University of Bergen
Department of Informatics

Automatic Drum Transcription: Optimizing Generalization through Architectural and Dataset Choices

Author: Runar Fosse
Supervisor: Pekka Parviainen

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

August, 2025

Abstract

Automatic Drum Transcription (ADT) remains a challenging task within the field of Music Information Retrieval (MIR), especially when drum sounds are mixed with melodic instruments, otherwise called Drum Transcription in the Presence of Melodic Instruments (DTM). The current standard methodology for solving such tasks is through deep learning. This thesis investigates how different architectures and dataset composition affect ADT performance. Two studies were conducted: one comparing the performances of five different neural network architectures over four standard ADT datasets, and another evaluating how training models on different dataset combinations impacts their generalization ability both on- and Out-Of-Distribution (OOD). A novel dataset, SADTP, is introduced for this latter study is utilized for OOD evaluation. The first study concluded convolutional recurrent architectures perform the best across datasets, however strictly recurrent architectures and the Vision Transformer show promising performance on larger datasets. The second study shows that training on large and diverse datasets from multiple sources could improve both on- and Out-Of-Distribution (OOD) generalization. These findings offer insight into how architecural and dataset choices influence generalization for ADT.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Pekka Parviainen, for allowing me to choose my own master thesis subject, for always being available, both physically and digitally, as well as for his continuous support and guidance throughout the year. I also want to thank the University of Bergen, specifically the Institute of Informatics, for their provision of computational resources like Birget. Lastly, I want to thank everyone who have shown excitement and engagement into my master thesis topic, something which undeniably has helped me through this last part of my studies.

Runar Fosse
Tuesday 5th August, 2025

Contents

1 Introduction **2**

1.1 Thesis statement 2

1.2 Thesis Outline 3

2 Background **5**

2.1 Automatic Music Transcription 5

2.1.1 Transcription using Deep learning 5

2.2 Audio 6

2.2.1 Fourier Transform 7

2.2.2 Discrete Fourier Transform 9

2.2.3 Nyquist frequency 9

2.2.4 Fast Fourier Transform 10

2.2.5 Short-time Fourier Transform 11

2.2.6 Spectrogram 13

2.2.7 Loudness of Magnitude 15

2.2.8 Filters 16

2.3 Transcription 18

2.3.1 Music Notation 18
2.3.2 MIDI Annotations 19
2.3.3 Activation Functions 20
2.4 Automatic Drum Transcription 22
2.4.1 The Drum Set 23
2.4.2 Transcription Task 24
2.5 Performance Measure 26
2.5.1 Correct Predictions 26
2.5.2 Accuracy 26
2.5.3 F1-score 26
2.5.4 Micro vs. Macro 27

3 Architectures **29**

3.1 Recurrent Neural Network 30
3.1.1 Implementation 31
3.2 Convolutional Neural Network 32
3.2.1 Implementation 32
3.3 Convolutional RNN 34
3.3.1 Implementation 34
3.4 Convolutional Transformer 35
3.4.1 Implementation 36
3.5 Vision Transformer 39
3.5.1 Patch Embedding 39
3.5.2 Architecture Modifications 40
3.5.3 Implementation 40

4 Datasets **43**

4.1 ENST+MDB 43

4.1.1 Splits 44

4.1.2 Mapping 44

4.2 E-GMD 45

4.2.1 Mapping 46

4.3 Slakh 47

4.3.1 Mapping 47

4.4 ADTOF-YT 48

4.4.1 Mapping 48

4.5 SADTP 49

4.5.1 Mapping 49

4.6 Summary 50

5 Methodology **51**

5.1 Data Preparation 51

5.1.1 Audio Files 51

5.1.2 Annotations 52

5.1.3 Splitting and Storing 53

5.2 Preprocessing 53

5.3 Training 54

5.4 Postprocessing 55

5.5 Model Selection 56

5.6 Hyperparameter Tuning 57

5.6.1 Search Strategies 57

5.6.2 Hyperparameters 57

6 Architecture Study **59**

6.1 Methodology 59

6.2 Results 60

6.3 Discussion 61

7 Dataset Study **64**

7.1 Methodology 64

7.2 Results 65

7.3 Discussion 66

8 Conclusion **69**

Bibliography **71**

A ENST+MDB Splits **78**

Chapter 1

Introduction

Within the field of Music Information Retrieval (MIR), the task of Automatic Music Transcription (AMT) is considered to be a challenging research problem. It describes the process of generating symbolic music notation from audio. The majority of instruments are melodic, where key information for transcription would be to discern pitch, onset time, and note duration. This stands in contrast to percussive instruments, where instead of pitch and duration one would focus on instrument classification and onset detection. This sets the stage for Automatic Drum Transcription (ADT), a subfield of AMT specifically focusing on transcribing drums and percussive instruments [58]. Specifically, this thesis will focous on Drum Transcription in the Presence of Melodic Instruments (DTM), the hardest subtask of ADT.

Previously, a popular approach to ADT was using signal processing, later developing into using classical machine learning methods [58]. In later years however, the standard has evolved into utilizing deep learning, showing to be quite effective. Therefore, the recent focus of most authors has been to find the best performing deep learning approaches by either; constructing and analysing different deep learning model architectures, striving to find a best performer, or by creating datasets which, when trained on, allow models to heighten their generalization ability [61].

1.1 Thesis statement

This leads us to my two primary questions.

1. Which deep learning architecture is the best suited for solving an ADT task like DTM?
2. Which factors make an ADT dataset optimal through making models generalize for DTM?

I will tackle both of these questions through two different studies performed in this thesis.

For the former, we will train several different model architectures on different, well-known ADT datasets. These different architectures are specifically, the recurrent neural network, the convolutional neural network, the convolutional-recurrent neural network, the convolutional transformers and, novel to the field of ADT, the Vision Transformer. By comparing their performances we will be able to gauge which of these are the best suited for the ADT task, specifically DTM.

For the latter we will select the best performing model architecture from the first question and train it over several different combinations of the ADT datasets. For this I also introduce SADTP, a novel ADT dataset solely used for Out-Of-Distribution (OOD) evaluation purposes. By performing cross-dataset evaluations we could analyse and figure out what makes a good ADT, dataset and how it would enhance a suitable, well-selected model architecture. I will also compare my best performing models with models from those from other literature, giving a comparative analysis which will strengthen and increase the robustness of my final conclusion.

1.2 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2: Background — Covers information needed to fully understand the ADT training and inference pipeline. Specifically it will give a thorough understanding into AMT and ADT, how audio is transformed and represented for deep learning, what the outputs of an ADT pipeline look like, and which performance measures that best describe a well-performing ADT model.

Chapter 3: Architectures — Presents and goes in-depth into each of the different deep learning architectures which are utilized in this thesis, specifically for the first study.

Chapter 4: Datasets — Presents each of the different datasets trained on within this thesis, as well as comparing their characteristics. In addition, I introduce the novel, DTM, test-only dataset, SADTP.

Chapter 5: Methodology — Covers the specific methodology used in this thesis by explaining how each dataset is prepared, what pre- and postprocessing are techniques are used, and how the hyperparameter tuning, model selection and training procedure is implemented.

Chapter 6: Architecture Study — Trains and compares the performances different deep learning architectures trained over each dataset, discusses the different results and concludes by selecting the best performing one.

Chapter 7: Dataset Study — Trains and compares the best performing model from the previous study trained over several different combinations of multiple datasets, and discusses evaluation results through both on- and OOD evaluation.

Chapter 8: Conclusion — Concludes this thesis by reflecting on the results of the different studies, as well as giving an outlook into what should be covered in future work.

Chapter 2

Background

2.1 Automatic Music Transcription

In general, transcription refers to the process of retrieving information from audible data, such as sounds or music. Specifically when applied on music, the information we seek is often an annotation in music notation. This is what is referred to as *music transcription*. Generally, transcribing a musical piece is expensive, requiring both extensive experience within a specific instrumental field, as well as a lot of manual, time-consuming work. Early on in 1986, Martin Piszczalski stated that *"The learned, human skill of transcribing music is one of the most sophisticated auditory-based pattern-recognition tasks that humans perform."* [38]. A decade prior he helped introduce the term Automatic Music Transcription (AMT) covering a field in which now, many years later, has evolved significantly [39].

2.1.1 Transcription using Deep learning

Currently, the majority of state-of-the-art within AMT utilizes deep learning [58, 61, 30]. With enough data one can train a Deep Neural Network (DNN) to, given an input sequence of music, automatically output a transcribed sequence representing musical notation. Musical notation such as this could be extensive in their information, but by far the most important parts could be reduced down to *"which instruments are playing"* and *"when are they playing"*.

Relating this to a common deep learning task, this could be described as a multi-label sequence tagging/sequence labelling task. For each element in an input sequence output a combination of one or more labels. Each element within this sequence represents a timestep in the audio recording, with a combination of labels representing one or more instruments being played. This constitutes the most basic form of AMT.

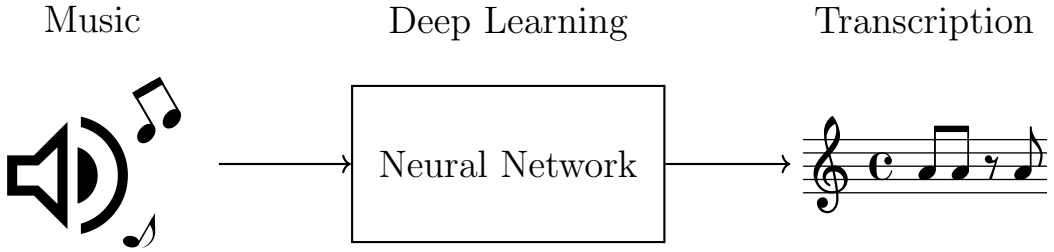

Figure 2.1: An abstract visualization of the Automatic Music Transcription (AMT) process using deep learning. Music is input, processed by a neural network, and ultimately converted into a symbolic transcription.

2.2 Audio

Sound is described by *"the sensation caused in the nervous system by vibration of the delicate membranes of the ear."* [1]. In short, sound is the human perception of acoustic waves in a transition medium, like air. These waves, consisting of vibrating molecules, get sensed by our auditory organs and perceived by the brain.

Thus sound can be described as the propogation and perception of waves. Mathematically, waves can be studied as signals [12]. To represent these sounds digitally, as *audio*, one can express these waves as a signal, giving rise to the *waveform*. The waveform is a timewise representation of a signal as a graph, charting the amplitude, or strength of the signal, over time.

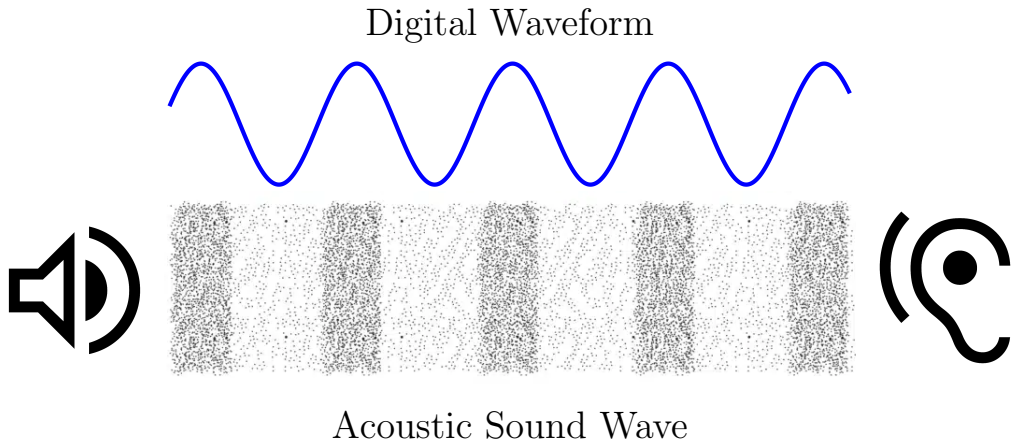

Figure 2.2: The relationship between an acoustic sound wave and its corresponding digital waveform. The waveform represents variations in the air pressure over time. Regions of higher and lower pressure in the sound wave correspond to *peaks* and *troughs* in the waveform, respectively.

For monophonic sound, this waveform is a one-dimensional representation. Even though this is an excellent way of storing audio digitally, informationally it is very dense. Although there do exist deep learning models working directly with these waveforms, such as Oord et al.'s WaveNet [52], the task of parsing and perceiving such a signal is a complex one. Luckily, there exists some processing techniques which can be applied to a waveform, lowering its complexity whilst keeping information intact.

2.2.1 Fourier Transform

The Fourier Transform is a mathematical transformation which, given a signal and frequency, computes the frequency's significance, or intensity, in the original signal. As we've established, audio is represented as a signal, meaning we therefore can use this transform, turning an audio signal into frequency space.

The fourier transform is a complex transformation. Specifically given a signal f, we can compute the integral

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

for a frequency $\xi \in \mathbb{R}$, computed over all values $x \in \mathbb{R}$ representing time, resulting in a *complex* number on the form $\widehat{f}(\xi) = a + bi$. This number consists of a *real* part a and an *imaginary* part b. We can represent this complex number in polar form

$$a + bi = re^{i\theta},$$

where $r = \sqrt{a^2 + b^2}$ denotes the magnitude (amplitude) and $\theta = \arctan\left(\frac{b}{a}\right)$, adjusted for the right quadrant, computes the phase of the respective frequency in the original signal f. Frequencies with higher amplitudes play a more prominent role in shaping the original sequence, indicating a greater significance. This information is what allow us to figure out which frequencies a signal is made out of and how each frequency contributes.

By doing such a transform we turn our temporal data, like a waveform, into spectral data, its *spectrum*. This inuitively *untangles* our signal into its respective base frequencies. By ridding our data with the highly dense temporal aspect, replacing it with a more informationally independent and sparse spectral representation, a transformation such as this could lessen the complexity of the task, making the audio easier to *understand*; in a metaphorical sense.

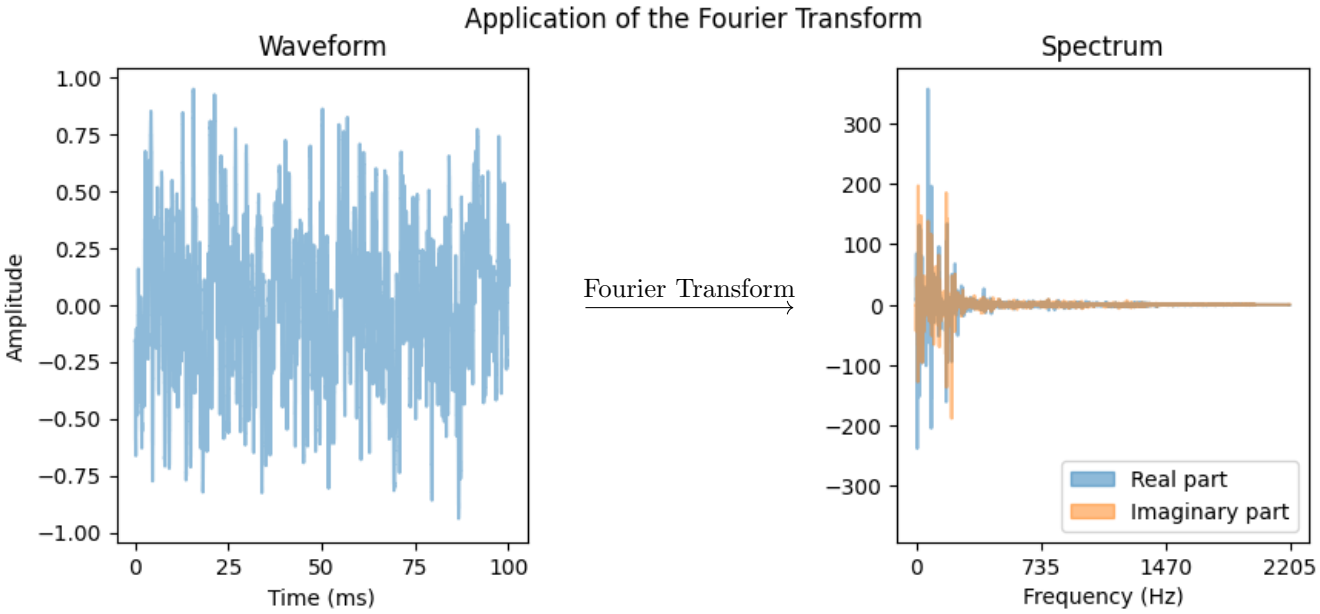

Figure 2.3: The Fourier Transform decomposes a waveform into its frequency components. The time-domain waveform signal on the left is measured in amplitude, where higher amplitudes correspond to higher sound wave air pressure levels. The frequency-domain spectrum on the right displays the amplitude of each frequency, indicating its contribution to the original signal.

Note that the Fourier Transform is invertible, meaning that the original signal can be reconstructed from its frequency components via an inverse transformation. This property is fundamental to the Fourier Transform's integral and widely exploited within signal processing.

2.2.2 Discrete Fourier Transform

The Fourier Transform is defined as an integral over continuous time. On computers, instead of storing signals continuously we instead store signals using a discrete number of samples. Each signal's *sampling rate* describes how many samples a signal contains per second of audio, and is denoted in *Hz*.

To extract frequency values from these signals, we instead have to use the Discrete Fourier Tranform (DFT). Intuitively this works identically to the normal Fourier Transform, just ported to work on discretely sampled signals. It is given by the formula

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi \frac{k}{N} n},$$

where k denotes a certain frequency bin and N the number of discrete samples. Note that contrary to the normal Fourier transformer, instead of computing the magnitude of each specific real-valued frequency, we instead compute the magnitude of frequencies covered by certain frequency bins. The number, or granularity, of these bins are directly dependent on the number of samples N, and the respective frequencies covered by each bin is given by the signal's sampling rate.

2.2.3 Nyquist frequency

When a continuous signal, such as an audio wave traveling through the air, is discritized by recording it on a computer, some information may be lost in the process. The discrete representation of the signal is an *approximation* which quality is directly dependent on the sampling rate. The higher the sampling rate, the *closer* we are to the original, continuous signal. However a higher sampling rate comes at the cost of needing to store these signals at a higher precision. A lower sampling rate would need less information stored, but this could also mean a less precise signal approxmation.

Aliasing is the phenomenon where new, incorrect frequencies emerge in undersampled signals. The *Nyquist frequency*, defined as half the sampling rate, represents the highest frequency that can be accurately captured by a discrete signal. Thus to prevent aliasing, the sampling rate must be at least twice the maximum frequency present in the original signal.

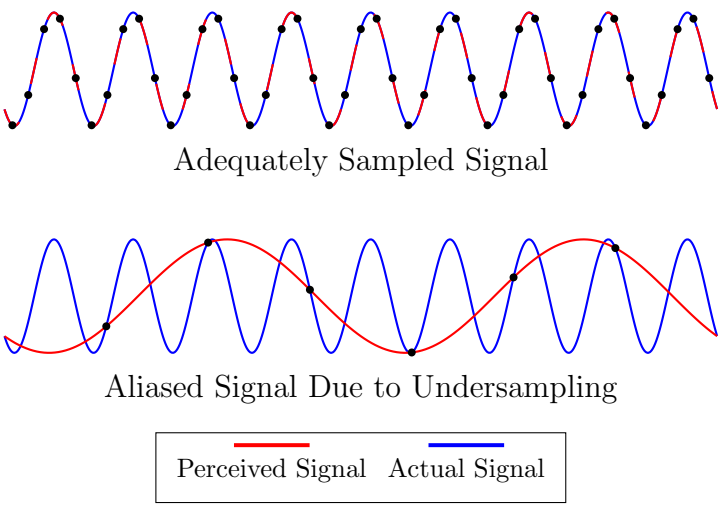

Figure 2.4: Example of aliasing due to undersampling. The original signal has a true frequency of 9 Hz. In the upper plot, sampling at a sufficiently high rate of 45 Hz allows the signal to be perceived and reconstructed accurately. In the lower plot, however, a lower sampling rate of 7 Hz causes aliasing, and the signal is incorrectly perceived as having a frequency of 2 Hz.

In the case of the DFT, it directly follows that the maximum frequency from which accurate information can be extracted is proportional to the signal's sampling rate. Specifically, it is equal to the signal's Nyquist frequency.

