the **Multilayer Perceptron (MLP)** and excels at a wide range of tasks. For processing visual data, **Convolutional Neural Networks (CNNs)** are commonly used, as they are especially good at extracting visual features from images. The **AMT** field has had recent success with **Recurrent Neural Networks (RNNs)**, deep learning models suitable for time-series or temporal data, where data's order of appearance carries information. Different types of deep learning architectures can also be combined to produce networks with processing capabilities from multiple domains.

- **MLP**, often referred to as a fully connected neural network, is a network where each neuron (perceptron) is connected to all neurons in the following layer. [Figure 2.1](#) illustrates how neurons are connected in an **MLP**.
- **CNN** is a type of neural network architecture that utilise convolutional kernels to extract visual features from data ([O'Shea and Nash, 2015](#)). The convolutional kernel is a matrix of weights, making up a filter, which can be passed over the input data, such as an image. The weights in the matrix are then used to create a weighted sum of each corresponding pixel in the image. Different configurations of weights in the kernel filter allow for the extraction of different types of features in an image, such as edge detection or blurred or sharper parts of an image. Additionally, convolutional layers are effective ways of reducing the size of an image, and their parameters can be adjusted to output specific sizes depending on their input image.
- **RNN** is another type of neural network architecture utilising recurrent connections to extract temporal data. The recurrent connections produce an effective cycle in the information flow through the network, as parts of the output from an **RNN** layer are passed as input to itself when the network processes subsequent data points. A widespread implementation of **RNNs** is the **Long Short-Term Memory (LSTM)** unit by [Hochreiter and Schmidhuber \(1997\)](#), illustrated in [Figure 2.2](#). The **LSTM** unit has an internal state, making it possible to process data differently depending on the data the unit has already processed. A variant of the **LSTM** unit is the **Bidirectional Long Short-Term Memory (BiLSTM)** unit, which consists of two **LSTM** layers processing data in opposite directions. One processes data from left to right, while the other processes data from right to left. In **AMT**, this would give the network access to the audio both preceding and succeeding a particular point in time.

2. Background and Theory

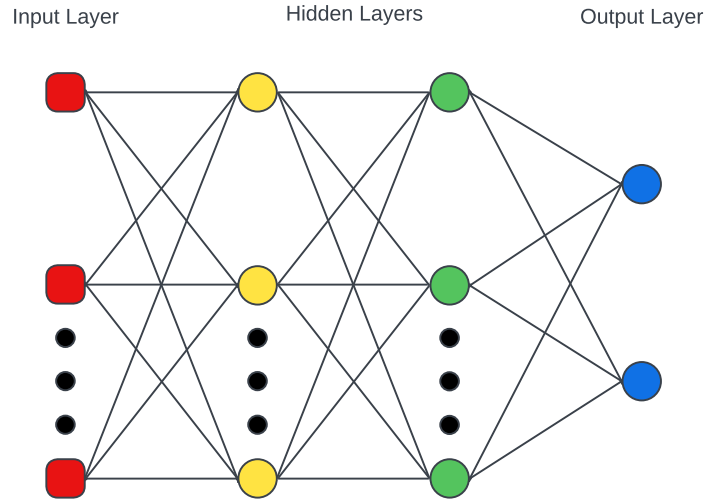


Figure 2.1.: Illustration of an **MLP**, and how each perceptron is connected to all perceptrons in the previous and following layers.

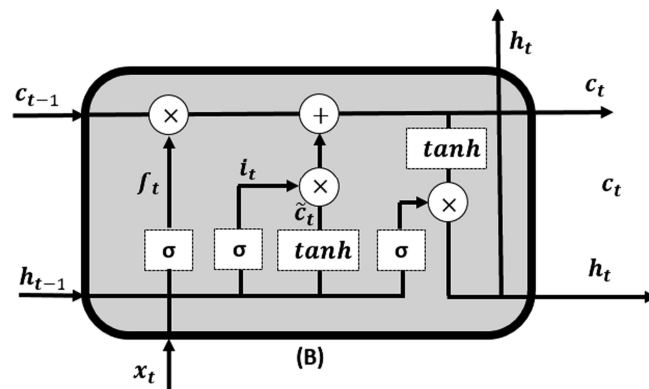


Figure 2.2.: The **LSTM** unit receives two extra inputs in addition to the output from the upstream layer (x_t): the hidden state h_{t-1} and the cell state c_{t-1} from running the previous data point through the **LSTM** layer. The cell outputs its new hidden state h_t and the new cell state c_t . Reprint of Figure 1 from [ArunKumar et al. \(2022\)](#) under CC BY 4.0 license.

2.2.3. Transposed Convolution

A convolutional layer has the ability to down-scale the dimensions of its input image. To achieve the opposite effect, a transposed convolutional layer is used. This effect can be achieved by adding columns and rows containing the value 0 around and in between the input data. By then passing this data to a standard convolutional layer, the output will have an increased size compared to the original input data. The amounts of padded rows and columns can be adjusted depending on the desired output size. Figure 2.3 illustrates how a transposed convolutional layer is able to upscale a 3x3 image to a 5x5 image.

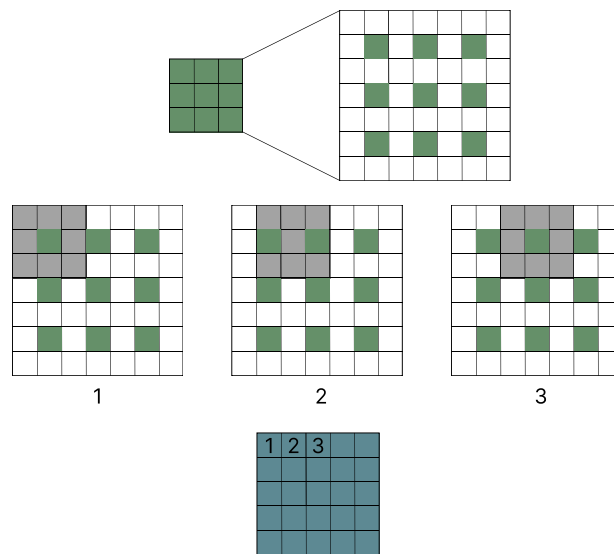


Figure 2.3.: Illustration of a transposed convolutional layer. The input data (green) is a 3x3 pixel image. White squares in the intermediate grid have the value 0. A kernel of size 3x3 is passed over the intermediate grid, producing an output image of 5x5 pixels (blue).

2.2.4. Residual Neural Network

Residual neural networks are neural networks that take advantage of shortcuts in the model architecture. These shortcuts let the data skip past some layers in the model and can be observed in Figure 2.4 (right). Their function is to send the output of one layer, not just to the following layer but also to layers further down the line. Shortcuts can have different lengths, and this length is denoted by the number of layers the shortcut skips plus one. Meaning a shortcut of length two skips one layer and passes the data to the second downstream layer, as shown in Figure 2.4. Additionally, each layer also receives data from the previous layer.

2. Background and Theory

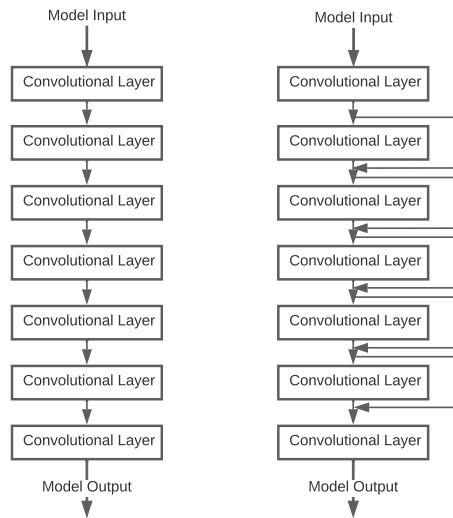


Figure 2.4.: Comparison between a normal CNN (left) and a residual neural network (right). In the residual neural network, residual connections are illustrated with arrows, each skipping one layer of the network. Typically, shortcuts skip one or two layers.

2.2.5. Siamese Neural Networks

A Siamese neural network is a type of deep learning architecture utilising two identical networks on different inputs. The output from this operation produces an output vector from each sub-network, often an encoding, preserving information about important features in the input. A popular use case for Siamese neural networks is to measure the similarity between two inputs by computing the difference between the output vectors produced by the two sub-networks.

2.2.6. Pre-Training

Pre-training in deep learning is the process of training parts of a neural network prior to the main training step. This is usually performed by training a small neural network on one dataset and later incorporating this network into another model, which is then trained on a different dataset. Pre-training can be grouped into three domains: Transfer Learning, Classification and Feature extraction. Transfer Learning is when a model uses the knowledge it has learned in one problem to solve another somewhat unrelated problem. For instance, a model can use its knowledge from a car detection problem to solve a building detection problem. The second category is Classification. In image classification, models can be pre-trained on datasets containing thousands of classes. These models can later be used for specialised classification tasks with only a few classes. This often leads to better results than training a model from scratch directly on the specialised classification problem. Additionally, the amount of data needed for the

specialised classification problem is often reduced when utilising this approach. Feature extraction, the third category, refers to the transformation of raw data into features that preserve the most amount of information. During pre-training using this method, a model is normally trained to output a representation much smaller than its input data. This forces the model to pack as much information into the small representation as possible. This method often yields better results when the downstream task is trained on these representations rather than the original raw data.

2.2.7. Image Augmentations

Image augmentations are modifications applied to an image and are utilised in machine learning to artificially increase the size and diversity of a dataset. An image is augmented by altering or changing it in some way, making it different from the original. Examples of image augmentations include image rotation, image flipping, colour adjustments or blurring.

2.3. Music Theory

A pure tone is a sinusoidal wave at a specific frequency (Alm and Walker, 2002). This frequency determines the note's pitch. The frequency of a pure A_4 tone is usually defined as 440Hz. As the frequency is doubled, the pitch increases by one octave. An A_5 is therefore 880Hz, and an A_3 is 220Hz. Tones generally have a **Fundamental Frequency (F0)** accompanied by overtones which are each different multiples of the **F0**. Western music has 12 semitones within each octave. One semitone is made up of 100 cents, meaning that if an A_4 is increased by 100 cents, it would become an $A\sharp_4$. In terms of note pitch, an $A\sharp_4$ is identical to a Bb_4 . However, different names for identical notes are used in different circumstances depending on their context.

The presence of a pitch is called a note, and the start of a note is referred to as the note onset. This corresponds to the exact time when a string on a guitar is plucked, or a key on a piano is pressed. Note offset refers to when the note stops. This does not necessarily mean the point in time when the key is released but rather when the pitch is no longer present. However, it is not required to be absent, but its volume must be below a certain threshold to be considered absent. Usually, this threshold is set to a particular fraction of the note's maximum volume.

Another way of describing a note is through **Attack Decay Sustain Release (ADSR)**. Each time a note is pressed, it goes through these four stages. Attack is the period from the note being pressed until it reaches its maximum volume or velocity. Decay describes the time from this peak until the sound reaches a sustain level. Sustain is the duration the note volume held at a constant level, which lasts until the key is released. Lastly, the release is the time it takes from when the key is physically released until the note offset. When comparing **ADSR** with onsets and offsets, the onset is the start of the attack stage, and the offset is the end of the release stage.

2.4. Fourier Transform

The Fourier transform is a mathematical transformation commonly used for isolating individual frequency components in a signal. The Fourier transform of any mathematical function is itself a function showing the individual frequencies making up the original function and their amplitudes. This is especially useful when processing audio signals to differentiate between different notes in a recording. In [Equation 2.1](#), $\hat{f}(\xi)$ is the Fourier transform of the function f at frequency ξ . Plotting $\hat{f}(\xi)$ over a range of values for ξ produces a frequency domain indicating the different frequencies making up f and their amplitudes.

$$\hat{f}(\xi) = \int_{-\infty}^{\infty} f(x)e^{-i2\pi\xi x} dx \quad (2.1)$$

Due to the discrete nature of all digitally stored data, computers are limited to a simplified version of the Fourier transform called the [Discrete Fourier Transform \(DFT\)](#), shown in [Equation 2.2](#). The [DFT](#) runs on a set of data points x_n where $n \in [0, N - 1]$, and produces the frequency domain X_k where $k \in [0, N - 1]$. This algorithm is, however, computationally quite complex, running in $O(n^2)$ time. Therefore, computers normally use a specialised faster algorithm for calculating the [DFT](#). This algorithm, commonly referred to as the [Fast Fourier Transform \(FFT\)](#), reduces the complexity to $O(n \log(n))$ ([Cochran et al., 1967](#)).

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \quad (2.2)$$

2.5. Audio Representation

There are several different useful formats for representing music and audio digitally or in written form. The most ubiquitous written format is staff notation, commonly used in sheet music worldwide and is detailed in [Section 2.5.1](#). Digitally represented audio comes in many formats, such as *MP3* or *WAVE*. The *WAVE* format is one of the most popular formats for storing uncompressed digital audio and is detailed in [Section 2.5.2](#). A spectrogram represents audio as a heatmap, showing frequencies present at different times in an audio recording, detailed in [Section 2.5.3](#). Finally, [MIDI](#) is a digital music representation format comparable to staff notation and is often used as target values in [AMT](#) models. [MIDI](#) is detailed in [Section 2.5.4](#).

2.5.1. Staff Notation

Staff notation is a language that allows humans to read music. It represents information on which notes to play and how long they should be played. [Gerou and Lusk \(1996\)](#) presents an overview of rules for staff notation. It is a 160-page book covering all there is to know about sheet music and staff notation. The first four measures of Sibelius' Op. 76, 2nd movement are displayed in [Figure 2.5](#), and incorporate some of these rules.

For instance, the $\frac{2}{4}$ fraction indicates the time signature of the piece, meaning there are two quarter notes in each measure. The \sharp symbol indicates that certain notes should be raised one semitone.

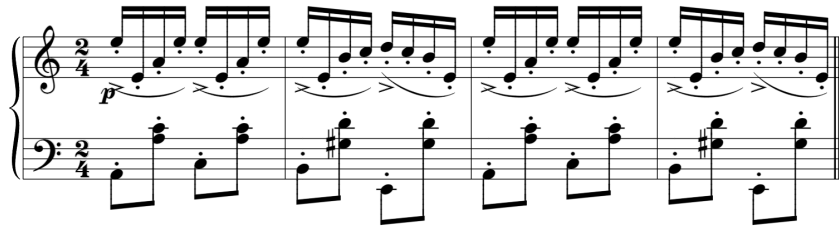


Figure 2.5.: The first four measures of Sibelius' Op. 76, 2nd movement for piano in staff notation. The piece is in $\frac{2}{4}$ time signature, indicating each measure contains two quarter notes. The right and left hands play the upper and lower staves, respectively.

2.5.2. Waveform

Waveform is one of the most common ways of storing uncompressed audio digitally. This involves storing the audio wave samples directly without any compression. Several different waveform audio file formats exist, but the most commonly used today are Microsoft and IBM's WAVE format and Apple's AIFF format.

2.5.3. Spectrograms

A spectrogram is a visual representation of audio. Spectrograms visually show the different frequencies present in an audio recording. Figure 2.6 shows an example of a spectrogram. Time is represented on the horizontal axis and frequency on the vertical axis. The amplitude of a given frequency then determines the colour of each pixel at any given time. In Figure 2.6, the lighter areas correspond to notes in the original recording, and their position on the vertical axis indicates their frequencies.

The process of generating spectrograms from audio utilises the FFT algorithm to break up audio waves into a sum of sine waves. This is achieved by breaking the audio recording into small chunks and analysing them individually. By running each chunk through the DFT to get its fundamental frequencies, it is possible to create individual columns in the spectrogram. These columns can then be combined to create the full spectrogram of the recording. This process for creating spectrograms is known as the Short-Time Fourier Transform (STFT).

When listening to music, humans interpret the pitch of notes in a logarithmic manner. Therefore, several transformations to spectrograms have been proposed to simulate how humans comprehend pitch. One of these transformations, proposed by Stevens et al. (1936), is the mel scale. The mel scale is a subjective scale designed such that notes in

2. Background and Theory

the scale are an equal distance from each other, as judged by humans. This decreases the amount of detail for lower and higher frequencies in the spectrogram while focusing on the pitches in the typical range of human perception.

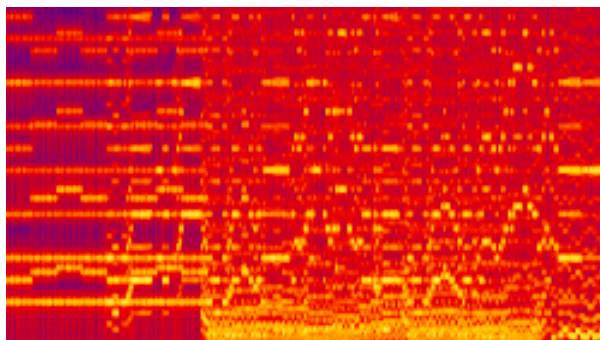


Figure 2.6.: A spectrogram of a piece of music, the most common input format for [AMT](#) models, converted using the [STFT](#). Light spots resembling notes in a piano roll are individual notes and their corresponding overtones with increasing frequency on the vertical axis and time on the horizontal axis.

2.5.4. MIDI

[MIDI](#) is a protocol that allows computers and electronic instruments to communicate by sending instructions to each other ([MIDI Manufacturers Association, 2009](#)). [MIDI](#) has a wide range of use cases. For example, a musician can record his performance using a [MIDI](#) instrument. This recording can later be played back using the same instrument or any other sound. With a [MIDI](#) recording or a [MIDI](#) file, it is also possible to edit the data by changing each note's pitch or timing. This is typically done using a [Digital Audio Workstation \(DAW\)](#). Another feature the [DAW](#) has is the ability to manually add notes, meaning that recording someone playing an instrument is not required. This is a popular tool for producers, especially those producing electronic music. Popular [DAWs](#) include *FL Studio*, *Ableton* and *Logic Pro*. The piano roll, a feature present in most [DAWs](#), makes it easier for humans to read the [MIDI](#) messages. An example of the piano roll is illustrated in [Figure 2.7](#). Here, a short [MIDI](#) file spanning eight bars is displayed.



Figure 2.7.: The piano roll in the FL Studio DAW showing eight bars with different chords. In the piano roll, the note pitch is indicated on the vertical axis and the note onset and offset on the horizontal axis.

2.6. Evaluation

AMT models are typically trained to predict note pitch, onset and offset. During this process, a way to evaluate the resulting transcription, and thus the model’s performance, is needed. In the field of Music Information Retrieval (MIR), this is usually achieved using precision, recall, and F1 scores. These metrics are commonly used in information retrieval and machine learning to evaluate the performance of different systems. AMT is considered a binary classification problem, meaning the predictions it produces are either correct or incorrect. A note prediction can, therefore, either be correct (true) or incorrect (false) and either present (positive) or absent (negative). This produces four categories in which a note prediction can be: true positive (TP), true negative (TN), false positive (FP), and false negative (FN). These four prediction categories are used to calculate the following metrics.

2.6.1. Precision

Precision measures the number of correct predictions relative to the number of total predictions. In AMT, this is the percentage of the predicted notes that also appear in the ground truth MIDI.

$$Precision = \frac{TP}{TP + FN} \quad (2.3)$$

2. Background and Theory

2.6.2. Recall

Recall measures the number of correct predictions relative to the number of desired correct predictions. In [AMT](#), this is the percentage of notes in the ground truth [MIDI](#) that are also present in the predicted transcription.

$$Precision = \frac{TP}{TP + FN} \quad (2.4)$$

2.6.3. F1 Score

In most cases, it is desirable to balance the model to perform acceptably regarding both precision and recall simultaneously. To achieve this, the F1 score is used to evaluate the model by combining precision and recall into one single metric. The α parameter allows us to assign a weight to either precision or recall.

$$F1 = \frac{1}{\alpha \frac{1}{Precision} + (1 - \alpha) \frac{1}{Recall}} \quad (2.5)$$

To weigh precision and recall equally, α is usually set to 0.5, which results in the following simplified equation.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \quad (2.6)$$

3. Datasets

This section presents the relevant datasets containing different types of labelled or unlabelled datasets for use with [MIR](#). The Slakh2100, MTG-Jamendo and MNIST datasets are utilised in the experiments detailed in [Chapter 6](#) and discussed further in [Section 7.2.3](#). The remaining datasets are utilised by the related work presented in [Chapter 4](#). Most of the datasets presented below contain either musical recordings, [MIDI](#) data or both. [Sections 3.1 through 3.10](#) are derived from [Nottveit and Strømsodd \(2022\)](#) with minor modifications.

3.1. MAPS

The MAPS (MIDI Aligned Piano Sounds) dataset contains real audio recordings with ground truth [MIDI](#) transcriptions explicitly created for use with [AMT \(Emiya, 2010\)](#). It consists of 65 hours of recorded piano performances and is divided into four parts ISOL, RAND, UCHO and MUS. ISOL contains isolated notes. RAND and UCHO contain recordings of different chords played on a piano. The former is made up of chords with random pitches, while the latter contains chords commonly associated with western music. The last part of the dataset, MUS, contains actual recordings of musical pieces performed on a grand piano.

3.2. MAESTRO

MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) is a dataset explicitly created for [AMT](#). It contains high-quality recordings and transcriptions created using the Yamaha Disklavier ([Hawthorne et al., 2019](#)). The Yamaha Disklavier is a high-end grand piano with integrated [MIDI](#) capture capabilities, such that [MIDI](#) transcriptions and audio recordings can be captured simultaneously. Nine years of piano competitions performed on this piano resulted in a dataset containing over 172 hours of recorded piano performances and corresponding [MIDI](#) transcriptions.

3.3. GiantMIDI-Piano

GiantMIDI-Piano is a dataset of piano recordings transcribed to [MIDI](#) using the deep learning [AMT](#) model created by [Kong et al. \(2021\)](#). The transcriptions contain data for note pitch, onset, offset and velocity, and onsets and offsets for the sustain pedal. GiantMIDI-Piano consists of 10,885 piano recordings totalling 1,237 hours, transcribed

3. Datasets

into [MIDI](#) files. These [MIDI](#) transcriptions are regarded to be of high quality due to the transcriber model achieving an onset F1-score of 96.72% on the MAESTRO dataset ([Kong et al., 2022](#)).

3.4. Million Song Dataset

The Million Song dataset consists of 280GB of metadata and audio features for one million songs ([Bertin-Mahieux et al., 2011](#)). Each song’s dataset contains information about the artist, time signature, release date, and artist location. It does not contain audio but does, however, provide code for downloading corresponding audio from 7digital.

3.5. MusicNet

The MusicNet dataset contains 34 hours of recordings and [MIDI](#) transcriptions of classical pieces performed by chamber music ensembles ([Thickstun et al., 2017](#)). The 330 pieces in the dataset are written by ten classical composers, including Beethoven, Schubert, and Brahms. The dataset is heavily skewed toward pieces by Beethoven, as around half of the dataset in terms of time consists of music written by Beethoven.

