University of Bergen
Department of Informatics

Automatic Drum Transcription:

Generalization via Architecture and Dataset

Author: Runar Fosse
Supervisor: Pekka Parviainen

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

August, 2025

Abstract

Automatic Drum Transcription (ADT), particularly in the form of Drum Transcription in the Presence of Melodic Instruments (DTM), remains a challenging task within the field of Music Information Retrieval (MIR) due to the complexity of recognizing and isolating drum events from polyphonic mixtures. Deep learning is the current approach for tackling this problem, but the roles of architectural design and dataset composition in supporting generalization remain underexplored.

This thesis investigates how different neural network architectures and dataset configurations affect ADT performance, both within-domain and under Out-of-Distribution (OOD) conditions. Two studies are presented. The first evaluates five architectures: Recurrent Neural Network, Convolutional Neural Network, Convolutional Recurrent Neural Network, Convolutional Transformer, and Vision Transformer, across four public ADT datasets. Results show that the Convolutional Recurrent Neural Network (CRNN) achieves the strongest overall performance, though the Recurrent Neural Network and Vision Transformer also perform competitively on larger datasets, warranting further investigation.

The second study examines how combining datasets with varying properties influences generalization. Models trained on diverse, especially crowdsourced, datasets exhibit stronger performance both within-domain and OOD. A novel evaluation dataset, SADTP, is introduced to support realistic zero-shot testing.

Together, these findings provide valuable insight for building generalizable ADT models. They highlight the importance of selecting robust architectures and constructing heterogeneous, task-aligned datasets to support strong transcription performance across real-world audio conditions.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Pekka Parviainen, for allowing me to pursue this topic for my master's thesis, and for his continuous support and guidance throughout the year.

I would also like to thank the University of Bergen, particularly the Department of Informatics, for providing computational resources such as *Birget*, which played a critical role in the execution of all experiments in this thesis.

Lastly, I would like to thank everyone who showed interest and enthusiasm for my thesis topic. Your encouragement has been deeply appreciated and has helped carry me through these final stages of my studies.

`Runar Fosse`
Sunday 10$^{\text{th}}$ August, 2025

Contents

Chapter 1

Introduction

Within the field of Music Information Retrieval (MIR), the task of Automatic Music Transcription (AMT), automatically generating symbolic music notation from audio, is widely regarded as a challenging research problem. Most musical instruments are melodic, requiring transcription systems to detect pitch, onset time, and note duration. Percussive instruments, by contrast, require a different focus: pitch is typically absent or uninformative, and the goal shifts to accurate onset detection and instrument classification. This distinction defines the subtask of Automatic Drum Transcription (ADT), which is concerned specifically with transcribing drums and other percussive instruments. This thesis focuses on the most difficult form of ADT, known as Drum Transcription in the Presence of Melodic Instruments (DTM), where drum events must be transcribed from within full polyphonic musical mixtures [62].

Earlier approaches to ADT relied heavily on signal processing techniques and classical machine learning methods [62]. However, in recent years, deep learning has become the standard, offering significant improvements in transcription performance. As a result, current research has shifted toward understanding how to best apply deep learning to this problem, either by developing or analysing new neural network architectures, or by constructing datasets that help models generalize more effectively [65].

1.1 Thesis statement

This thesis addresses two primary research questions:

1. Which deep learning architecture is best suited for solving a task like ADT?
2. What makes an ADT dataset optimal for training models to generalize on DTM?

To answer these, I divide the thesis into two main studies.

For the first, I train and compare several neural network architectures on established ADT datasets. The evaluated architectures include a Recurrent Neural Network, a Convolutional Neural Network, a Convolutional Recurrent Neural Network, a Convolutional Transformer, and, novel to the field of ADT, the Vision Transformer. By comparing their performances, I aim to identify which architectural designs are most effective for the ADT task, with a focus on DTM.

For the second study, I take the best-performing architecture from the first and train it across multiple combinations of existing ADT datasets. To support Out-of-Distribution (OOD) evaluation, I also introduce SADTP, a novel DTM dataset created specifically for this thesis. Through systematic cross-dataset evaluation, I analyze what makes a dataset effective for improving generalization and how training data can enhance the performance of a well-chosen architecture.

Finally, I compare my top-performing models with results from existing literature to better contextualize performance and generalization. This strengthens my conclusions and positions my findings within the current state of ADT research.

1.2 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2: Background — Provides the foundational concepts necessary to understand the ADT training and inference pipeline. This includes an overview of AMT and ADT, how audio is transformed and represented for deep learning, what the outputs of an ADT system look like, and which evaluation metrics best capture model performance.

Chapter 3: Architectures — Describes and analyses the deep learning architectures used in this thesis, particularly in the first study.

Chapter 4: Datasets — Presents the datasets used for training and evaluation, compares their characteristics, and introduces SADTP, a novel DTM test-only dataset developed for this thesis.

Chapter 5: Methodology — Details the overall experimental methodology, including data preprocessing, training procedures, output postprocessing, model selection, and hyperparameter tuning.

Chapter 6: Architecture Study — Compares the performance of various architectures trained separately on each dataset. Results are analyzed to determine the best-performing model for ADT.