2.2.4 Fast Fourier Transform

Keen-eyed computer scientists may note that the DFT has a time complexity of $\mathcal{O}(n^2)$, since it requires summing over N values for each of the N different frequencies. As a result, the DFT scales poorly in regards to input size. For instance, given that the standard audio sampling rate is 44.1kHz, the computational cost becomes significant, making the DFT impractical for real-time or large-scale signal analysis [4].

The Fast Fourier Transform (FFT) algorithm addresses this limitation by computing the same result as the DFT, but with a time complexity of $\mathcal{O}(n \log n)$ instead. Described by Gilbert Strang as *"the most important numerical algorithm of our lifetime"* [51], the FFT effectively solves the prior scaling problem, allowing us to efficiently extract spectral information from a signal even at high sampling rates.

There exist several different implementations of the FFT. Among them, the Cooley–Tukey algorithm is by far the most widely used, and optimizes its calculations through a *divide and conquer* approach, reducing redundant computation by reusing intermediate results [15].

2.2.5 Short-time Fourier Transform

The Fourier Transform comes with certain limitations; most notably in how, by transforming a signal into the frequency domain, we lose its temporal structure. Although this loss of time information may be acceptable for certain tasks, it poses a challenge for applications like music transcription and ADT where timing of events is vital. So far, we've seen how the Fourier Transform unravels the frequency content of a signal as a whole. But what happens if we had applied it to smaller, localized partitions of the signal instead?

This leads us to the Short-time Fourier Transform (STFT). Rather than transforming the entire signal at once, the STFT applies the Fourier Transform to smaller, overlapping *windows* of the signal. This allows us to extract frequency information while preserving much of its temporal resolution. The result is a two-dimensional representation, revealing the intensity of different frequencies over time.

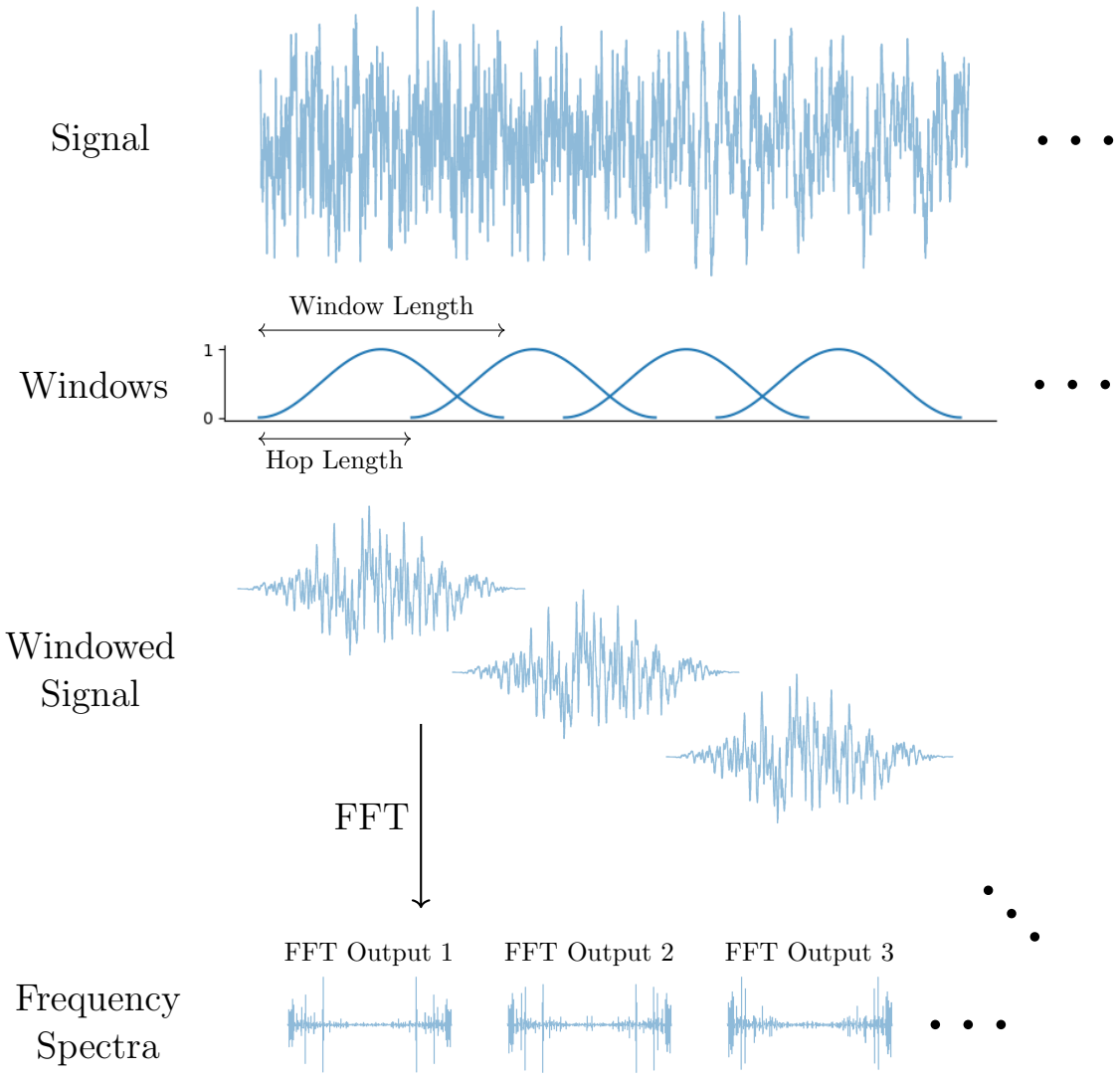

Figure 2.5: Application of the STFT to a signal. The original waveform is multiplied by multiple offset window functions, producing a set of overlapping signal partitions. The FFT is then applied to each partition individually, resulting in a sequence of frequency spectra that preserve localized time–frequency information.

The STFT has several parameters that affect the structure and resolution of its output; most notably the *window function*, *window length*, and *hop length*. Due to a phenomenon called *spectral leakage*, where spectral information bleeds into other frequencies, a windowing function is applied to each partitioned segment. A common choice is the *Hann window*, illustrated in Figure 2.6.

The window length plays a crucial role in determining the time–frequency resolution. A longer window improves the frequency resolution, allowing for more precise magnitude estimation of each frequency component, but comes at the cost of reduced time resolution.

This is because a large window spans a longer segment of the signal, causing individual windows to overlap more and blur timing information. On the contrary, shorter windows improve temporal precision but blur frequency content, due to each window consequently consisting of fewer samples. This duality is known as the *time–frequency tradeoff*.

The hop length determines how far the window shifts between each segment and directly affects the temporal granularity of the output. A smaller hop length results in more overlap and a finer temporal resolution, but also increases the computational cost by requiring more FFTs. Additionally, the hop length defines the time step between frames, allowing each window to be assigned a temporal index.

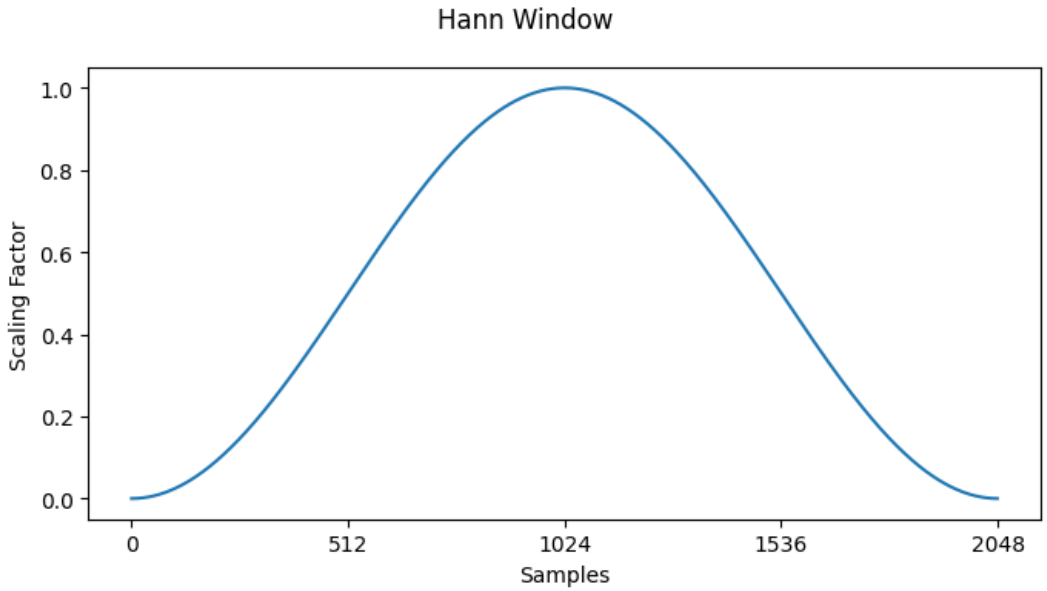

Figure 2.6: The Hann window function, commonly applied to each partition during the STFT to reduce the effects of spectral leakage. It does so by scaling the amplitude of the signal down toward zero at the window's edges. This example shows a window length of 2048 samples.

For ADT, a common configuration is to use the Hann window, with a window size of 2048 samples and a hop length corresponding to 10 ms, i.e., the sampling rate divided by 100 [58, 54, 56, 61].

2.2.6 Spectrogram

The STFT, like the standard Fourier Transform, produces complex-valued output. To convert this into strictly real-valued data without discarding meaningful information, we

compute the *spectrogram*. Specifically, a magnitude spectrogram is obtained by taking the absolute value of each complex result from the STFT.

The result is a two-dimensional, real-valued representation of the signal, where one axis corresponds to time and the other to frequency. While this representation can be modeled as a time series, it is also structurally equivalent to an image. As such, spectrograms are often visualized as heatmaps, offering an intuitive way to observe how frequency information evolves over time.

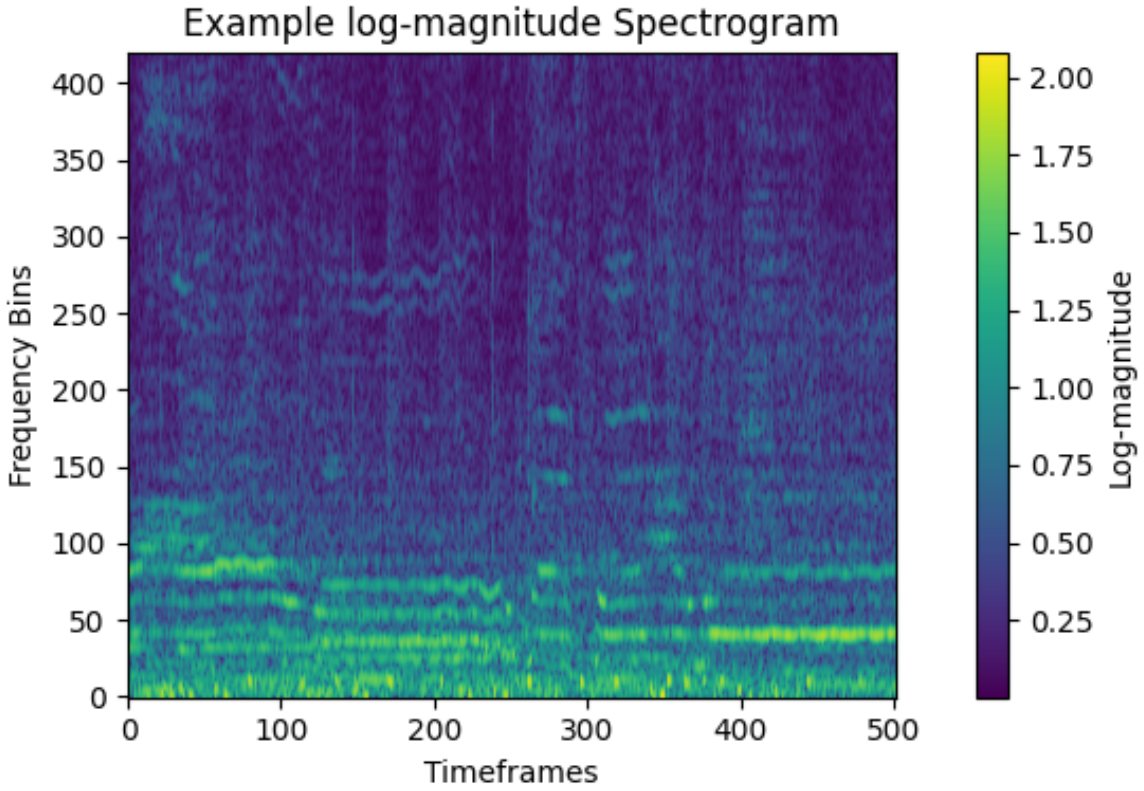

Figure 2.7: Log-magnitude spectrogram of the SADTP track *"Red Swan"*, showing the segment from 3:35 to 3:40 minutes. The spectrogram is computed using 2048 FFTs, with a window length of 512 samples and a hop length corresponding to 10 ms. Only the first 420 frequency bins are visualized. The color represents the log-magnitude of each time–frequency bin.

The number of frequency bins (represented on the y-axis of Figure 2.7) in a spectrogram is equal to half the number of FFTs used in its computation. These bins contain information about linearly spaced frequencies, ranging from 0 Hz up to the Nyquist frequency of the signal. The number of time frames (the x-axis) is determined by the hop length of the STFT, with smaller hop lengths resulting in a greater number of frames. In

this way, one has significant control over the dimensionality of the spectrogram through parameter selection.

One drawback of the spectrogram is that it discards all information about signal's phase. As previously mentioned, phase is encoded in the angle of each complex value, which is lost when computing the magnitude. This means it is not possible to perfectly reconstruct the original signal from a magnitude-only spectrogram. However, approximate reconstruction is possible using iterative algorithms such as Griffin–Lim [24].

2.2.7 Loudness of Magnitude

The human perception of loudness is approximately logarithmic. A soundwave that carries ten times more energy is not perceived as ten times louder. Instead, a doubling in perceived loudness corresponds roughly to a 10 Decibel (dB) increase in signal amplitude. The Decibel is a relative unit of measurement that expresses the power the ratio between two signals, where an increase of 1 dB corresponds to a power ratio of $10^{\frac{1}{10}}$. The power of a signal is computed as the square of its magnitude.

This relationship has two major consequences. First, the magnitude scale used in standard spectrograms does not align with human perception of loudness; quiet and loud signals may differ significantly in magnitude, even if they perceptually seem close. Second, this mismatch causes the standard (linear) spectrogram to be highly variable in its magnitude representation, often making it difficult to identify quieter frequency components among louder ones.

To address this, a log-magnitude spectrogram is often used (as in Figure 2.7), which applies a logarithmic transformation to the magnitudes. This logarithm is typically base 10, aligning with the Decibel scale and better reflecting human auditory perception.

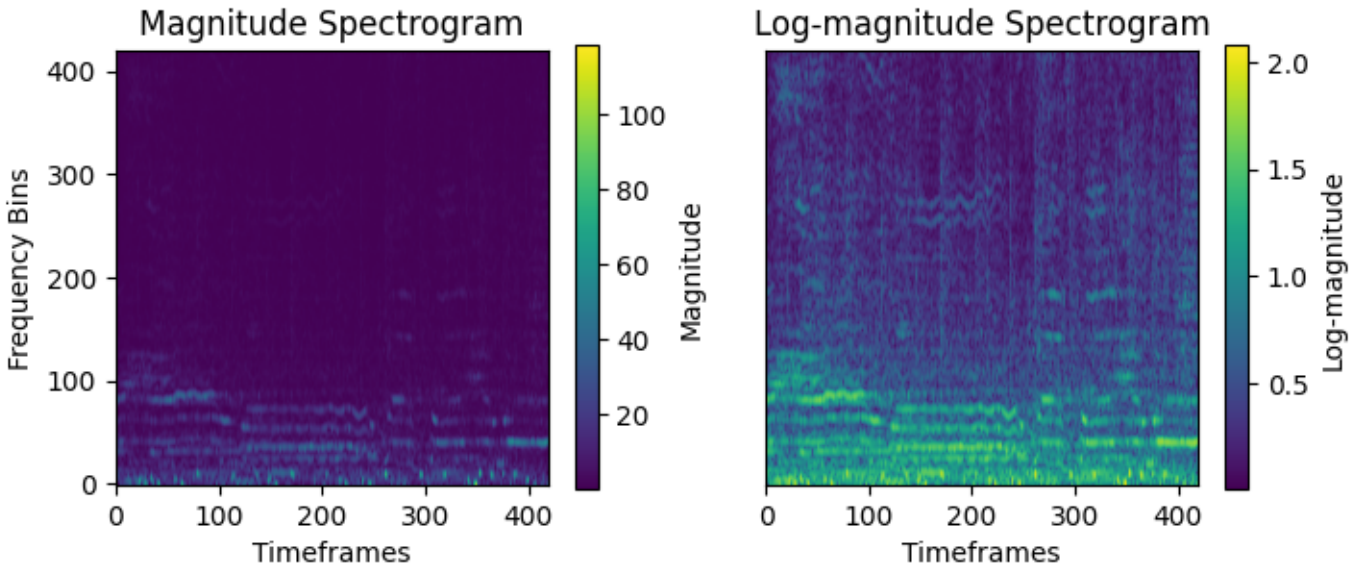

Figure 2.8: Comparison between a linear magnitude spectrogram and a log-magnitude spectrogram. When scaled linearly, differences in magnitude are harder to distinguish, especially for quieter frequency components. The logarithmic scale makes both loud and quiet components visually more distinguishable.

2.2.8 Filters

Signal frequencies and human perception have a non-linear relationship. Just as loudness is perceived on a logarithmic scale, so too is pitch. Humans perceive logarithmic differences in frequency as a linear difference in pitch, and are more sensitive to changes in lower frequencies than in higher ones. For example, the notes A_2 and B_2 differ by approximately 13.47 Hz, while D_7 and E_7 differ by nearly 287.70 Hz, yet both are perceived as being one semitone apart, meaning they audibly have the same perceived difference in pitch. Because the frequency bins in a standard spectrogram are spaced evenly in frequency, they do not reflect how pitch is actually perceived, with finer sensitivity in lower regions and coarser sensitivity in higher ones.

To better reflect this, we apply a set of *filters*, each emphasizing a certain frequency range. A filter is typically a triangular weighting function centered at a specific frequency, gradually tapering off toward neighbouring bins. Multiple filters are combined into a *filterbank*, which can be applied to a spectrogram using matrix multiplication. This transforms the frequency axis into a new scale that better aligns with how we hear changes in pitch.

Mel Spectrograms

The Mel scale, introduced by Stevens, Volkmann, and Newmann in 1937, transforms the frequency axis into a perceptual pitch scale where equal steps correspond to equal perceived pitch differences. In other words, a linear difference in mels is perceived as a linear difference in pitch. Applying a set of triangular Mel-filters results in a *Mel spectrogram*, a representation widely used in audio-related machine learning tasks. Mel spectrograms have shown successful application in AMT. [19, 13, 23, 57, 58]

Logarithmic Filters

While the Mel scale was designed to reflect how humans perceive pitch, a different trend has emerged within Automatic Drum Transcription (ADT). Rather than using perceptual spacing, many recent ADT approaches apply *logarithmically spaced filters*, centered on the note A_4 (440 Hz), resulting in what is referred to in this thesis as a *logarithmically filtered spectrogram*.

This approach does not attempt to mimic human perception directly, but instead reshapes the frequency axis to reflect musical structure, spacing filters uniformly on a logarithmic scale, with 12 filters per octave to match the 12 semitones in Western music. The filters are designed to cover the human hearing range, from 20 Hz to 20,000 Hz, resulting in a filterbank with 84 frequency bins. Each filter is triangular and area-normalized, such that the area under each filter curve equals 1.

This representation has been adopted in several recent ADT studies, particularly by Vogl et al., and is often preferred when preserving musical relationships is important. Compared to Mel spectrograms, it provides a harmonically consistent frequency resolution across octaves, while also reducing input dimensionality [55, 56, 31, 61, 62].