3.6. Lakh

Lakh is a dataset of [MIDI](#) files matched to audio recordings from the Million Song Dataset ([Raffel, 2016](#)). From more than 176,000 [MIDI](#) files, 160,000 [MIDI](#) files were matched with audio recordings from the Million Song Dataset. All 160,000 files had a duration of more than 30 seconds, and the matches had varying levels of confidence. From an [AMT](#) point of view, the Lakh dataset only contains ground truth [MIDI](#) transcriptions and no audio recordings. Although every [MIDI](#) file in Lakh has a match in the Million Song Dataset, the overall lack of confidence in these matches makes the dataset infeasible to use with [AMT](#) directly.

3.7. Slakh2100

Slakh2100, or Synthesised Lakh, is a subset of [MIDI](#) files from the Lakh dataset digitally synthesised into audio recordings ([Manilow et al., 2019](#)). It contains 2100 high-quality audio-[MIDI](#) pairs totalling over 145 hours of recordings. Created for music source separation, the Slakh dataset is well suited for use with [AMT](#) due to the high note-level correlation between ground truth [MIDI](#) and the synthesised audio. Every audio recording in Slakh contains piano, bass, guitar, and drums, while most contain at least one additional instrument.

3.8. Cerberus4

Cerberus4 is a derivative dataset of Slakh2100, consisting of 542 hours of synthesised audio spread across 1327 different pieces (Gardner et al., 2022). It was created by extracting subsets of instruments for each recording. These were selected such that each resulting subset contains precisely one of each of piano, guitar, bass, and drums.

3.9. URMP

The University of Rochester Multi-modal Music Performance (URMP) is a multi-track and multi-instrument dataset containing both audio and video recordings, as well as sheet music and MIDI transcription of 44 classical performances (Li et al., 2019). The dataset is designed for use in MIR and AMT, and therefore also contains high accuracy ground truth frame and note-level transcriptions in addition to the MIDI data. The recordings are performed on a selection of 14 instruments, each piece on different subsets of these instruments. The ensemble includes four woodwinds, four brass and six different string instruments.

3.10. GuitarSet

The GuitarSet dataset presented by Xi et al. (2018) contains over three hours of live polyphonic guitar recordings. The recordings are different guitarists playing the same pieces, such as Autumn Leaves and Pachelbel’s Canon. All recordings contain acoustic guitar recordings. This dataset also provides aligned transcriptions in the form of MIDI-files.

3.11. MTG-Jamendo

The MTG-Jamendo (MTG-J) dataset presented by Bogdanov et al. (2019) provides more than 55000 full audio tracks spanning 87 different genres performed on 40 different instruments. The dataset is unlabelled and thus contains no MIDI-files representing the notes played in the recordings.

3.12. MNIST

The MNIST dataset contains more than 58000 different handwritten digit images (LeCun et al., 1998). The digits are written by 500 different people, resulting in a wide range of different styles for the handwritten digits.

4. Related Work

This chapter will present previous research within [AMT](#), including both preliminary approaches and modern [SOTA](#) methods utilising deep learning and neural networks. The latter will be the main focus, as all current [SOTA](#) solutions utilise neural network-based approaches. First, the Onsets and Frames model by Google Magenta and similar models based on this architecture are presented. Then, the transformer-based model, MT3, is presented followed by a self-supervised approach to [AMT](#). This chapter also provides a comparison between [SOTA](#) methods, comparing their training datasets and performance. Lastly, related work on self-supervised representation learning is presented. Sections 4.1 through 4.4 are based on [Nottveit and Strømsodd \(2022\)](#) with minor modifications.

4.1. Preliminary Approaches

In the late 1970s, a team from the University of Michigan created one of the first [AMT](#) systems ([Piszczałski and Galler, 1977](#)). Their goal was to transcribe sound into staff notation representation. This was achieved by mapping the sound into the frequency domain, which was done using the [FFT](#). They created a three-dimensional representation of the sound by calculating the frequencies, with time, frequency and amplitude on each axis. They then used this representation to pick the most substantial frequency in each period. The end of a note was defined as either the beginning of the following note or at the point in time in which the note’s amplitude fell below a certain threshold. The note’s pitch was then assigned to the average most substantial frequency in the period the note was active. After repeating this process for the whole recording, the results were converted to staff notation.

One of the main drawbacks of this model is that each time frame could only be assigned the average most substantial frequency. It can only assign one pitch to each point in time, resulting in a monophonic transcription. Another issue with this model is that it relies heavily on the notes having a strong [F0](#). This works well on instruments like the flute or electronically generated sine waves, as they have a strong [F0](#). However, this is not always the case for other instruments, and the system could, in these cases, produce worse transcriptions.

Later, another team proposed a new method for [AMT](#) using [Non-Negative Matrix Factorisation \(NMF\)](#) ([Smaragdis and Brown, 2003](#)). This method is capable of polyphonic music transcription, meaning it can transcribe pieces with multiple notes playing simultaneously. [NMF](#) transcription assumes that the harmonic profile in a musical recording is static. In practice, all notes should have the same relative overtones. The advantage of this approach is that any previous musical knowledge is not required, and the model will

4. Related Work

learn by observing the notes in the recording. This, however, also leads to one of the main drawbacks of [NMF](#), as it cannot separate notes that do not appear on their own in the recording.

4.2. Onsets and Frames

Since the 90s, the use of artificial neural networks has vastly increased. Along with most other fields, [MIR](#) and [AMT](#) have also seen a leap in performance due to the rise of neural networks. Today, most leading [AMT](#) models are based on the Onsets and Frames architecture developed by Google Magenta. The following section presents some of these models.

Created by Google Magenta in 2017, the Onsets and Frames model is an [AMT](#) model capable of transcribing polyphonic piano recordings with high accuracy ([Hawthorne et al., 2018](#)). This method innovated the field by the way it predicts individual notes. The model is divided into two parts, an onset predictor and a frame predictor. First, note onsets are predicted by the onset predictor to mark the beginning of each note in the piece. The onset predictions include information about the note’s pitch and in which frame the onset is located. These predictions, along with the spectrogram, are then given to the frame predictor, which then predicts the pitches of notes present in each frame. By using the onset predictions as part of the input to the frame predictor, frame predictions are restricted by onset predictions such that frames without an accompanying note onset are less likely to appear.

The model was trained on the MAPS dataset, covered in [Section 3.1](#), consisting of actual piano recordings and aligned [MIDI](#) data. The model takes audio converted to log mel-spectrograms as input. These were generated using the [FFT](#) with 229 logarithmically spaced frequency bins, a hop length of 512 and a window length of 2048 at a sample rate of $16kHz$ ([Hawthorne et al., 2018](#)). The complete spectrogram is then passed through the network without segmentation.

The onset predictor and the frame predictor consist of two separate neural networks, a convolutional neural network and a recurrent neural network. The convolutional neural networks are designed to pre-process the input spectrogram such that the most valuable features are extracted from the input. Convolutional neural networks, such as spectrograms, are often applied when processing image data because they are good at extracting visual features from images. The [CNNs](#) in each predictor is followed by an [RNN](#). The first layer is the [BiLSTM](#) layer with 128 units running in both directions. The output of this [LSTM](#) is then passed through a fully-connected layer of 512 nodes and a sigmoid activation function followed by an output layer of 88 nodes. These 88 nodes represent each key on a standard 88-key grand piano. For the frame predictor, the input to the [RNN](#) is the output from the [CNN](#) concatenated with the output from the onset predictor. Both predictors use the cross-entropy loss function, and the total loss of the model is defined as the sum of these individual losses. This architecture is illustrated in [Figure 4.1](#).

When trained on the MAPS dataset, the model achieves a frame-level F1 score of 78.30% and a note-level F1 score of 82.29% (Hawthorne et al., 2018). This frame-level F1 score is slightly better than previous SOTA models, while the onset F1 score is a 52% relative increase from previous SOTA models. The release of the MAESTRO dataset by Hawthorne et al. (2019), covered in Section 3.2, prompted the team to push the results further. The team trained a larger version of their previous Onsets and Frames model on the new and larger dataset to receive a frame-level F1 score of 90.15% and an onset F1 score of 95.32%. This newly modified Onsets and Frames model included an offset predictor, whose predictions were also used as inputs to the frame predictor. The new model also increased the size of the CNN, the RNN and the fully-connected layers.

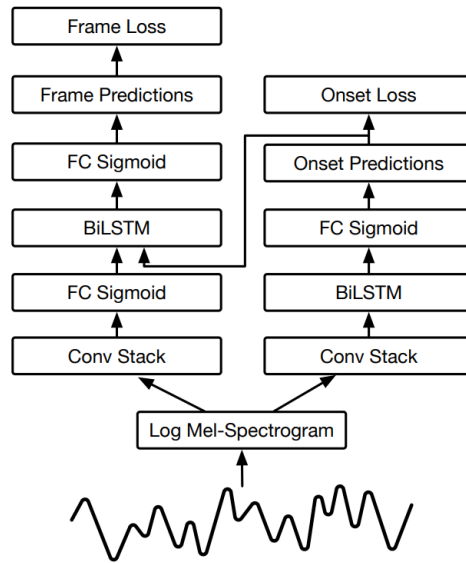


Figure 4.1.: The initial Onsets and Frames architecture presented by Hawthorne et al. (2018). The model contains an onset and a frame predictor, each given a log mel-spectrogram as input. In addition to the spectrogram input, the onsets predictions are passed through parts of the frame predictor. Reprint of Figure 1 from Hawthorne et al. (2018) under CC BY 4.0 license.

The initial Onsets and Frames model proposed by Hawthorne et al. (2018) encodes the presence of a note onset in a binary fashion for each frame. I.e. the model predicts which frames have note onsets but not at what time an onset happens within the 32ms frame. To improve the timing accuracy of these onset predictions, ByteDance, another research group, proposed a new model for onset representation (Kong et al., 2021). This representation uses a regression-based onset encoding, enabling arbitrary resolution of note onset location within each frame. The team also modify the architecture to include velocity predictions in the input to the onset predictor’s RNN. They also swap the BiLSTM with bi-directional Gated Recurrent Unit (GRU) in the RNN stage of each predictor.

4. Related Work

The new model presented by Kong et al. (2021) outperforms previous SOTA results receiving a note onset F1 score of 96.72% compared to Onsets and Frames’ 95.60% (reproduced by ByteDance) when trained on the MAESTRO dataset. Frame-level F1 score is 89.62%, which is somewhat lower than SOTA performance at 91.40%. After achieving this result, Kong et al. (2021) proceeded to create high-quality MIDI transcriptions of a large dataset of piano recordings, ultimately resulting in the creation of the GiantMIDI-Piano dataset detailed in Section 3.3.

The two models presented so far have only focused on single-instrument transcription and even specialised their model architectures for use with piano transcription. A master’s student at the Norwegian University of Science and Technology (NTNU), expanded the previous work with the Onsets and Frames model by training a similar model on a multi-instrument dataset (Grønbech, 2021). The model architecture, illustrated in Figure 4.2, resembles the previous Onsets and Frames model, with the addition of an offset and velocity predictor. The frame predictor then utilises both onset, offset and velocity predictions for frame prediction. The main difference, however, is adding a U-Net, a fully convolutional neural network. The U-Net works by downsampling and then upsampling the input data using convolutional layers. Then, the U-Net is trained to predict the noise in the input image, which is then subtracted from the original image to remove noise. The U-Net architecture is illustrated in Figure 4.3. During the upsampling process, data from the corresponding layer in the encoder is concatenated to the data from the layer below. This noise removal process is performed on the spectrograms before it is passed on to each predictor.

The model proposed by Grønbech (2021) was trained on the Slakh2100 dataset and achieves promising results when compared to contemporary SOTA single-instrument AMT models. When evaluating electric bass transcription in a multi-instrument setting without source separation, the model achieves a note F1 score of 92.9%. In the same experiment, piano and guitar transcription results yield note F1 scores of 74.3% and 64.7%, respectively.

4.3. MT3

Concurrently with the work done by Grønbech (2021), another team focusing on the same task published their findings in late 2021. They argue that AMT systems should be capable of both transcribing multiple voices playing simultaneously and played by different combinations of instruments. The team, therefore, proposed a new model, the Multi-Task Multitrack Music Transcription (MT3) model Gardner et al. (2022), capable of performing these tasks. The model can transcribe an arbitrary amount of instruments given a spectrogram generated from an audio recording.

The architecture builds on previous work in natural language processing and incorporates the T5 architecture developed by Raffel et al. (2019). This model is an autoencoder and transformer architecture capable of predicting the most probable following tokens given an input sequence. For language models, tokenisation is achieved by representing each word as a single token. This enables the model to predict word sequences. However,

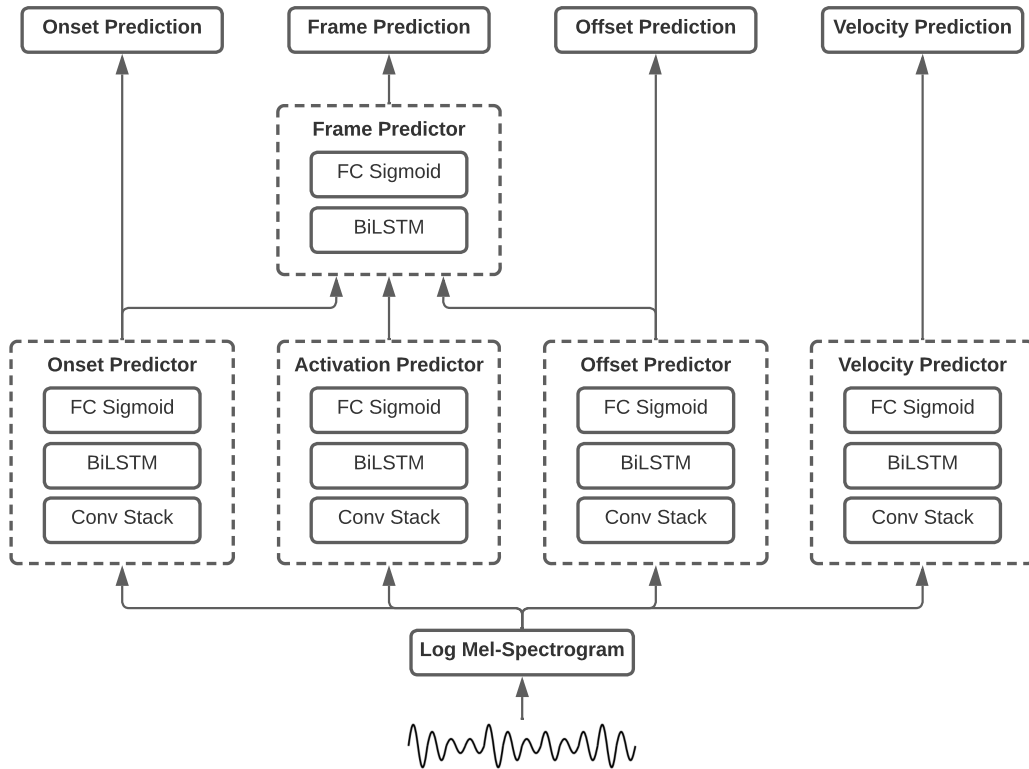


Figure 4.2.: The extended Onsets and Frames architecture proposed by Grønbech (2021), featuring offset and velocity predictors. Offset, onset and activation predictions are concatenated and passed as input to the frame predictor. Reprint of Figure 5.1 from Grønbech (2021) with permission.

when such models are used for *AMT*, a token vocabulary capable of describing music must be constructed. The MT3 model uses a vocabulary of tokens describing musical events similar to the *MIDI* standard, such as note pitch, instrument type, note-on/note-off and end-of-string events. The instrument type token is crucial for multi-instrument transcription for the model to specify to which instrument the following events belong. Like most previous *SOTA AMT* models, the MT3 model utilises log mel-spectrograms as its input. The model then produces a sequence of token outputs which is later converted to standard *MIDI*.

The MT3 model achieves *SOTA* performance when evaluated on the MAESTRO dataset with frame and onset F1 scores of 86% and 95%, respectively, when trained on all datasets (Gardner et al., 2022). Onset F1 scores for the remaining test datasets Cerberus4, GuitarSet, MusicNet, Slakh2100 and URMP are 92%, 90%, 50%, 76%, 77%, respectively. According to the team behind MT3, their model outperforms all other models in their comparison when evaluated on multitrack datasets such as Cerberus4, Slakh2100 and MusicNet.

4. Related Work

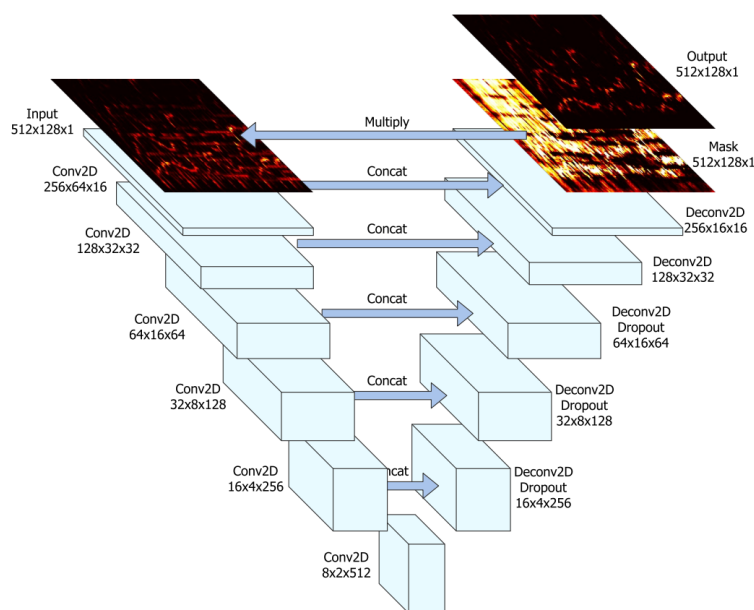


Figure 4.3.: The U-Net architecture illustrates how a spectrogram is passed through the encoder-decoder style architecture. The U-Net outputs a mask, which is then subtracted from the input for denoising. Reprint of Figure 1 in [Jansson et al. \(2017\)](#) under CC BY 4.0 license.

4.4. NoteEM

The NoteEM model by [Maman and Bermano \(2022\)](#) is one of the few SOTA AMT models utilising self-supervised learning. The model is trained using the [Expectation Maximisation \(EM\)](#) algorithm, and the training process is divided into three steps: pre-training, E-step and M-step. A simplified overview of the model is shown to the far left in [Figure 4.4](#). During the first step, the team bootstraps the training process by training the model on labelled data similar to data used by previous SOTA models. [Maman and Bermano](#) use the MIDI Pop Dataset for this bootstrapping process. This large dataset consists only of MIDI data. It contains more than 77,000 MIDI representations of pop songs, from which the team synthesised their audio.

After pre-training, the team continues with the E-step. This training step utilises the MusicNet dataset containing multiple recordings of pieces and a single MIDI-file for each piece. There is no aligned MIDI for every single recording, so the MIDI data is only a somewhat good audio representation. The team also uses a small set of self-collected data and MusicNet for training. The authors refer to the combined dataset used in this step as the unaligned dataset. During the E-step, spectrograms of the recordings are passed through the network. The network’s output is combined with the unaligned ground truth MIDI using Dynamic Time Warping (DTW). This produces a new audio transcription closer to the aligned ground truth than the unaligned MIDI.

This new transcription can then be used for fully supervised training during the M-step. Although the training data is not fully aligned, the training procedure is identical to how models by Hawthorne et al. were trained. The network architecture is also identical to that of Hawthorne et al.. However, the NoteEM network is around 1.5 times larger in terms of trainable parameters. The E-step and M-step can be repeated multiple times to further improve the model and the unaligned training data.

The NoteEM model achieves frame and onset F1 scores of 77.0% and 89.7%, respectively, when tested on piano from the MAESTRO dataset. The model can also be used on other instruments as it is a multi-instrument transcriber. However, these scores are lower than piano transcription scores.

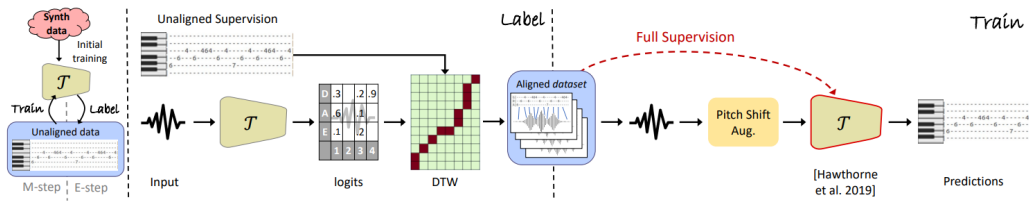


Figure 4.4.: Illustration of the Note_{EM} training algorithm. Left: An overview of the training process. Middle: The E-step during training. Right: The M-step during training. Reprint of Figure 1 in Maman and Bermano (2022) under CC BY-NC-SA 4.0 license.

4. Related Work

4.5. Comparison

Table 4.1 shows a comparison between the presented related work in AMT. The table presents the different datasets used to train each model, which instruments the models can transcribe, and each team’s reported onset F1 score. For the multi-instrument transcriber models presented in this chapter, the reported onset F1 score shown in Table 4.1 is the onset F1 for the instrument and dataset, yielding the highest F1 score.

Table 4.1.: Comparison between architecture and performance in SOTA AMT models. The Onset F1 column shows the onset F1 scores reported by each team from experimental results.

Author	Architecture	Instrument	Dataset	Onset F1 (%)
Hawthorne et al. (2018)	Onsets and Frames	Piano	MAPS	82.89
Hawthorne et al. (2019)	Onsets and Frames	Piano	MAESTRO	95.32
Kong et al. (2021)	Onsets and Frames	Piano	MAESTRO	96.72
Grønbech (2021)	Ext. Onsets and Frames	Multi	Slakh2100	92.9 ¹
Gardner et al. (2022)	MT3	Multi	Multi	96 ²
Maman and Bermano (2022)	Note _{EM}	Multi	Multi	89.7 ³

¹The onset F1 score for the Ext. The Onsets and Frames model was 92.9% for the instrument achieving the best performance (electric bass).

²The onset F1 score for the MT3 model was 96% on the test dataset achieving best performance (MAESTRO).

³The onset F1 score for the Note_{EM} model was 89.7% for the instrument and test dataset achieving best performance. (Piano transcription on MAESTRO).

4.6. Self-Supervised Pre-Training

The following section presents the related work on self-supervised pre-training methods, specifically Siamese representation learning, which is utilised to pre-train an encoder model on unlabelled data in the target domain. The two methods presented below are closely related, and both utilise Siamese neural networks during pre-training.