Chapter 7: Dataset Study — Uses the selected architecture from the previous chapter to evaluate how training on different dataset combinations affects generalization, with a focus on both within-domain and Out-of-Distribution (OOD) performance.

Chapter 8: Conclusion — Summarizes the findings of both studies, revisits the research questions, and offers suggestions for 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."* [41]. A decade prior he helped introduce the term Automatic Music Transcription (AMT) covering a field in which now, many years later, has evolved significantly [42].

2.1.1 Transcription using Deep Learning

Currently, the majority of state-of-the-art models within AMT utilize deep learning [62, 65, 32]. 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, it could be described as a multi-label sequence tagging/sequence labeling task. For each element in an input sequence, output a combination of one or more labels. Each element of this output sequence represents a timestep in the input audio recording, consisting of a combination of labels, each representing an instrument being played. This constitutes the most basic form of AMT. Figure 2.1 illustrates an abstract AMT pipeline.

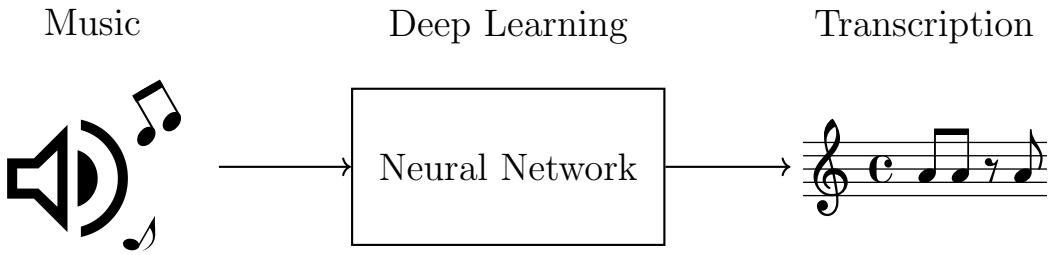

Figure 2.1: An abstract visualization of the Automatic Music Transcription (AMT) process using deep learning. Music is input, processed by a neural network, and ultimately converted into a symbolic transcription.

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 propagation and perception of waves. Mathematically, waves can be studied as signals [13]. To represent these sounds digitally, as *audio*, one can express these waves as a signal, giving rise to the *waveform*. The waveform is a time-wise representation of a signal as a graph, charting the amplitude, or strength of the signal, over time. This sound wave to waveform relationship is visualized in Figure 2.2.

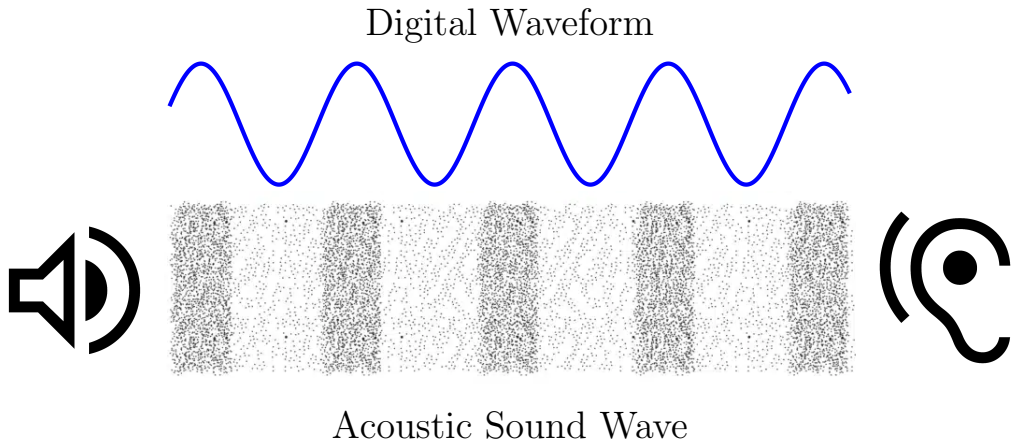

Figure 2.2: The relationship between an acoustic sound wave and its corresponding digital waveform. The waveform represents variations in the air pressure over time. Regions of higher and lower pressure in the sound wave correspond to *peaks* and *troughs* in the waveform, respectively.

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 [56], 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 of 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 allows us to figure out which frequencies make up the signal and how each frequency contributes.

By doing such a transform (seen in Figure 2.3) we turn our temporal data, like a waveform, into spectral data, its *spectrum*. This intuitively *untangles* our signal into its respective base frequencies. By ridding our data of 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.