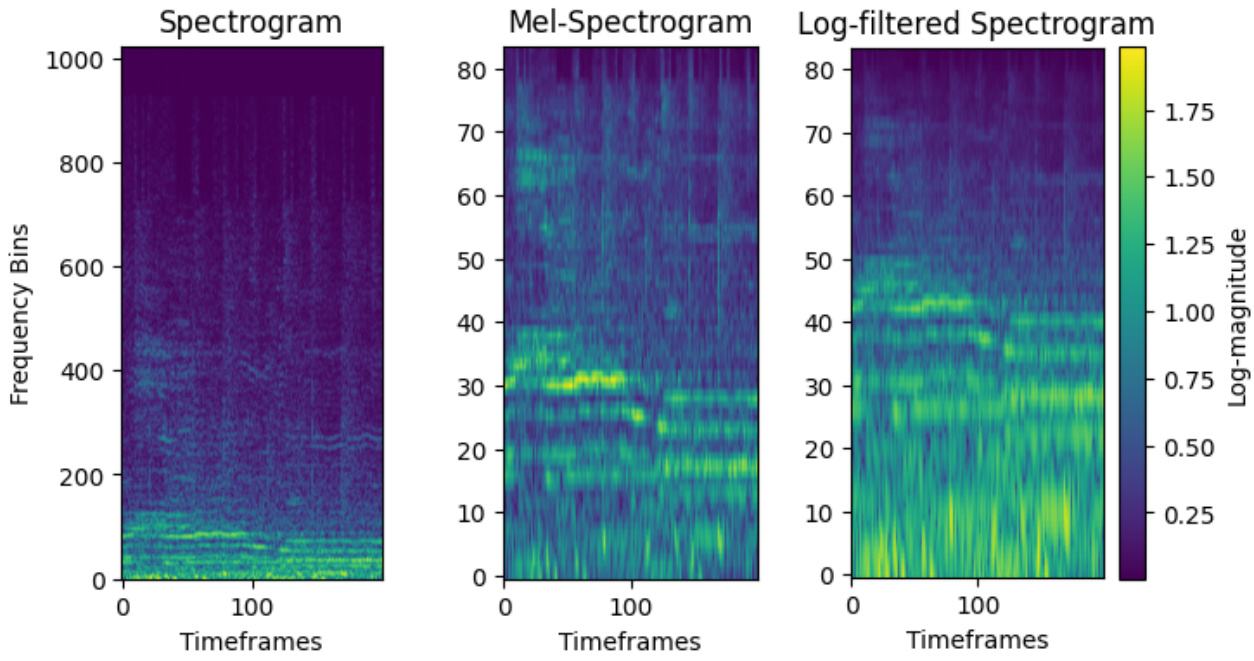

Figure 2.9: Comparison between a full spectrogram, a Mel spectrogram, and a log-filtered spectrogram. All are presented on a log-magnitude scale. Both the Mel and log-filtered spectrograms apply 84 area-normalized filters, resulting in a non-linear frequency axis. While the Mel spectrogram emphasizes perceptual resolution, with greater detail in lower frequencies, the log-filtered spectrogram maintains uniform resolution across octaves, aligning with musical intervals.

2.3 Transcription

Transcription refers to the process of converting an auditory signal, such as music, into a structured representation in another medium. In a musical context, this representation serves as a description of the original performance and may take several different forms.

2.3.1 Music Notation

Modern staff notation is a symbolic transcription systen that provides a structured set of instructions for a musician to reproduce a performance. It has become the standard way of documenting music and is widely used by musicians across many different genres and instruments.

Sheet music written in this notation is typically highly descriptive, containing information such as note onsets, pitch (for pitched instruments), instrument type, velocity,

and tempo. Time is represented horizontally from left to right, while pitch or instrument assignment is encoded vertically. For pitched instruments, the vertical position of the note corresponds to its pitch, while for instruments like percussion, the vertical position indicates the specific drum instruments. The duration of the note is denoted through the shape of the notehead, stem, or additional note decorations.

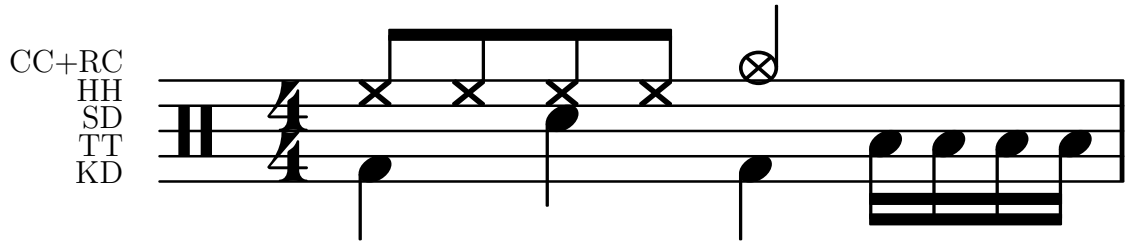

Figure 2.10: An example drum sheet containing all instruments used in a 5-instrument ADT task. The instrument associated with each row is shown on the left. From bottom to top, they are the Kick Drum (KD), Tom-Tom (TT), Snare Drum (SD), Hi-Hat (HH), and Crash and Ride (CC+RC).

Note that rest marks such as 𝄻, 𝄽, 𝄾, and 𝄿 may appear in notation transcriptions. These indicate brief silences or pauses in the music. Although they may appear inline with other onset events, they do not represent instrument activity and should not be interpreted as onsets.

2.3.2 MIDI Annotations

Musical Instrument Digital Interface (MIDI) is the industry standard for representing and controlling digital music. It is an event-based protocol, encoding sequences of commands used to synthesize or trigger musical sounds. As it is stored in a binary format, MIDI is not directly readable to humans without translating it into a more interpretable representation.

When computers play MIDI sequences, the events are parsed in order at a fixed rate, using *note on*/*note off* messages seperated by time deltas. Similar to sheet music, MIDI is highly descriptive, capturing information such as pitch, velocity and timing. Intuitively, one might think of MIDI as serving the same role for computers as sheet music does for musicians.

Recently, several works in AMT have explored generating transcriptions in a MIDI-like format, with promising results. Using an NLP approach, Gardner et al. introduced

MT3 [19], a model that takes log-Mel spectrograms as input and autoregressively predicts MIDI events. This format was further developed by Chang et al. in YourMT3+ [13], utilizing an LLM instead.

```
MetaEvent DeviceName SmartMusic SoftSynth 1   start : 0  delta : 0
MetaEvent SequenceName Instrument 2           start : 0  delta : 0
CC  Ch: 1 C: MAIN_VOLUME value: 101  start : 0  delta : 0
CC  Ch: 1 C: PANPOT value: 64        start : 0  delta : 0
ON:  Ch: 1 key: 67 vel: 96     start : 3072     delta : 3072
OFF: Ch: 1 key: 67 vel: 0      start : 4096     delta : 1024
ON:  Ch: 1 key: 67 vel: 96     start : 4096     delta : 0
OFF: Ch: 1 key: 67 vel: 0      start : 5120     delta : 1024
ON:  Ch: 1 key: 66 vel: 96     start : 5120     delta : 0
OFF: Ch: 1 key: 66 vel: 0      start : 6144     delta : 1024
ON:  Ch: 1 key: 62 vel: 96     start : 6144     delta : 0
OFF: Ch: 1 key: 62 vel: 0      start : 7168     delta : 1024
ON:  Ch: 1 key: 64 vel: 96     start : 7168     delta : 0
OFF: Ch: 1 key: 64 vel: 0      start : 7680     delta : 512
ON:  Ch: 1 key: 62 vel: 96     start : 7680     delta : 0
OFF: Ch: 1 key: 62 vel: 0      start : 8192     delta : 512
ON:  Ch: 1 key: 60 vel: 96     start : 8192     delta : 0
OFF: Ch: 1 key: 60 vel: 0      start : 9216     delta : 1024
ON:  Ch: 1 key: 62 vel: 96     start : 9216     delta : 0
```

Figure 2.11: Example MIDI arrangement in text format. The first lines contain metadata, followed by note on/off events. Each event includes temporal information (start, delta), as well as musical attributes such as channel, key, and velocity, which determine the instrument and sound [50].

2.3.3 Activation Functions

In machine learning, the task of detecting instrument onsets can be framed as a multi-label sequence labeling problem. For each time frame in a sequence, the model predicts a confidence value (which can be interpreted as a probability), that a given instrument onset occurs at that moment. In the context of MIR and AMT, it has become common practice to describe these confidence distributions over time as *activation functions* (not to be confused with the activation functions used within neural networks, such as ReLU or sigmoid) [44, 7, 47, 58, 56].

Frame-level prediction of activation functions is a common approach in onset detection for ADT, and is the format adopted in this thesis. Each drum instrument's activation function forms a row in a matrix, with time progressing along the columns.

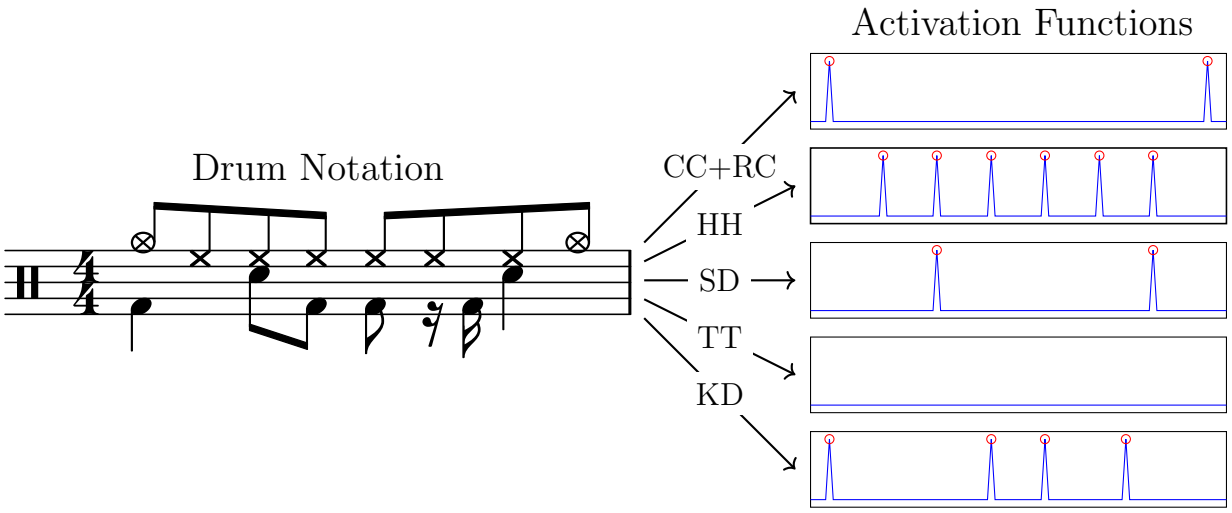

Figure 2.12: Activation function representation for a 5-instrument ADT task, corresponding to a drum pattern written in standard staff notation. Each activation function is zero throughout, except at onset positions, where it takes a value of one. The $\mathcal{7}$ symbol indicates a sixteenth-note rest in the Kick Drum (KD) instrument, and should not be interpreted as an onset.

Peak-picking

When predicting activation functions, a separate post-processing step is required to convert the continuous confidence distributions into discrete onset events. This is commonly achieved using a standard *peak-picking* algorithm, which isolates and enhances peaks in the activation functions.

The peak-picking algorithm, introduced in its current form by Böck et al. [6], defines an onset at time frame n if the predicted activation $\hat{y}_n$ satisfies the following three conditions:

$$\begin{aligned} \hat{y}_n &= \max(\hat{y}_{n-m}, \dots \hat{y}_n, \dots \hat{y}_{n+m}), \\ \hat{y}_n &\geq \text{mean}(\hat{y}_{n-a}, \dots \hat{y}_n, \dots \hat{y}_{n+a}) + \delta, \\ n &\geq n_{\text{last onset}} + w. \end{aligned}$$

Here, m defines the local window size for detecting peaks, a controls how much context is used when averaging surrounding values, δ sets how far above the average the peak must be, and w defines the minimum spacing between two detected onsets. For appropriately trained deep learning models, Vogl et al. [56] found that the peak-picking parameters that gave the best results were $m = a = w = 2$ and $\delta = 0.1$. Consequently, these parameter values are also adopted in this thesis.

2.4 Automatic Drum Transcription

As mentioned, Automatic Drum Transcription (ADT) refers to the task of transcribing symbolic drum notation from audio recordings. More specifically, the field can be divided into several subtasks, ordered from simpler to more complex settings [58]. These include:

- DSC: Drum sound classification, which involves identifying the instrument class of isolated, single-event drum recordings.
- DTD: Drum transcription from recordings containing only drum instruments.
- DTP: Drum transcription with percussive accompaniment, where the model must transcribe the drum instruments while ignoring non-drum percussion.
- DTM: Drum transcription from full musical mixtures, where both melodic and percussive instruments are present and only drum onsets should be transcribed.

This thesis focuses on the most complex of these tasks, namely Drum Transcription in the Presence of Melodic Instruments (DTM). Intuitively, the goal is to develop a deep learning model that, given input audio in the form of music containing both drums and melodic instruments, can detect and classify drum instrument onsets while selectively ignoring unrelated melodic content. This setting introduces challenges not present in the simpler variants. As Zehren et al. [61] note, *"melodic and percussive instruments can overlap and mask eachother..., or have similar sounds, thus creating confusion between instruments"*.

Deep learning has proven to be an effective approach for DTM, and a range of architectures have been explored with promising results. Vogl et al. [55, 56] achieved strong results using both convolutional and convolutional-recurrent networks. Zehren et al. [61, 62] emphasized the importance of dataset size and quality, showing that both are critical to achieving strong performance. Most recently, Chang et al. [13] proposed an autoregressive language-model approach for multi-instrument transcription (AMT), with competitive results on DTM.

These works highlight that multiple modeling directions remain viable and that further improvements in ADT and DTM performance are still possible.

2.4.1 The Drum Set

A drum set is a collection of percussive instruments, typically including drums, cymbals, and possibly different auxiliary percussion. While the exact configuration can vary, a standard drum kit typically includes a Snare Drum (SD), a KD, one or more Tom-Toms (TTs) (toms), one or more cymbals (Crash Cymbal (CC) and Ride Cymbal (RC)), and a Hi-Hat (HH) cymbal [37].

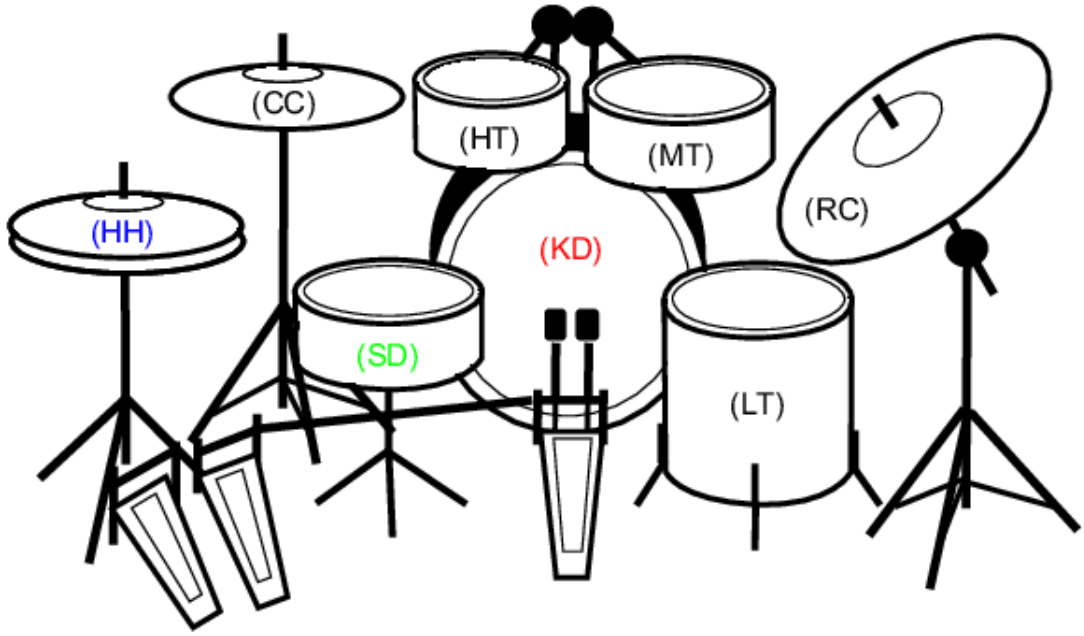

Figure 2.13: The standard drum set configuration, including the Kick Drum (KD), Snare Drum (SD), Hi-Hat (HH), Crash Cymbal (CC), Ride Cymbal (RC), High Tom (HT), Mid Tom (MT), Low Tom (LT) [58].

Percussion instruments like those in a drum set stand apart from melodic instruments in that different components, such as the snare, kick, and hi-hat, produce distinctly different acoustic signatures. Even within a single drum, variations in playing technique can significantly affect the resulting sound or *"audible footprint"*. The snare drum, kick drum, and hi-hat differ significantly in timbre, frequency range, loudness, and function, making them acoustically and musically distinct.

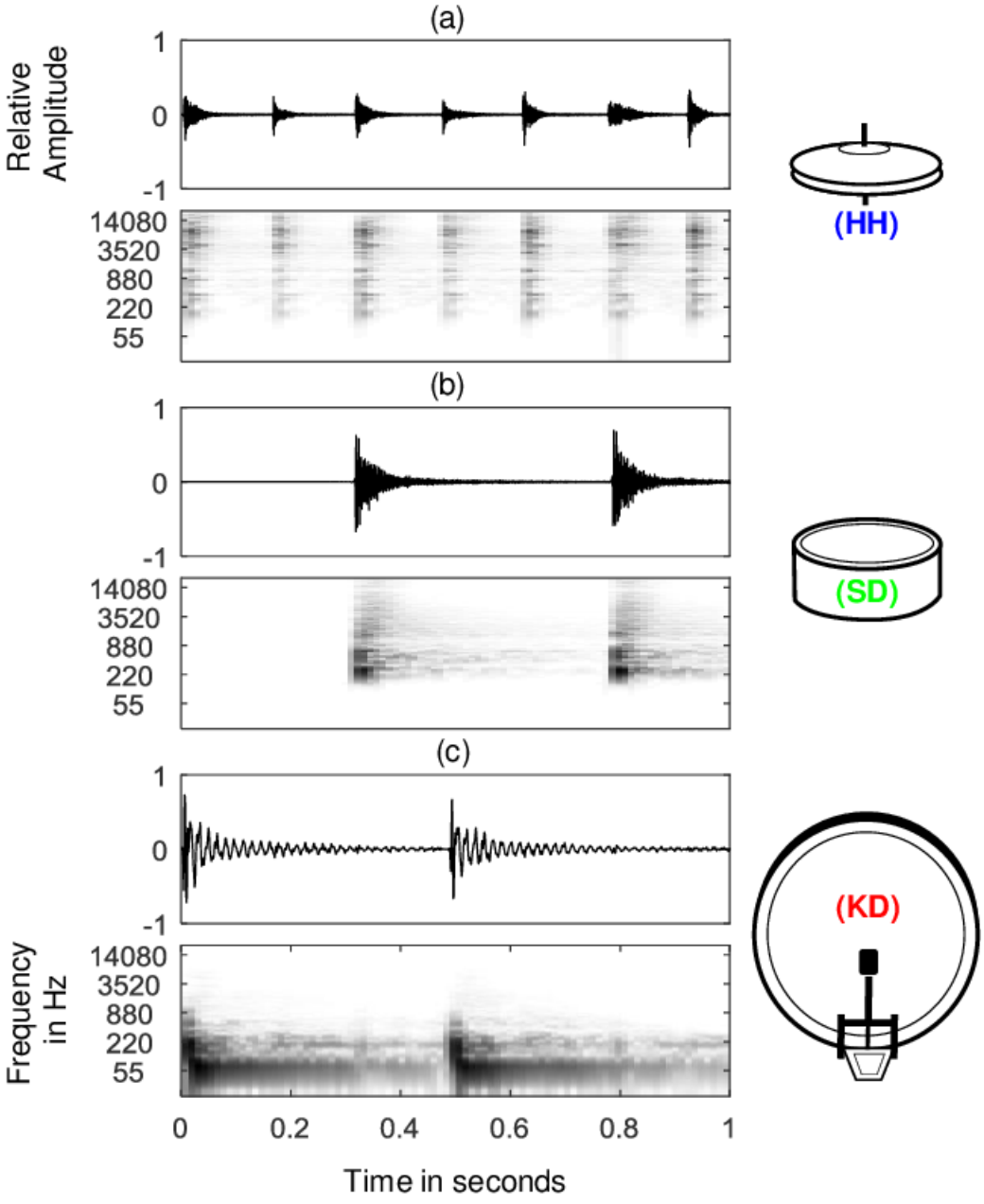

Figure 2.14: Examples of the different audible footprints of drum set components. Plotted are the waveforms of three drum instruments played at varying patterns, each paired with a corresponding spectrogram. The panels highlight how each instrument's events produce distinct patterns in both the time and frequency domains [58].