4.6.1. Simple Siamese Representation Learning

Simple Siamese Representation Learning (SimSiam), is a method for unsupervised image representation learning presented by [Chen and He \(2021\)](#). It can be used for pre-training an encoder model on unlabelled data and is often used for image classification. The way **SimSiam** works is by applying two sets of image augmentations to an input image from the training data. This results in two different versions of the input image, each passing through the encoder network. This produces two encodings of the image, one for each augmentation. A predictor is then given these image encodings and is trained to predict the encoder output for the other augmentation. A loss is calculated using cosine-similarity between the prediction and the actual encoding. Finally, losses are added together, and gradients are propagated through the predictor and the encoder. After pre-training, the predictor is discarded and the encoder can be utilised in a wide range of downstream machine learning tasks.

4.6.2. BYOL Representation Learning

Bootstrap Your Own Latent (BYOL), is another method for unsupervised image representation learning presented by [Grill et al. \(2020\)](#). Its use case is identical to that of **SimSiam**, and its implementation is fairly similar. Like **SimSiam**, **BYOL** applies two different sets of image augmentations to the input image. However, these two augmented images are now passed through two encoder models. The weights of the second encoder model are a moving average of the previous weights of the main encoder. This is known as a momentum encoder and, along with **BYOL**'s loss function, differentiates this algorithm from **SimSiam**.

[Grill et al. \(2020\)](#) present a list of augmentations and individual probabilities for applying these augmentations to the input of each encoder. [Chen and He \(2021\)](#) utilise the same augmentations and probabilities in **SimSiam** as the ones presented by [Grill et al. \(2020\)](#).

5. Architecture

This chapter presents the system architecture used to conduct each experiment presented in Chapter 6. The overall architecture is designed to be trained in two steps. First, a self-supervised training step utilising only unlabelled data, followed by a fully supervised training step with labelled data. The second step makes use of the model pre-trained in the self-supervised training step. In order for the models to extract information from the input data more easily, the datasets need to be processed before training can begin. Furthermore, the raw output from the networks also needs processing before model performance can be evaluated and **MIDI** output can be constructed from the network's predictions.

The overall architecture of the system, illustrated in Figure 5.1, is comprised of four main parts presented in this chapter. First, the pre-processor, responsible for audio to spectrogram conversion and parsing of ground truth **MIDI**, is presented. The self-supervised training algorithm and encoder architectures are then presented, followed by the extended Onsets and Frames architecture and its supervised training procedure responsible for frame, onset and offset predictions. Finally, the post-processing step is presented, detailing how the model's predicted features are converted to the output **MIDI** format.

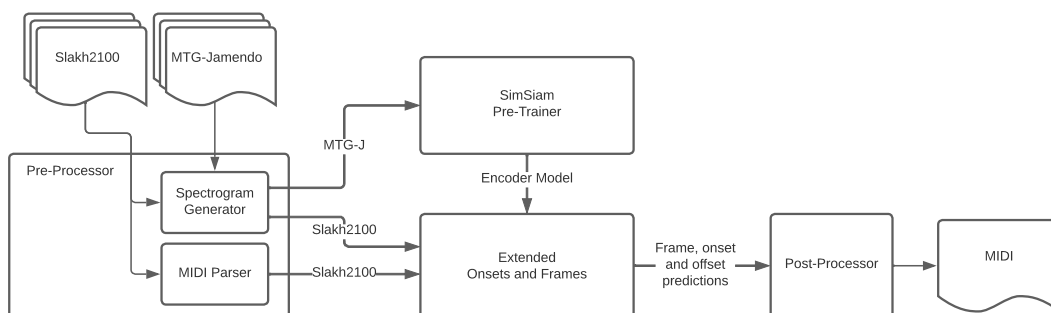


Figure 5.1.: Overall architecture of the system implemented in this thesis. The data flows through the pre-processor, the output of which passes through the pre-trainer and the main Onsets and Frames model. The predictions produced by the latter after training are passed to the post-processor capable of producing **MIDI**-files of the transcribed audio.

5. Architecture

5.1. Pre-Processing

The system’s pre-processing steps handle the data before it is used during training. As the system trains on both labelled and unlabelled data, the pre-processing step must handle both forms. A central part of the pre-processing step is the conversion of input audio to a format the neural network can read. This conversion is identical in both the self-supervised and the fully supervised training steps. During this pre-processing step, the audio is converted to log mel-spectrograms using the [STFT](#). This algorithm creates spectrograms in matrix form, suitable in size for direct input to the neural network.

Data from the Slakh2100 dataset was loaded using the `slakh-dataset` Python library. This library was set up to load each minibatch of audio and accompanying [MIDI](#) transcription. Each time an audio track was loaded, a random 20-second chunk from the source audio file, along with its [MIDI](#) counterpart, was selected and loaded. For the [MTG-J](#) dataset, a similar data loader was created by adapting the source code from the `slakh-dataset` library to work on [MTG-J](#). This improved code quality and was necessary for the parallelisation of data loading during pre-training.

While loading the Slakh2100 dataset, the system must handle the conversion of ground truth [MIDI](#) to a matrix representation identical to that of the network’s output. The output has dimensions 88 by 640. 88 is the number of different notes the network can predict, while 640 is the number of time steps these notes can occur in. For the network to learn during the fully supervised training step, each converted ground truth label must match the input spectrogram to a high degree of accuracy in terms of timing. Using these converted [MIDI](#) labels, the neural network is trained to produce the desired [MIDI](#) pattern when prompted with the corresponding input spectrogram.

As the audio in the Slakh2100 dataset is rendered at a 44.1 kHz sample rate and the Onsets and Frame model demands data at a 16 kHz sample rate, data had to be resampled. This conversion was performed only once before training. First, data in the original [FLAC](#) format was converted to the raw [WAV](#) format. These [WAV](#)-files were then resampled into the desired 16 kHz sample rate and were finally converted back to the original [FLAC](#) format required by the Onsets and Frames model.

The data in [MTG-J](#) was distributed in the compressed [MP3](#) format. These files were directly loaded to produce raw samples of the audio with a 44.1 kHz sample rate. However, unlike for the Slakh2100 dataset, the loaded audio recordings from the [MTG-J](#) dataset were directly resampled to 16 kHz during loading.

Due to how the extended Onsets and Frames model is implemented, spectrograms from both datasets are rotated 90 degrees in a clockwise direction compared to the spectrograms detailed in [Section 2.5.3](#). This results in time being represented on the vertical axis and pitch on the horizontal axis with minimum values for both axes in the top left of the spectrogram.

5.2. Self-Supervised System

To compensate for the lack of labelled data, a self-supervised system was implemented. This addition divides the training procedures into two separate steps. First, training on unlabelled data is performed, after which training on labelled data occurs. To differentiate the two training methods, training on unlabelled data is referred to as pre-training, while training on labelled data is referred to as supervised training.

The pre-training step utilises the [Simple Siamese Representation Learning \(SimSiam\)](#) algorithm by [Chen and He \(2021\)](#) presented in Section 4.6.1. This step aims to train an encoder architecture to produce meaningful representations of its input spectrograms. The [SimSiam](#) algorithm is illustrated in [Figure 5.2](#). It consists of two neural networks: the encoder network f and the predictor network h . [Figure 5.2](#) illustrates the flow of data as two instances of the encoder network f . However, in the actual implementation of [SimSiam](#), the encoder network f is a single neural network. During pre-training, the encoder is given two augmented versions, x_1 and x_2 , of the same input image. This results in two different embeddings, z_1 and z_2 . z_1 is then passed on to the predictor h , whose task is to predict z_2 (the embedding produced by the encoder given x_2). Simultaneously, a prediction p_2 for z_1 is produced by h , given z_2 . Loss is then calculated using the loss function defined in [Equation 5.1](#).

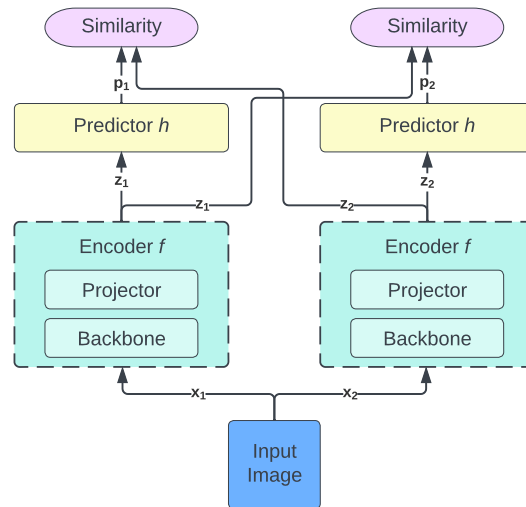


Figure 5.2.: Simple Siamese training algorithm. Two augmentations, x_1 and x_2 , of the input image are passed through the encoder f . For each input, the encoder produces respective embeddings z_1 and z_2 , which are passed along to the predictor h . This predictor produces predictions p_1 and p_2 which are compared to their counterpart and a loss is calculated based on this similarity.

5. Architecture

$$Loss = -\frac{CosSim(p_1, z_2) + CosSim(p_2, z_1)}{2} \quad (5.1)$$

$$CosSim(p, z) = \frac{p \cdot z}{max(\|p\|_2 \cdot \|z\|_2, \epsilon)} \quad (5.2)$$

The loss function, defined in Equation 5.1, utilises Cosine Similarity, defined in Equation 5.2. The Cosine Similarity formula computes the product of the l2-norms of p and z . The max -function is used with $\epsilon = 10^{-8}$ to avoid division by zero errors.

During pre-training of the encoder, there is a possibility that the model collapses, which means that the encoder outputs the same encoding regardless of its input. To decrease the probability of this happening during pre-training, it is vital that gradients are not applied to the encoder directly along the path of z_1 and z_2 , and only through the predictor h (Chen and He, 2021).

5.2.1. Data Augmentations

Different image augmentations are needed during self-supervised training of the encoder using SimSiam. Here, one or more image augmentations are applied to each batch to produce the model inputs x_1 and x_2 . Both augmented spectrograms are then passed through the encoder-predictor stack. The following augmentations were considered relevant for audio spectrograms, based on the augmentations utilised by Grill et al. (2020) and Chen and He (2021) for image classification.

- **Gaussian Blur:** The Gaussian blur augmentation blurs the image, making the image seem out of focus.
- **Erase:** The erase augmentation removes or erases a random rectangle in the image. A random rectangle is selected, and all values within are replaced with the spectrogram’s overall minimum value.
- **Noise Injection:** Adds Gaussian noise to the spectrogram creating a low-quality effect on the image.
- **Crop and Stretch:** Crops the spectrogram in the time dimension by slicing away a random amount of rows in the matrix on the top and bottom. The spectrogram is then resized to its original dimensions to achieve a stretching effect.
- **Pitch Shift:** Moves the spectrogram left or right a random amount resulting in either an increased or decreased frequency. The empty area at the far left or far right of the spectrogram is filled with the spectrogram’s minimum value to keep the matrix’s original shape.

Augmentation Selection Algorithm

Having multiple augmentations to choose from, we need some way of selecting which augmentations to apply. In the [SimSiam](#) training algorithm, two different augmented versions of the same spectrogram is created. For this purpose, two different algorithms for augmentation selection were designed. The first algorithm, Augmentation Selection Algorithm A, selects an augmentation randomly, which is then applied to an image. This augmentation is then prevented from being randomly chosen when picking the following augmentation. The second algorithm, Augmentation Selection Algorithm B, operates probabilistically. In this algorithm, each augmentation has a probability of being chosen, meaning an image could be augmented by more than one augmentation. Since different augmented images are desired, the set of probabilities for applying the augmentations differs between the first and second time a spectrogram is augmented. Derived from the probabilities of similar image augmentations presented by [Grill et al. \(2020\)](#), [Table 5.1](#) presents the probabilities utilised in Augmentation Selection Algorithm B.

Table 5.1.: Probabilities for the different augmentations in Augmentation Selection Algorithm B. The two columns x_1 and x_2 show probabilities for applying each augmentation to the input image to produce the two augmented images x_1 and x_2 .

Augmentation	x_1	x_2
Random crop and stretch	1.0	1.0
Random erase	0.2	0.2
Gaussian blur	0.8	0.1
Random pitch shift	0.0	0.3
Noise injection	0.3	0.3

5.2.2. Encoder Architectures

During the experiments carried out in this thesis, several architectures were tested. In accordance with [Chen and He \(2021\)](#), the encoder is divided into two parts: the backbone and the projector. The backbone makes up most of the encoder in terms of trainable parameters, while the projector is usually an [MLP](#) or a [CNN](#). The following section presents the different architectures utilised as the backbone and projector in the encoder network. In all experiments, the predictor architecture is almost identical to the projector architecture, with only layer sizes differentiating the two. Overall, three different encoder architectures were implemented. All architectures utilise a residual neural network as the backbone, while the projector differs more between architectures. [Appendix B](#) details the projectors utilised in each architecture. The main difference between the three encoder architectures is the output shape of the encoding. The first architecture outputs an encoding of shape 640 by 229. The second architecture cuts this roughly in half and outputs an encoding with the shape 640 by 128. The third and final encoder architecture outputs a much smaller encoding of only 1 by 768 values.

5. Architecture

Encoder Architecture 1: Resnet34 With Large Output

The backbone of Encoder Architecture 1 is the default Resnet34 implementation from PyTorch. This architecture is based on residual neural networks, detailed in Section 2.2.4, and consists of 34 convolutional layers with residual connections between layers. The final two layers in the network, an Average Pooling layer and a fully connected layer with 1000 nodes, were removed to avoid an unnecessary bottleneck between the backbone and the projector. Following the backbone is a small projector CNN containing five two-dimensional transposed convolutional layers, responsible for upscaling the encoder’s output from 640 by 128 to 640 by 229. This projector is detailed in Table B.1.

Encoder Architecture 2: Resnet34 With Medium Output

The backbone of Encoder Architecture 2 is identical to the backbone of Encoder Architecture 1. However, the projector network following the backbone is altered such that the output from the encoder is the same shape as the output from the backbone, 640 by 128. As a consequence of this change, the projector in Encoder Architecture 2 is redesigned to both receive and produce tensors of size 640 by 128. This projector consists of three two-dimensional convolutional layers followed by three two-dimensional transposed convolutional layers. A two-dimensional batch normalisation layer follows each fully connected layer in the projector, and a ReLU activation function follows each internal layer. This projector is detailed in Table B.2.

Encoder Architecture 3: Resnet18 With Small Output

In the third revision of the Encoder Architecture, the backbone was changed to Resnet18 in an attempt to avoid overfitting the encoder model. The last two layers in this backbone were kept as is and were not removed as in the two previous encoder architectures. Because of this change, the backbone produced tensors of size 1 by 1000, and the projector was modified accordingly. The projector in Encoder Architecture 3 consists of three linear layers, each followed by batch normalisation layers and ReLU activation functions following the internal layers. Each hidden layer in the projector has a size of 2048, while the output layer has a size of 768. This projector is detailed in Table B.3.

5.3. Fully Supervised System

After the pre-training step, training moves on to the fully supervised step, training on labelled data. Here, the pre-trained encoder model is incorporated into a modified version of the existing Onsets and Frames architecture used by Grønbech (2021). In this thesis, velocity predictions are not utilised, and the velocity predictor shown in Figure 4.2 is thus removed. Instead, this modified Onsets and Frames architecture processes the representation created by the encoder network trained in Section 5.2.

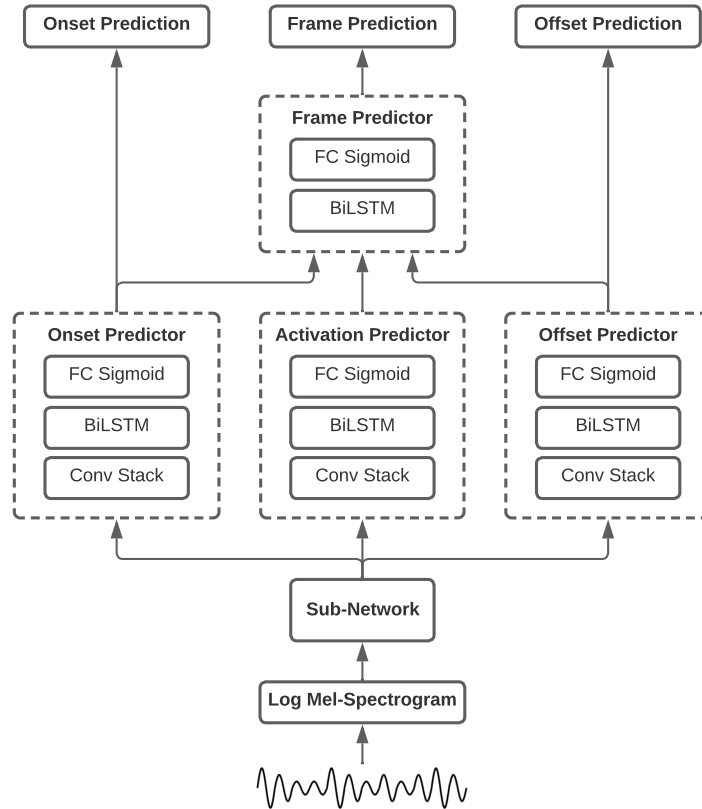


Figure 5.3.: The modified Onsets and Frames architecture utilised in the replacement and early concatenation approaches. The architecture features a sub-network at the beginning of the model. The experiments in this thesis utilise two different sub-networks.

In the experiments carried out in this thesis, three main architectures were tested. The first architecture is referred to as the replacement method and is illustrated in Figure 5.3. In this method, sub-network A, shown in Figure 5.4, is used as the sub-network in Figure 5.3. The sub-network consists only of the pre-trained encoder, which receives a log mel-spectrogram. An output encoding is produced by the encoder and is sent to the rest of the model. The main Onsets and Frames model following the sub-network consists of four main predictors, each responsible for predicting different features in the audio. The output from the encoder is passed through three identical predictors with different target values. The onset and offset predictors are fitted on the notes' beginning and end, respectively. The output from the onset, offset and activation predictors are concatenated and passed along to the frame predictor, which is responsible for predicting notes present in individual frames. Finally, the activation prediction is discarded, and the onset, frame, and offset predictions form the outputs from the network.

5. Architecture

The second architecture, referred to as early concatenation, shares the architecture of the Onsets and Frames model, shown in Figure 5.3, with the replacement method. This architecture differs from the replacement method by utilising sub-network B instead of sub-network A from Figure 5.4. The sub-network receives the same input as before: a log mel-spectrogram passed through a pre-trained encoder. The encoded spectrogram is then concatenated to the original input spectrogram, and this combined input is passed as the input to the Onsets and Frames model.

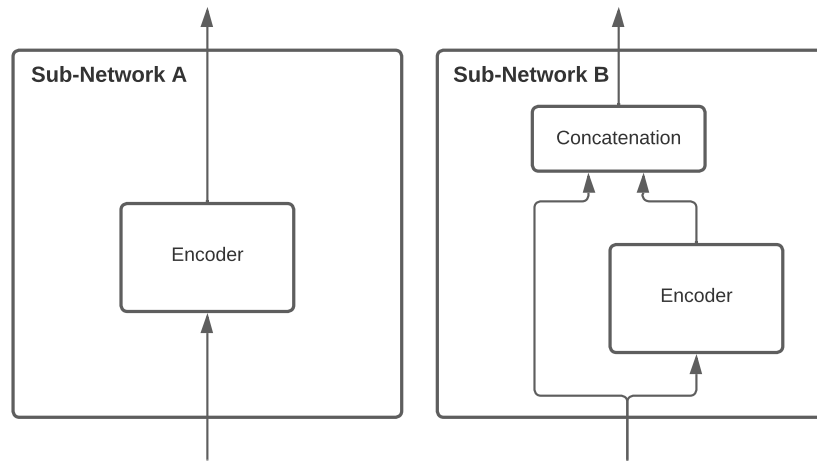


Figure 5.4.: The two different sub-networks featured in the experiments. Sub-network A consists simply of an encoder network passing its output as the output of the sub-network. Sub-network B concatenates the encoder output to the log mel-spectrogram input to the sub-network and outputs this concatenated tensor.

The third architecture, referred to as late concatenation, is illustrated in Figure 5.5. Similarly to the extended Onsets and Frames model by Grønbech (2021), only the log mel-spectrogram is sent to the predictors. However, after the BiLSTM-layers in the Onset, Offset and Activation predictors, the encoding of the original input is concatenated into the network. More specifically, the output from the BiLSTM-layer in each predictor is concatenated with the pre-trained encoder output. Each BiLSTM-layer outputs a tensor of size 640 by 768, while the encoder outputs a tensor of size 1 by 768. The encoder output is therefore repeated along the vertical axis to create a 640 by 768 tensor. This tensor is concatenated with the BiLSTM's output along the horizontal axis, resulting in a 640 by 1536 tensor. Finally, the tensor is passed through the fully connected layers and frame predictor as usual.

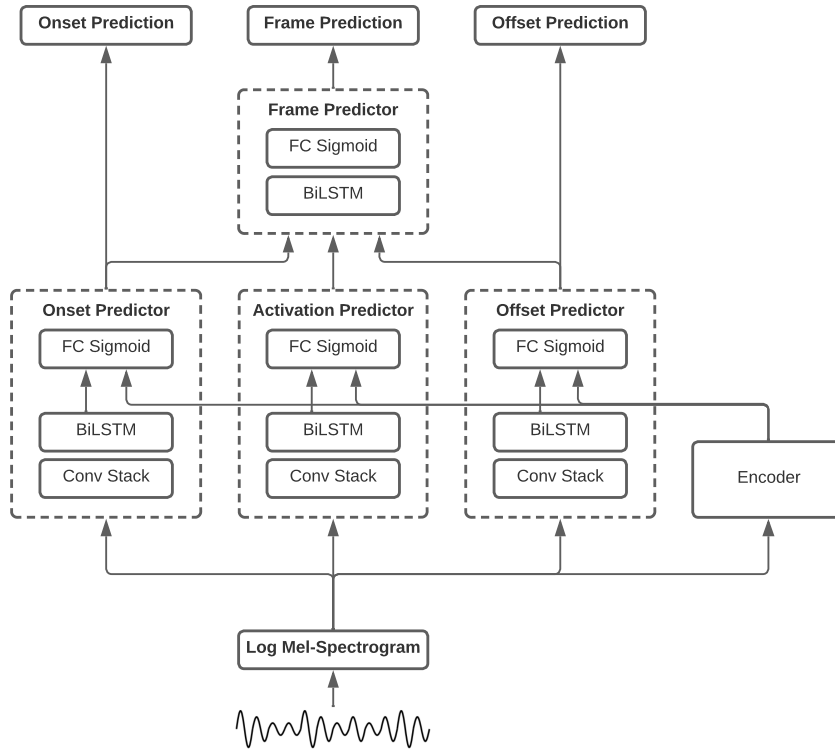


Figure 5.5.: The modified Onsets and Frames architecture utilised in the late concatenation approach. The log mel-spectrogram is passed to each predictor and the encoder. The output from the encoder is then concatenated into the network following the BiLSTM layer in each predictor.