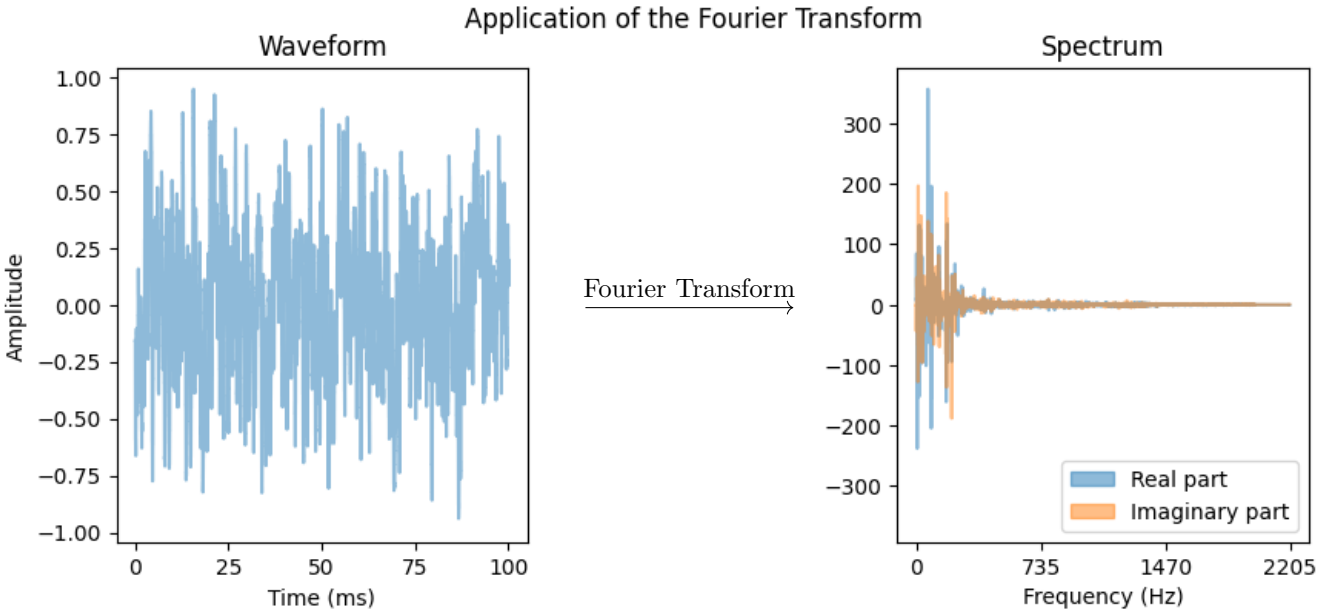

Figure 2.3: The Fourier Transform decomposes a waveform into its frequency components. The time-domain waveform signal on the left is measured in amplitude, where higher amplitudes correspond to higher sound wave air pressure levels. The frequency-domain spectrum on the right displays the amplitude of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

Aliasing is the phenomenon where new, incorrect frequencies emerge in undersampled signals (illustrated in Figure 2.4). 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.

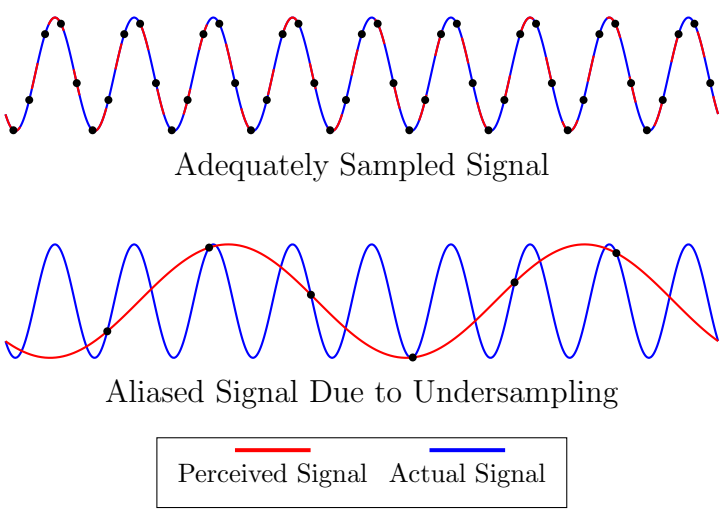

Figure 2.4: Example of aliasing due to undersampling. The original signal has a true frequency of 9 Hz. In the upper plot, sampling at a sufficiently high rate of 45 Hz allows the signal to be perceived and reconstructed accurately. In the lower plot, however, a lower sampling rate of 7 Hz causes aliasing, 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 with regard 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"* [55], 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 [16].

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. An example showing this application of the STFT is shown in Figure 2.5.