2.4.2 Transcription Task

Understanding the transcription pipeline, specifically in the context of ADT, is essential for this thesis. The process begins with a raw audio waveform representing the musical recording to be transcribed, typically segmented into smaller, non-overlapping

partitions [56, 19]. Each segment is transformed into a spectrogram, which captures the frequency content over time while reducing temporal resolution, making it easier for models to interpret.

The spectrogram is then passed into an ADT model, such as a DNN, which predicts, for each time frame, the likelihood that a given drum instrument is played. These continuous probability estimates, commonly referred to as activation functions, are then post-processed into a more interpretable format, such as discrete onsets or drum notation.

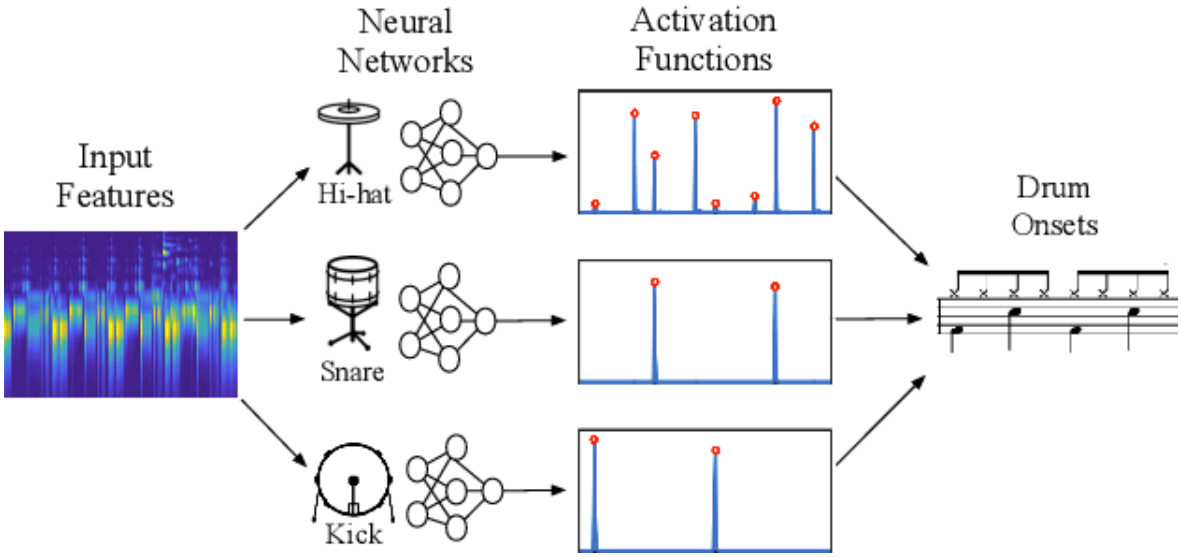

Figure 2.15: Example of a typical ADT pipeline. Given an input spectrogram, the model predicts activation functions for each instrument, which are then quantized into drum onsets and converted into symbolic sheet music [47].

Designing such a pipeline involves several key choices. For instance: How are the input and output of a model structured? While the input here is always a spectrogram, its size and structure can vary depending on parameters such as window length, hop length, and number of frequency bins, as discussed previously. The model's output is also structured as a sequence, with length tied to the input's time resolution and dimensionality determined by the number of instruments being transcribed.

Early work in ADT often adopted a 3-instrument setup, predicting only the Kick Drum (KD), Snare Drum (SD) and Hi-Hat (HH) [54]. While this configuration offered a useful proof-of-concept for early model development, it omits essential components of the drum set. To better capture realistic use cases, I instead use a 5-instrument setup, which also includes cymbals (combining both Crash Cymbal (CC) and Ride Cymbal (RC)) and Tom-Tom (TT) (covering all toms). This adds complexity to the problem but allows the system to represent the full standard drum kit.

2.5 Performance Measure

2.5.1 Correct Predictions

ADT models predict instrument onsets on a frame-level basis. To account for slight misalignments between predicted and ground truth events, evaluation is typically performed using a *tolerance window*, where a prediction is considered correct if it falls within a fixed time window around the reference onset, commonly between 25 ms and 50 ms [54]. This approach shifts the evaluation focus from framewise labels to discrete event matching and introduces challenges when defining standard classification metrics.

2.5.2 Accuracy

Although accuracy is a widely used performance measure in classification, it is ill-suited for ADT. Most frames contain no onsets, creating a heavy class imbalance that allows a naïve model to achieve high accuracy by simply predicting silence. Moreover, due to the event-based evaluation and tolerance window, the number of True Negatives (TNs) are generally undefined, rendering the standard accuracy formula:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}},$$

inapplicable in practice.

2.5.3 F1-score

Due to the issues outlined above, the *F1-score* has become the standard evaluation metric in ADT. It balances two competing objectives: *precision*, which measures the proportion of correct predictions among all predicted events;

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}},$$

and *recall*, which measures the proportion of correctly predicted events among all actual events;

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

High precision indicates that the model rarely produces False Positives (FPs) (i.e., it predicts an onset only when there truly is one), while high recall indicates that the model rarely produces False Negatives (FNs) (i.e., it doesn't predict an onset only if there truly isn't one). The F1-score captures both by computing their harmonic mean:

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}.$$

This encourages a balance between precision and recall and is particularly well-suited for onset detection tasks, where both types of error (misses and false alarms) are critical.

2.5.4 Micro vs. Macro

In multi-label tasks like ADT, there are multiple ways to aggregate per-class F1-scores into a single performance metric. Although they may appear similar, the distinction between *macro* and *micro* F1-score has important implications for model evaluation.

Macro F1-score is computed as the unweighted arithmetic mean of F1-scores for each class. It treats all classes equally, regardless of their frequency in the dataset. In the context of ADT, this means emphasizing balanced transcription performance across all instruments, including those that occur infrequently, such as toms or cymbals.

Micro F1-score, by contrast, is computed using global counts of True Positives (TPs), FPs, and FNs, effectively weighting each class according to its frequency. This tends to favor models that perform well on common instruments like the snare drum or bass drum, even if their performance on rarer instruments is weaker.

For ADT, micro F1-score is often preferred, as it better reflects a model's overall transcription ability on real-world music. In practice, frequent instruments form the backbone of a drum performance, and prioritizing their transcription tends to align better with musical utility. While macro F1-score offers useful diagnostic insight, especially for evaluating per-instrument weaknesses; micro F1 is typically used as the main evaluation criterion.

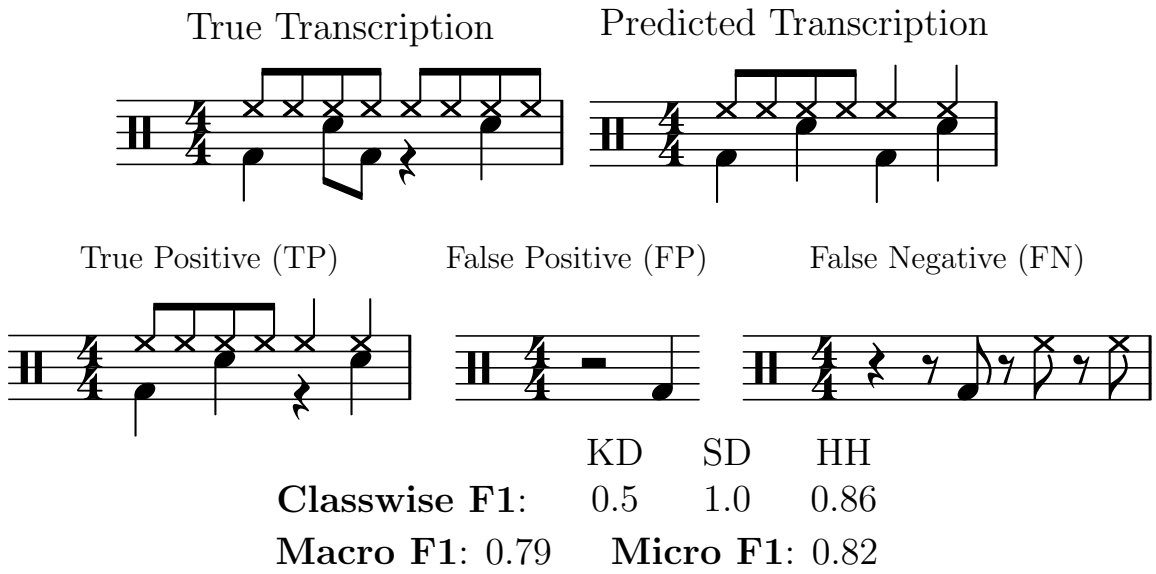

Figure 2.16: Illustration of micro and macro F1-score computation for an example ADT transcription. Symbols such as 𝄽, 𝄾, and 𝄿 denote pauses, and not onsets. Macro F1 reflects the average performance across individual instruments, while micro F1 captures the overall transcription quality across the entire sequence. For clarity, F1-scores are reported with two decimal places.

Chapter 3

Architectures

Choosing an suitable architecture is crucial for achieving strong performance in deep learning. The choice of model defines not only the capacity for learning complex patterns, but also the inductive biases that guide generalization. In the context of ADT, these models must operate on time-frequency representations of audio and produce accuracte, frame-level onset predictions.

Each input to the model is a spectrogram of shape $(T \times D_{STFT})$, where T is the sequence length (i.e., number of frames) and D_{STFT} is the number of frequency bins. The corresponding output is a matrix of shape $(T \times C)$, where C is the number of drum instruments being predicted.

In this thesis, I use log-magnitude spectrograms filtered with 12 logarithmically spaced, area-normalized filters per octave, spanning 20 Hz to 20,000 Hz. This results in $D_{STFT} = 84$ frequency bins. The task is to predict a common 5 class drum vocabulary, consisting of Kick Drum (KD), Snare Drum (SD), Tom-Tom (TT), Hi-Hat (HH), and Crash and Ride (CC+RC), giving $C = 5$ [62].

While the input sequence length T may vary, it is fixed to $T = 400$ for all experiments in this thesis, corresponding to 4 seconds of audio with 10 ms hop length. Thus, the models considered in this chapter operate with input and output shapes of $(T \times 84)$ and $(T \times 5)$, respectively.

3.1 Recurrent Neural Network

Recurrent Neural Networks (RNNs) are a foundational architecture for modeling sequential data and have demonstrated promising results on a wide range of audio-related tasks. Their core component is the *recurrent unit*, which processes an input sequence one frame at a time while maintaining a hidden state that carries information from previous timesteps. This enables the model to capture temporal dependencies within the input.

To include information from both past and future frames, RNNs can be extended to their *bidirectional* variant, where one recurrent layer processes the sequence forward in time and another in reverse. This is particularly useful in tasks such as ADT, where relevant auditory information may span multiple frames; for example, due to instrument timbre persisting after onset.

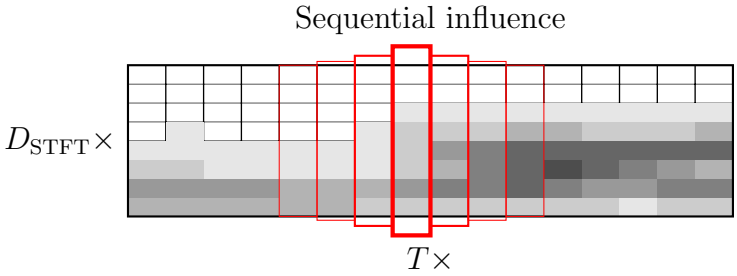

Figure 3.1: Illustration of how bidirectional RNNs include information from surrounding timesteps when predicting the current frame. The background represents a mock spectrogram of shape $(T \times D_{STFT})$, and the red boxes indicate the relative temporal influence on the middle frame. Box height reflects influence strength, gradually tapering off as distance from the center sequentially increases.

Despite their effectiveness, traditional RNNs suffer from the *vanishing gradient problem*, which limits their ability to learn long-range dependencies. To address this, more advanced variants have been proposed, including the Gated Recurrent Unit (GRU) by Cho et al. [14] and the Long Short-Term Memory (LSTM) by Hochreiter and Schmidhuber [26].

Both GRUs and LSTMs have been successfully applied to ADT-related tasks [47, 54, 55, 61].

3.1.1 Implementation

I implemented the RNN architecture using a stack of Bidirectional Recurrent Units (BiRUs), followed by a framewise linear projection. These bidirectional recurrent layers extract and combine local temporal features at each time frame, allowing the model to consider both past and future context. The final linear layer then maps each resulting hidden state to onset probabilities for each of the five drum instruments.

As part of the model search, I experimented with both bidirectional GRUs and LSTMs as the recurrent unit, treating this choice as a tunable hyperparameter. I also varied the number of layers L and the hidden size H.

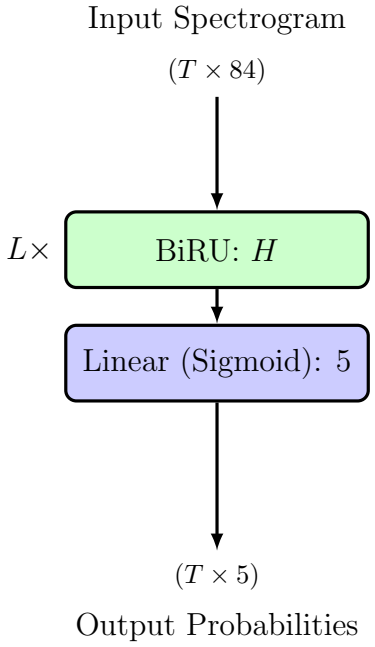

Figure 3.2: Model architecture of the Recurrent Neural Network used in this thesis.

Hyperparameter		Values
L	Number of layers	{2, 3, 4, 5, 6}
H	Hidden size	{72, 144, 288, 576}
BiRU	Bidirectional Recurrent Unit	{GRU, LSTM}

Table 3.1: Hyperparameters and their corresponding search spaces for training the Recurrent Neural Network.

3.2 Convolutional Neural Network

Although spectrograms are often processed as time sequences, we've seen that they also naturally resemble images, with time and frequency forming the horizontal and vertical axes. This makes them well-suited for Convolutional Neural Networks (CNNs), which are designed to extract local patterns in data.

A CNN operates by applying learnable filters, called *kernels*, across the input. These filters aggregate spatially local information, allowing each output unit to incorporate context from its surrounding region. When applied to spectrograms, convolutional layers enable the model to detect patterns in time-frequency space. This makes them a strong candidate for tasks like ADT, where relevant features often span multiple adjacent time frames and frequency bins.

CNNs have also been shown to perform well in ADT, likely because this contextual information helps the model more easily learn the characteristics of instrument onsets. They are also relatively efficient to train and run, which may also help explain their reasonable performance [55].

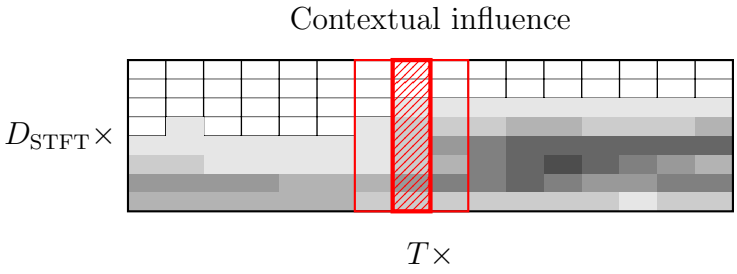

Figure 3.3: Illustration of how CNNs use a fixed-size receptive field to incorporate local context from surrounding time frames. The background represents a mock spectrogram. The red shaded region marks the center time frame being predicted, and the red boxes indicate the receptive field, the neighboring frames that influence the prediction.

3.2.1 Implementation

My CNN architecure begins with I initial convolutional blocks, each designed to preserve the temporal resolution of the input, ensuring that the output retains the same sequence length T. Within these blocks, the number of kernels K_i increases with block depth i, using $K = \{32, 64, 96\}$ to enable deeper layers to capture more complex time–frequency

ptterns. This kernel progression is fixed (i.e., not treated as a hyperparameter), and follows a common design convention of increasing the filter count with depth [46].

These convolutional blocks iteratively combine local features from the spectrogram, forming a richer internal representation of the input. This representation is then passed through L fully connected layers, which project the features into an H-dimensional latent space for final interpretation.

Each convolutional and fully connected layer is followed by a Rectified Linear Unit (ReLU) activation function. Finally, an output layer computes onset probabilities for each of the five drum instruments.

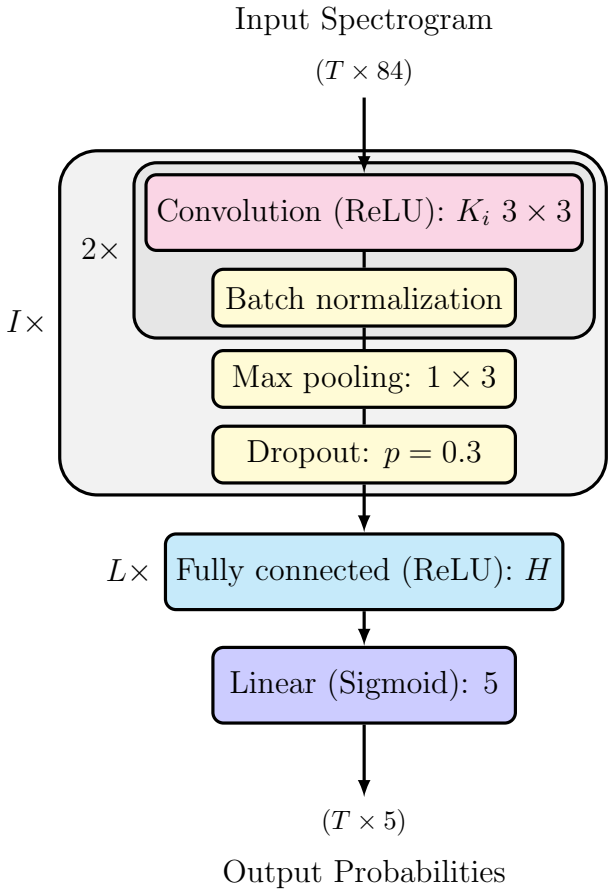

Figure 3.4: Model architecture of the Convolutional Neural Network used in this thesis.

	Hyperparameter	Values
I	Number of convolutions	{1, 2, 3}
L	Number of layers	{2, 3, 4}
H	Hidden size	{72, 144, 288, 576}

Table 3.2: Hyperparameters and their corresponding search spaces for training the Convolutional Neural Network.

3.3 Convolutional Recurrent Neural Network

The strengths of the recurrent layers and convolutions are not mutually exclusive. Theoretically, they can harmonize when combined into a unified architecture, the Convolutional Recurrent Neural Network (CRNN), where each component complements the other.

Intuitively, the ability of CNNs to extract local time–frequency patterns from spectrograms, together with the ability of RNNs to model temporal dependencies, makes this combination particularly well-suited for ADT. This merging of contextual representation and cross-timestep memory has shown promising results in prior work [55, 56, 61].

3.3.1 Implementation

I implemented the CRNN using a fixed stack of $I = 2$ initial convolutional blocks, a setup inspired by prior work in ADT [55, 61]. These convolutional blocks use an increasing number of kernels $K = \{32, 64\}$, allowing deeper layers to extract more complex time–frequency patterns. As with the pure CNN, the convolutions preserve the input's temporal resolution.

The resulting convolutional output is passed to a BiRU, which models temporal dependencies across frames. As in the RNN architecture, I experimented with both GRUs and LSTMs. Each timestep's hidden state is then passed into the final linear layer, which computes and outputs onset probabilities for each of the five drum instruments.