5.4. Post-Processing


As the system's goal is to produce [MIDI](#) Transcriptions, additional processing must be applied to the raw output from the system. A threshold cutoff is used to filter out faint frame, onset and offset predictions such that only the strongest predictions remain. This threshold is normally set to 0.5. As the network is able to predict any value between 0 and 1, only frame, onset and offset predictions stronger than this threshold are preserved.


A simple mapping from the frame predictor to [MIDI](#) could easily be done. However, additional processing steps can be applied to avoid unnecessary errors. For instance, if the output from the frame predictor indicates a note is being played, but no output from the onset predictor indicates the beginning of this note, the frame predictions will be discarded. The same principle applies to the offset predictor. When the offset predictor predicts an offset, and the frame predictor predicts that the same note is present even after the offset, the note will end at the offset. Any extra frames predicted by the frame predictor for that note will not be converted into [MIDI](#) unless another onset is predicted.

5. Architecture

Figure 5.6 illustrates the different ways the model can predict notes with onsets, frames and offsets and how the post-processor handles each of these cases to produce the final MIDI output from the transcriber. Notice how the note is entirely discarded in the MIDI output if the onset is either absent or predicted without any accompanying frames.

Model Prediction	MIDI Output
	
	
	
	
	
	
	

 Frame

 Note onset


 Note offset

Figure 5.6.: Outcome for different note predictions after post-processing step. The left column shows prediction from the Onsets and Frames model. Red indicates a note onset prediction, green indicates a frame prediction, and blue indicates a note offset prediction. The right column shows the resulting MIDI output after the post-processing step.

6. Experiments and Results

This chapter presents the experiments conducted in this thesis and their respective results. Each experiment is designed to answer one of the research questions presented in Section 1. First, the experimental plan is presented, covering the details of each experiment and its purpose. Then, the experimental setup is described, presenting the parameters and information necessary to reproduce the results from each experiment. This includes details about third-party libraries utilised in the implementation, datasets used during training and parameters for the training procedures and spectrogram generation. Finally, detailed results from all experiments are presented and some important results are highlighted.

6.1. Experimental Plan

This section presents the experimental plan for the thesis, in which specific details about each planned experiment are presented. They are designed to answer one of the research questions directly or to provide a point of comparison for later experiments.

The first two experiments serve as baseline experiments. Experiment 0 is designed to verify the implementation of the training algorithm for the pre-training step, and experiment 1 is designed to provide a point of comparison for all following experiments. Experiments 2, 3 and 4 investigate different ways of incorporating a pre-trained encoder model in the extended Onsets and Frames architecture. Experiment 5 is designed to investigate the effect of how augmentations are selected during pre-training, and experiment 6 investigates the effect of increasing the amount of training data during the pre-training step.

6.1.1. Baseline Experiments

The first two experiments act as baseline experiments. The goal is to ensure that the implementations of both the Onsets and Frames model and the Simple Siamese training algorithm are correct and to provide a benchmark for comparison with results produced by later experiments.

Experiment 0: Simple Siamese Verification

To verify our implementation of the Simple Siamese training algorithm, the first experiment compares the difference in accuracy between two models, trained to classify handwritten digits from the MNIST dataset. Both models are identical, with two convolutional layers followed by two fully connected layers with ten nodes in the output

6. Experiments and Results

layer. The first model was trained on the MNIST handwritten digits in 28 by 28 pixel monochrome images. The second model was trained in the same manner. This time, however, the input images were first passed through an encoder producing 28 by 28 pixel image encodings. The encoder consisted of four convolutional layers and was pre-trained using the Simple Siamese training algorithm. Both classifier networks were trained on 16.6% of the MNIST training set, while the encoder was trained using the remaining 83.4% of the training data, stripped of its labels. Finally, both trained classifier networks were tested on the MNIST test set to evaluate their performances.

Experiment 1: Onsets and Frames Baseline

To verify that we can match the results from Grønbech (2021), we will rerun experiments 0, 1 and 3 on *electric bass* and *all instruments*, defined by Grønbech (2021). The results from these runs will be compared to the corresponding results in Grønbech (2021) to ensure that subsequent experiments can be safely compared with the results.

6.1.2. Experiments on Input Replacement

Experiment 2 is designed to answer Research Question 1. For this experiment, an encoder will be added to the Onsets and Frames model. The spectrograms generated from the audio will first pass through the encoder, after which the encoder’s output is sent through the Onsets and Frames model. In this experiment, the encoder will be trained using Simple Siamese Representation Learning (SimSiam) on 20% of the MTG-Jamendo (MTG-J) dataset. The Onsets and Frames model will be trained in a supervised manner, using the same dataset as in experiment 1. To observe how freezing the encoder’s weights affect the result of the transcriber, some experiments have the weights of the encoder frozen after pre-training. The weights are frozen in experiments 2a, 2b, 2d and 2e, while in experiment 2c, they are not.

Experiment 2a: Single Channel Audio

This experiment utilises Encoder Architecture 1, a Resnet34 with a large output encoding, detailed in Section 5.2.2. This encoder consists of a modified version of the default Resnet34 implementation in PyTorch, followed by the projector, responsible for upscaling the output to the desired dimensions. The size of this encoder architecture’s input spectrogram and output embedding is identical (640 by 229 values). The encoder’s weights are frozen after pre-training, and the input spectrograms only contain the electric bass audio channel.

Experiment 2b: Mixed Audio

This experiment is nearly identical to experiment 2a, but the input spectrograms now contain the whole mix of all instruments in each track. The encoder architecture is identical to experiment 2a, and the transcriber model is still trained to only transcribe electric bass.

Experiment 2c: Unfrozen Encoder

In this experiment, the encoder weights are not frozen to investigate how this affects the final result. The encoder architecture is identical to experiments 2a and 2b.

Experiment 2d: Smaller Encoding

This experiment utilises Encoder Architecture 2, a Resnet34 with a medium-sized output encoding, detailed in Section 5.2.2. In this architecture, the projector network following the Resnet34 is modified to produce tensors of size 640 by 128, identical to the output of the modified Resnet34. Encoder weights are once again frozen after pre-training.

Experiment 2e: Removal of Augmentation

In this experiment, the *Random Pitch Shift* augmentation, detailed in Section 5.1, is removed from the set of possible augmentations during pre-training. The encoder architecture is identical to experiment 2d and encoder weights are frozen after pre-training.

6.1.3. Experiments on Concatenation

The following experiments are designed to answer Research Question 2. Both experiments 3 and 4 investigate using a pre-trained encoder in the Onsets and Frames architecture by concatenating an encoding to the data passing through the network.

Experiment 3: Early Concatenation

The same basic setup as in experiment 2d is used in this experiment. However, the Early Concatenation approach detailed in Section 5.3 is utilised. In this method, the output from the pre-trained encoder is concatenated to the original spectrogram, and this combined tensor is passed as the input to the transcriber network. Experiment 3 uses Encoder Architecture 2, detailed in Section 5.2.2 outputting embeddings of size 640 by 128. This embedding results in an input to the Onsets and Frames model of size 640 by 357 when concatenated with the original spectrogram (640 by 229). To test whether the model ignores the encoding or can obtain useful information, the same experiment is run with random initial weights for the encoder in experiment 3b. Essentially, an untrained encoder is used. Encoder weights are not frozen in either experiment.

6. Experiments and Results

Experiment 4: Late Concatenation

In this experiment, the Late Concatenation approach detailed in Section 5.3 is utilised. In this method, the input spectrogram is passed both to the transcriber model, as in the baseline experiments, and to the encoder model. This experiment utilises Encoder Architecture 3, a Resnet18 with a small output encoding, detailed in Section 5.2.2. This encoder produces spectrogram embeddings of size 1 by 768 values. Experiment 4 is divided into two parts, 4a and 4b, and encoder weights are frozen after pre-training in both experiments. The model in 4a transcribes electric bass only, while the model in 4b transcribes all instruments simultaneously.

6.1.4. Experiments on Augmentation Selection Mode

Experiment 5 is designed to answer Research Question 3 directly. In all previous experiments, augmentations are applied using Augmentation Selection Algorithm A, detailed in Section 5.2.1. In this experiment, the setup from experiment 4 is reused, only altering how augmentations are selected.

Experiment 5: Augmentation Selection

Experiment 5 swaps Augmentation Selection Algorithm A with Augmentation Selection Algorithm B, detailed in Section 5.2.1. In experiment 5a, the model is trained to transcribe electric bass, while in experiment 5b, the model is trained to transcribe all instruments simultaneously. Just like in experiment 4, Encoder Architecture 3 is used, and encoder weights are frozen after pre-training.

6.1.5. Experiments on Size of Unlabelled Dataset

Experiment 6 is designed to directly answer Research Question 4, by investigating the effect of altering the amount of unlabelled training data during pre-training.

Experiment 6: More Unlabelled Data

Experiment 6 trains an encoder model on more unlabelled data during pre-training. All previous experiments involving a pre-trained encoder utilised 20% of the data in the *MTG-J* dataset. This experiment increases the amount of data to 50% of *MTG-J*. Experiment 6a trains a model to transcribe electric bass only, while experiment 6b trains a model to transcribe all instruments. The setup is otherwise identical to experiment 5.

6.2. Experimental Setup

This section contains details about the experimental setup to provide reproducibility of the results presented in Section 6.3. It will cover the datasets used, parameters of the predictor and encoder models, spectrogram generation, hardware details, and any specific third-party libraries used in the implementation¹.

6.2.1. Datasets

Two datasets are used in the experiments, [MTG-Jamendo \(MTG-J\)](#) and [Slakh2100](#). [MTG-J](#), the unlabelled dataset, is only used during pre-training of the encoder model. In this thesis, only 50% of this dataset was used during training. This was done to limit the ratio of unlabelled data compared to labelled data. For experiments 2-5, 20% of [MTG-J](#) was used to achieve a split of 90% unlabelled and 10% labelled data. In experiment 6, the amount of unlabelled data was increased to 50% of [MTG-J](#). Additionally, an advantage of not using all of the data from [MTG-J](#) is that this significantly decreases the time required to pre-train the encoder model.

During supervised training, the [Slakh2100](#) dataset was used. Parts of the dataset were omitted during training due to significant errors in either the [MIDI](#)-files or the generated audio. All audio tracks containing pitch bends were also removed, as the [Onsets and Frames](#) architecture is currently unable to transcribe such audio features.

6.2.2. Third-Party Libraries

The neural network models utilised in all experiments were implemented in Python 3.8.6, using [PyTorch 1.9.0](#) and its corresponding [TorchAudio](#) and [TorchVision](#) implementations. [TorchAudio](#) was also used when generating spectrograms from audio files. For evaluating the quality of the transcriptions produced by the model during and after training, the Python library [mir_eval](#) by [Raffel et al. \(2014\)](#) was used. This library is commonly used for evaluating [MIR](#) tasks. A list of all third-party libraries and versions used in all experiments is presented in [Table A.1](#) in [Appendix A](#).

6.2.3. Spectrogram Generation

To ensure that the mel-scaled spectrograms are consistent across both datasets, the two datasets needed to have the same sample rate when generating the spectrograms. However, the audio in [MTG-J](#) has a sample rate of 44.1 kHz, while the audio in [Slakh2100](#) was converted to a 16 kHz sample rate prior to training. Thus, it was necessary to downsample the [MTG-J](#) audio to 16 kHz during loading.

¹<https://github.com/haakon8855/multi-instrument-onsets-and-frames>

6. Experiments and Results

The audio from both datasets is converted to mel-scaled spectrograms before being passed through either model. These spectrograms are generated using PyTorch’s *MelSpectrogram* implementation in order to take advantage of parallel computations on the GPU. Table 6.1 shows the parameters used for spectrogram generation and are identical to the ones utilised by Grønbech (2021).

Table 6.1.: Parameters for mel-scaled spectrogram calculation.

Parameter	Value
Sample rate	16 kHz
Minimum frequency	30 Hz
Maximum frequency	8000 Hz
Window length	2048
FFT size (<code>n_fft</code>)	2048
Hop length	512
Frequency bins	229
Power	1.0

6.2.4. Network Parameters

The two neural networks in the architecture presented in Chapter 5 are set up with a range of different parameters, most of which are constant across all experiments. All such parameters are presented in Table 6.2. The encoder is trained using the SGD optimiser provided by PyTorch. A custom formula, presented in Equation 6.1 by Chen and He (2021), is used to calculate the learning rate for the pre-training step. In all experiments, this results in a learning rate of 0.0078125 due to the constant batch size of 40. This batch size was chosen due to hardware limitations, as this was the maximum possible size on the available A100-GPUs. All experiments utilising the encoder model are trained for 100 epochs, identical to the training duration suggested by Chen and He (2021).

For the Onsets and Frames model, all parameter values are identical to the ones used in Grønbech (2021). This model uses the Adam optimiser, provided by PyTorch, and the learning rate is considerably lower than in the pre-training step. Additionally, this model utilises a learning rate decay procedure with a decay rate of 0.98, every 10000 iterations. Along with the learning rate, the batch size is also considerably smaller compared to the encoder. Some parameters for the Onsets and Frames model are no longer subject to the hardware limitations present in the work by Grønbech (2021). However, all such parameters are kept identical to Grønbech (2021) to ensure consistency and comparability of results.

$$LR = \frac{LR_{base} * BatchSize}{256} \tag{6.1}$$

Table 6.2.: Parameters for both neural networks utilised across all experiments.

Parameter	Value
Encoder	
Base learning rate	0.05
Weight decay	0.0001
Momentum	0.9
Batch size	40
Epochs	100
Onsets and Frames	
LR (Learning rate)	0.0006
LR decay rate	0.98
LR decay interval	10000
Momentum	0.9
Batch size	8
Iterations	50000

6.2.5. Hardware

All experiments are run on the IDUN HPC Cluster at NTNU, containing numerous enterprise-level GPUs. The IDUN HPC Cluster has several P100, V100 and A100-GPUs. The latter are available in both 40GB and 80GB VRAM variants. Due to the large batch size requirement of the Simple Siamese training algorithm, and the large size of each input spectrogram, all experiments carried out had to be run on the 40GB A100-GPUs.

To accommodate the large amount of data, a PyTorch data loader was created for the [MTG-J](#) dataset in order for data to be loaded properly from disk during training. To speed up this process, the loader was set up to utilise 10 separate processes, each loading one training example in parallel. During the pre-training step, this sped up the loading times between batches by a factor of 10.

6.3. Experimental Results

This section contains the results from the experiments presented in section 6.1. All the results from every experiment are presented in the tables throughout this section. First, the baseline experiments are presented, followed by the experiments on [SimSiam](#) in [AMT](#). At the end of this section, averaged results from running the same experiments multiple times are presented.

6. Experiments and Results

6.3.1. Experiment 0: Simple Siamese Verification

Table 6.3.: Results from experiment 0 when running the two classifiers on the MNIST test set.

Model	Correct Classifications	Total Classifications	Accuracy
Baseline	9253	10000	92.53%
Pre-Trained	9277	10000	92.77%

The results from experiment 0 are presented in [Table 6.3](#). The first model, trained only on 10000 labelled examples from MNIST achieved an accuracy of 92.53%. The second model, whose input was first passed through a pre-trained encoder trained using the Simple Siamese training algorithm, achieved an accuracy of 92.77%.

6.3.2. Experiment 1: Onsets and Frames Baseline

The results from experiment 1 can be found in [Table 6.4](#). It is clear that experiment 1a performs better than the other experiments. All F1 scores from experiment 1a, are above 95%. It is worth noting that experiment 1a operates on individual source audio, while the rest operates on a mix of instruments in the audio. Results for experiment 1b are, however, still quite high with respect to the F1 scores for both frame and note onset, while there is a significant drop in the note with offset F1 score. Experiment 1c transcribes all instruments simultaneously, and not just electric bass. In this experiment, there is a noticeable difference in all scores compared to experiments 1a and 1b. The experiments that have a counterpart in [Grønbech \(2021\)](#), presented in [Table 6.5](#), all have slightly lower scores. This difference is around 1-2 percentage points. The results from experiments 1b and 1c are arguably the most important results from experiment 1, as all later experiments are compared with these result.

Table 6.4.: Results from experiment 1 and 2

Experiment	Instrument	Source	Frame			Note			Note /w offset		
			P	R	F ₁	P	R	F ₁	P	R	F ₁
1a	Electric Bass	Individual	97.2	96.2	96.6	97.5	96.5	96.9	96.0	95.0	95.5
1b	Electric Bass	Mix	93.0	90.1	91.3	91.5	90.3	90.7	83.9	82.7	83.1
1c	All	Mix	75.2	68.7	71.1	81.7	69.7	74.3	40.6	35.4	37.4
1d	Electric Bass	Mix	93.6	91.7	92.5	91.2	92.4	91.6	84.4	85.5	84.8
2c	Electric Bass	Mix	88.7	47.0	58.8	92.9	39.7	52.3	74.9	33.7	43.8

²Experiment 1c attempted to transcribe all instruments simultaneously while other parameters remained equal to that of experiments 1a-1c in [Grønbech \(2021\)](#).

Table 6.5.: Correlation between rerun-experiments in experiment 1 and their counterpart in Grønbech (2021).

Experiment	Equivalent in Grønbech (2021)
1a	0a
1b	1a
1c	No equivalent ²
1d	3a

6.3.3. Experiment 2: Input Replacement

The following experiments investigate the effects of incorporating a pre-trained encoder model in the Onsets and Frames model. Table 6.4 shows the results from experiment 2, focusing on input replacement. This was tested by passing the input spectrogram through a pre-trained encoder before passing the produced encoding through the Onsets and Frames model. For experiments 2a, 2b, 2d and 2e, encoder weights were frozen after pre-training, such that the encoder is not altered when training the Onsets and Frames model. These experiments received precision, recall and F_1 scores of 0.0% for all metrics. This indicates that these models were unsuccessful in transcribing any notes. Experiment 2c tests an unfrozen encoder model, enabling the fully supervised training step to further train the encoder as it trains the Onsets and Frames model. Results from this experiment are presented in Table 6.4. Experiment 2c is the only model experiment 2 to achieve note F_1 -scores above 0.0%.

6.3.4. Experiment 3: Early Concatenation

Both experiments in experiment 3 have quite similar results, which can be observed in Table 6.6. Experiment 3b, utilising an untrained encoder with random initial weights, yields slightly better results than the model trained in experiment 3a. This can be seen especially in the note with offset F_1 score in experiment 3b.

Table 6.6.: Results from experiments 3, 4 and 5

Experiment	Instrument	Source	Frame			Note			Note /w offset		
			P	R	F_1	P	R	F_1	P	R	F_1
3a	Electric Bass	Mix	91.0	87.6	89.1	89.5	88.5	88.9	80.4	79.5	79.8
3b	Electric Bass	Mix	91.2	88.6	89.8	90.2	87.5	88.6	84.0	81.4	82.5
4a	Electric Bass	Mix	91.1	86.9	88.8	90.6	87.6	88.9	84.0	81.2	82.4
4b	All	Mix	77.9	70.4	73.5	81.6	69.5	74.2	42.9	37.7	39.6
5a	Electric Bass	Mix	93.2	89.6	91.2	91.7	88.8	90.0	83.8	81.1	82.3
5b	All	Mix	77.7	69.2	72.7	81.7	68.8	73.5	42.8	37.2	39.1

6. Experiments and Results

6.3.5. Experiment 4: Late Concatenation

The results of experiment 4 can be observed in [Table 6.6](#). Experiment 4a, which only operates on electric bass, has substantially better scores than experiment 4b. This is especially true when it comes to offset, where the difference in score is over 40 percentage points. The models trained in experiments 4a and 4b achieve note F_1 scores of **88.9** and **74.2**, respectively. While the scores from experiment 4a are lower compared to their baseline equivalents in experiment 1b, scores from experiment 4b seem to increase compared to the equivalent results from experiment 1c.

6.3.6. Experiment 5: Augmentation Selection Mode

[Table 6.6](#) shows the results from experiment 5. It is apparent from these results that the change to Augmentation Selection Algorithm B has improved the scores when transcribing only electric bass from the audio mix. Comparing experiment 5b with 4b, however, it seems as though the scores decrease slightly in experiment 5b. After seeing these results, experiments 1c and 5b were rerun four times each. This provided five runs for each experiment which were then averaged to decrease any variation present in the results from individual runs. [Table 6.7](#) presents the averaged results from these repeated runs. These results do not exhibit the apparent increase in performance between experiments 1c and 5b observed in [Table 6.6](#), and indicate that the models trained in experiment 5b are, on average, equal to the models trained in experiment 1c. Individual results from each rerun are available in [Appendix C](#).

Table 6.7.: Averaged results from repeated runs. Source data from individual runs are available [Appendix C](#).

Experiment	Instrument	Source	Frame			Note			Note /w offset		
			P	R	F_1	P	R	F_1	P	R	F_1
1c	All	Mix	77.3	68.6	72.0	83.7	68.3	74.3	42.9	36.0	38.7
5b	All	Mix	77.7	68.3	72.0	83.3	67.8	73.7	43.0	36.1	38.7
6a	Electric Bass	Mix	89.8	87.9	88.6	89.1	88.4	88.5	82.0	81.1	81.4
6b	All	Mix	77.2	67.1	71.1	83.8	67.7	73.7	42.9	35.8	38.5

6.3.7. Experiment 6: More Unlabelled Data

The two experiments in experiment 6 were both run five times each to provide a more exact point of comparison to earlier experiments. The averages from these runs are presented in [Table 6.7](#) and do not seem to exhibit any increase in performance compared to the results of experiments 1b or 1c. This is most evident when comparing the averaged results of experiments 1c and 6b. The results from each individual rerun are available in [Appendix C](#).

7. Evaluation and Discussion

This chapter will evaluate and discuss the results presented in Section 6.3. First, the research questions are evaluated by investigating the experimental results and comparing these against results from the baseline experiments and the related work. This is followed by discussing possible reasons for the observed results and how this affected certain architectural choices during experimentation.

7.1. Evaluation

This section will analyse the results in detail and attempt to answer the research questions and, finally, the overall thesis goal: *Examine if self-supervised pre-training can be used to increase the performance of AMT models.* In general, the results presented in Section 6.3 do not indicate any direct benefit of combining SimSiam with AMT. In most experiments, the model including a pre-trained encoder performs equal to or slightly worse than the baseline models. However, some interesting findings in the raw output from the models in some experiments indicate that the encoder can extract some information from its input data given the proper parameters. This calls for further investigation into why this happens and if it ultimately can be utilised to improve SOTA multi-instrument AMT performance.

Experiments 0 and 1 served as baseline experiments, not to directly contribute to any specific research question but to verify the implementation and provide points of comparison for all later experiments. The primary purpose of experiment 0 was to verify the implementation of the SimSiam training algorithm. It also served as a way to test that the hardware used for later experiments was set up correctly. The results from experiment 0 were uplifting. While the pre-trained model only marginally outperformed the baseline model, it provided evidence that the SimSiam-algorithm implementation was correct.