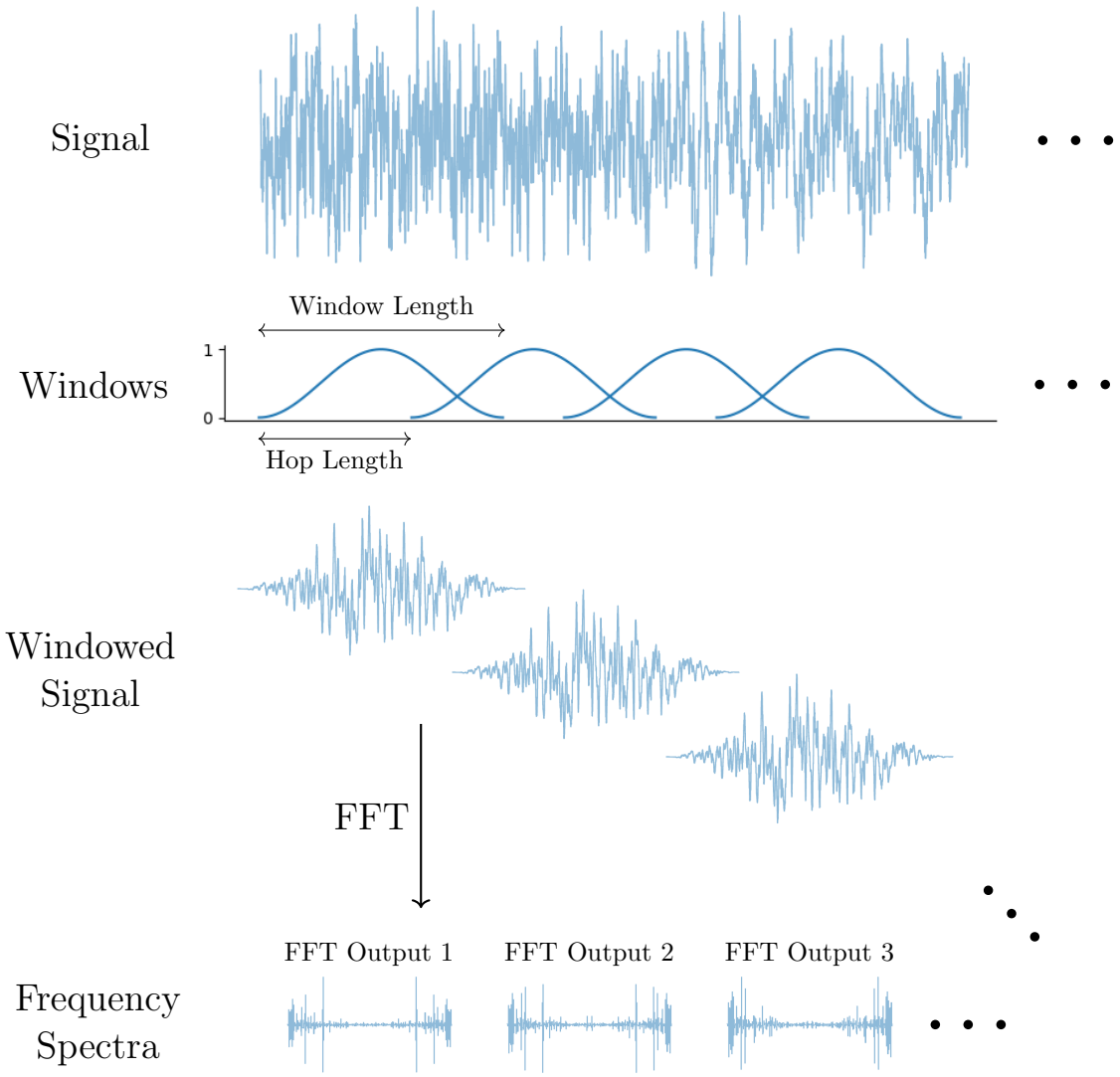

Figure 2.5: Application of the STFT to a signal. The original waveform is multiplied by multiple offset window functions, producing a set of overlapping signal partitions. The FFT is then applied to each partition individually, resulting in a sequence of frequency spectra that preserve localized time–frequency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

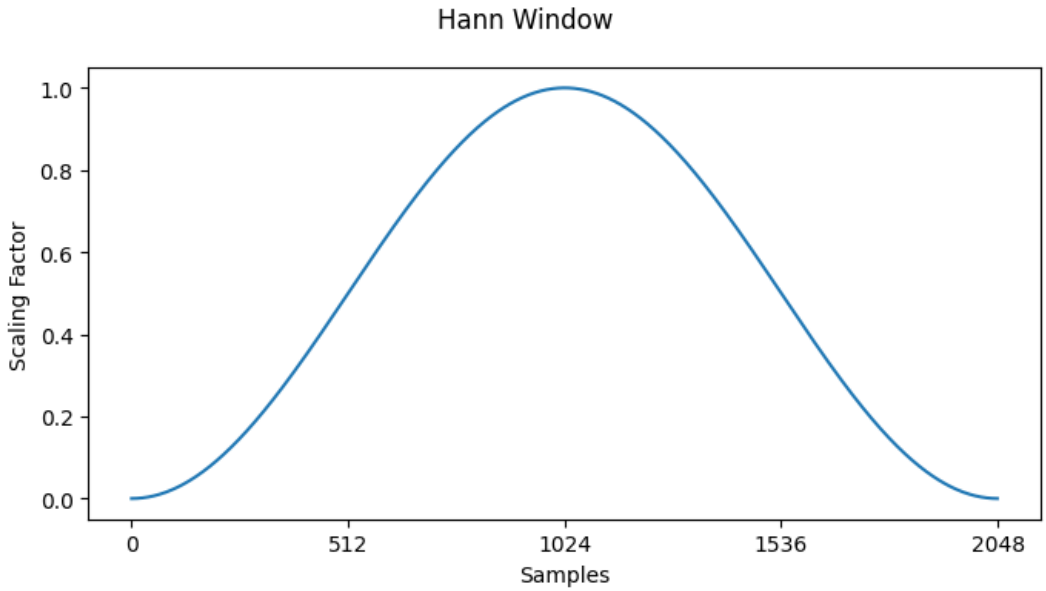

Figure 2.6: The Hann window function, commonly applied to each partition during the STFT to reduce the effects of spectral leakage. It does so by scaling the amplitude of the signal down toward zero at the window's edges. This example shows a window length of 2048 samples.

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 [62, 58, 60, 65].

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, as seen in Figure 2.7, offering an intuitive way to observe how frequency information evolves over time.