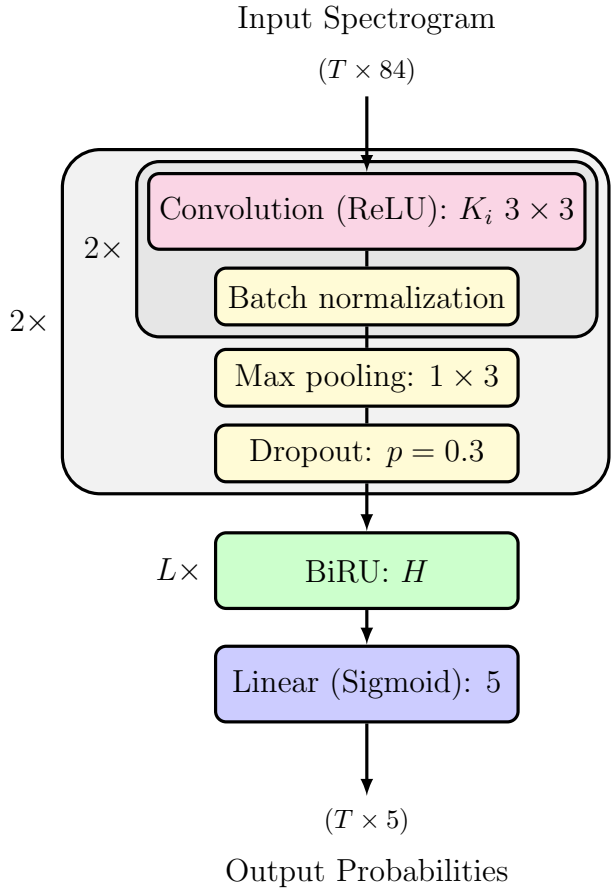

Figure 3.5: Model architecture of the Convolutional Recurrent Neural Network used in this thesis.

Hyperparameter		Values
L	Number of layers	{2, 3, 4, 5}
H	Hidden size	{72, 144, 288, 576}
BiRU	Bidirectional Recurrent Unit	{GRU, LSTM}

Table 3.3: Hyperparameters and their corresponding search spaces for training the Convolutional Recurrent Neural Network.

3.4 Convolutional Transformer

While CRNNs have proven effective in combining local time–frequency pattern extraction with temporal modeling, their ability to capture long-range dependencies still remain limited. Recurrent layers often struggle to maintain distant information through their hidden states, and convolutional layers are constrained by the size of their fixed receptive field.

To address these limitations, a major architectural shift was instigated, most notably with the introduction of the *attention* mechanism in Google's "Attention Is All You Need" [53]. This mechanism allows models to dynamically focus, or *attend*, to different parts of an input sequence by learning relationships between its elements. Replacing recurrent layers with stacks of attention-based blocks gave rise to a new class of models: the *transformers*.

Unlike recurrent units, which propagate information sequentially through hidden states, attention layers allow each timestep to directly access information from every other timestep in the sequence. This gives the model a more flexible way to capture a global context. Intuitively, it enables each element to *"intelligently"* decide which other parts of the sequence it wants to focus on, rather than depending on neighbouring timesteps to pass information forward.

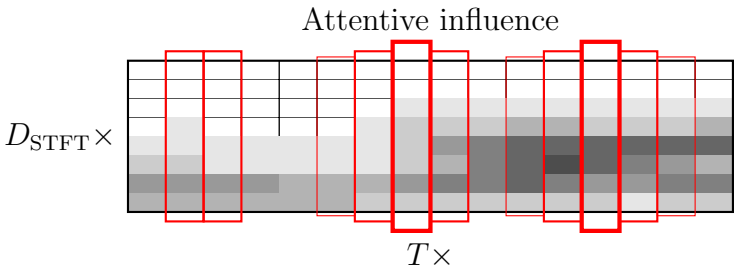

Figure 3.6: Ilustration of how attention layers enable influence from time frames across the entire sequence. The background represents a mock spectrogram, and the red boxes indicate the time frames influencing the prediction on the middle frame. Box height reflects the reltive influence strength, as determined by attention weights.

Attention mechanisms have recently shown strong performance in both AMT and ADT, in some cases outperforming RNNs. Replacing recurrent layers with transformer blocks may therefore improve the model's ability to capture long-range dependencies, particularly when combined with convolutional layers that extract local time–frequency patterns [29, 19, 61, 13, 62].

3.4.1 Implementation

I implemented the convolutional transformer architecture using an initial fixed stack of $I = 2$ convolutional blocks, following the same configuration as in the CRNN. These blocks use an increasing number of kernels $K = \{32, 64\}$, as in the CRNN setup.

The convolutional output is then projected into a lower-dimensional embedding space if size D_e, which reduces computational load and acts as input to the transformer blocks. A sinusoidal positional encoding is added to this embedding to provide the model with temporal ordering information, compensating for the transformer's lack of inherent sequence structure.

The core of the model consists of L transformer block with pre-layer normalization, a design shown to improve training stability compared to post-layer normalization [59]. Each block contains multi-head self-attention with H heads, enabling the model to capture global dependencies across the input sequence. The first layer of each block's feed-forward component uses the Gaussian Error Linear Unit (GELU) activation function, which is standard in transformer-based models and has shown improved performance over ReLU [16, 25].

Finally, a linear output layer computes onset probabilities for each of the five drum instruments.

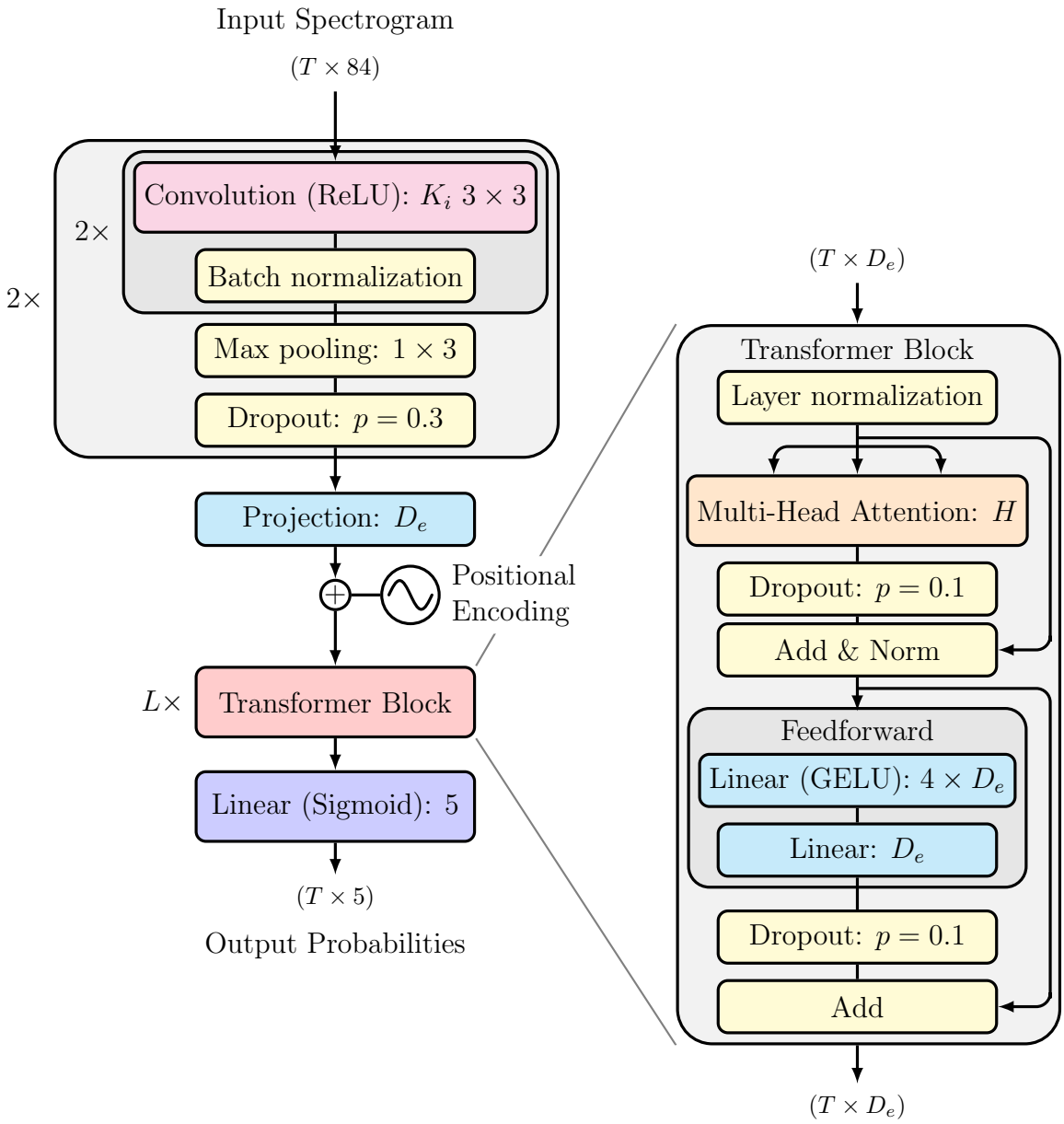

Figure 3.7: Model architecture of the Convolutional Transformer used in this thesis.

Hyperparameter		Values
H	Number of heads	{2, 4, 6, 8}
L	Number of layers	{2, 4, 6, 8, 10}
D_e	Embedding dimension	{72, 144, 288, 576}

Table 3.4: Hyperparameters and their corresponding search spaces for training the Convolutional Transformer.

3.5 Vision Transformer

The introducing of the transformer naturally raises a question: is it possible to build an architecture composed solely of attention layers, eliminating convolutions and removing the need for hybrid convolutional-transformer designs? This question was explored in Google's "An Image Is Worth 16x16 Words" [17], which introduced the Vision Transformer (ViT).

Originally developed for image recognition tasks, the Vision Transformer has since been applied to audio-based problems as well, demonstrating strong performance in both domains [17, 23]. However, to the best of my knowledge, its application to ADT remains unexplored, making it a novel approach evaluated in this thesis.

While Vision Transformers have shown excellent performance, they typically require substantially more data, or benefit heavily from large-scale pre-training, to reach their full potential. [17].

3.5.1 Patch Embedding

A key component of the Vision Transformer is the use of patch embeddings. The input image is first divided into patches, each of which is flattened and linearly projected into a latent vector. These latent vectors, known as patch embeddings, together form a sequence that represents the entire input image. To retain spatial information, a positional encoding is added to each vector, often implemented as a learnable embedding.

Although Vision Transformers are often described as convolution–free, patch embeddings are typically implemented using a single 2D convolutional layer with a kernel size and stride equal to the patch size (assuming non-overlapping patches). This acts as a linear projection over each patch and does not involve any activation function.

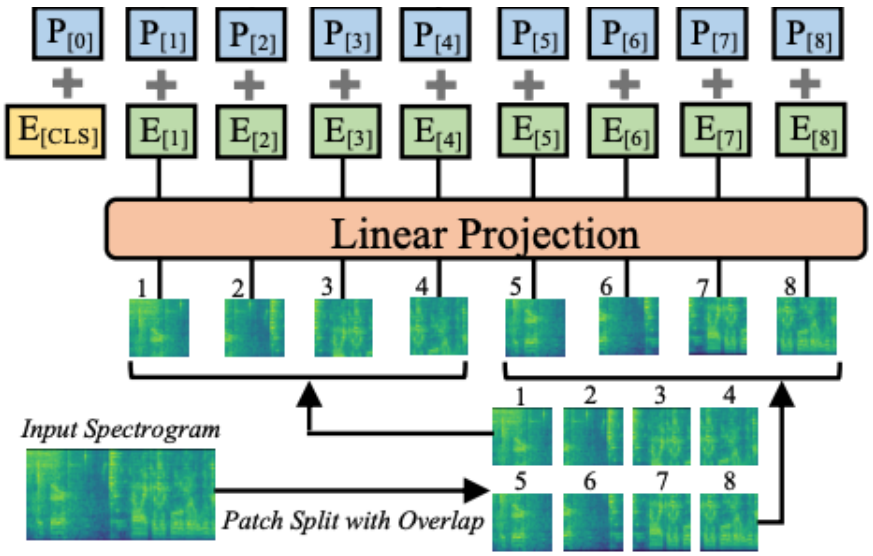

Figure 3.8: Creation of a sequence of patch embeddings from an input spectrogram. The spectrogram is split into overlapping patches, each of which is linearly projected into a one-dimensional embedding vector. A learnable positional embedding is then added to each patch embedding. An initial Class Token ([CLS]) is also included, as is common in classification tasks [23].

3.5.2 Architecture Modifications

The original Vision Transformer was designed for classification tasks, where the output is a single vector rather than a sequence. In contrast, ADT is a sequence labeling task, requiring an output sequence whose temporal resolution matches that of the input.

To accommodate this, we reinterpret groups of patch embeddings as representing individual time frames. This is feasible as long as the number of timesteps divides evenly into the total number of patches, allowing the output to be reshaped to match the desired sequence length.

Since we do not perform classification, the additional Class Token ([CLS]) typically used to summarize the input for class prediction is unnecessary and thus omitted [17].

3.5.3 Implementation

I implemented the Vision Transformer by first converting the input spectrogram into a sequence of patch embeddings with shape $(T \times D_e)$. This is done using a 2D convolutional

layer with kernel size and stride equal to the patch size $(1 \times P)$, where P is the patch height. This value is chosen such that the spectrogram's 84 frequency bins can be evenly divided into $n_P = \frac{84}{P}$ patches. To control the output embedding dimensionality D_e, the convolution uses $\frac{D_e}{n_P}$ kernels.

Since each patch has a width of 1, it retains information only about the frequencies within its own time frame, helping preserve the temporal resolution of the input spectrogram. A learnable positional embedding of shape $(\frac{D_e}{n_P} \times n_P)$ is then added to each patch to encode spectral position and kernel identity. These patches are then permuted and flattened into a sequence, forming the final patch embedding representation.

While each patch embedding encodes its relative position within a time frame, it lacks information about which time frame it belongs to. To introduce this temporal ordering, I add a sinusoidal positional encoding across the time dimension. The resulting sequence is then passed through L transformer blocks with pre-layer normalization and multi-head self-attention using H heads, following the same design as in the convolutional transformer. Finally, a linear output layer computes onset probabilities for each of the five drum instruments.

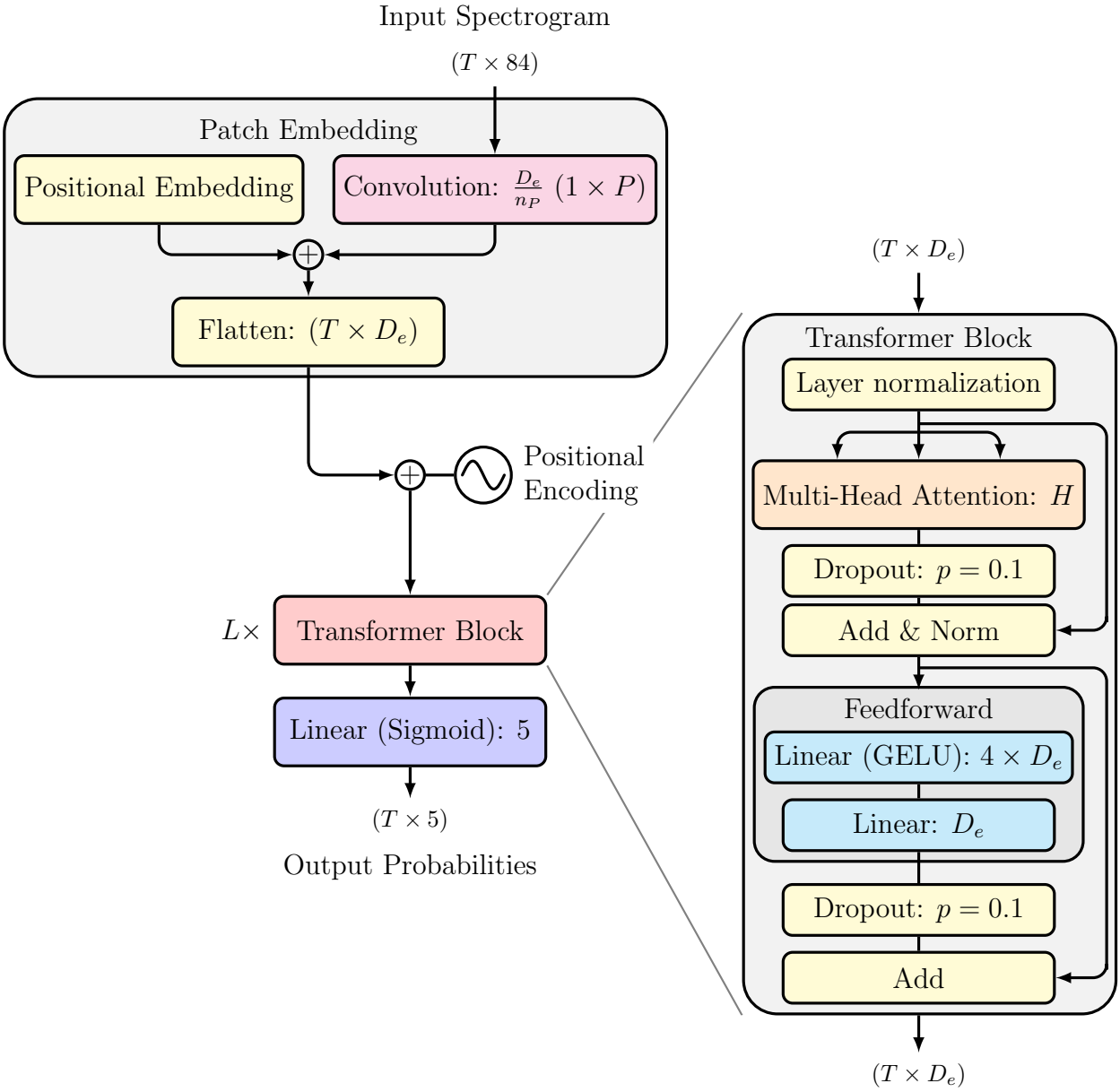

Figure 3.9: Model architecture of the Vision Transformer used in this thesis.

Hyperparameter		Values
P	Patch height	{7, 14, 21}
H	Number of heads	{2, 4, 6, 8}
L	Number of layers	{2, 4, 6, 8, 10}
D_e	Embedding dimension	{72, 144, 288, 576}

Table 3.5: Hyperparameters and their corresponding search spaces for training the Vision Transformer.

Chapter 4

Datasets

4.1 ENST+MDB

The ENST-Drums dataset by Gillet and Richard [20] is one of the most commonly used ADT datasets [58]. It features thoroughly annotated, real drum recordings performed by three drummers across various musical genres. Most tracks are drum-only, except for the *minus-one* subset, which includes accompaniment. Since this thesis focuses on DTM, I limit my use to subset.

The dataset provides separate audio files for the drum performance and the accompaniment. I additively combine these to form a single mixture track. Although multiple microphone recordings from different spatial positions are available for each performance, I exclusively use the *"wet mix"* as it combines the individual channels into a polished recording that closely resembles a professionally mixed track.

ENST-Drums contains 1.02 hours of music across 64 tracks and was accessed via Zenodo [21].

The MDB Drums dataset by Southall et al. [48] is another well-known dataset used for ADT. It is built on Bittner et al.'s MedleyDB (MDB) dataset [5], but has been re-annotated and specifically adapted for ADT-related transcription tasks. Like ENST-Drums, it is split into different stem tracks, including isolated drum recordings and accompaniement. However, it also provides pre-mixed *full mix* tracks, which I use in this thesis.

MDB Drums include transcriptions at two levels of granularity: a *class* file with 6 broad drum classes, and a *subclass* file with 21 more specific drum instruments. While the 6-class transcriptions may seem more aligned with a 5-instrument setup, as used in this thesis, I found that the 21-class transcriptions offered more precise control over mapping specific instruments into the 5-instrument vocabulary. Therefore, I use the 21-instrument subclass transcriptions as annotations.

MDB Drums contains 0.35 hours of music across 23 tracks and was accessed via GitHub [49].

Both datasets distribute their audio as waveform files and their annotations as plain text files, formatted by onset time and instrument label. They are also relatively small in size and contain real, thoroughly annotated data. Due to these similarities, I combine them into a single, slightly larger dataset referred to as ENST+MDB.

In total, ENST+MDB contains 1.37 hours of music across 87 tracks, split into 896, 236, and 152 respective training, validation and test sequences.

4.1.1 Splits

Neither dataset comes with predefined train/validation/test splits, such I construct my own. From ENST-Drums, I use recordings from *drummer1* and *drummer2* for training. The remaining drummer, *drummer3*, is split evenly between validation and test sets. Since MDB Drums does not include explicit drummers, I instead split the data by musical genre. Full details of the splits are provided in Appendix A.

4.1.2 Mapping

The ENST-Drums and MDB Drums datasets contain transcriptions of 20 and 21 distinct drum instruments, respectively. This does not align with the 5-instrument vocabulary used in this thesis. To utilize all the available annotations, I apply a mapping from each original annotated instrument to one of the five target classes. Table 4.1 shows the mapping used for the combined ENST+MDB dataset.