Experiment 1 was designed to provide a clear point of comparison for later experiments. Most of the later experiments in this thesis share a fundamental similarity to one of the sub-experiments in experiment 1. However, the results from experiment 1 are slightly worse than those from the equivalent experiments carried out by Grønbech (2021). Experiment 1a achieves frame, onset and onset with offset F_1 scores of **96.6**, **96.9** and **95.5**, respectively. These scores are consistently lower than in Grønbech’s setup, achieving F_1 scores of respectively **98.5**, **99.1** and **97.9** in the equivalent experiment. Although not as profound, similar results can be observed when comparing experiments 1b and 1d to their counterparts in Grønbech (2021). This suggests that some part of our setup could be set up differently than in Grønbech (2021), which may have been the

7. Evaluation and Discussion

cause for this discrepancy. To isolate the effects of adding a pre-trained encoder to the transcriber model, results from later experiments will be compared mainly with results from experiment 1 rather than directly with results from the related work.

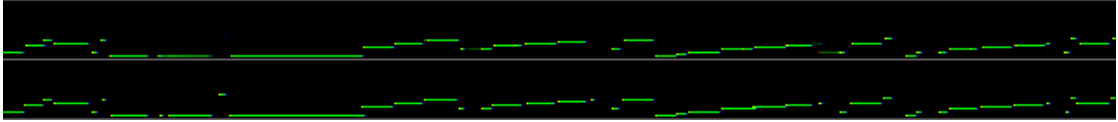


Figure 7.1.: Model predictions (top) and ground truth MIDI (bottom) from experiment 1b when evaluated on audio from the validation dataset.

To further evaluate the models’ performance, model predictions from the validation phase can be compared manually to the ground truth MIDI as shown in Figure 7.1. From the model output shown on top, it is clear that most notes are present and correct. While some errors occur in the baseline models, evident by the models’ F_1 scores, these errors seem to be caused mainly by missing onset predictions or wrong timing for predicted frames, onsets or offsets. Similar detailed model predictions from all experiments are available in Appendix E.

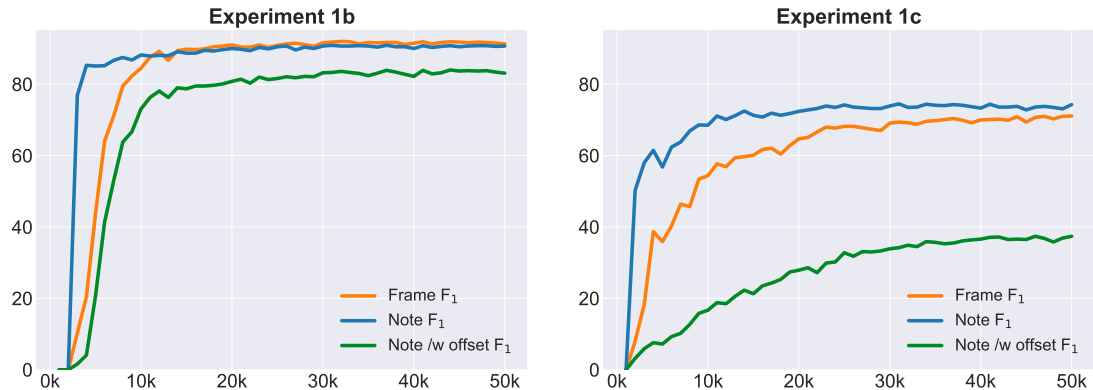


Figure 7.2.: Frame, note and note-with-offset F_1 scores for experiment 1b and 1c.

During supervised training, the model is evaluated on the validation dataset every 1000 iterations. Figure 7.2 shows the F_1 score during training in experiments 1b and 1c as a function of iterations on the horizontal axis. Consistent across all experiments transcribing electric bass is the F_1 score of **0.0** during the first few thousand iterations. On the other hand, when transcribing all instruments simultaneously, like in experiment 1c, the F_1 score seems to increase earlier, after just 2000 iterations. However, after training, the F_1 score is much lower when transcribing all instruments than in experiments transcribing electric bass only. The F_1 score graphs from experiment 1 in Figure 7.2 are consistent with results from the related work (Grønbech, 2021). However, the maximum F_1 scores after training seem to be somewhat lower in our reruns.

Research Question 1 *How does the use of a pre-trained encoder, used to replace the input to the transcriber network, affect the performance of an AMT model?*

Experiment 2 was designed to answer this research question by passing the input spectrograms through an encoder before sending this encoding through the transcriber network. The encoder was trained using the [SimSiam](#) training algorithm. The results from experiments 2a and 2b show that replacing the transcriber input during supervised training did not look very promising. Both models trained in these experiments achieved F_1 scores of **0.0** for both frame, note onset and note-with-offset predictions. This indicates that the models did not produce any predicted frames, onsets or offsets when validation spectrograms were passed through the model. This conclusion is also supported by the raw model outputs presented in [Appendix E](#). Experiments 2a and 2b are equivalent to experiments 1a and 1b, respectively, except that the inputs in experiments 2a and 2b are passed through the encoder before reaching the transcriber network. As this was the only difference between the models, we would expect somewhat similar results if the encoder was able to produce meaningful embeddings of the input spectrograms.

Experiment 2c was the first model including a pre-trained encoder to successfully produce predictions after training, evidenced by its non-zero precision and recall scores. The model achieves a note F_1 score of **52.3**, much lower than the expected score of around **90.7** from experiment 1b. It is worth noting that the onset precision score for experiment 2c of **92.9** is a little higher than its equivalent in baseline experiment 1b of **91.5**. However, the recall score is more than **50** percentage points lower in experiment 2c. The high precision score indicates that when the model in 2c predicts an onset, it is usually correct. Still, the low recall score indicates that it fails to predict the majority of the notes present in the recording. Additionally, it is difficult to tell whether this increase in precision is even significant without repeating the experiment.

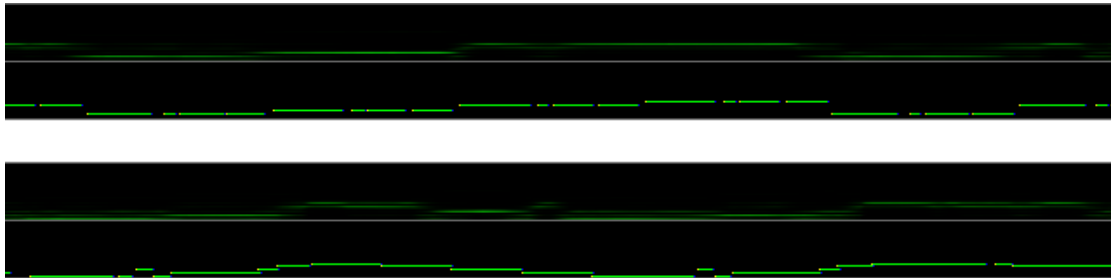


Figure 7.3.: Predictions and ground truth from experiment 2d (top two rows) and experiment 2e (bottom two rows). For each experiment, predictions on validation data are shown in the upper row, and ground truth [MIDI](#) is shown in the lower row.

In an attempt to mitigate the apparent collapse of the encoder in earlier experiments, experiments 2d and 2e were designed to create smaller spectrogram embeddings. Once again, the transcriber model is unable to predict any notes, with precision, recall, and F_1 scores of **0.0** across the board. However, looking at the raw output from the transcriber

7. Evaluation and Discussion

in Figure 7.3, the model seems to exhibit some ability to produce faint frame predictions. The same behaviour is also observed in experiment 2e. Sadly, the model output in experiments 2d and 2e is entirely devoid of onset predictions, so the post-processor does not preserve any notes, resulting in the F_1 score of **0.0**. However, because encoder weights were frozen during experiments 2d and 2e, the results from these experiments are the most interesting yet. They prove the encoder does not collapse and is, in fact, able to preserve some information from the input spectrogram.

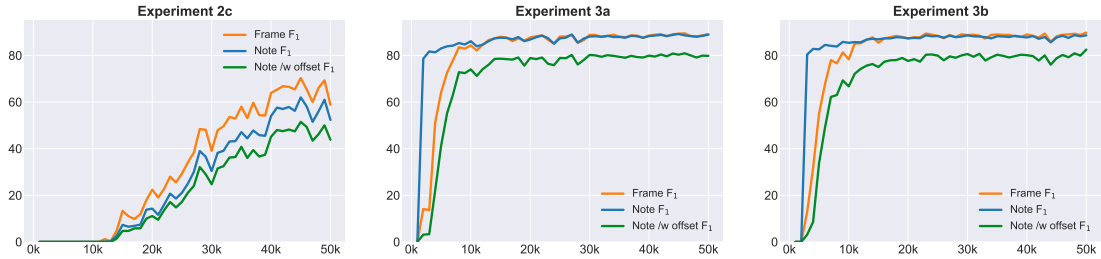


Figure 7.4.: Frame, note and note-with-offset F_1 scores for experiment 2c, 3a and 3b.

When looking at the F_1 scores during training from experiment 2c in Figure 7.4, it is clear that the model has problems learning as quickly as in the baseline experiments. The model achieves F_1 scores of **0.0** until training is over 20% complete. Equivalent graphs for all experiments, including experiments 2b and 2d, are included in Appendix D. In these experiments, F_1 scores were consistently at **0.0** during the entire training procedure for both frame, onset and onset with offset metrics.

Research Question 2 *How does the use of a pre-trained encoder, used to supplement the input to the transcriber network affect the performance of an AMT model?*

Experiments 3 and 4 were designed to answer Research Question 2, investigating the concatenation of spectrogram encoding into the transcriber model. Experiment 3 concatenates this encoding to the input spectrogram, resulting in a larger network input during the fully supervised training phase. Experiment 3a achieves a note F_1 score of **88.9**. Again, lower than the baseline F_1 score of **90.7** from experiment 1b. However, this model easily outperforms the model utilising the replacement method from experiment 2c by achieving a much better balance between the precision and recall metrics for both frame, note and note-with-offset.

Experiment 3b was designed to verify the value of the pre-trained encoder in the transcriber model by rerunning experiment 3a with an untrained encoder. Interestingly, the results in experiment 3b are difficult to differentiate from the results in 3a. Metrics for both frame and note transcription are nearly identical, while only the metrics for note-with-offsets differ. The experiment utilising the untrained encoder model outperforms the model using a pre-trained encoder, especially for the note-with-offset precision scores of **84.0** and **80.4**, respectively. This could indicate that the encoder model has difficulty producing meaningful embeddings of the input spectrograms, which is further supported

by the F_1 plots in Figure 7.4 for experiments 3a and 3b. Both plots resemble results from experiment 1b. A notable observation is that the F_1 score in experiment 3a seems to increase earlier than in experiment 3b. However, this could be explained by random variations in the initial model weights.

Experiment 4 attempts to answer Research Question 2 by concatenating the spectrogram encoding into the transcriber network deeper in the network. Thus, the results from this experiment could be valuable when compared to both experiment 3 and experiment 1. The encoder utilised in experiment 4 outputs a significantly smaller encoding than in earlier experiments. Additionally, the encoder in experiment 4 was frozen after pre-training, which was not the case for experiment 3. The model trained in experiment 4a achieves similar results to the model in experiment 3b. It is worth pointing out that, as in experiment 3b, the note-with-offset precision score for experiment 4a is comparable to that of baseline experiment 1b. At the same time, other metrics are consistently lower in experiments 3b and 4a.

Experiment 4b utilises the same pre-trained encoder as in 4a while attempting to transcribe all instruments simultaneously, and not just electric bass, as in earlier experiments. Experiment 4b’s equivalent baseline experiment, without an encoder model, is experiment 1c. This model outperforms the baseline model by over 2 percentage points in all metrics except for note precision, recall and F_1 score. This might indicate that the encoder model is able to extract some information from the input spectrograms, which may be valuable when transcribing all instruments simultaneously. Even though the increase in performance is relatively small, it indicates that further investigation may be necessary.

The F_1 scores during training of experiments 4a and 4b, illustrated in Figure 7.5, also seem to increase faster than in the baseline experiments. Especially noticeable is how the frame F_1 score follows the note F_1 score much more closely in experiments 4a and 4b than in experiments 1b and 1c. It is also apparent from Figure 7.5 that even though the F_1 scores end up at a similar level as in experiment 1, the models seem to learn faster during the beginning of training and are eventually caught up by the baseline models.

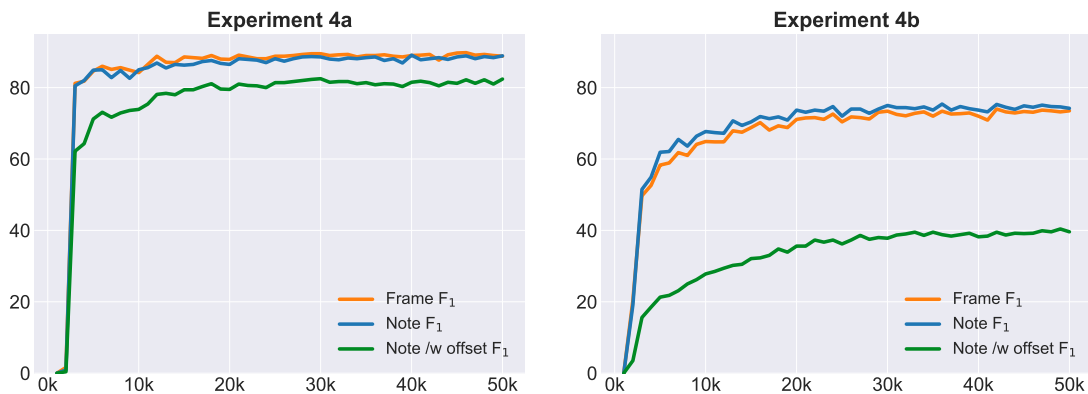


Figure 7.5.: Frame, note and note-with-offset F_1 scores for experiment 4a and 4b.

7. Evaluation and Discussion

Research Question 3 *How do different modes of selecting data augmentations affect the performance of an AMT model?*

Experiment 5 was designed to directly answer Research Question 3 by reusing the setup from experiments 4a and 4b while swapping Augmentation Selection Algorithm A with Algorithm B. This algorithm utilised an augmentation selection scheme similar to the related work presented in Section 4.6. In terms of model architecture, the models trained in experiments 5a and 5b are identical, with the former trained to transcribe electric bass only, while the latter was trained to transcribe all instruments simultaneously. Results from 5a are overall better than those of 4a, with a note F_1 score of **90.0**. Compared to the baseline experiments, 5a performs quite similarly to Experiment 1b. For Experiment 5b, the opposite is true. The model’s performance is very similar to that of the model trained in experiment 4b, and both models seem to outperform the baseline model trained in 1c.

The fact that 5b seemed to slightly outperform experiment 1c called for further investigation to discover whether or not this was a coincidence. Reruns of experiments 1c and 5b were therefore executed to expose the variance present in each run of these experiments. The average values of these reruns indicate that the difference was, in fact, due to a random variation in the results and not because either model performed better on average. The results from reruns of experiments 1c and 5b seem to indicate that the encoder model present in 5b does not affect the model’s transcription ability after training.

Research Question 4 *How does the amount of unlabelled data affect the results of the AMT model?*

Experiment 6 was designed to directly answer Research Question 4 and thus investigate the effect of increasing the amount of unlabelled training data without increasing the amount of labelled data. After the significant variance between individual runs was discovered in experiment 5, experiments 6a and 6b were also run multiple times each. The presented average scores from experiment 6a in Table 6.7 are lower than the results from experiment 5a for all metrics. This difference is most significant for the frame prediction metrics but is still present for onset and onset with offset predictions, especially in the F_1 scores.

The results from experiment 6b similarly indicate a decrease in performance when the unlabelled dataset is increased. However, compared with the equivalent experiment 5b without the increased dataset, this increase only holds for frame prediction metrics, and the difference is less significant than between experiments 6a and 5a. These results seem to indicate that increasing the size of the unlabelled dataset to this extent does not contribute positively to the performance of the final transcription model.

7.2. Discussion

This section will discuss the results presented in Section 6.3. Section 7.1 discovered that most experiments carried out in this thesis were unable to improve the performance of the extended Onsets and Frames model proposed by Grønbech (2021). This discussion will attempt to shed light on possible causes for this performance in light of the results, the architecture and the training data.

7.2.1. Discussion of Results

The models trained in experiments 2a and 2b had some obvious problems, evident by the precision, recall, and F_1 scores all equal to **0.0** after training. With precision and recall scores at **0.0**, the model is unable to predict any notes correctly. This can be verified by looking directly at the models' outputs from these experiments in the figures presented in Appendix E. A possible explanation for the empty transcriptions produced during experiments 2a and 2b could be that the pre-trained encoder model produces similar embeddings for all input spectrograms. If this were the case, it would result in similar inputs to the transcriber model, which in turn would lack the required information to transcribe the notes present in the original input spectrogram. This behaviour of the encoder is a possible outcome when utilising the SimSiam training algorithm and is referred to as *model collapse* by Chen and He (2021). It is likely possible to investigate whether the encoder collapsed or not. For instance, by using some defined metric to calculate the difference between embeddings of different spectrograms. The standard deviation between embeddings is one such metric. However, this investigation was not attempted during experimentation. In hindsight, this could provide clues as to why the models trained in experiments 2a and 2b were unable to produce frame, onset and offset predictions.

In experiment 2c, the model was able to produce both onset, offset and frame predictions. However, the only difference between experiments 2b and 2c was that the encoder's weights were no longer frozen during supervised training. As a result, gradients were calculated for both the encoder and transcriber models, allowing the encoder to be trained further after its initial pre-training. The results point to the conclusion that the encoder in experiment 2c did not learn enough during pre-training and instead received most of its training during the supervised training step.

In experiments 2d and 2e, the models were able to produce faint frame predictions, evident by their raw output illustrated in Figure 7.3. However, these frame predictions seem to be washed out and stretched in the horizontal time dimension. To achieve this effect, some information from the input spectrogram regarding the timing of each note must be missing from the embedding produced by the encoder. As the encoder is only trained in a self-supervised manner, it receives no direct penalty if the time dimension of its input data is neglected in the embedding. Its only goal is to preserve as much information as possible from the input spectrogram such that the SimSiam predictor is able to predict the encoder's output for both spectrogram augmentations. Nonetheless, the results prove that the encoder model used in experiments 2d and 2e does not collapse

7. Evaluation and Discussion

and is able to create meaningful embeddings of the input spectrograms. While later experiments were unable to take advantage of this, the results from experiments 2d and 2e do call for further investigation into the use of self-supervised representation learning in combination with AMT.

The observed stretched quality of the frame predictions from experiments 2d, and 2e could also be the result of a wrong choice of augmentations. Of the five augmentations utilised in this thesis, only the Gaussian Blur and the Random Crop-and-Stretch augmentations move data in the spectrogram’s time dimension (vertical). In experiment 2e, the Random Crop-and-Stretch augmentation was removed from Augmentation Selection Algorithm A. However, this did not produce any significant difference in the model’s performance or output. In hindsight, it may have proven more valuable to explore the effect of removing other augmentations or decreasing the intensiveness of the augmentations applied.

During experiment 3, a transcriber model was trained using the early concatenation approach with a pre-trained encoder in experiment 3a and an untrained encoder in experiment 3b. Surprisingly, the transcriber model with the untrained encoder outperformed its pre-trained counterpart. A possible explanation for this behaviour is that the encoder from experiment 3a has overfitted on the unlabelled training data and is unable to handle the Slakh2100 dataset. Model collapse during pre-training could also be another possibility and would explain why an encoder with random weights could lead to the observed results. However, this explanation is less likely because of the transcriber outputs seen in experiments 2d and 2e and the fact that the encoder architecture is identical across experiments 2d, 2e, 3a and 3b.

During all experiments utilising the late concatenation approach, encoder weights were frozen after pre-training. Similarly, most runs in experiment 2 also utilised frozen encoder weights. The fact that encoder weights were not frozen during experiment 3 devalues the results from this experiment and makes it difficult to compare experiment 3 with other experiments. In hindsight, encoder weights should have been kept frozen during experiment 3 as well.

Another possible issue, which may lead to the observed results, is the augmentations applied during pre-training. The model trained in experiment 5b seemed to decrease performance compared to experiment 4b, with the only difference being the Augmentation Selection Algorithm. This could be caused by the fact that Augmentation Selection Algorithm B has the potential to apply multiple augmentations at once. If the individual augmentations were too aggressive, meaning they altered the spectrogram too much, it could cause problems during pre-training.

During experiment 6, two transcriber models were trained using encoders pre-trained on a larger unlabelled dataset than the encoders in earlier experiments. These models were unable to increase performance compared to their equivalent models from experiment 5. Increasing the amount of training data will, in most cases, give a model better chances of being able to generalise well to new unseen data and should reduce the chances of overfitting. Thus, the results from experiment 6 seem to contradict the previous explanation of the encoder model overfitting on the unlabelled dataset. If overfitting did happen, experiment 6 should have seen an increase in performance.

When looking at the results from all experiments, it seems the difference between precision and recall is larger in the models that include an encoder. In these experiments, precision is always higher than recall. Additionally, in experiments 2 through 6, it seems that the corresponding recall score is always lower when the precision score is better than in the equivalent baseline experiment. This suggests that the encoder encourages the model to predict less often, but the prediction is more often correct when a prediction occurs. However, as this observation also holds for experiment 3b, containing the untrained encoder, it might only be the presence of the encoder network itself and not the pre-trained encoder model causing this behaviour.

7.2.2. Discussion of Architecture

If the encoder is unable to produce meaningful embeddings of the spectrogram it is presented with, the transcriber model will, in turn, have a hard time predicting notes using the replacement approach. A potential explanation for this behaviour is that the encoder network may have been either too large or too small. If the network is too small, the encoder could not pass through enough information from the input spectrogram to the encoding. Each input spectrogram covers 20 seconds of audio and a frequency range from 30 Hz to 8000 Hz. In an average piece of western music, a 20-second excerpt could include a considerable amount of information. Perhaps too much information for the encoder to handle. Even though we would still expect the encoding to contain some information, the amount could be minimal and even entirely irrelevant to the downstream task. This is especially relevant in experiments 4 and onward, where the backbone in the encoder architecture was swapped with a smaller ResNet, compared to the encoder architectures used in earlier experiments.

On the other hand, a network too large for the encoding size could easily be overfitting on the unlabelled data and perform poorly when presented with the labelled dataset. This could explain the results in experiment 2, as the two datasets utilised in this thesis are not perfectly similar. The audio in the Slakh2100 dataset is rendered from the target [MIDI](#), while the audio in the [MTG-J](#) dataset is mostly a collection of recordings of human performances. The sound quality of individual instruments would vary more greatly between tracks in [MTG-J](#) than in Slakh2100, and unlike the Slakh2100 dataset, some tracks in [MTG-J](#) even contain vocal performances.