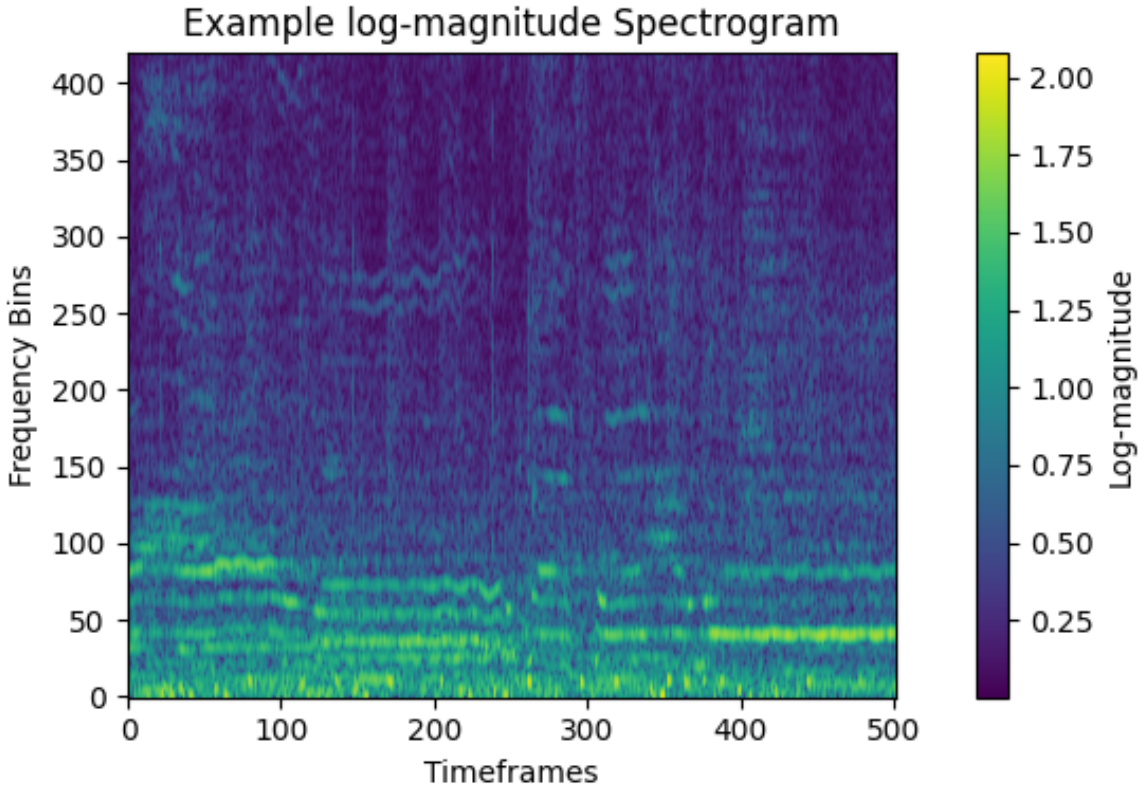

Figure 2.7: Log-magnitude spectrogram of the SADTP track *"Red Swan"*, showing the segment from 3:35 to 3:40 minutes. The spectrogram is computed using 2048 FFTs, with a window length of 512 samples and a hop length corresponding to 10 ms. Only the first 420 frequency bins are visualized. The colour 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 [26].

2.2.7 Loudness of Magnitude

The human perception of loudness is approximately logarithmic. A sound wave 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 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, 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. A comparison between a magnitude spectrogram and a log-magnitude one can be seen in Figure 2.8.