Vocabulary mapping	ENST-Drums	MDB-Drums
Kick Drum (KD)	*bd*	*KD*
Snare Drum (SD)	*sd*, *sweep*, *rs*, *sticks*, *cs*	*SD*, *SDB*, *SDD*, *SDF*, *SDG*, *SDNS*, *SST*
Tom-Tom (TT)	*mt*, *mtr*, *lmt*, *lt*, *ltr*, *lft*	*HIT*, *MHT*, *HFT*, *LFT*
Hi-Hat (HH)	*chh*, *ohh*	*CHH*, *OHH*, *PHH*, *TMB*
Crash and Ride (CC+RC)	*cr*, *spl*, *ch*, *rc*, *c*, *cb*	*RDC*, *RDB*, *CRC*, *CHC*, *SPC*

Table 4.1: Mapping from the original instrument annotations in ENST-Drums and MDB Drums to the 5-instrument vocabulary used in this thesis. Description of the original instruments are available in their respective source papers [20, 48].

It should be noted that ENST-Drums includes more specific instrument annotations, where some instrument labels end with a number (e.g., *"rc3"*) to indicate that multiple instances of the same instrument were present in the drum set, with the number specifying which was played. Additionally, certain snare drum events end with a hyphenated suffix (e.g., *"sd-"*), indicating that the onset corresponds to a softer *ghost note* hit [20]. Following general ADT practice and for consistency with other datasets, I ignore these suffixes and treat such events as standard onsets for their respective instruments.

4.2 E-GMD

The Expanded Groove MIDI Dataset (E-GMD) from Callender et al. [9] is a large ADT dataset consisting of audio recordings from human drum performances annotated in MIDI. It is an expansion of Gillick et al.'s Groove MIDI Dataset (GMD) [22].

GMD was created through recording human performances in MIDI format through a Roland TD-11 electronic drumkit. This dataset does not contain audio recordings, only MIDI files, such that it is not applicable for ADT. To expand this dataset to be used with ADT, Callender et al. re-recorded the MIDI sequences on a Roland TD-17 electronic drumkit in real-time on a Digital Audio Workstation (DAW). These re-recordings were done over a large amount of differing soundfonts (confusingly also called *drumkits* on electronic drumkits), synthesizing several differently sounding audio recordings from a single MIDI performance [22, 9].

In contrast to the other dataset's utilized in this thesis, this dataset is not created for a DTM task, but rather a Drum Transcription of Drum-only Recordings (DTD) task. This is due to all recordings containing solo, drum-only performances. In addition, as

this data is recorded from human performances in a semi-manual nature, there exist some errors from the recording process *"that resulted in unusable tracks."* [9]. The magnitude of these errors are not stated, however other authors propose that it might be as high as 20.5% of the tracks with varying amounts of discrepancies [27].

This dataset contains 444.5 hours of audio recordings, from 1,059 unique drum performances resampled to 45,537 MIDI sequences, accessed through Zenodo [10].

4.2.1 Mapping

The E-GMD dataset is annotated on a Roland electric drumkit and stored in MIDI files. These files store the instruments as specific pitches, with the pitch-to-instrument mapping being given by the standard *General MIDI* percussion key map, as well as specific Roland default MIDI mappings [2].

This *General MIDI* percussion mapping is extensive, containing MIDI pitches for vast amounts of percussive instruments. Many of these either do not appear in drumset transcriptions, or do not have a logical 5-instrument mapping. Such instruments are ignored, and are omitted from the mapping.

General MIDI is a widely used standard, and is extensively documented. The specific MIDI percussion pitch to drum instrument can be found in e.g. "MIDI: A comprehensive introduction" by Rothstein [42], and the Roland specific MIDI pitches in their documentation [2]. In case there are two different instruments marked with the same MIDI pitch, the one from Roland is selected.

Show the mapping directly from Groove MIDI dataset instead (GMD) and Cite!

Vocabulary mapping	MIDI Pitch
Kick Drum (KD)	*35*, *36*
Snare Drum (SD)	*37*, *38*, *39*, *40*
Tom-Tom (TT)	*41*, *43*, *45*, *47*, *48*, *50*, *58*
Hi-Hat (HH)	*22*, *26*, *42*, *44*, *46*, *54*
Crash and Ride (CC+RC)	*49*, *51*, *52*, *53*, *55*, *56*, *57*, *59*

Table 4.2: The mapping for each of the Roland TD-11 and General MIDI percussion pitches onto the 5-instrument vocabulary used in this thesis. MIDI pitches omitted do not have a intuitive 5-instrument drum mapping and are simply ignored.

4.3 Slakh

The Synthesized Lakh 2100 (Slakh) Dataset from Manilow et al. [35] is a synthesized version of a subset of the Lakh Dataset from Raffel [40], that subset being a 2100 randomly selected tracks from Lakh where the MIDI files contain at least a piano, bass, guitar, and drums. These MIDI files are rendered and mixed into combined audio files, stored together with their respective original MIDI performances.

As mentioned, this dataset contains more instruments than just the drumset, such that it is originally meant for a AMT task. However, due to each track being guaranteed to contain a drum performance, converting it to an ADT task is trivial, and is done by selectively only utilizing the MIDI files corresponding to the drumset as our labels.

The original Slakh dataset was found to have data leakage between the different splits, and it is therefore recommended for transcription tasks to use a smaller subset, Slakh2100-redux, where this issue has been solved. Therefore, this is the dataset used for this thesis. Needs citation, other than the github and zenodo?
https://github.com/ethman/slakh-utils, https://zenodo.org/records/4599666

This dataset contains 115 hours of audio recordings over 1709 different songs, accessed through Zenodo [36].

4.3.1 Mapping

The Slakh dataset stores its annotations in MIDI files, similarly to E-GMD. Contrary to E-GMD it strictly uses the General MIDI percussion mapping. This mapping is very similar, only excluding the specific MIDI pitches *22*, *26*, and *58*, seen in Table 4.3.

Vocabulary mapping	MIDI Pitch
Kick Drum (KD)	*35*, *36*
Snare Drum (SD)	*37*, *38*, *39*, *40*
Tom-Tom (TT)	*41*, *43*, *45*, *47*, *48*, *50*
Hi-Hat (HH)	*42*, *44*, *46*, *54*
Crash and Ride (CC+RC)	*49*, *51*, *52*, *53*, *55*, *56*, *57*, *59*

Table 4.3: The mapping for each of General MIDI percussion pitches onto the 5-instrument vocabulary used in this thesis. MIDI pitches omitted do not have an intuitive 5-instrument drum mapping and are simply ignored.

4.4 ADTOF-YT

The Automatic Drums Transcription On Fire YouTube (ADTOF-YT) dataset from Zehren et al. [61] is a large ADT dataset containing crowdsourced data, hence the YouTube suffix. Due to the crowdsourced nature of this dataset, it is possible to utilize large amounts of non-synthetic, human data, but with the tradeoff in which we can not guarantee its quality. It is also worth to be noted ADTOF-YT seems to be biased towards metal and rock musical genres, as noticed by Zehren et al. [61].

Contrary to the other datasets, this one is distributed in prepackaged TensorFlow datasets for each split, with datapoints as pairs of logarithmically filtered log-spectograms and sequence of instrument onset probabilities. This dataset also comes with a vocabulary of 5, and is thus the least diverse dataset in this thesis.

In their original paper, "High-Quality and Reproducible Automatic Drum Transcription From Crowdsourced Data", they state that their spectrograms are stored as log-magnitude, log-frequency spectrograms [61]. This contradicts information from where they distribute their dataset, where they state that *"The data as mel-scale spectrograms."* [60]. Through manual verification and approximation of the original waveforms, we propose that this latter information is an error, and confirms that the spectrograms are indeed stored as logarithmically filtered spectrograms, following the same format used by Vogl et. al [55] and what is otherwise used in this thesis. Is this enough justification? Should I provide more information? If so, what exactly?

The dataset contains 245 hours of music over 2924 tracks, and was accessed through Zenodo [60].

4.4.1 Mapping

This dataset is distributed already on a 5-instrument vocabulary fitting the one used in this thesis. Therefore, there was no need to remap the annotations; the dataset was used "as is".

4.5 SADTP

The SADTP (Small Automatic Drum Transcription Performance) dataset is a novel dataset introduced in this thesis. It is a small dataset comprised of 16 songs with corresponding MIDI transcriptions.

The *performance* name alludes to the transcription being recorded live while listening to the songs on playback, with only minor post-processing. The transcriptions were recorded on a Roland TD-11 electric drumset, recording the MIDI perfomance to Apple's Garageband DAW, and extracting them to separate MIDI files. This comes with a similarly to E-GMD, as this dataset also was recorded in a semi-manual nature, which opens the possibility for slight, human induced errors. The magnitude of such errors are however not known, but we speculate that it is small but not insignificant.

This dataset stands out, as it is the only one in this thesis not used for training. Its sole purpose is for cross-dataset evaluation, and to provide information on the generalization ability for models trained on data from other sources.

The dataset contains 1.08 hours of music, which can be split into 977 non-overlapping 4 second datapoints (includes zero padding certain pieces for even partitioning), accessed through GitHub [18].

4.5.1 Mapping

Similarly to E-GMD, the SADTP dataset stores its annotations in MIDI files recorded from a Roland TD-11 electric drumset. Hence, it applies the same Roland MIDI mapping given in Table 4.2 making it suitable for a 5-instrument vocabulary.

4.6 Summary

Dataset	Duration (h)	Number of tracks	Melodic	Synthetic
ENST+MDB	1.37	87	✓	×
E-GMD	444.5	45,537	×	✓
Slakh	115	1709	✓	✓
ADTOF-YT	245	2915	✓	×
SADTP	1.08	977	✓	×

Table 4.4: Comparison of each dataset's characteristics including total duration and number of tracks. Melodic datasets denote being a part of DTM, otherwise they are DTD. Synthetic datasets contain music which is synthesized digitally, with an often automatic generation scheme [62].

Dataset	Train	Validation	Test
ENST+MDB	896	236	152
E-GMD	324,266	49,424	48,571
Slakh	80,662	16,643	9963
ADTOF-YT	134,332	28,984	18,650
SADTP	×	×	977

Table 4.5: Comparison of the sizes of each dataset's different splits. Each dataset contains sequences of size 400 (corresponding to 4 second audio clips). These only summarize the direct size of each dataset, but does not take specific dataset augmentation into account, e.g. oversampling a MIDI sequence with differently sounding synthesized instruments.

In Table 4.4 we have a comprehensive summary of the different aspect of each of the dataset, as well as presenting them in a comparable format. These dataset span a good range of different characteristics, which could prove beneficial for training generalizing ADT models. Table 4.5 explicitly shows us the size differences between these datasets, and will be crucial once we start comparing performances across datasets.

Chapter 5

Methodology

In this thesis we perform 2 different studies. Although they differ in both intention and result comparison, their dataset preparation and model selection pipeline remains identical.

5.1 Data Preparation

As mentioned, the different datasets are distributed in differening formats. A few transformations are done to unify them all as PyTorch datasets.

Due to the preprocessed nature of the ADTOF-YT dataset, the unification is trivial and simply denotes a transformation from a stored TensorFlow dataset to a PyTorch dataset. Otherwise it is kept "as is".

5.1.1 Audio Files

The others are however distributed in the Waveform Audio File Format (with the suffix *.wav*) or using the Free Lossless Audio Codec (with the suffix *.flac*). Both of these formats are loaded using the PyTorch library `Torchaudio` and converted to monophonic format through meaning over each side's waveform. If any datasets contain distinct drum and accompaniement audio (like e.g. ENST-Drums), these are additively mixed together.

After each track is loaded into a waveform, a zero-padding is added to the end of each sequence allowing for even partitioning into 4 second partitions. Then we turn the track into a spectrogram with 2048 fft's, and a window length of 2048. By keeping the hop length equal to the sampling rate divided by 100, the resulting spectrogram's timesteps, or frames, represent a 10ms window of the original waveform. As each sequence consists of 4 seconds of audio, all resulting sequences will have a sequence length of $T = 400$ frames.

After this, a filterbank is computed by generating 12 normalized logarithmically spaced filters, centered at 440Hz, and bounded over the interval [20Hz, 20,000Hz]. Applying this filterbank as a simple matrix multiplication over the spectrogram results in a logarithmically filtered spectrogram with $D_{\text{STFT}} = 84$ number of frequency bins. Lastly, we turn it into a log-spectrogram by applying a $\log_{10}$ operation to each cell, following an addition of 1 (preventing $\lim_{x \to 0} \log_{10}(x) = -\infty$ situations).

5.1.2 Annotations

The annotations are either distributed in specific formats as text files (with the suffix *.txt*), or in MIDI files (with the suffix *.mid* or *.midi*), each one having a different transformation into a sequence of instrument onset probabilities. Although more specific information might be contained in the datasets, such as velocity, I solely utilize the time and specific drum-instrument for each onset.

The datasets declaring onsets in text files (ENST-Drums and MDB-Drums) follow a similar format, storing onsets on separate lines, each one containing the time in seconds for the onset, and its respective instrument ID, separated by a space or tab respectively. To convert this into the onset probability sequence, we convert the time into a timeframe index by turning the time into milliseconds, dividing by 10 to group into 10ms intervals, and rounding to the nearest integer. After this, we map the instrument ID into its respective class, and set that specific (timeframe, class) cell value to 1.

The data given in MIDI format, the annotations are parsed using the library `Partitura`, and loads the information into an array of MIDI events called *notes*. These *notes* contain information for each event, importantly time, pitch, and velocity. These events are very thorough, but strict instrument onsets can be isolated by restricting our view to notes with a non-zero velocity. Instruments are denoted by the event's pitch, and a mapping is done from each pitch to a respective class. The time is converted identically

to the loading of the text annotations, turning them into timeframe indices. At last, we also here set each specific registered onset (timeframe, class) cell to 1.

In addition to this, we apply a *target widening* step, setting values in timeframes adjacent to an instrument onset with a lower weight, equal to 0.5. These additional neighbouring *soft labels* have shown to be benifical in countering sparsity in our labels following multiple works on beat transcription [29, 61].

5.1.3 Splitting and Storing

These spectrogram/onset sequence pairs are stored together in PyTorch's TensorDatasets, separated into each track's respective train/validation/test split, and stored into PyTorch pickle files (with the suffix *.pt*). By doing all this preprocessing in advance, minimal preprocessing has to be done during runtime, increasing the efficiency of training.

5.2 Preprocessing

Most of the preprocessing is done during the data preparation step, however there are some remaining. Most importantly, a given model computes the mean and standard deviation of its training dataset, and uses these parameters to standardize its input data before prediction during runtime.

Data normalization like this has been shown to increase the speed and stability of convergence during training and, in summary, producing models which better generalize to unseen data. Although the specific benefits depend on the normalization technique used, the general consensus is that normalization in itself is beneficial in machine learning, hence their ubiquitous use in state-of-the-art models [28].

Another preprocessing step motivated by ADT specific methods is the use of infrequency weights, which framewise weighs the loss based on the instrument onsets that are present at each frame. These weights are precomputed from the training dataset, and are, for each instrument, computed by what Cartwright and Bello call *"the inverse estimated entropy of their event activity distribution"* [11]. Although they apply this to account for sparsity in data along different tasks, Zehren et al. [61] applied it to give more weight to infrequent instruments.

These weights are computed by calculating the probability of an instrument i appearing $p_i = \frac{n_i}{T}$ as the total number of onsets n_i divided by the total number of timesteps T. With this probability we compute its inverse entropy, giving us our final weights $w_i = (-p_i \log p_i - (1 - p_i) \log 1 - p_i)^{-1}$. Note that our probability computation differs from the work of Cartwright and Bello, as we do not divide by the number of instruments [11].

Should I mention anything on how the entropy is symmetric over 0.5? Such that a probability over 0.5 would lower our weights again, but how that is not a problem in ADT due to instrument onset inherently being sparse?

5.3 Training

As mentioned, the ADT task could be thought of as a sequence labeling task, where we each timeframe could have several instrument onsets present, taking the form of a 0 if an instrument is not present, and a 1 if it is. A natural loss function for this, where each value is handled as a separate independent probability distribution is the binary cross-entropy loss. Due to the numerical instability which can appear by applying a sigmoid activation function to our logits before computing the loss, we instead output our logits directly and utilize PyTorch's `BCEWithLogitsLoss` loss function, as recommended in their documentation Do I need to cite this?. It increases the numerical stability by taking advantage of the log-sum-exp trick, increasing numerical precision by avoiding underflow or overflow problems followed by significantly small or large input values.

With the choice by what to use, Adam was considered. This is an optimizer so ubiquitously used within the field of deep learning that when questioned with what optimizer to generally use authors like Sebastian Ruder state that *"., Adam might be the best overall choice* [43]. However contrary to this, the Adam optimizer has been shown to contain some issues, like its coupling of the weight decay term inside its gradient-based updates. Due to this, the choice instead fell on AdamW, a modified Adam implementation decoupling weight decay in whole from the gradient-based updates, and displaying a better ability to generalize. [32, 8, 34]

It was observed during training that the magnitude of the loss values could vary greatly and often displayed a tendancy to explode. To counteract this observation, we clip the gradients with a maximum norm set to 2. This addition significantly lowered the observed chance of exploding gradients occuring.

Another addition which is frequenct in other ADT works is the use of a learning rate scheduler. A learning rate scheduler keeps track of recent validation loss values, and if the minimum loss achieved plateaus (meaning it stop decreasing) for a certain number of epochs, the learning rate gets reducing by a given factor. In this thesis we reduce the learning rate by a factor of 5 if we observe 5 epochs of plateauing, with no improvement to our minimal validation loss [13, 61]. We also keep track of the general count of epochs since validation loss last improvement and perform an early stop if we ever observe 15 epochs without improvement.

5.4 Postprocessing

As mentioned, the model outputs a sequence of activation values, a 2 dimensional matrix with values on the interval $(0, 1)$ interpreted as the model's confidence in an instrument onset being present per frame. This can be utilized directly when computing our loss during training, however it is a difficult format to work with when talking about general performance, due to it rather representing a continuous confidence rather than discrete predictions. To suit this purpose, additional postprocessing is performed on the output.

First, we apply the aforementioned peak picking algorithm to isolate peaks in the model's onset confidence, intuitively being frames where the model is most confident in an instrument onset happening [6, 56]. Afterwards, we count a predicted onset if the given peak has a value larger or equal to 0.5. From this, it is trivial to compare predicted onsets with actual onsets, by greedily iterating our output sequence from the beginning, counting a prediction as a true positive if it happens within a 5 frame (50 ms) interval of a true onset, false positive if it happens outside such an interval, or false negative if a true onset happens with no prediction within said interval.

Could be useful showing transformation from output, to peak picked, to correct vs incorrect prediction.

These TP, FP, FN predictions are then added together and used to compute the previously mentioned micro F1-score. For additional analysis, we also compute and store instrumentwise F1-scores.

5.5 Model Selection

For model training and selection we utilize the RayTune library [33]. It simplifies the training of models by allowing us to input a training function, which metric to optimize for, hyperparameter spaces and search strategy, and additional configs. This simplifies our training, and through per-epoch reporting, it handles both checkpointing of model weights and best performing model selection for us.

Through RayTune, we train 15 different models (25 solely for the smallest dataset ENST+MDB), with hyperparameters tuned through bayesian optimization (see the next section 5.6). Each model is also ran for at most 100 epochs. As mentioned, we perform an early stop if performance stops increasing. We utilize PyTorch's dataloaders for iterating the datasets, with a batch size of 128. And using the AdamW optimizer, as previously mentioned.

During each epoch of training, we evaluate the model on the training dataset's corresponding validation data. After training, the model with the highest validation Micro F1-score is selected. Due to the pre-split nature of most of our datasets, we utilize hold-out validation, with separate train, validation and test datasets. This is not only very common within ADT [54, 58, 13], but throughout the whole field of deep learning. Raschka mentions that *"The holdout method is inarguably the simplest model evaluation technique"* [41], and might be one of the reasons for its popularity. Then, we estimate its performance on unseen data through evaluating it on the test dataset. At last, said model is stored together with its corresponding weights, training config, and metrics.

As just mentioned, every selected model is evaluated on test splits of the datasets. Although these test splits are traditionally reserved for a final performance evaluation, acting as a model's general performance on "unseen" data, they are here used to compare either architectural or dataset factors (respective to each study), and analysing how each of these controllable factors could be beneficial towards a model's generalization ability. No model selection nor hyper parameter tuning is done based on test evaluation, maintaining the validity of these performances as estimated performance on "unseen" data.