If the problems with the encoder did not occur because of its size, it is still a plausible explanation that the bottleneck in the encoder could be sub-optimal. In encoder architectures 1 and 2, the last two layers in the ResNet were removed to avoid this

7. Evaluation and Discussion

bottleneck. Typically, the ResNet produces an output of size 1 by 1000 values. After this alteration, the ResNets in encoder architectures 1 and 2 produced outputs of size 640 by 229 and 640 by 128, respectively. In encoder architecture 3, however, the ResNet was unaltered and thus produced outputs of size 1 by 1000 values. The final output from the encoder in this architecture was of size 1 by 768 values, almost 200 times smaller than the output encoding from encoder architecture 1. This change was carried out to produce encodings of similar size to that of the encoder used by [Chen and He \(2021\)](#) and to allow for concatenation into the transcriber network. In hindsight, however, an encoding of this size will likely not capture all the necessary information in the spectrogram. An important distinction between the machine learning tasks tested in the related work ([Chen and He, 2021](#); [Grill et al., 2020](#)) and *AMT* is that the related work only tests their methods on image classification. For image classification, where there may be around 1000 image classes at most, a ResNet output of 1000 values may be acceptable. However, in *AMT*, where the model is trained to produce a prediction of 640 by 88 values, an encoding of this size may not be feasible.

Self-supervised representation learning algorithms are, in most cases, utilised for simple classification tasks where the model input belongs to exactly one of several categories. While *AMT* is considered a binary classification problem, the input can contain an arbitrary amount of notes. Thus, the number of classifications equals the number of possible combinations of notes in the ground-truth *MIDI*. This could make extracting data from the input exponentially harder than in simple classification tasks, which could be another reason for the results observed in this thesis.

During all experiments, batch sizes during pre-training were set to 40 songs. This batch size is much smaller than what is utilised by the related work on self-supervised representation learning. [Chen and He \(2021\)](#) recommend a batch size of at least 256, and show that performance decreases as batch size decreases. Due to the size of the encoder model utilised in this thesis and the hardware limitations presented in [Section 6.2.5](#), larger batch sizes were unfortunately impossible and may have contributed negatively to the encoder’s performance.

An inherent flaw with the concatenation approach tested in experiments 3 onward is that the transcriber model could, during supervised training, learn to ignore the encoder model’s output. One could also argue that this could happen more easily in the late concatenation experiments (experiments 4-6). These experiments utilise a considerably smaller encoding, concatenated much later in the transcriber network. In this case, the encoder output is smaller than the data it is concatenated to, and the concatenated data ultimately passes through fewer layers than in the early concatenation approach. This explanation is also backed by the results from experiment 3b, where the transcriber model trained on the untrained encoder showed no significant performance decrease when compared to experiment 3a.

7.2.3. Discussion of Data

The datasets utilised in this thesis were chosen to be as similar as possible and provide a sufficient amount of unlabelled data in relation to labelled data. Due to the synthetic nature of the audio in the Slakh2100 dataset, perfect similarity between the datasets would be nearly impossible when considering the currently available datasets presented in Section 3. The [MTG-J](#) dataset was chosen due to its size, instrumental variety and similarity in style to the Slakh2100 dataset. A difference is, however, still quite noticeable and may have impacted results in an unfortunate manner. When comparing audio from both datasets, the audio from Slakh2100 appears to have a robotic [MIDI](#)-quality to it. Furthermore, audio in [MTG-J](#) often contains singing and other instruments not present in Slakh2100. This leads to the assumption that if the encoder was able to learn and extract musical features during training on [MTG-J](#), it might have been too difficult to adapt this knowledge for the audio in Slakh2100.

8. Conclusion and Future Work

In this chapter, we will attempt to summarise the findings from the experiments and the discussion of these results, followed by a presentation of notable contributions resulting from the work in this thesis. Finally, ideas for future work in the field of [AMT](#) as a whole are presented.

8.1. Conclusion

This thesis started out with the goal of investigating if self-supervised pre-training can be used with [AMT](#) models to improve their transcription performance. While this is not the first work investigating self-supervised learning in the field of [AMT](#), none of the related work has explored the use of Siamese representation learning methods to mitigate the low amount of labelled training data available for [AMT](#). Overall, the results from the experiments carried out in this thesis indicate that combining the [SimSiam](#) training algorithm with the Onsets and Frames architecture did not positively affect transcription performance. Several different architectures were tested, and it was shown in experiments 2 and 4 that the encoder was able to learn. While this was not enough to improve the performance of the transcription model above the baseline results, it implies that further research on this method could be worthwhile.

When evaluating the first research question, examining the use of a pre-trained encoder to replace the input to the transcription model, it was clear from the results that this approach for combining the [SimSiam](#) training algorithm with the Onsets and Frames model was not suitable. Performance was severely lacking during experimentation on this method and, in most cases, the model failed to produce any predictions when encoder weights were frozen. While freezing the weights in this experiment resulted in F_1 scores of **0.0**, the raw output from the model proved that the encoder could learn something during pre-training. This significant result prompted further experimentation on how the encoder was combined with the transcription model.

The second research question in this thesis was designed to examine different ways of concatenating the encoder output into the transcription model. Neither the early concatenation nor the late concatenation approaches managed to outperform the baseline models. Still, the results from these experiments matched the performance of the baseline models more closely than earlier tests. In general, the difference between precision and recall seemed to be more prominent in all the models that included an encoder, with precision being higher than recall. The assumption is that this was caused by the encoder output, which likely lacks information. When concatenated into the transcriber network, this output may divert attention from the original input spectrogram. It was also shown

8. Conclusion and Future Work

that the models trained using the late concatenation approach appeared to reach peak F_1 scores faster than in the baseline experiments, especially when transcribing all instruments simultaneously. This indicates that the encoder can provide the transcription model with some extra information, proving beneficial initially but ultimately preventing the model from surpassing the performance of the baseline experiments.

The third research question concerns how data augmentations are selected during the **SimSiam** pre-training step. Results from experiments designed to answer this research question did not surpass the baseline experiments. Instead, they decreased performance compared to earlier experiments utilising a simpler augmentation selection algorithm. The inherent intensity of Augmentation Selection Algorithm B is assumed to be the most probable cause for this behaviour. This algorithm can apply multiple augmentations to the same input. With each augmentation possibly too aggressive, this could lead to spectrograms augmented beyond recognition.

The final research question revolves around how the quantity of unlabelled data impacts the encoder’s performance and, ultimately, the transcriber model. Though the amount of unlabelled data was increased by a factor of 2.5, no performance increase was observed. This result, consistent with results from all earlier experiments, highlights the encoder’s inability to extract features from its input such that the produced encoding can benefit the transcriber model.

While several contributions resulted from the work done in this thesis, the main contribution is the extensive testing and experimentation on different encoder architectures and different methods of combining these encoders with the extended Onsets and Frames architecture created by Grønbech (2021). The system created by Grønbech (2021) was reused and expanded in this thesis to include the **SimSiam** training algorithm and encoder architectures such that the transcriber model can automatically incorporate the encoder trained in pre-training. This thesis also includes a presentation of the current **SOTA** solutions within both **AMT** and Siamese representation learning, as well as an introduction to the necessary background theory utilised in later parts of the thesis. A PyTorch data loader, similar to the one presented by Grønbech (2021) for Slakh2100, was created for the **MTG-J** dataset. This enables parallelisation during the loading of unlabelled data utilised by the pre-training system, capable of speeding up loading times between individual batches by a factor of ten. Finally, this thesis also presents recommendations for future work on self-supervised learning in **AMT** and the field of **AMT** as a whole.

8.2. Future Work

A machine learning model is never better than its training data. In all fields of machine learning, the amount of training data available is crucial for a successful model regardless of the training algorithm utilised. Identified in both the preliminary report and from the results in this thesis is that there is still a severe lack of labelled multi-instrument training data for [AMT](#). Creating new and larger datasets containing such data could easily push the performance of current [SOTA](#) models further without needing modifications to the architecture. However, creating these transcriptions is a tedious task that is currently impossible to fully automate. To alleviate the amount of work required to annotate multi-instrument performances, a possible approach would be to first transcribe the music using a [SOTA](#) multi-instrument model and then fine-tune these transcriptions manually to achieve the required quality. Another possible approach for creating more multi-instrument training data is synthesising data directly from [MIDI](#). This method was used to create the [Slakh2100](#) dataset but is a less desirable solution, as current synthesis methods cannot imitate real performances accurately.

The quality of the available training data also plays an essential role in the final result. A perfectly trained model can imitate the training data flawlessly, but any faults or shortcomings present in the data will, in turn, be reflected in the final model. This can be mitigated using several methods, such as increasing the amount of training data or identifying and removing faulty data points. The latter method was utilised during the work by [Grønbech \(2021\)](#) on the [Slakh2100](#) dataset and reused in this thesis. Additionally, it was discovered from our experiments that the audio in the two datasets used during training ([MTG-J](#) and [Slakh2100](#)) did not resemble each other as much as first presumed. It would therefore be beneficial to explore how the architecture investigated in this thesis would perform when pre-trained on data with a higher level of similarity to the labelled data.

Even though the hardware utilised to run the experiments in this thesis was necessary to include the encoder in the final model, it still imposed some limitations on the encoder architecture. The size of the encoder model, its input and output size, and ultimately the batch size was severely limited by the amount of memory available on the [GPU](#) during pre-training. Compared to the recommendations from the related work on [SimSiam \(Chen and He, 2021\)](#), the batch size utilised in all experiments in this thesis was reduced by an order of magnitude. Given the necessary hardware, it would be helpful to investigate the effect of increasing the batch size during pre-training.

Even though current [SOTA](#) multi-instrument transcription models do not perform on the same level as [SOTA](#) single-instrument transcribers, they are most likely already good enough to provide some value for musicians. However, most [SOTA](#) transcription models are difficult to use if they do not provide a graphical user interface. Some models may even require extensive knowledge from the end user about programming and the system's specific architecture to produce transcriptions. Therefore, a natural next step could be to develop an application with a user-friendly interface incorporating a multi-instrument transcription model. Future [SOTA](#) transcription models would also likely share similar

8. Conclusion and Future Work

input and output formats after training compared to current models. Swapping the transcriber model in such a system with newer and better **SOTA** models should therefore be trivial compared to the work required to improve the model itself.

The **SimSiam** training algorithm did not provide as much information for the transcriber model as first anticipated. Several possible reasons for this failure were identified in the discussion. However, it would be interesting to see how other self-supervised representation learning methods handle the unlabelled training data and if they are able to pre-train an encoder more efficiently than the **SimSiam** algorithm. One closely related training algorithm to **SimSiam** is the **BYOL** training algorithm, detailed in Section 4.6.2. This method also utilises Siamese networks, but the training algorithm and its loss function are more complicated than that of **SimSiam**.

Another possibility would be to investigate how one of these self-supervised representation learning algorithms affects other **SOTA AMT** architectures such as the MT3 architecture by Gardner et al. (2022) or the NoteEM architecture by Maman and Bermano (2022) which both differ significantly from the extended Onsets and Frames architecture utilised in this thesis.

During pre-training in experiments 2 through 6, all augmentations utilised were adapted from the related work on self-supervised learning (Chen and He, 2021; Grill et al., 2020). However, these augmentations are not directly designed for use with spectrograms, and some modifications to the augmentations had to be made. It could nonetheless be quite valuable to thoroughly investigate what types of data augmentations are best suited to music in the form of spectrograms and how these would affect the final model.

Another way to augment the data would be to load two different random parts of the same song rather than augmenting one spectrogram using traditional image augmentations. As western music, especially pop music, is quite similar throughout an entire song, the two extracted segments would most likely share many features such as tempo, key and time signature.

Bibliography

- Jeremy F. Alm and James S. Walker. Time-Frequency Analysis of Musical Instruments. *SIAM Review*, 44(3), January 2002. ISSN 0036-1445, 1095-7200. doi:10.1137/S00361445003822. URL <http://epubs.siam.org/doi/10.1137/S00361445003822>.
- K.E. ArunKumar, Dinesh V. Kalaga, Ch. Mohan Sai Kumar, Masahiro Kawaji, and Timothy M. Brenza. Comparative analysis of Gated Recurrent Units (GRU), long Short-Term memory (LSTM) cells, autoregressive Integrated moving average (ARIMA), seasonal autoregressive Integrated moving average (SARIMA) for forecasting COVID-19 trends. *Alexandria Engineering Journal*, 61(10):7585–7603, October 2022. ISSN 11100168. doi:10.1016/j.aej.2022.01.011. URL <https://linkinghub.elsevier.com/retrieve/pii/S1110016822000138>.
- 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), December 2013. ISSN 0925-9902, 1573-7675. doi:10.1007/s10844-013-0258-3. URL <http://link.springer.com/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, January 2019. ISSN 1053-5888, 1558-0792. doi:10.1109/MSP.2018.2869928. URL <https://ieeexplore.ieee.org/document/8588423/>.
- Thierry Bertin-Mahieux, Daniel P. W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.
- Dmitry Bogdanov, Minz Won, Philip Tovstogan, Alastair Porter, and Xavier Serra. The MTG-Jamendo Dataset for Automatic Music Tagging. *Proceedings of the 36th International Conference on Machine Learning*, 2019. URL <http://hdl.handle.net/10230/42015>.
- Xinlei Chen and Kaiming He. Exploring Simple Siamese Representation Learning. *IEEE*, June 2021. URL <https://ieeexplore.ieee.org/document/9578004/>.
- W.T. Cochran, J.W. Cooley, D.L. Favon, H.D. Helms, R.A. Kaenel, W.W. Lang, G.C. Maling, D.E. Nelson, C.M. Rader, and P.D. Welch. What is the fast Fourier transform?

Bibliography

- Proceedings of the IEEE*, 55(10), 1967. ISSN 0018-9219. doi:10.1109/PROC.1967.5957. URL <http://ieeexplore.ieee.org/document/1447887/>.
- Valentin Emiya. MAPS Database: a Piano database for multipitch estimation and automatic transcription of music, July 2010. URL <https://adasp.telecom-paris.fr/resources/2010-07-08-maps-database/>.
- Josh Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse Engel. MT3: Multi-Task Multitrack Music Transcription, March 2022. URL <http://arxiv.org/abs/2111.03017>.
- Tom Gerou and Linda Lusk. *Essential dictionary of music notation*. Alfred Publishing Company, Van Nuys, CA, January 1996.
- Jean-Bastien Grill, Florian Strub, Florent Alché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised Learning, September 2020. URL <http://arxiv.org/abs/2006.07733>.
- Henrik Grønbech. Multi-Instrument Automatic Music Transcription with Deep Learning. *NTNU Open*, June 2021. URL <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2979217>.
- Curtis Hawthorne, Erich Elsen, Jialin Song, Adam Roberts, Ian Simon, Colin Raffel, Jesse Engel, Sageev Oore, and Douglas Eck. Onsets and Frames: Dual-Objective Piano Transcription, October 2018. URL <http://arxiv.org/abs/1710.11153>.
- Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling Factorized Piano Music Modeling and Generation with the MAESTRO Dataset, 2019. URL <https://openreview.net/pdf?id=r11YRjC9F7>.
- Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. URL <https://ieeexplore.ieee.org/abstract/document/6795963>.
- Andreas Jansson, Eric Humphrey, Nicola Montecchio, Rachel Bittner, Aparna Kumar, and Tillman Weyde. Singing Voice Separation With Deep U-NET Convolutional Networks. *Proceedings of the 18th International Society for Music Information Retrieval Conference*, 2017. URL <https://openaccess.city.ac.uk/id/eprint/19289/>.
- Qiuqiang Kong, Bochen Li, Xuchen Song, Yuan Wan, and Yuxuan Wang. High-resolution Piano Transcription with Pedals by Regressing Onset and Offset Times, July 2021. URL <http://arxiv.org/abs/2010.01815>.

- Qiuqiang Kong, Bochen Li, Jitong Chen, and Yuxuan Wang. GiantMIDI-Piano: A large-scale MIDI dataset for classical piano music, April 2022. URL <http://arxiv.org/abs/2010.07061>.
- Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 1998. URL <http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf>.
- Bochen Li, Xinzhaoh 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, February 2019. ISSN 1520-9210, 1941-0077. doi:10.1109/TMM.2018.2856090. URL <https://ieeexplore.ieee.org/document/8411155/>.
- Ben Maman and Amit H. Bermano. Unaligned Supervision For Automatic Music Transcription in The Wild, April 2022. URL <http://arxiv.org/abs/2204.13668>. arXiv:2204.13668 [cs, eess].
- 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, September 2019. URL <http://arxiv.org/abs/1909.08494>.
- MIDI Manufacturers Association. An Introduction to MIDI, 2009. URL https://www.midi.org/images/easyblog_articles/43/intromidi.pdf.
- Mats Jaer Nottveit and Håkon Anders Strømsodd. Automatic Music Transcription using Deep Learning. Specialisation project, Dept. of Computer Science, Norwegian University of Science and Technology, December 2022.
- Keiron O’Shea and Ryan Nash. An Introduction to Convolutional Neural Networks, December 2015. URL <http://arxiv.org/abs/1511.08458>. arXiv:1511.08458 [cs].
- Martin Piszczalski and Bernard A Galler. Automatic music transcription. *Computer Music Journal*, 1977. URL <https://www.jstor.org/stable/40731297>.
- Colin Raffel. Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching, 2016. URL <https://www.proquest.com/openview/4dc9b57b251e7806ff9041a948b7c320/1?pq-origsite=gscholar&cbl=18750>.
- Colin Raffel, Brian McFee, Eric J Humphrey, Justin Salamon, Oriol Nieto, Dawen Liang, and Daniel P W Ellis. A TRANSPARENT IMPLEMENTATION OF COMMON MIR METRICS. *Proceedings of the 15th International Society for Music Information Retrieval Conference*, 2014. URL <https://doi.org/10.5281/zenodo.1416528>.
- 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

Bibliography

- Learning with a Unified Text-to-Text Transformer, July 2019. URL <http://arxiv.org/abs/1910.10683>. arXiv:1910.10683 [cs, stat].
- Magnus Sjölander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure, 2019.
- Paris Smaragdis and Judith C Brown. Non-negative matrix factorization for polyphonic music transcription, 2003. URL <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1285860>.
- S S Stevens, J Volkman, and E B Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *JASA*, 1936. URL <https://asa.scitation.org/doi/pdf/10.1121/1.1915893>.
- John Thickstun, Zaid Harchaoui, and Sham Kakade. Learning Features of Music from Scratch, April 2017. URL <http://arxiv.org/abs/1611.09827>.
- Qingyang Xi, Rachel M Bittner, Johan Pauwels, Xuzhou Ye, and Juan P Bello. GUITARSET: A DATASET FOR GUITAR TRANSCRIPTION. *ISMIR*, page 8, 2018. URL https://ismir2018.ismir.net/doc/pdfs/188_Paper.pdf.

A. List of Third-Party Python Libraries

All experiments were run using Python 3.8.6 and the following packages installed. Some packages may depend on other packages not listed in this table.

Table A.1.: The complete list of third-party Python libraries and their versions used to run all experiments.

Package	Version
ffmpeg-python	0.2.0
mido	1.2.10
mir-eval	0.7
numpy	1.22.4
openunmix	1.1.2
pillow	9.4.0
pretty-midi	0.2.9
sacred	0.8.4
scipy	1.10.0
slakh-dataset	0.1.20
soundfile	0.10.3.post1
spleeter	2.3.2
tensorboard	2.11.2
torch	1.9.0+cu111
torchaudio	0.9.0
torchsummary	1.5.1
torchvision	0.10.0+cu111
tqdm	4.64.1

B. Detailed Projector Architectures

The following tables present the detailed architecture and parameters utilised in the PyTorch implementations of the three encoder architectures. The upscaler-projector in Encoder Architecture 1 is detailed in [Table B.1](#). The projector in Encoder Architecture 2 does not perform any upscaling. Its parameters are detailed in [Table B.2](#). The projector in Encoder Architecture 3 is an MLP and thus purely built on PyTorch’s linear layers. This network is detailed in [Table B.3](#).

Table B.1.: Detailed architecture and parameters for the PyTorch implementation of the upscaler-projector in Encoder Architecture 1

Layer	Input Channels	Output Channels	Kernel Size	Stride	Padding
ConvTranspose2d	512	256	(2, 3)	(2, 2)	(0, 1)
ConvTranspose2d	256	128	(2, 3)	(2, 2)	(0, 1)
ConvTranspose2d	128	64	(2, 2)	(2, 2)	(0, 0)
ConvTranspose2d	64	32	(2, 3)	(2, 2)	(0, 1)
ConvTranspose2d	32	1	(2, 3)	(2, 2)	(0, 1)

B. Detailed Projector Architectures

Table B.2.: Detailed architecture and parameters for the PyTorch implementation of the projector in Encoder Architecture 2

Layer	Input Channels	Output Channels	Kernel Size	Stride	Padding
Conv2d	1	64	(3, 3)	(2, 2)	(1, 1)
BatchNorm2d	64	64	-	-	-
ReLU	-	-	-	-	-
Conv2d	64	256	(3, 3)	(2, 2)	(1, 1)
BatchNorm2d	256	256	-	-	-
ReLU	-	-	-	-	-
Conv2d	256	512	(3, 3)	(2, 2)	(1, 1)
BatchNorm2d	512	512	-	-	-
ReLU	-	-	-	-	-
ConvTranspose2d	512	256	(2, 2)	(2, 2)	(0, 0)
BatchNorm2d	256	256	-	-	-
ReLU	-	-	-	-	-
ConvTranspose2d	256	64	(2, 2)	(2, 2)	(0, 0)
BatchNorm2d	64	64	-	-	-
ReLU	-	-	-	-	-
ConvTranspose2d	64	1	(2, 2)	(2, 2)	(0, 0)
BatchNorm2d	1	1	-	-	-
ReLU	-	-	-	-	-

Table B.3.: Detailed architecture and parameters for the PyTorch implementation of the projector in Encoder Architecture 3

Layer	Input Features	Output Features
Linear	1000	2048
BatchNorm1d	2048	2048
ReLU	-	-
Linear	2048	2048
BatchNorm1d	2048	2048
ReLU	-	-
Linear	2048	768
BatchNorm1d	768	768

C. Individual Results from Repeated Runs

This section contains tables with precision, recall, and F_1 scores from individual reruns utilised to calculate the averaged results presented in section 6. Results from reruns of experiment 1c are presented in Table C.1. Table C.2 presents results from reruns of experiment 5b. Finally, results from reruns of experiments 6a and 6b are presented in Table C.3 and Table C.4, respectively.

Table C.1.: Results from individual runs used to calculate the averaged result for experiment 1c presented in Table 6.7 in Section 6.3.