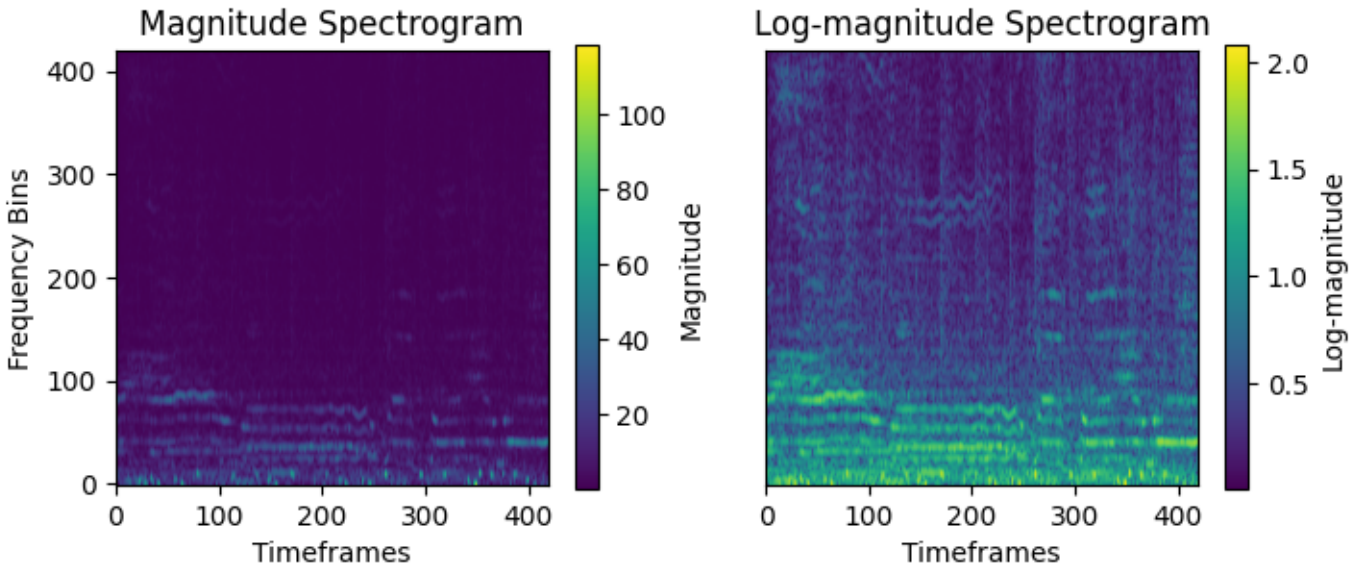

Figure 2.8: Comparison between a linear magnitude spectrogram and a log-magnitude spectrogram. When scaled linearly, differences in magnitude are harder to distinguish, especially for quieter frequency components. The logarithmic scale makes both loud and quiet components visually more distinguishable.

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. [21, 14, 25, 61, 62]

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 [59, 60, 33, 65, 66].

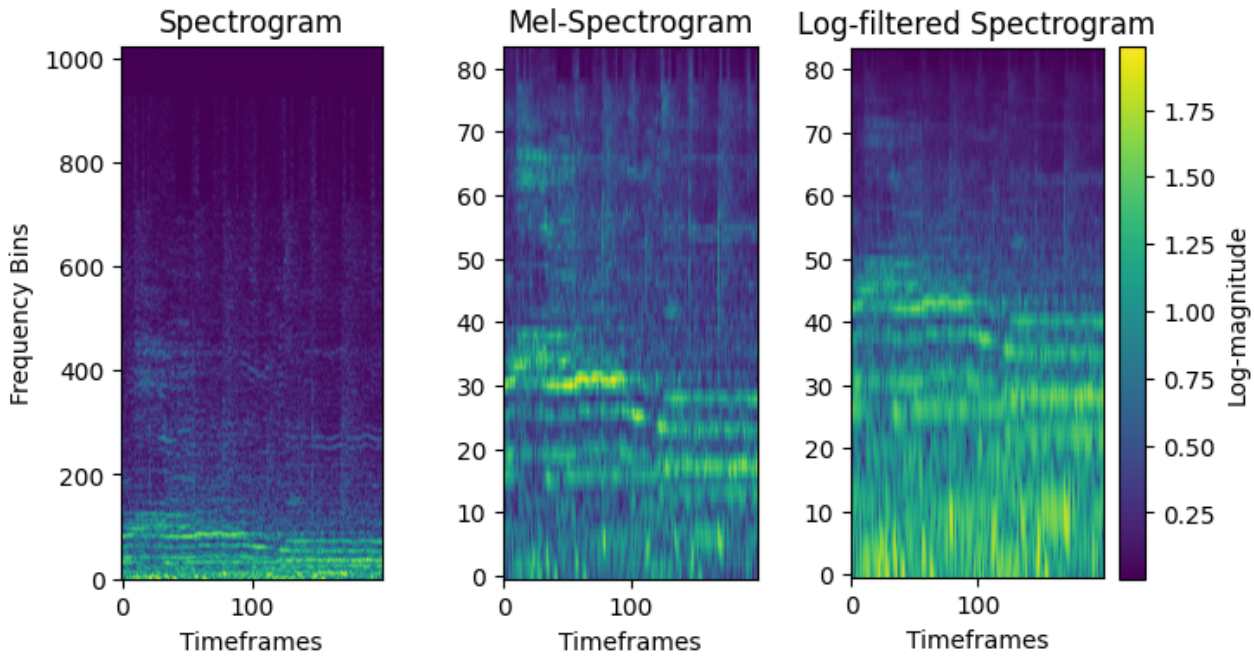

Figure 2.9: Comparison between a normal spectrogram, a Mel spectrogram, and a log-filtered spectrogram. All are presented on a log-magnitude scale. Both the Mel and log-filtered spectrograms apply 84 area-normalized filters, resulting in a non-linear frequency axis. While the Mel spectrogram emphasizes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

Figure 2.9 compares the normal, Mel, and logarithmically filtered spectrogram (all with log-magnitude scaling). Both of the latter two reshape the frequency axis, but in different ways.

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 system 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.

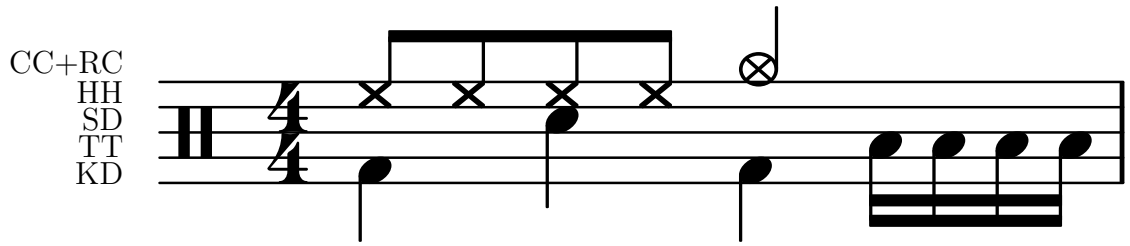

Figure 2.10: An example drum sheet containing all instruments used in a 5-instrument ADT task. The instrument associated with each row is shown on the left. From bottom to top, they are the Kick Drum (KD), Tom-Tom (TT), Snare Drum (SD), Hi-Hat (HH), and Crash and Ride (CC+RC).