5.6 Hyperparameter Tuning

5.6.1 Search Strategies

There are several hyperparameters that are tuned for each model. One of the most thorough ways to tune these would be through a grid-search regime. Then, one would train and evaluate each possible hyperparameter combination, to find the best performing. However, this grows out of proportion fast, with an exponentially growing number of combinations based on the number of hyperparameters and their values, making it computationally expensive. Another way would be to use a random-search regime, where one randomly picks the combination of hyperparamters. This regime sacrifices hyperparameter combination coverage by increasing computational efficiency. The drawback with this approach however is its reliance on probability. During a given training trial, we might be unlucky with all our randomly selected hyperparameter combinations, leading to a suboptimal performing model.

A combination of these two regimes, almost utilizing "the best of both worlds" would be to use a bayesian optimization regime. It chooses its hyperparameter combinations in a random fashion, like random-search, but also uses previously trained models' performance values to "intelligently" perform later choices, focusing in on promising combinations. In this way, we reap the rewards of the random-search regime's computational efficiency, while keeping some of the thoroughness from the grid-search regime.

We utilize RayTune's implementation of `OptunaSearch`, a hyperparameter searching regime which uses Optuna, an automatic hyperparameter optimization software framework based on bayesian optimization [3]. It has shown to be efficient in finding a good tradeoff between computation and performance, with authors like Shekhar et al. performing benchmarks on neural networks and showiung that *"The performance score of Optuna is the highest for all datasets"* [45].

How much do I need to cite here?

5.6.2 Hyperparameters

RayTune allows one to easily declare the search space for each of the hyperparameters. Every architecture-specific hyperparameter is chosen based on a random choice. In addition to this, we also tune the optimizer-specific learning rate and weight decay using

a logarithmically uniform random sample. The specific spaces can be found below, in Table 5.1.

Hyperparameter	Search Space
Learning Rate	Log-uniform over $[1 \cdot 10^{-4}, 5 \cdot 10^{-3}]$
Weight Decay	Log-uniform over $[1 \cdot 10^{-6}, 1 \cdot 10^{-2}]$
Architecture-specific Hyperparameters	Random choice over each

Table 5.1: The search space for each of the different hyperparameters. The architecture-specific hyperparameters can be found under each architecture in the Architecture chapter 3

Chapter 6

Architecture Study

This study's main purpose is to figure out how suited each architecture is for Automatic Drum Transcription (ADT), more specifically Drum Transcription in the Presence of Melodic Instruments (DTM) tasks. This could help us figure out which architecture is superior for ADT, if there are any similarities between architectures who perform similarly well, or if there are any architectures who perform poorly.

6.1 Methodology

For each of the different datasets, we train several different models on the respective training dataset, and model select the best performing model on the respective validation dataset. At last we test said selected model on that dataset's respective test split. As a result, we are left with performance measures on unseen data from the same distribution as those each model was trained on. In other words, the model evaluated on a test split is strictly trained on that dataset's respective train split. This will give us a good intuition into each architecture's ability to learn the task of ADT and could help us estimate their generalization ability.

6.2 Results

Architecture	ENST+MDB	E-GMD	Slakh	ADTOF-YT
Recurrent Neural Network	0.67	**0.90**	0.86	**0.96**
Convolutional Neural Network	0.78	0.88	0.83	0.84
Convolutional Recurrent Neural Network	**0.81**	**0.90**	**0.90**	0.93
Convolutional Transformer	0.77	0.89	0.88	0.95
Vision Transformer	0.54	0.89	0.88	**0.96**

Table 6.1: The test Micro F1-score for each architecture, strictly trained and tested on that same dataset. The performances which are **bolded** represent the highest F1-score, and thus best performance, for that respective dataset.

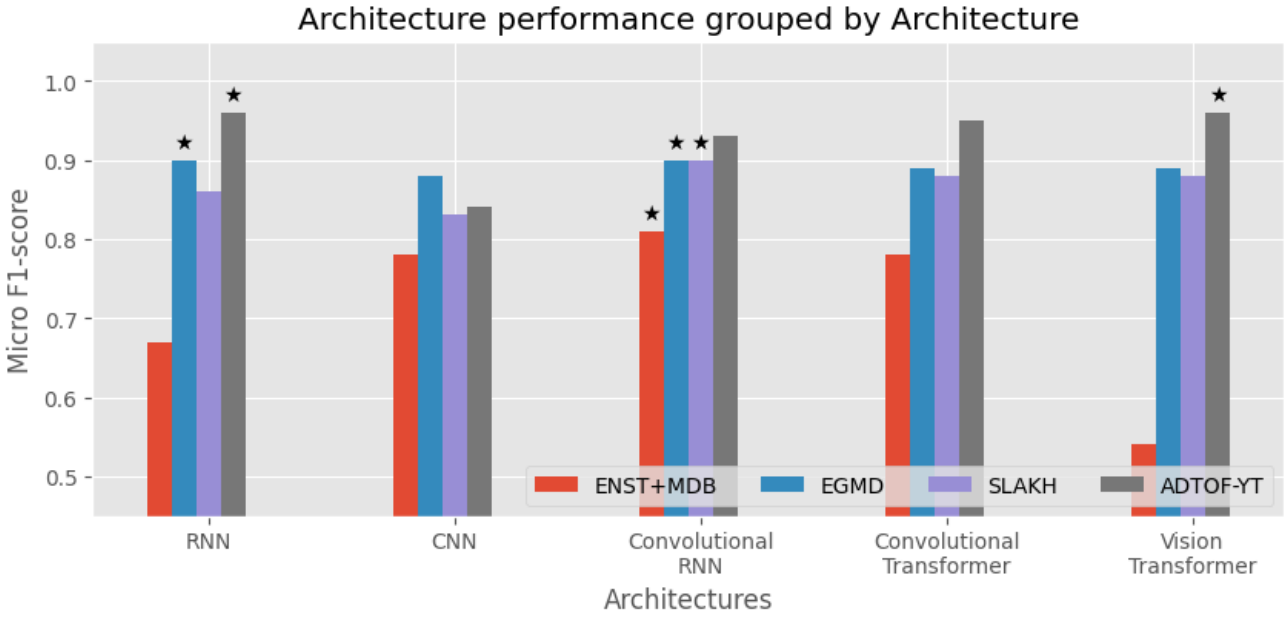

Figure 6.1: Comparison of test Micro F1-scores for each dataset across the different architectures. Bars marked with a ($\star$) indicate the best performing architecture for each respective dataset.

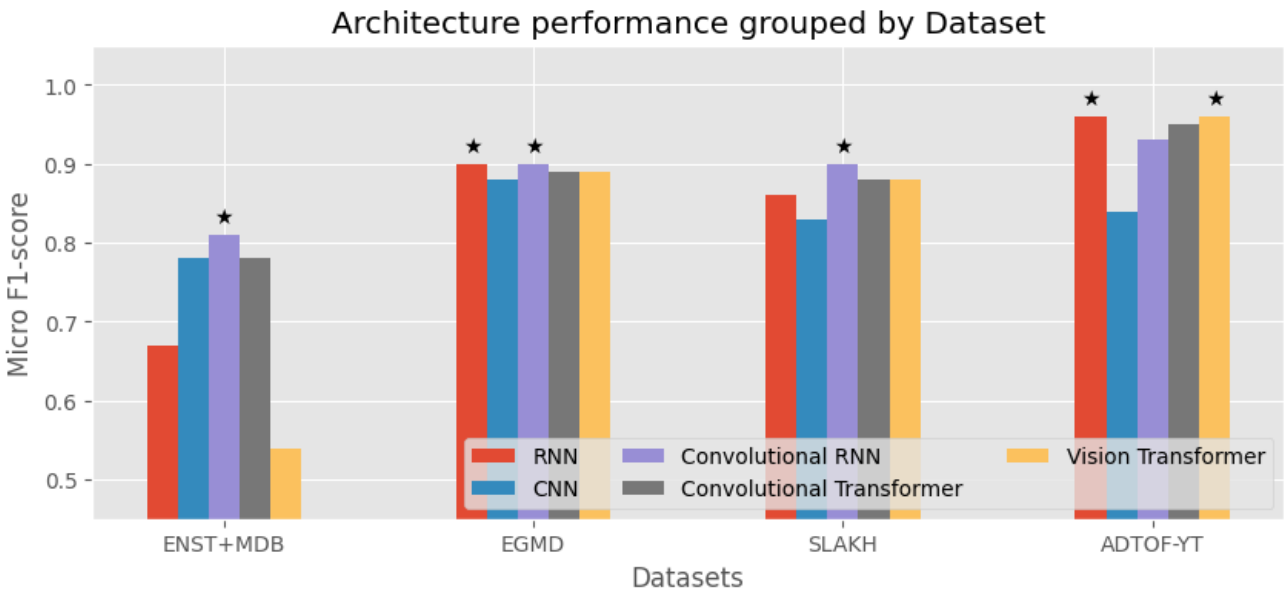

Figure 6.2: Comparison of test Micro F1-scores for each architectures across the different dataset. Bars marked with a ($\star$) indicate the best performing architecture for each respective dataset.

6.3 Discussion

The results from the architecture study, summarized in Table 6.1 and Figures 6.1 and 6.2, indicate that there is no single superior architecture for ADT. Instead, performance is variant on the properties of each dataset, as well as the inherent inductive biases of each architecture.

Firstly, it is evident that there does not seem to be one superior architecture when it comes to ADT. There does not exist a single architecture outperforming the others across all the datasets, and the different architectures often share similarly high performances on most of the datasets.

However, the convolutional recurrent neural network demonstrates the highest Micro F1-score on three of the four datasets (namely ENST+MDB, E-GMD and Slakh). It also provides a high, but not exceptional F1-score on the fourth (ADTOF-YT). The consistency of its high performance across the different characteristics of each dataset suggests that it is able to handle a wide variety of ADT tasks, and that its performance may be high independent on the training dataset's size and complexity. In other words, it displays properties of being a architecture highly suitable for ADT tasks. One could speculate

that this suitability is provided by the strong inductive bias from the combination of convolutions and recurrent units, allowing for both initial spatial feature extraction as well as short-range temporal modelling.

The convolutional neural network displays a moderate performance across all datasets. It shows adequate performance on the smallest dataset (ENST+MDB), having the second highest F1-score. However, across all others (E-GMD, Slakh and ADTOF-YT) it displays the lowest. This relatively poor overall performance seems to indicate that the convolutional neural network currently is an inferior architecture for ADT tasks, unless dataset size is limited, in which it could provide a viable architecture. It also shows the importance of explicitly leveraging the temporal dependencies within ADT, as solely relying on the inductive bias of the convolutions didn't prove sufficient here.

This importance seems to be strengthened by the performance of the purely recurrent neural network, which has a surprisingly high performance across the datasets. In a way, it displays the opposite behaviour compared to the CNN, displaying inadequate performance for the smallest dataset (ENST+MDB), but a relatively high performance for two of the others (E-GMD and Slakh), sharing the crown for the highest F1-score on the last (ADTOF-YT). The first dataset's low performance suggests that the inductive bias of the recurrent layers are "weaker" than the ones from convolutions, relying on a larger amount of data to accurately learn the task. Notably, it significantly outperforms the CRNN on the last dataset, hinting that the inductive bias of the convolutions might "overpower" the model, proving so strong that it ends up hindering its performance.

The convolutional transformers exhibits performance comparable to something in between the RNN and CRNN, and shows what can be described as the most consistent relatively high performance across all the four datasets. It puts itself just under the CRNN for the first three datasets (ENST+MDB, E-GMD and Slakh), and a good step over for the last (ADTOF-YT). This does seem to indicate that the transformer models provide enough of a temporal inductive bias to accurately perform well on ADT tasks, however it might also hint that the long range dependencies more easily modelled with attention layers are not as important as the accurate short range aggregation from the recurrent layers on most ADT datasets. Although by the performance displayed, the convolutional transformer could be chosen as a viable architecture for ADT.

At last we have the Vision transformer, which performs similar to the RNN. For the smallest dataset (ENST+MDB) it displays a subpar performance with by far the lowest F1-score. For the two middle ones (E-GMD and Slakh) it exhibits a relatively average

performance. For the last dataset (ADTOF-YT) it shows excellent performance, having an identical F1-score to the RNN. This seems to agree with the previous conclusion that the inductive bias of the convolutional layers are "strong", putting a higher responsibility on a large amount of quality data when omitting it. The significantly poor performance on the small dataset might also indicate that the attention layers and transformer blocks come with an even "weaker" inductive bias than the recurrent layers, hightening the reliance on data amount. This aligns well with existing literature, where vision transformers generally require extensive training data or pre-training on larger datasets to obtain optimal performance [17].

In summary, our results strongly indicate that the Convolutional Recurrent Neural Network is the current most suited architecture for ADT tasks across different datasets. However, further research should be done into both the Vision Transformer and Recurrent Neural Network, gauging how these models scale with larger datasets.

Is it relevant to talk about future research here or should it be in the final conclusion?

Chapter 7

Dataset Study

This study's main purpose is to figure out how each dataset, and their characteristics, combined can influence a model's performance, both on- and OOD. This could help us figure out how to utilize and combine current datasets, or how to intelligently construct future datasets, to optimize models' performance and generalization ability for Automatic Drum Transcription (ADT) and Drum Transcription in the Presence of Melodic Instruments (DTM).

7.1 Methodology

We train convolutional recurrent neural networks, the best performing architecture from the first study, over several different combinations of the first four datasets ENST+MDB, E-GMD, Slakh, and ADTOF-YT. Datasets are combined as a union, where one epoch over the combined dataset would be equivalent to one epoch of both datasets separately. Each model is also evaluated on the remaining datasets, in addition to the SADTP dataset.

To get a comprehensive overview over how the different datasets could complement or contrast eachother's performances, without doing redundant experiments with diminishing returns, we select and train on a subset of all possible dataset combination. More specifically, we select 10 different combinations of datasets in such a way that we could extract valuable information about how their characteristics affect the resulting model's generalization ability.

7.2 Results

Training Datasets	ENST+MDB	E-GMD	Slakh	ADTOF-YT	SADTP
ENST+MDB	0.81	0.59	0.53	0.60	0.42
E-GMD	0.55	**0.90**	0.44	0.42	0.28
Slakh	0.80	0.73	0.90	0.59	0.48
ADTOF-YT	0.84	0.69	0.65	0.93	0.60
ENST+MDB + Slakh	0.84	0.73	0.90	0.63	0.48
ENST+MDB + ADTOF-YT	0.86	0.70	0.63	0.94	**0.62**
Slakh + ADTOF-YT	0.86	0.72	**0.90**	**0.97**	**0.62**
ENST+MDB + Slakh + ADTOF-YT	**0.88**	0.74	0.89	0.95	0.61
E-GMD + Slakh + ADTOF-YT	0.85	**0.90**	0.89	0.93	0.61
ENST+MDB + E-GMD + Slakh + ADTOF-YT	0.87	0.89	**0.90**	0.93	**0.62**

Table 7.1: The Micro F1-score for a convolutional recurrent neural network trained over different combination of datasets, and tested on all datasets. The performances which are **bolded** represent the highest overall F1-score for the given dataset. Cells which are coloured light blue represent OOD evaluations. Values which are underlined represent the highest possible F1-score of OOD evaluations for the respective dataset.

Model	ENST+MDB	E-GMD	Slakh	ADTOF-YT
Slakh + ADTOF-YT	0.86	0.72	**0.90**	**0.97**
ENST+MDB + Slakh + ADTOF-YT	**0.88**	0.74	0.89	0.95
E-GMD + Slakh + ADTOF-YT	0.85	**0.90**	0.89	0.93
OaF-Drums [9]	0.77/×*	0.83	×	×
MT3 (mixture) [19]	×	×	0.76	×
YPTF.MoE+M [13]	0.87/×**	×	0.85	×
ADTOF-RGW + ADTOF-YT [61]	0.78/0.81***	×	×	0.85

Table 7.2: The Micro F1-score of selectively chosen convolutional recurrent neural networks from this dataset study, compared with best performing models from other literature over the different test sets. The performances which are **bolded** represent the highest overall F1-score for the given dataset for all compared models. Cells which are coloured light blue represent OOD evaluations. (*) Callender et. al only tests ENST-Drums, and strictly on isolated drum stems handling it as a DTD task, on a different test split than ours [9]. (**) Chang et. al only tests ENST-Drums and tests on a different test split than ours [13]. (***) Zehren et. al tests on ENST-Drums and MDB-Drums separately and tests on different test splits than ours [61].

Add E-GMD to other literature and discuss it in discussion.

7.3 Discussion

The results from the dataset study, summarized in Table 7.1 highlight important insights regarding how dataset composition could affect and impact model generalization. These results clearly demonstrate that strategically combining ADT datasets improve both on- and Out-Of-Distribution (OOD) generalization.

Firstly, it is evident that each dataset exhibits great performance on test splits which distribution overlaps with their own training dataset, but a lesser performance when testing OOD. There are several hypotheses one could propose for why this is the case, however one plausible one is that the datasets are characteristically varied enough in such a way that they each contrast eachother. This could give rise to this generalization penalty we see and explains the drop in performance. This generalization penalty also seem to be very uniform for different datasets, except one (E-GMD) which is talked about in the following paragraph. The best performing OOD evaluations all lie in the Micro F1-score range of 0.6 to 0.7, except for the smallest dataset (ENST+MDB) which displays an F1-score of around 0.87. Interestingly, the best performing OOD evaluation for ENST+MDB is significantly close to its best performance overall. As we know, in contrast to the other datasets ENST+MDB consist of high quality, really precise DTM data. A hypothesis is that this transcription quality of ENST+MDB more easily show the strength of the other model's, which might be hidden in the lesser quality of other test splits.

Another important observation, as mentioned, is that of E-GMD. As known, this dataset differs from the others by being a Drum Transcription of Drum-only Recordings (DTD) dataset, instead of a DTM one. Despite its large size, it is the worst at generalizing to the others, and it is the worst performing model on all other datasets. However, it is worth to note that it also displays the best performance on itself. This helps reinforcing the differences between the tasks of DTD and DTM, showing how model's trained on differently tasked datasets are not directly applicable interchangeably.

Building on this information we could inspect the opposite relationship. Notably, the other model's OOD evaluation on E-GMD is comparable to that of their OOD evaluations on themselves. This also helps strengthen the hypothesis that these tasks, DTD and DTM, display a complexity hierarchy where one seems to build upon the other. In other words, DTM might be a more difficult transcription task than DTD, but in return, a model trained for DTM might be sufficient for DTD tasks, and exhibits a zero-shot generalization ability which does not hold for the opposite.

One results that might merit discussion is the correlation between high ADTOF-YT performance and high SADTP performance. A high F1-score in the ADTOF-YT test split is accompanied by a relatively high OOD F1-score for SADTP. Due to the crowdsourced nature of ADTOF-YT, and the SADTP consisting of contemporary and public music tracks, there might exist some overlap in their distribution which could explain this behaviour. This could also be explained by ADTOF-YT being a dataset containing characteristics which, when trained on, give the models a better ability at generalizing onto other ADT datasets, as these models trained on ADTOF-YT also perform well OOD on the other datasets.Maybe show correlation graph?.

An insightful observation is that the best performance of all DTM datasets from the architecture study, sees themselves beat by models trained on a combination of datasets. This is an important observation and shows that model performance often is enhanced by expanding the dataset size and variation. However, building upon what we discussed in the last paragraph, it is important to understand the differences between DTD and DTM datasets. As we observe, the best Micro F1-score for DTD dataset comes from training data solely of DTD data. The same holds the other way, in that the best performances on DTM datasets happen when solely trained on DTM data.

Secondly, take a look at the model trained on the largest amount of data, the combination of all datasets. This model exhibits superior performance across the board, however interestingly it does not achieve the highest Micro F1-score on any, though being very close. This strengthens the hypothesis that expanding the amount data will heighten generalization ability for a given model, and agrees with the current consensus that training dataset size is vital to achieving an optimally generalizing model [61, 29]. For future works it would be interesting to analyze how valuable data augmentation would be for ADT, and if it affects generalization positively.