Exp. 1c Run Nr.	Frame			Note onset			Note /w offset		
	P	R	F_1	P	R	F_1	P	R	F_1
1	75.2	68.7	71.1	81.7	69.7	74.3	40.6	35.4	37.4
2	78.9	68.3	72.7	84.8	67.4	74.1	44.7	36.7	39.8
3	78.1	68.8	72.5	85.6	67.7	74.6	44.4	36.4	39.4
4	75.8	66.9	70.2	83.1	67.9	73.9	40.9	34.1	36.8
5	78.4	70.4	73.6	83.2	69.0	74.6	44.1	37.5	40.1
Average	77.3	68.6	72.0	83.7	68.3	74.3	42.9	36.0	38.7
St. dev	1.663	1.252	1.355	1.535	0.971	0.308	2.013	1.310	1.497

Table C.2.: Results from individual runs used to calculate the averaged result for experiment 5b presented in Table 6.7 in Section 6.3.

Exp. 5b Run Nr.	Frame			Note onset			Note /w offset		
	P	R	F_1	P	R	F_1	P	R	F_1
1	77.7	69.2	72.7	81.7	68.8	73.5	42.8	37.2	39.1
2	78.3	69.2	72.9	84.8	67.7	74.4	45.5	37.4	40.6
3	77.1	69.9	72.6	81.3	68.5	73.5	41.7	36.3	38.4
4	78.1	69.0	72.8	84.2	67.5	73.7	43.6	36.0	38.8
5	77.5	64.0	69.1	84.5	66.3	73.3	41.5	33.5	36.6
Average	77.7	68.3	72.0	83.3	67.8	73.7	43.0	36.1	38.7
St. dev	0.477	2.406	1.636	1.663	0.979	0.427	1.627	1.558	1.439

C. Individual Results from Repeated Runs

Table C.3.: Results from individual runs used to calculate the averaged result for experiment 6a presented in Table 6.7 in Section 6.3.

Exp. 6a	Frame			Note onset			Note /w offset		
Run Nr.	P	R	F₁	P	R	F₁	P	R	F₁
1	90.6	87.4	88.8	89.2	87.8	88.3	81.9	80.6	81.1
2	90.0	89.2	89.4	89.0	90.1	89.3	82.0	82.7	82.2
3	89.9	86.9	88.2	88.2	88.2	88.0	80.7	80.1	80.3
4	89.5	88.5	88.7	90.3	87.7	88.7	82.7	80.2	81.2
5	89.2	87.4	88.1	88.8	88.1	88.3	82.5	81.9	82.0
Average	89.8	87.9	88.6	89.1	88.4	88.5	82.0	81.1	81.4
St. dev	0.532	0.942	0.522	0.768	0.983	0.502	0.780	1.147	0.764

Table C.4.: Results from individual runs used to calculate the averaged result for experiment 6b presented in Table 6.7 in Section 6.3.

Exp. 6b	Frame			Note onset			Note /w offset		
Run Nr.	P	R	F₁	P	R	F₁	P	R	F₁
1	75.7	66.4	70.0	83.0	67.7	72.6	41.4	35.0	37.4
2	77.3	65.7	70.2	83.8	68.8	74.5	42.1	35.4	37.9
3	78.0	68.3	72.2	83.9	68.6	74.5	44.4	37.6	40.2
4	78.5	68.2	72.4	83.8	67.4	73.6	43.9	36.5	39.2
5	76.5	66.8	70.5	84.5	66.0	73.2	42.5	34.5	37.6
Average	77.2	67.1	71.1	83.8	67.7	73.7	42.9	35.8	38.5
St. dev	1.127	1.139	1.148	0.534	1.118	0.829	1.254	1.247	1.199

D. F_1 Score Plots for Remaining Experiments

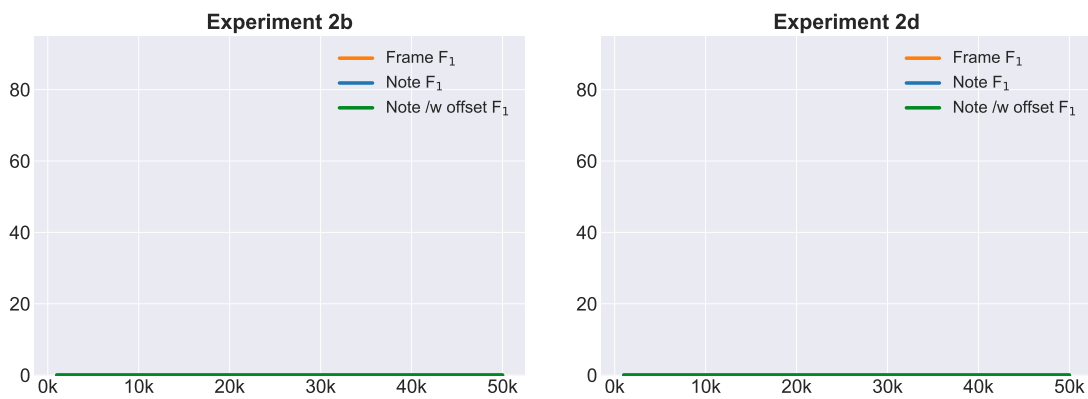


Figure D.1.: Frame, note and note with offsets F_1 scores for experiment 2b and 2d.

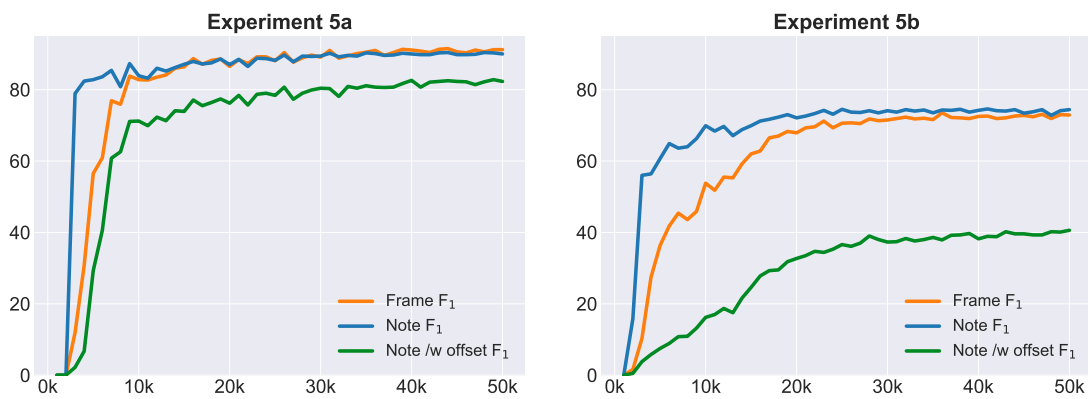


Figure D.2.: Frame, note and note with offsets F_1 scores for experiment 5a and 5b.

D. F1 Score Plots for Remaining Experiments

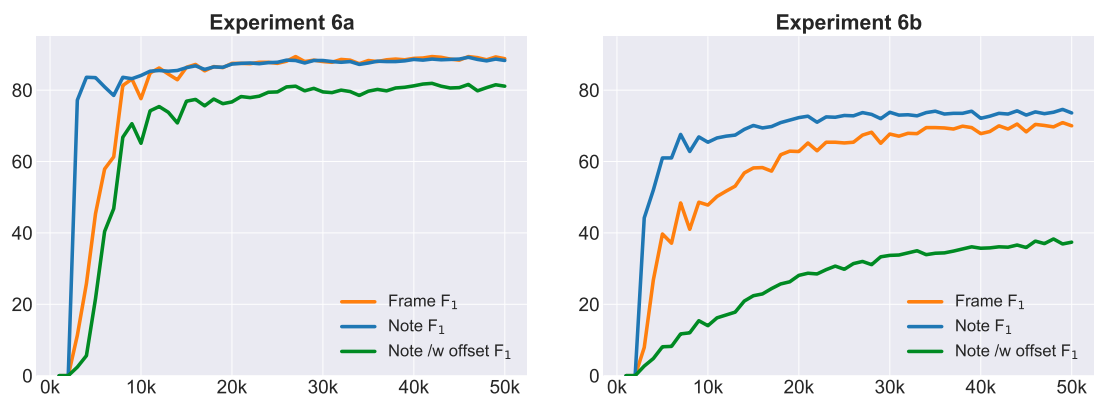


Figure D.3.: Frame, note and note with offsets F_1 scores for experiment 6a and 6b.

E. Example Transcriptions of Validation Data

The following figures show examples of transcribed music from the Slakh2100 validation split. In all figures, the top row shows the input spectrogram, the upper middle row shows the model predictions, the lower middle row shows the ground truth **MIDI** and the bottom row shows the difference between model predictions and ground truth **MIDI**. In the bottom row, grey pixels indicate correctly predicted features (frames, onsets and offsets), blue pixels indicate features missing from the prediction and red pixels indicate wrongly predicted features not present in the ground truth **MIDI**.

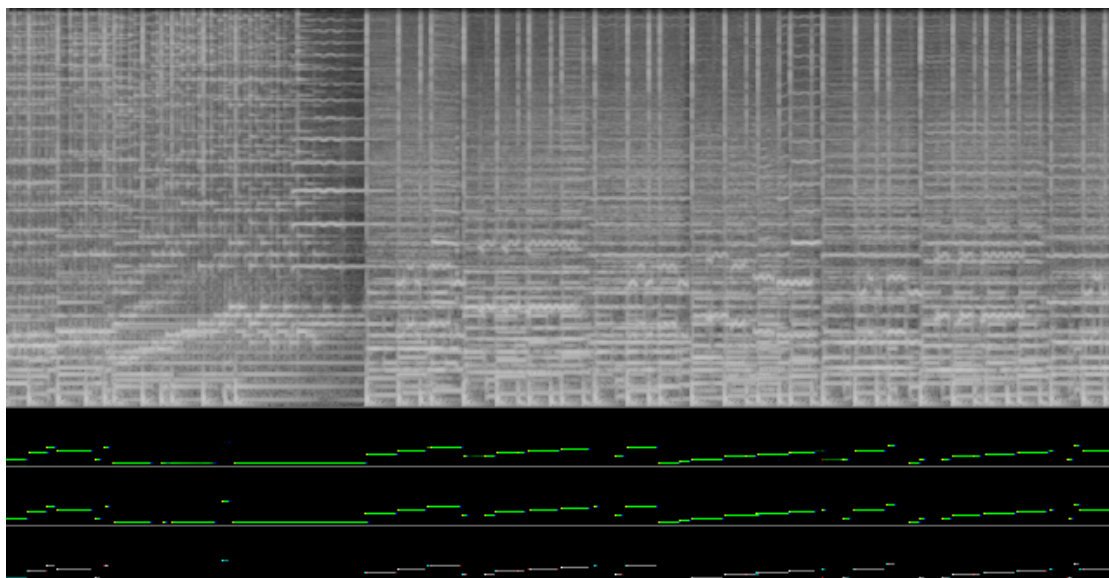


Figure E.1.: Transcribed validation spectrograms for experiment 1b.

E. Example Transcriptions of Validation Data

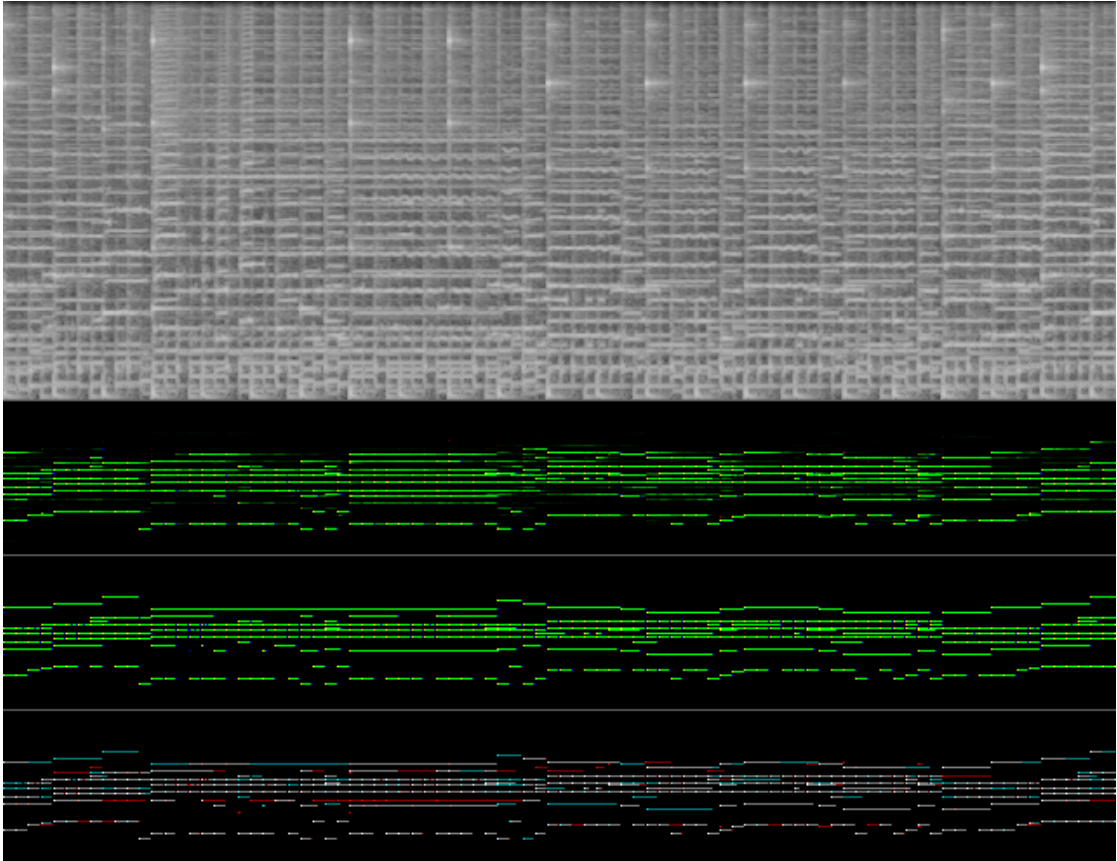


Figure E.2.: Transcribed validation spectrograms for experiment 1c.

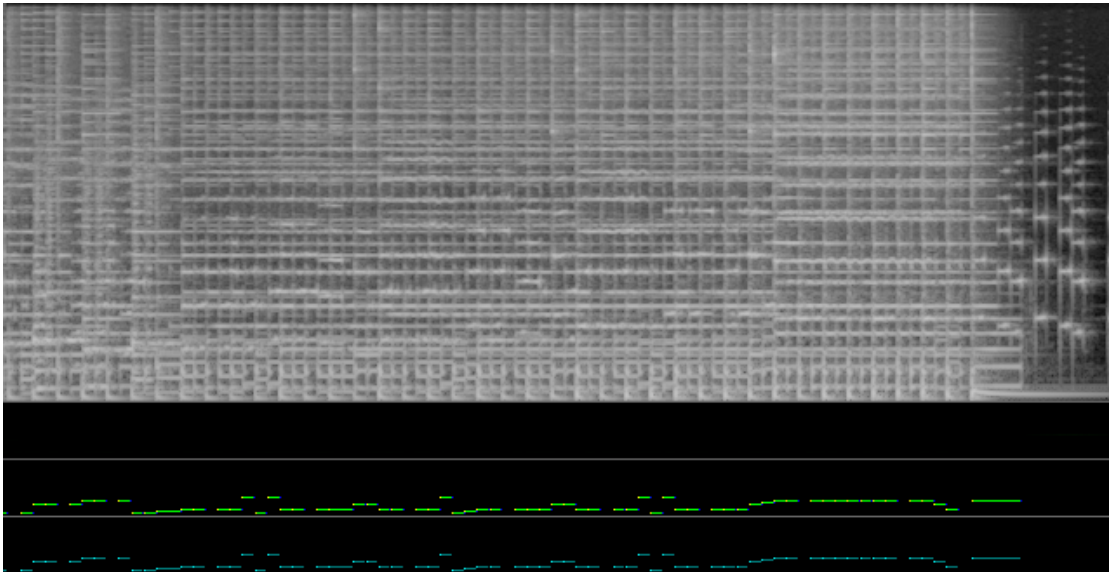


Figure E.3.: Transcribed validation spectrograms for experiment 2a.

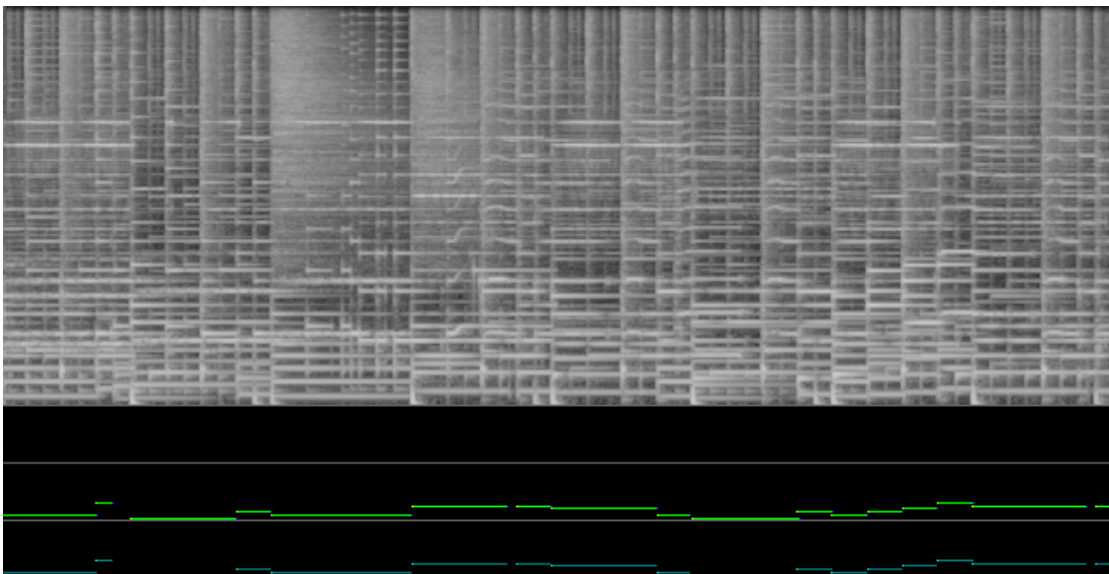


Figure E.4.: Transcribed validation spectrograms for experiment 2b.

E. Example Transcriptions of Validation Data

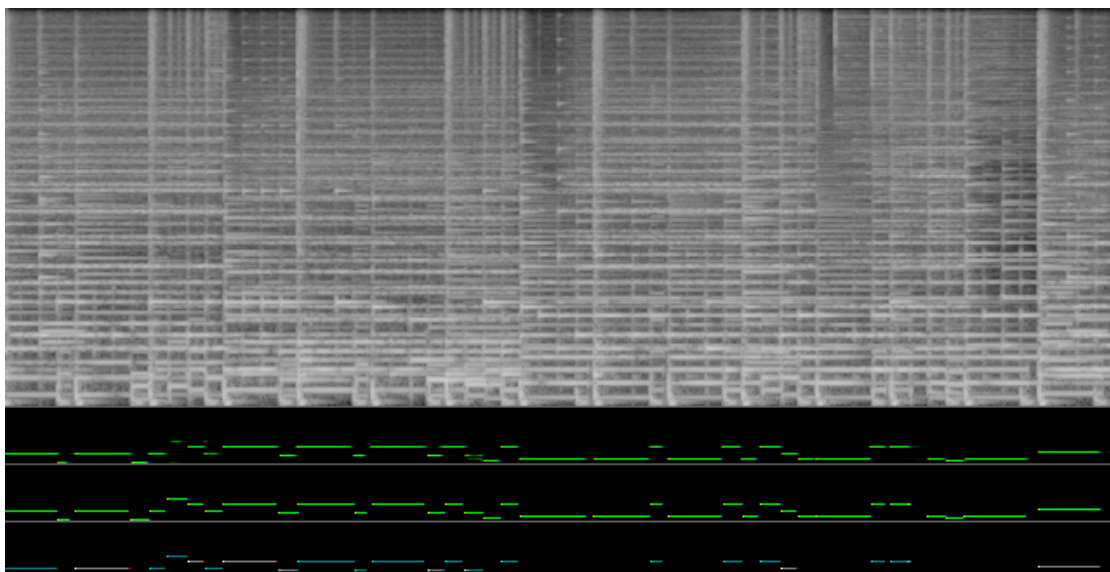


Figure E.5.: Transcribed validation spectrograms for experiment 2c.

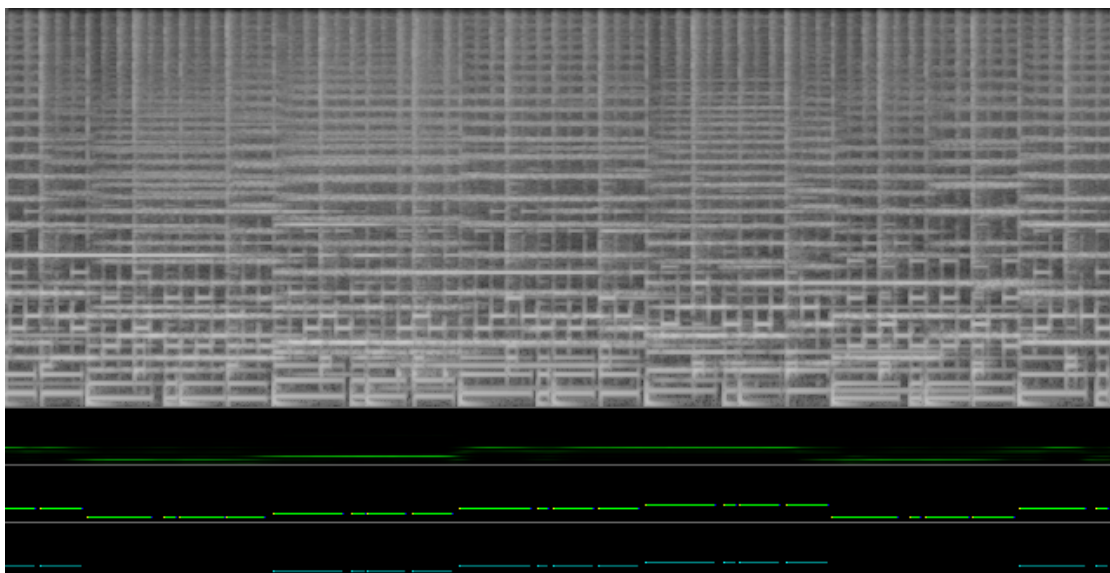


Figure E.6.: Transcribed validation spectrograms for experiment 2d.