Figure 2.10 illustrates a simple drum pattern using staff notation. As mentioned, each note's instrument is given by its vertical position, here labeled explicitly on the left. 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. Figure 2.11 illustrates a list of MIDI events written out in text format.

When computers play MIDI sequences, the events are parsed in order at a fixed rate, using *note on/note off* messages separated 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 [21], 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+ [14], 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 [54].

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). Figure 2.12 illustrates how a drum pattern in staff notation can be represented using these activation functions. [47, 8, 51, 62, 60].

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.

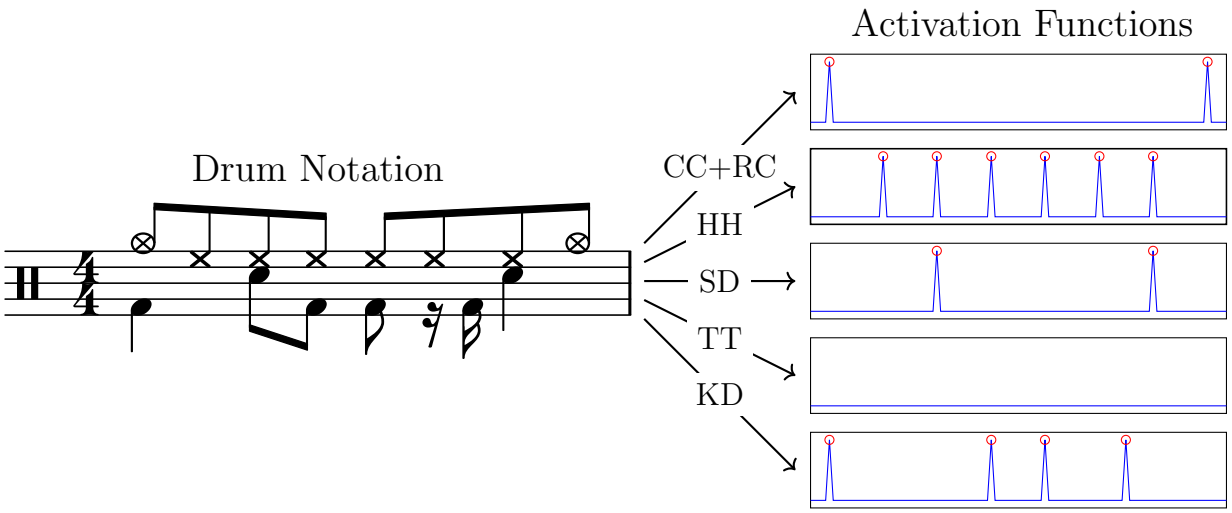

Figure 2.12: Activation function representation for a 5-instrument ADT task, corresponding to a drum pattern written in standard staff notation. Each activation function is zero throughout, except at onset positions, where it takes a value of one. The 𝅀 symbol indicates a sixteenth-note rest in the Kick 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. [7], 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. [60] 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 [62]. 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. [65] note, *"melodic and percussive instruments can overlap and mask each other... 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. [59, 60] achieved strong results using both convolutional and convolutional-recurrent networks. Zehren et al. [65, 66] emphasized the importance of dataset size and quality, showing that both are critical to achieving strong performance. Most recently, Chang et al. [14] 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 (shown in Figure 2.13) typically includes a Snare Drum (SD), a Kick Drum (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 [39].