Lastly, in Table 7.2 we compare our best performing model's with other literature and giving remarkable insight. Note that not much literature exist with models evaluated on thsee datasets. Gardner et. al's MT3 [19] and Chang et. al's YPTF.MOE+M [13] are general AMT models, predicting several different instruments in addition to drums. They also present using the Offset F1-score terminology, equivalent to our Micro F1-score (also used by Zehren et. al [61]). Zehren et. al's ADTOF-RGW + ADTOF-YT is a DTM model, similar to that of ours. Observably, we note that our model's outperform the others' by a singificant amount on both the Slakh and ADTOF-YT datasets. Due to the splits and datasets being a bit different for ENST+MDB, a direct comparison is a bit more difficult, but one could reason that our model's performance is comparable

to that of YPTF.MOE+M [13]. By comparing with results from other literature, we can put our model's performance into perspective and strengthens the argument that our combinations could in fact provide valuable advancements for the state-of-the-art in ADT and DTM.

In summary we can conclude that our results strongly indicate and strengthens our hypothesis that combining different datasets with variable characteristics, will help a model perform better on ADT tasks, and could make them generalize better towards Out-Of-Distribution (OOD) datasets. We also note that applicability of different ADT tasks follow a relationship, where DTM datasets generalize better on DTD datasets than the inverse.

I'm unsure if I've discussed enough (or too much of certain aspects) around the results presented in 7.1?

Chapter 8

Conclusion

This thesis set out to investigate how to achieve optimal performance on Automatic Drum Transcription (ADT) tasks, specifically Drum Transcription in the Presence of Melodic Instruments (DTM). Through exploring and analysing how different deep learning architectures and different dataset compositions affect model performance.

In the first study we trained and compared the performance of five deep learning architectures, namely a recurrent neural network, convolutional neural network, convolutional recurrent neural network, convolutional transformer, and vision transformer, each trained and tested over different dataset. The best architecture showing the strongest overall performance was the convolutional recurrent neural network, which performed the best performance on three of the four datasets. However, we also note that more experimentation should be done with the recurrent neural network, and the vision transformer, specifically on larger dataset, which both identically achieved the best performance on the last dataset.

In the second study we explored how combining datasets of different characteristics influences a model's generalization ability. We found out that combining datasets substantially improves both on- and Out-Of-Distribution (OOD) performance. However, we also identified that carefully constructing these datasets is vital, but that an increase in data amount generally leads to better performance.

Reflecting back on the questions raised in the introduction.

1. **Which deep learning architecture is the best suited for solving a task like DTM?**

   The Convolutional Recurrent Neural Network consistently achieves superior performance on different ADT and DTM datasets.

2. **What makes a ADT dataset optimal by making models generalize for DTM?**

   Training models on large, DTM datasets, covering a wide spectrum of musical characteristics is the most important factor of generalizability when used to train a model suitable for ADT and DTM.

This thesis contributes not only by comparing different architectural choices for DTM tasks, but also through evaluating different ADT datasets' affection on model generalization, this on both public datasets, and on a novel dataset composed and transcribed for this thesis, SADTP. These contributions could give valuable insight into ADT by emphasizing both choices on deep learning architectures as well as construction of datasets.

Future work should expand upon this research by performing rigorous architecture analyses over larger datasets, as well as optionally inspecting how the different number of drum instrument classes both affect and perform over these different datasets.

Lastly, this thesis demonstrates that deep learning model's can exhibit great performance on ADT and DTM tasks achievable through intelligent choices in both architecture and dataset.

List of Acronyms and Abbreviations

[CLS] Class Token.

ADT Automatic Drum Transcription.

AMT Automatic Music Transcription.

BiRU Bidirectional Recurrent Unit.

CC Crash Cymbal.

CC+RC Crash and Ride.

CNN Convolutional Neural Network.

CRNN Convolutional Recurrent Neural Network.

DAW Digital Audio Workstation.

dB Decibel.

DFT Discrete Fourier Tranform.

DNN Deep Neural Network.

DSC Drum Sound Classification.

DTD Drum Transcription of Drum-only Recordings.

DTM Drum Transcription in the Presence of Melodic Instruments.

DTP Drum Transcription in the Presence of Additional Percussion.

FFT Fast Fourier Transform.

FN False Negative.

FP False Positive.

GELU Gaussian Error Linear Unit.

GRU Gated Recurrent Unit.

HH Hi-Hat.

HT High Tom.

KD Kick Drum.

LLM Large Language Model.

LSTM Long Short-Term Memory.

LT Low Tom.

MIDI Musical Instrument Digital Interface.

MIR Music Information Retrieval.

MT Mid Tom.

NLP Natural Language Processing.

OOD Out-Of-Distribution.

RC Ride Cymbal.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

SD Snare Drum.

STFT Short-time Fourier Transform.

TN True Negative.

TP True Positive.

TT Tom-Tom.

ViT Vision Transformer.

Bibliography

[1] *Fundamentals of Telephony*. United States, Department of the Army, 1953.
URL: https://books.google.no/books?id=8nvJ6qvtdPUC.

[2] Td-17: Default (factory) midi note map, 2022.
URL: https://support.roland.com/hc/en-us/articles/360005173411-TD-17-Default-Factory-MIDI-Note-Map.

[3] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[4] Pras Amandine and Guastavino Catherine. Sampling rate discrimination: 44.1 khz vs. 88.2 khz. *Journal of the Audio Engineering Society*, (8101), may 2010.

[5] Rachel M Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. Medleydb: A multitrack dataset for annotation-intensive mir research. In *Ismir*, volume 14, pages 155–160, 2014.

[6] Sebastian Böck, Florian Krebs, and Markus Schedl. Evaluating the online capabilities of onset detection methods. In *International Society for Music Information Retrieval Conference*, 2012.
URL: https://api.semanticscholar.org/CorpusID:7379180.

[7] Sebastian Böck, Florian Krebs, and Gerhard Widmer. Joint beat and downbeat tracking with recurrent neural networks. In *ISMIR*, pages 255–261. New York City, 2016.

[8] Sebastian Bock, Josef Goppold, and Martin Weiß. An improvement of the convergence proof of the adam-optimizer, 2018.
URL: https://arxiv.org/abs/1804.10587.

[9] Lee Callender, Curtis Hawthorne, and Jesse Engel. Improving perceptual quality of drum transcription with the expanded groove midi dataset, 2020.
URL: `https://arxiv.org/abs/2004.00188`.

[10] Lee Callender, Curtis Hawthorne, and Jesse Engel. Expanded groove midi dataset, April 2020.
URL: `https://doi.org/10.5281/zenodo.4300943`. Accessed: 05.03.2025.

[11] Mark Cartwright and Juan Pablo Bello. Increasing drum transcription vocabulary using data synthesis. In *Proc. International Conference on Digital Audio Effects (DAFx)*, pages 72–79, 2018.

[12] Pragnan Chakravorty. What is a signal? [lecture notes]. *IEEE Signal Processing Magazine*, 35(5):175–177, 2018. doi: 10.1109/MSP.2018.2832195.

[13] Sungkyun Chang, Emmanouil Benetos, Holger Kirchhoff, and Simon Dixon. Yourmt3+: Multi-instrument music transcription with enhanced transformer architectures and cross-dataset stem augmentation. In *2024 IEEE 34th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2024.

[14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1724–1734. ACL, 2014. doi: 10.3115/V1/D14-1179.
URL: `https://doi.org/10.3115/v1/d14-1179`.

[15] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. ISSN 00255718, 10886842.
URL: `http://www.jstor.org/stable/2003354`.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages

4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423.
URL: https://aclanthology.org/N19-1423/.

[17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.
URL: https://arxiv.org/abs/2010.11929.

[18] Runar Fosse. Sadtp, 2025.
URL: https://github.com/RunarFosse/SADTP. Accessed: 16.06.2025.

[19] Josh Gardner, Ian Simon, Ethan Manilow, Curtis Hawthorne, and Jesse Engel. Mt3: Multi-task multitrack music transcription, 2022.
URL: https://arxiv.org/abs/2111.03017.

[20] Olivier Gillet and Gaël Richard. Enst-drums: an extensive audio-visual database for drum signals processing. In *International Society for Music Information Retrieval Conference (ISMIR)*, 2006.

[21] Olivier Gillet and Gaël Richard. Enst-drums: an extensive audio-visual database for drum signals processing, October 2006.
URL: https://doi.org/10.5281/zenodo.7432188. Accessed: 05.03.2025.

[22] Jon Gillick, Adam Roberts, Jesse Engel, Douglas Eck, and David Bamman. Learning to groove with inverse sequence transformations. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2269–2279. PMLR, 09–15 Jun 2019.
URL: https://proceedings.mlr.press/v97/gillick19a.html.

[23] Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram transformer, 2021.
URL: https://arxiv.org/abs/2104.01778.

[24] D. Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, 1984. doi: 10.1109/TASSP.1984.1164317.

[25] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2023.
URL: https://arxiv.org/abs/1606.08415.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.
URL: https://doi.org/10.1162/neco.1997.9.8.1735.

[27] Thomas Holz. Automatic drum transcription with deep neural networks. Master's thesis, Technische Universitaet Berlin (Germany), 2021.

[28] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization techniques in training dnns: Methodology, analysis and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(8):10173–10196, 2023. doi: 10.1109/TPAMI.2023.3250241.

[29] Yun-Ning Hung, Ju-Chiang Wang, Xuchen Song, Wei-Tsung Lu, and Minz Won. Modeling beats and downbeats with a time-frequency transformer. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 401–405, 2022. doi: 10.1109/ICASSP43922.2022.9747048.

[30] Fatemeh Jamshidi, Gary Pike, Amit Das, and Richard Chapman. Machine learning techniques in automatic music transcription: A systematic survey. *arXiv preprint arXiv:2406.15249*, 2024.

[31] Bijue Jia, Jiancheng Lv, and Dayiheng Liu. Deep learning-based automatic downbeat tracking: a brief review. *Multimedia Systems*, 25(6):617–638, 2019.

[32] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
URL: https://arxiv.org/abs/1412.6980.

[33] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training, 2018.
URL: https://arxiv.org/abs/1807.05118.

[34] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.
URL: https://arxiv.org/abs/1711.05101.

[35] 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. In *2019 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 45–49, 2019. doi: 10.1109/WASPAA.2019.8937170.

[36] Ethan Manilow, Gordon Wichern, Prem Seetharaman, and Jonathan Le Roux. Slakh2100, October 2019.
URL: `https://doi.org/10.5281/zenodo.4599666`. Accessed: 17.04.2025.

[37] Geoff Nicholls. *The Drum Handbook: Buying, maintaining, and getting the best from your drum kit*. San Francisco, CA: Backbeat Books, 2003.

[38] Martin Piszczalski. *A computational model of music transcription*. PhD thesis, USA, 1986. UMI order no. GAX86-21354.

[39] Martin Piszczalski and Bernard A Galler. Automatic music transcription. *Computer Music Journal*, pages 24–31, 1977.

[40] Colin Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching*. Columbia University, 2016.

[41] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning, 2020.
URL: `https://arxiv.org/abs/1811.12808`.

[42] Joseph Rothstein. Midi: A comprehensive introduction. volume 7, page 59. AR Editions, Inc., 1995.

[43] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.
URL: `https://arxiv.org/abs/1609.04747`.

[44] Jan Schlüter and Sebastian Böck. Improved musical onset detection with convolutional neural networks. In *2014 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 6979–6983. IEEE, 2014.

[45] Shashank Shekhar, Adesh Bansode, and Asif Salim. A comparative study of hyper-parameter optimization tools. In *2021 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, pages 1–6. IEEE, 2021.

[46] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[47] Carl Southall, Ryan Stables, and Jason Hockman. Automatic drum transcription using bi-directional recurrent neural networks. In *International Society for Music Information Retrieval Conference*, 2016.
URL: `https://api.semanticscholar.org/CorpusID:2891003`.

[48] Carl Southall, Chih-Wei Wu, Alexander Lerch, and Jason Hockman. Mdb drums: An annotated subset of medleydb for automatic drum transcription. 2017.

[49] Carl Southall, Chih-Wei Wu, Alexander Lerch, and Jason Hockman. Mdb drums, 2017.
URL: `https://github.com/CarlSouthall/MDBDrums`. Accessed: 22.04.2025.

[50] Oleg Starostenko and Omar Lopez-Rincon. *A 3D Spatial Visualization of Measures in Music Compositions*, page 127. 03 2019.

[51] Gilbert Strang. Wavelet transforms versus fourier transforms. *Bulletin of the American Mathematical Society*, 28(2):288–305, 1993.

[52] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016.
URL: `https://arxiv.org/abs/1609.03499`.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
URL: `https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf`.

[54] Richard Vogl, Matthias Dorfer, and Peter Knees. Recurrent neural networks for drum transcription. In *ISMIR*, pages 730–736, 2016.

[55] Richard Vogl, Matthias Dorfer, Gerhard Widmer, and Peter Knees. Drum transcription via joint beat and drum modeling using convolutional recurrent neural networks. In *International Society for Music Information Retrieval Conference*, 2017.
URL: `https://api.semanticscholar.org/CorpusID:21314796`.

[56] Richard Vogl, Gerhard Widmer, and Peter Knees. Towards multi-instrument drum transcription, 2018.
URL: `https://arxiv.org/abs/1806.06676`.

[57] Friedrich Wolf-Monheim. Spectral and rhythm features for audio classification with deep convolutional neural networks, 2024.
URL: `https://arxiv.org/abs/2410.06927`.

[58] Chih-Wei Wu, Christian Dittmar, Carl Southall, Richard Vogl, Gerhard Widmer, Jason Hockman, Meinard Müller, and Alexander Lerch. A review of automatic drum transcription. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(9):1457–1483, 2018. doi: 10.1109/TASLP.2018.2830113.

[59] Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10524–10533. PMLR, 13–18 Jul 2020.

URL: https://proceedings.mlr.press/v119/xiong20b.html.

[60] Mickael Zehren, Marco Alunno, and Paolo Bientinesi. Adtof datasets, November 2023.

URL: https://doi.org/10.5281/zenodo.10084511. Accessed: 23.10.2024.

[61] Mickaël Zehren, Marco Alunno, and Paolo Bientinesi. High-quality and reproducible automatic drum transcription from crowdsourced data. *Signals*, 4(4):768–787, 2023. ISSN 2624-6120. doi: 10.3390/signals4040042.

URL: https://www.mdpi.com/2624-6120/4/4/42.

[62] Mickaël Zehren, Marco Alunno, and Paolo Bientinesi. Analyzing and reducing the synthetic-to-real transfer gap in music information retrieval: the task of automatic drum transcription, 2024.

URL: https://arxiv.org/abs/2407.19823.

Appendix A

ENST+MDB Splits

	ENST-Drums	
Split	Drummer	Track
Train	**drummer_1**	**107_minus-one_salsa_sticks**
	drummer_1	**108_minus-one_rock-60s_sticks**
	drummer_1	**109_minus-one_metal_sticks**
	drummer_1	**110_minus-one_musette_brushes**
	drummer_1	**111_minus-one_funky_rods**
	drummer_1	**112_minus-one_funk_rods**
	drummer_1	**113_minus-one_charleston_sticks**
	drummer_1	**114_minus-one_celtic-rock_brushes**
	drummer_1	**115_minus-one_bossa_brushes**
	drummer_1	**121_MIDI-minus-one_bigband_brushes**
	drummer_1	**123_MIDI-minus-one_blues-102_sticks**
	drummer_1	**125_MIDI-minus-one_country-120_brushes**
	drummer_1	**127_MIDI-minus-one_disco-108_sticks**
	drummer_1	**129_MIDI-minus-one_funk-101_sticks**
	drummer_1	**131_MIDI-minus-one_grunge_sticks**
	drummer_1	**133_MIDI-minus-one_nu-soul_sticks**
	drummer_1	**135_MIDI-minus-one_rock-113_sticks**
	drummer_1	**137_MIDI-minus-one_rock'n'roll-188_sticks**
	drummer_1	**139_MIDI-minus-one_soul-120-marvin-gaye_sticks**
	drummer_1	**141_MIDI-minus-one_soul-98_sticks**
	drummer_1	**143_MIDI-minus-one_fusion-125_sticks**
	drummer_2	**115_minus-one_salsa_sticks**

...

Split	Drummer	Track
	drummer_2	**116_minus-one_rock-60s_sticks**
	drummer_2	**117_minus-one_metal_sticks**
	drummer_2	**118_minus-one_musette_brushes**
	drummer_2	**119_minus-one_funky_sticks**
	drummer_2	**120_minus-one_funk_sticks**
	drummer_2	**121_minus-one_charleston_sticks**
	drummer_2	**122_minus-one_celtic-rock_sticks**
	drummer_2	**123_minus-one_celtic-rock-better-take_sticks**
	drummer_2	**124_minus-one_bossa_sticks**
	drummer_2	**130_MIDI-minus-one_bigband_sticks**
	drummer_2	**132_MIDI-minus-one_blues-102_sticks**
	drummer_2	**134_MIDI-minus-one_country-120_sticks**
	drummer_2	**136_MIDI-minus-one_disco-108_sticks**
	drummer_2	**138_MIDI-minus-one_funk-101_sticks**
	drummer_2	**140_MIDI-minus-one_grunge_sticks**
	drummer_2	**142_MIDI-minus-one_nu-soul_sticks**
	drummer_2	**144_MIDI-minus-one_rock-113_sticks**
	drummer_2	**146_MIDI-minus-one_rock'n'roll-188_sticks**
	drummer_2	**148_MIDI-minus-one_soul-120-marvin-gaye_sticks**
	drummer_2	**150_MIDI-minus-one_soul-98_sticks**
	drummer_2	**152_MIDI-minus-one_fusion-125_sticks**
Validation	**drummer_3**	**126_minus-one_salsa_sticks**
	drummer_3	**127_minus-one_rock-60s_sticks**
	drummer_3	**128_minus-one_metal_sticks**
	drummer_3	**129_minus-one_musette_sticks**
	drummer_3	**130_minus-one_funky_sticks**
	drummer_3	**131_minus-one_funk_sticks**
	drummer_3	**132_minus-one_charleston_sticks**
	drummer_3	**133_minus-one_celtic-rock_sticks**
	drummer_3	**134_minus-one_bossa_sticks**
	drummer_3	**140_MIDI-minus-one_bigband_sticks**
Test	**drummer_3**	**142_MIDI-minus-one_blues-102_sticks**
	drummer_3	**144_MIDI-minus-one_country-120_sticks**
	drummer_3	**146_MIDI-minus-one_disco-108_sticks**
	drummer_3	**148_MIDI-minus-one_funk-101_sticks**

...

Split	Drummer	Track
	drummer_3	**150_MIDI-minus-one_grunge_sticks**
	drummer_3	**152_MIDI-minus-one_nu-soul_sticks**
	drummer_3	**154_MIDI-minus-one_rock-113_sticks**
	drummer_3	**156_MIDI-minus-one_rock'n'roll-188_sticks**
	drummer_3	**158_MIDI-minus-one_soul-120-marvin-gaye_sticks**
	drummer_3	**160_MIDI-minus-one_soul-98_sticks**
	drummer_3	**162_MIDI-minus-one_fusion-125_sticks**

Table A.1: Due to its small size and no explicit train/validation/test split for ENST-Drums existing, these are the respective splits for each ENST-Drums track in ENST+MDB.

	MDB-Drums
Split	Track
Train	**MusicDelta_Punk_MIX**
	MusicDelta_CoolJazz_MIX
	MusicDelta_Disco_MIX
	MusicDelta_SwingJazz_MIX
	MusicDelta_Rockabilly_MIX
	MusicDelta_Gospel_MIX
	MusicDelta_BebopJazz_MIX
	MusicDelta_FunkJazz_MIX
	MusicDelta_FreeJazz_MIX
	MusicDelta_Reggae_MIX
	MusicDelta_LatinJazz_MIX
	MusicDelta_Britpop_MIX
	MusicDelta_FusionJazz_MIX
	MusicDelta_Shadows_MIX
	MusicDelta_80sRock_MIX
Validation	**MusicDelta_Beatles_MIX**
	MusicDelta_Grunge_MIX
	MusicDelta_Zeppelin_MIX
	MusicDelta_ModalJazz_MIX
Test	**MusicDelta_Country1_MIX**
	MusicDelta_SpeedMetal_MIX
	MusicDelta_Rock_MIX
	...

Split	Track
	MusicDelta_Hendrix_MIX

Table A.2: Due to its small size and no explicit train/validation/test split for MDB-Drums existing, these are the respective splits for each MDB-Drums track in ENST+MDB.