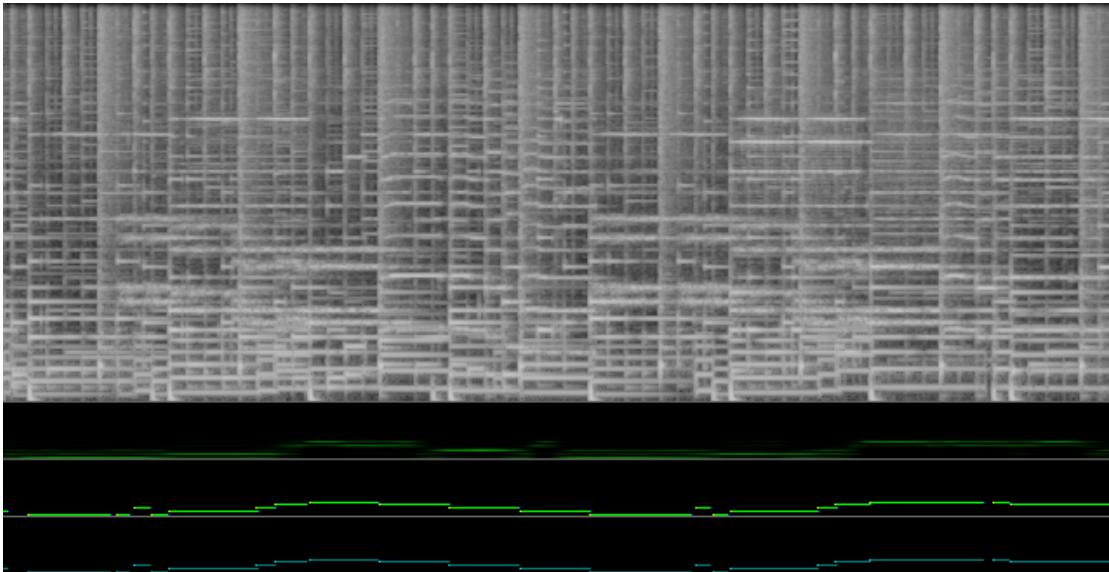


Figure E.7.: Transcribed validation spectrograms for experiment 2e.

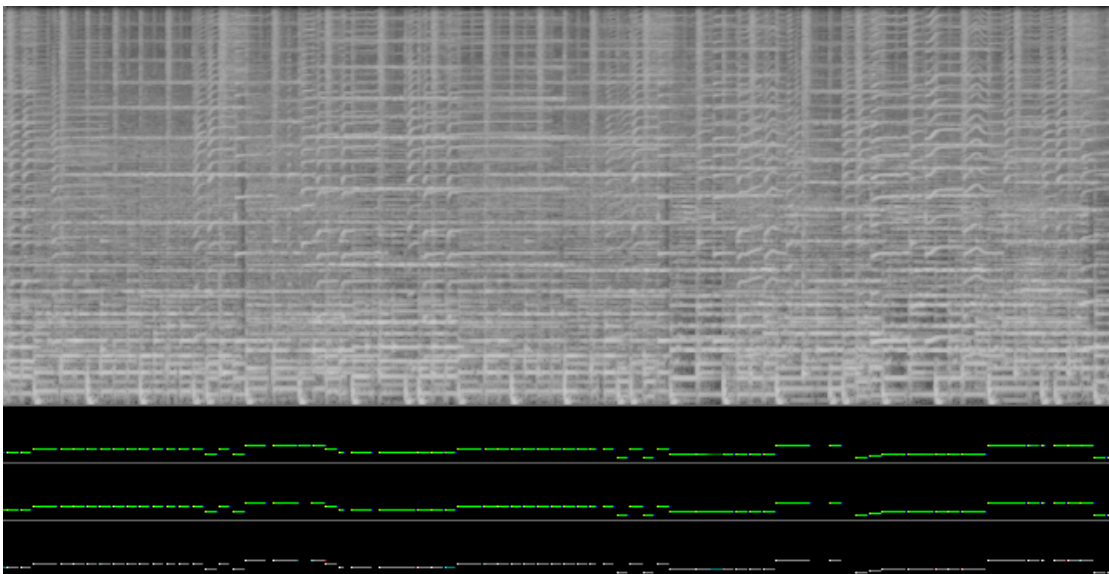


Figure E.8.: Transcribed validation spectrograms for experiment 3a.

E. Example Transcriptions of Validation Data

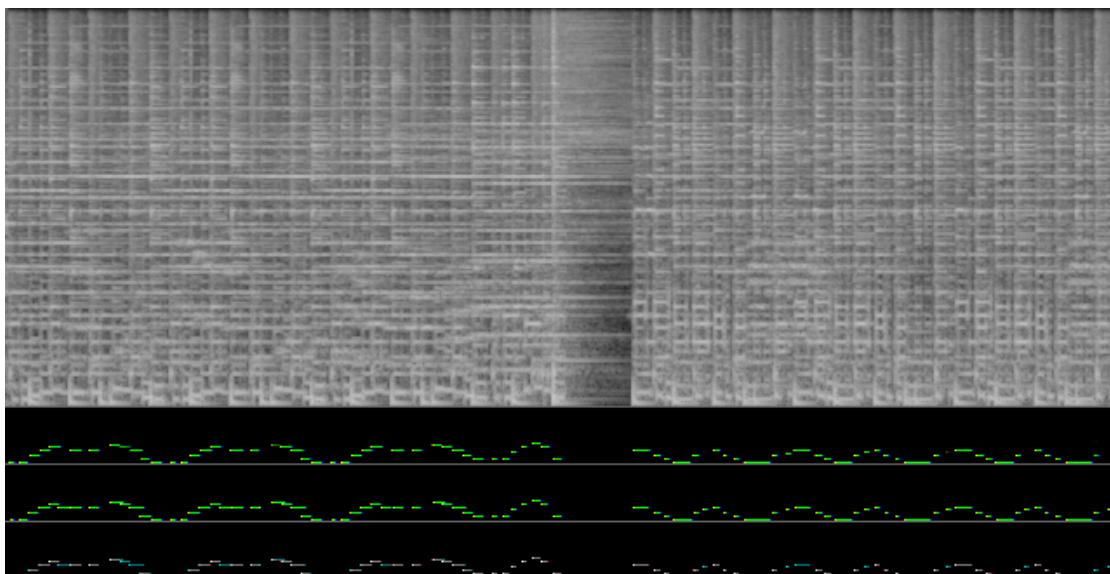


Figure E.9.: Transcribed validation spectrograms for experiment 3b.

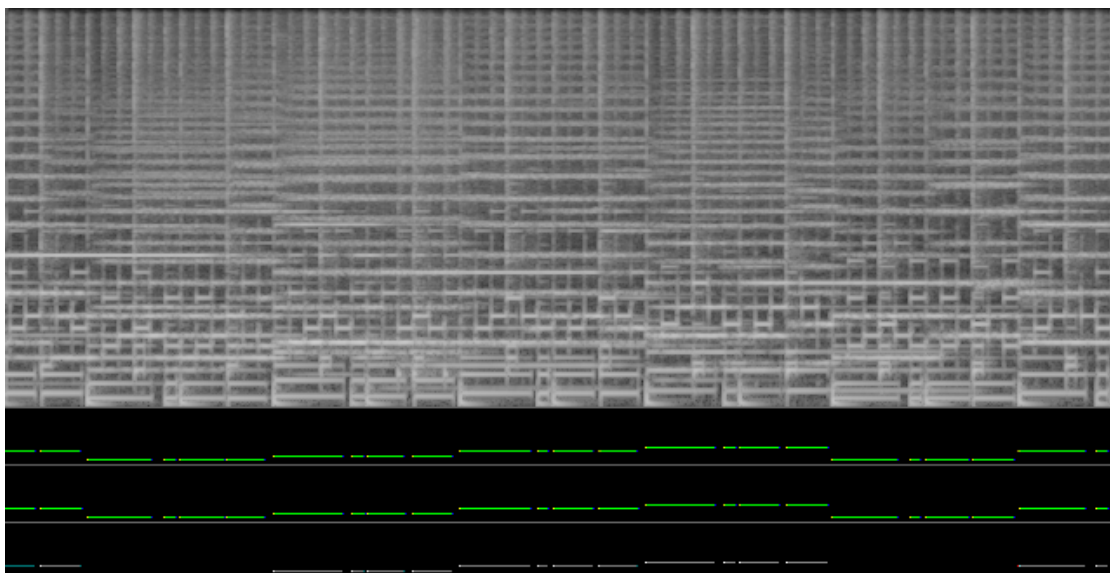


Figure E.10.: Transcribed validation spectrograms for experiment 4a.

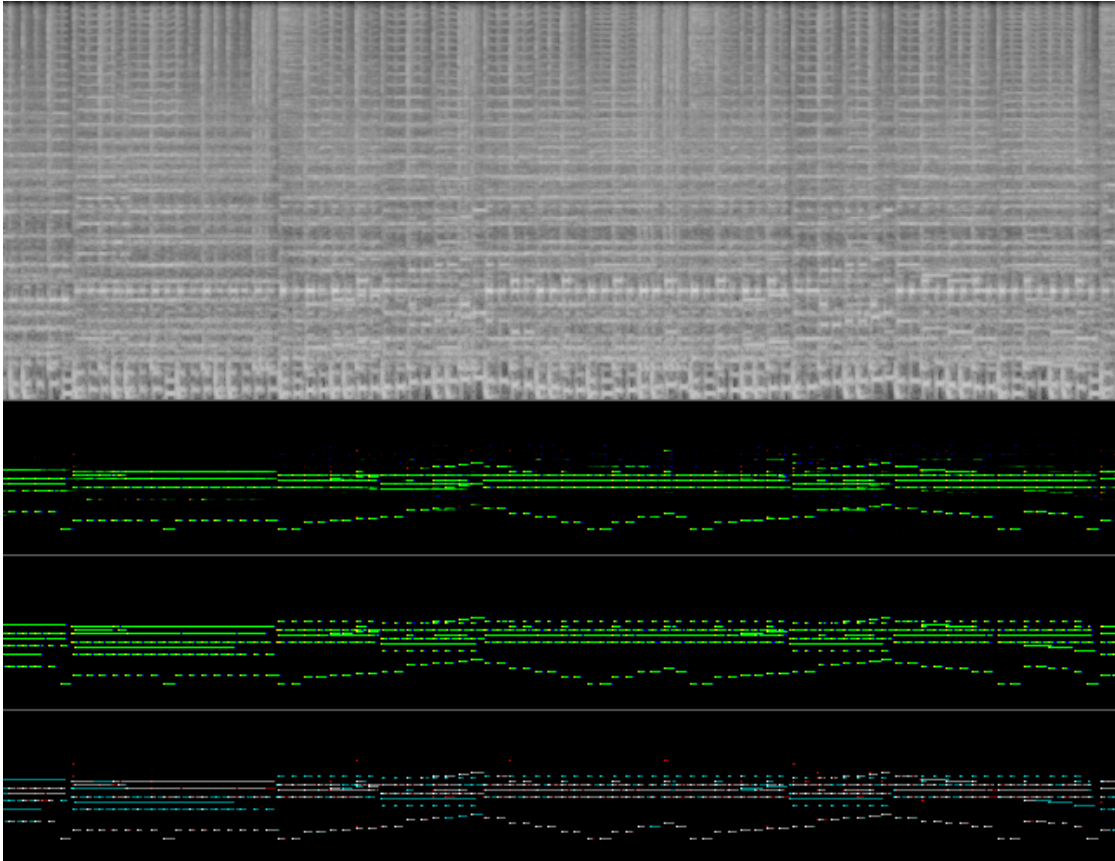


Figure E.11.: Transcribed validation spectrograms for experiment 4b.

E. Example Transcriptions of Validation Data

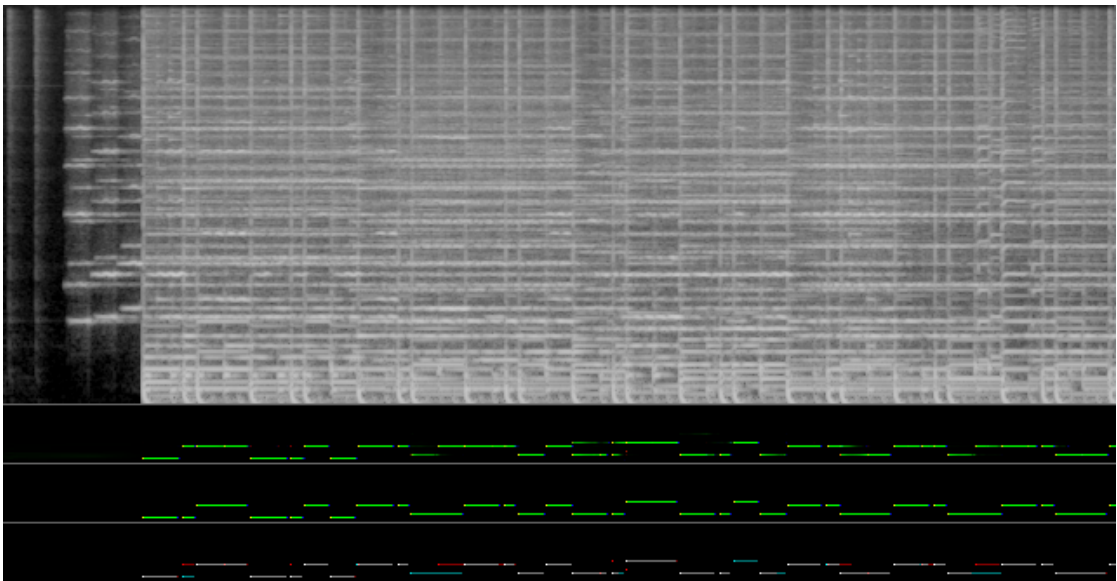


Figure E.12.: Transcribed validation spectrograms for experiment 5a.

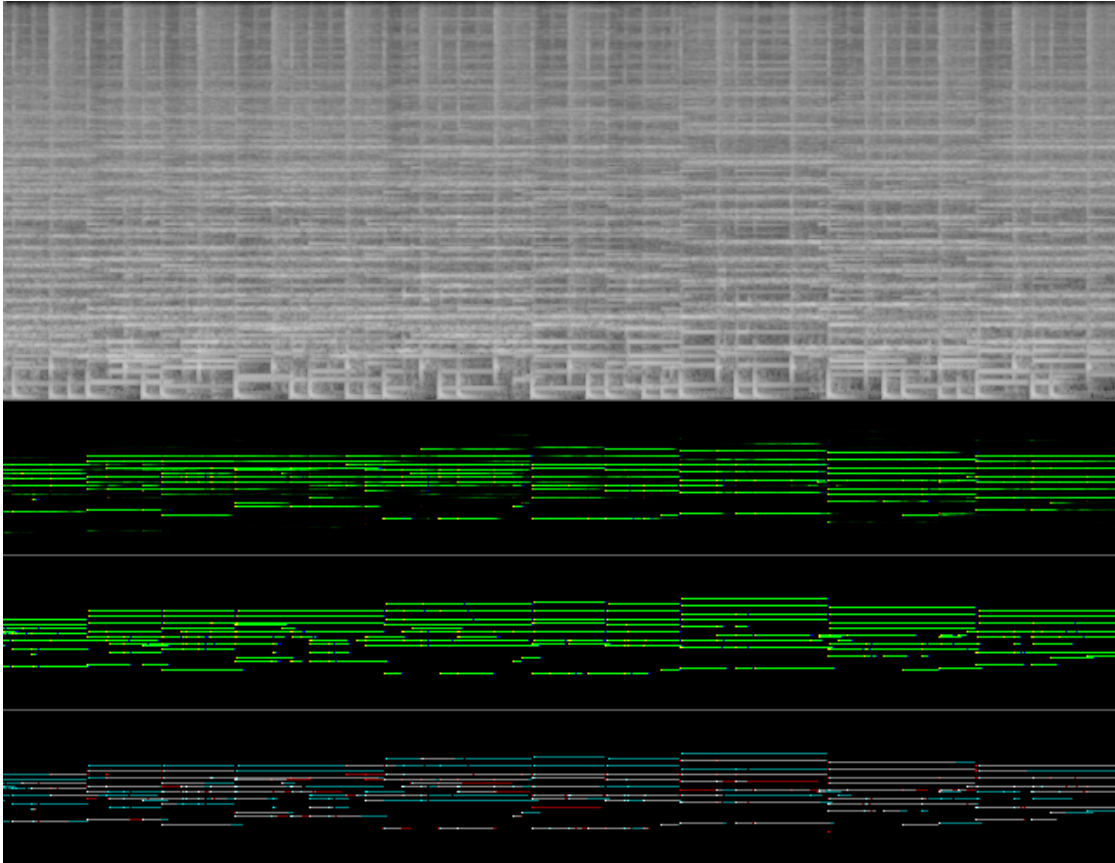


Figure E.13.: Transcribed validation spectrograms for experiment 5b.

E. Example Transcriptions of Validation Data

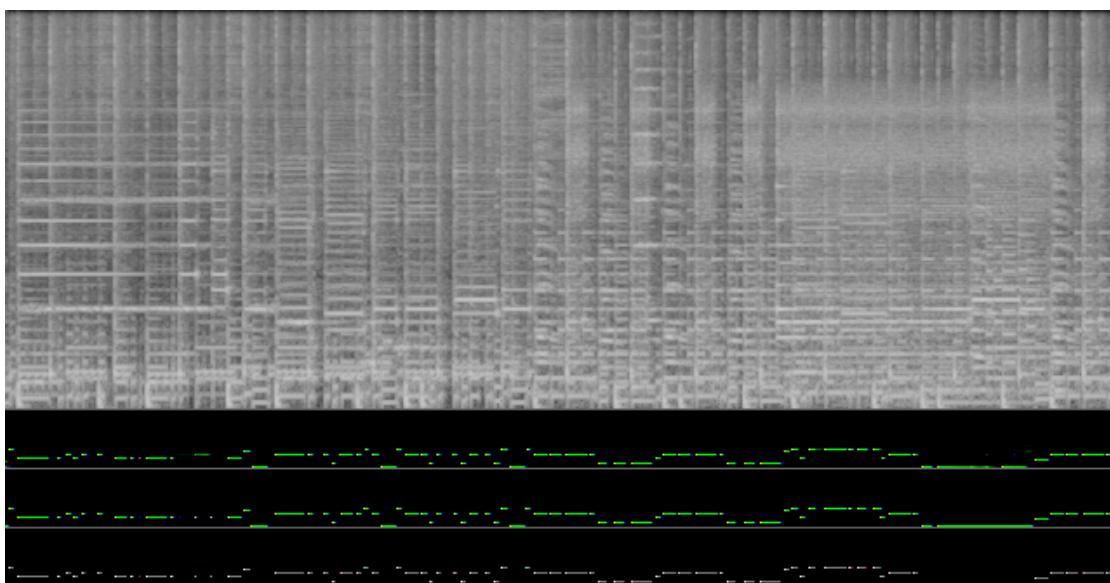


Figure E.14.: Transcribed validation spectrograms for experiment 6a.

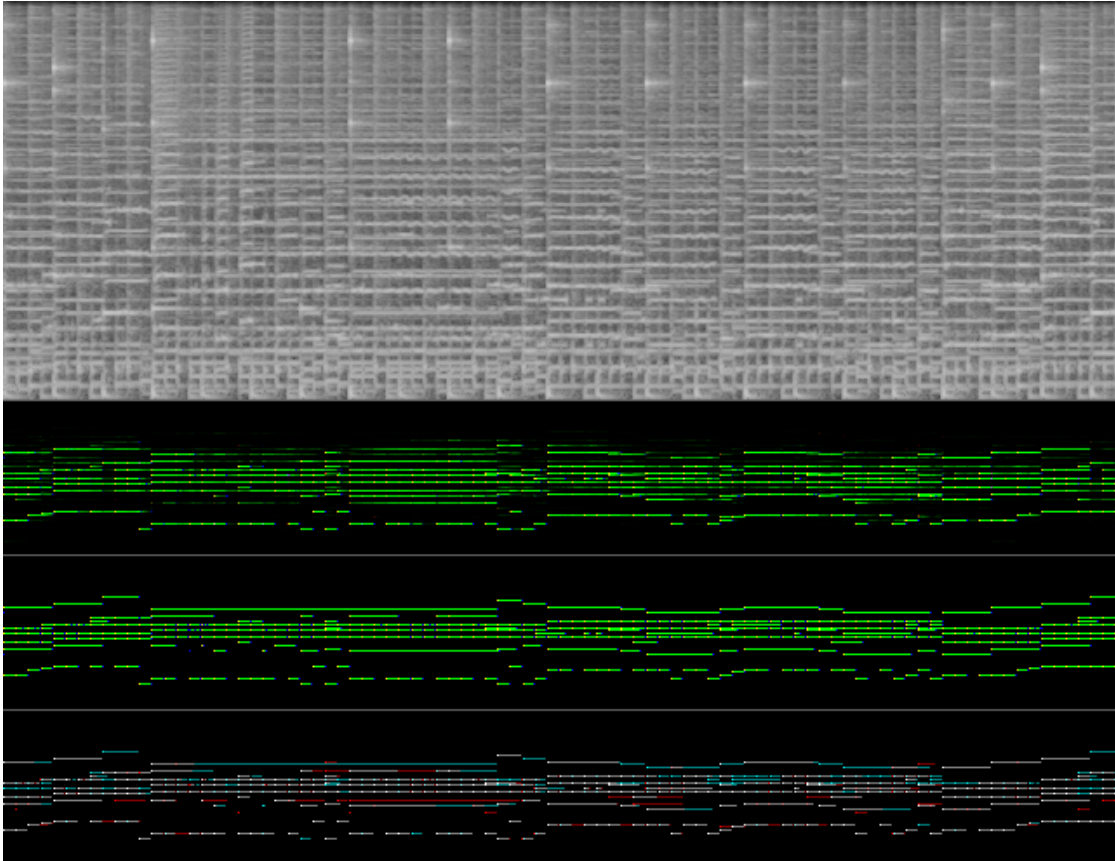
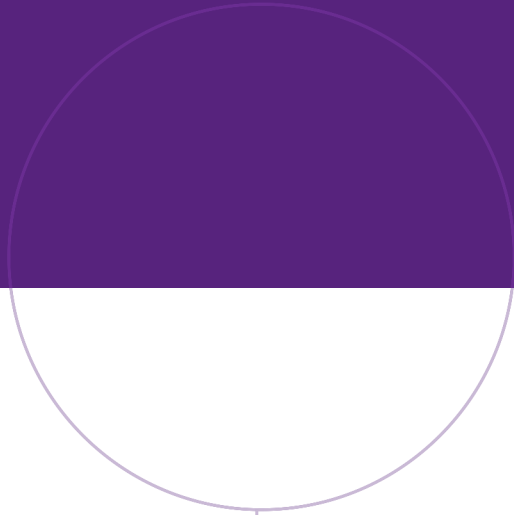


Figure E.15.: Transcribed validation spectrograms for experiment 6b.



Norwegian University of
Science and Technology