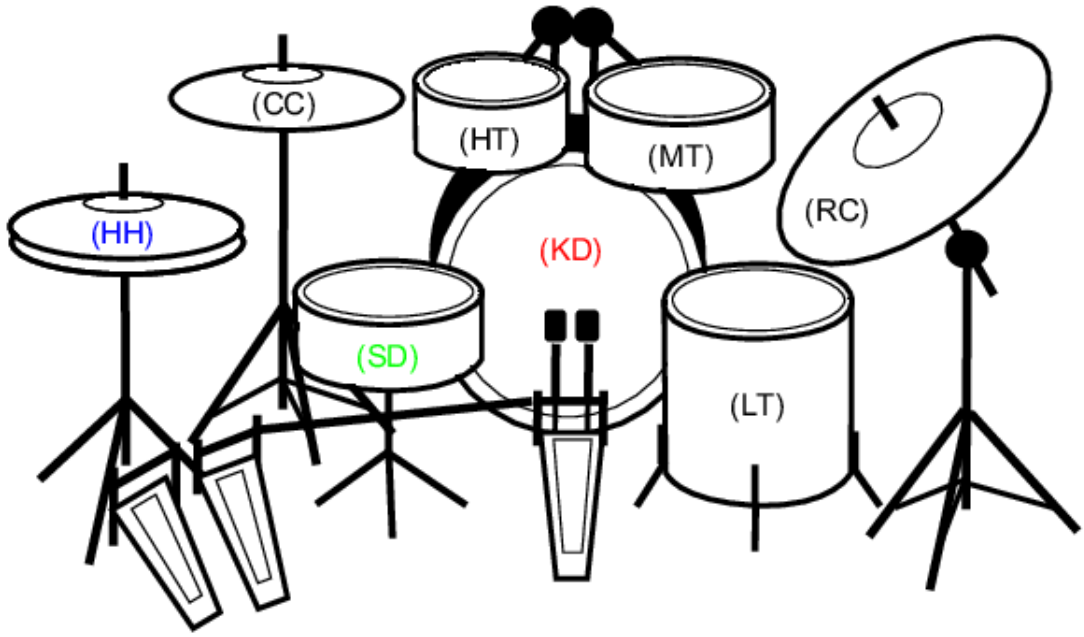

Figure 2.13: The standard drum set configuration, including the Kick Drum (KD), Snare Drum (SD), Hi-Hat (HH), Crash Cymbal (CC), Ride Cymbal (RC), High Tom (HT), Mid Tom (MT), Low Tom (LT) [62].

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 (as seen in Figure 2.14).

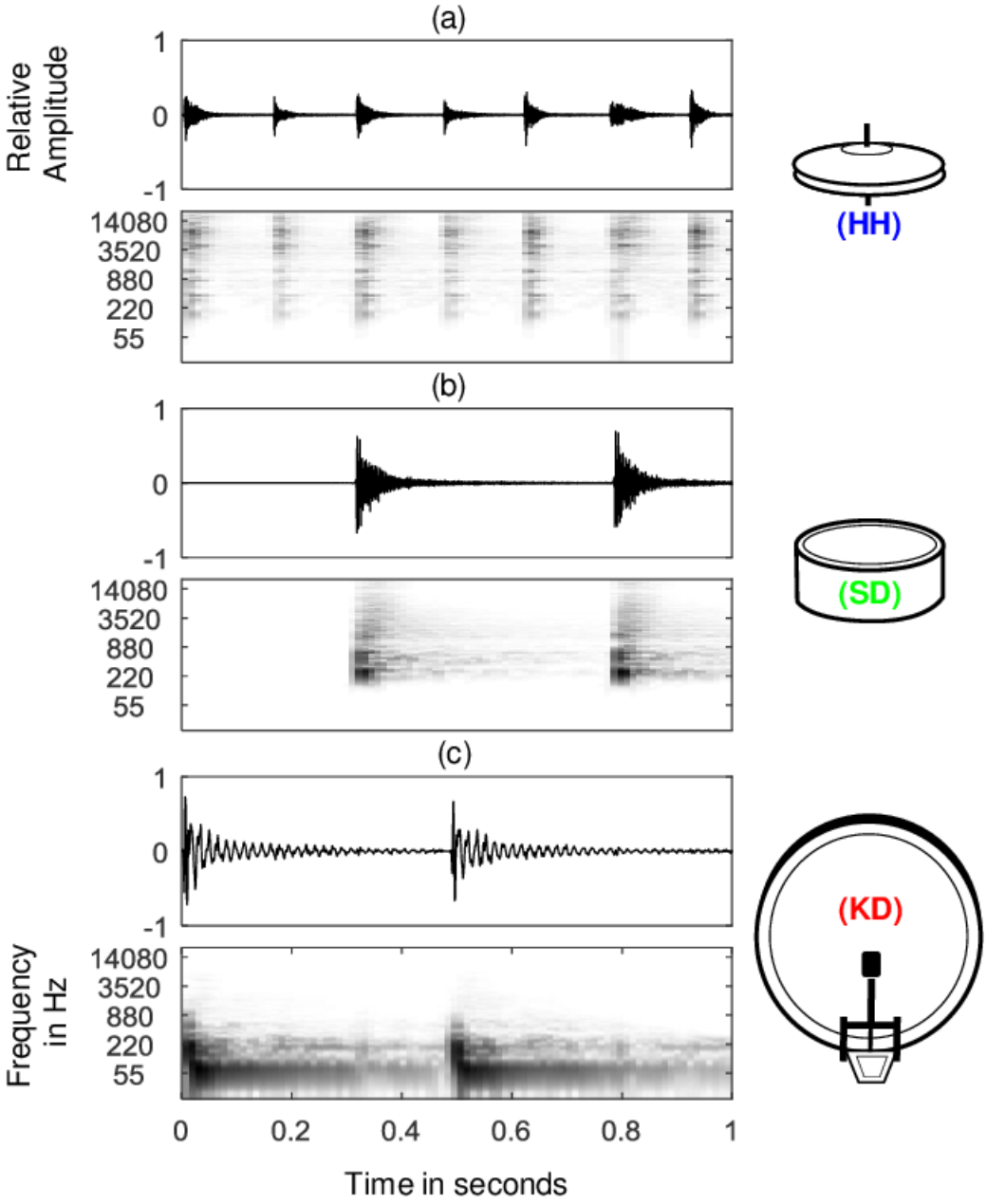

Figure 2.14: Examples of the different audible footprints of drum set components. Plotted are the waveforms of three drum instruments played at varying patterns, each paired with a corresponding spectrogram. The panels highlight how each instrument's events produce distinct patterns in both the time and frequency domains [62].

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 [60, 21]. 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. An example of this pipeline is illustrated in Figure 2.15.

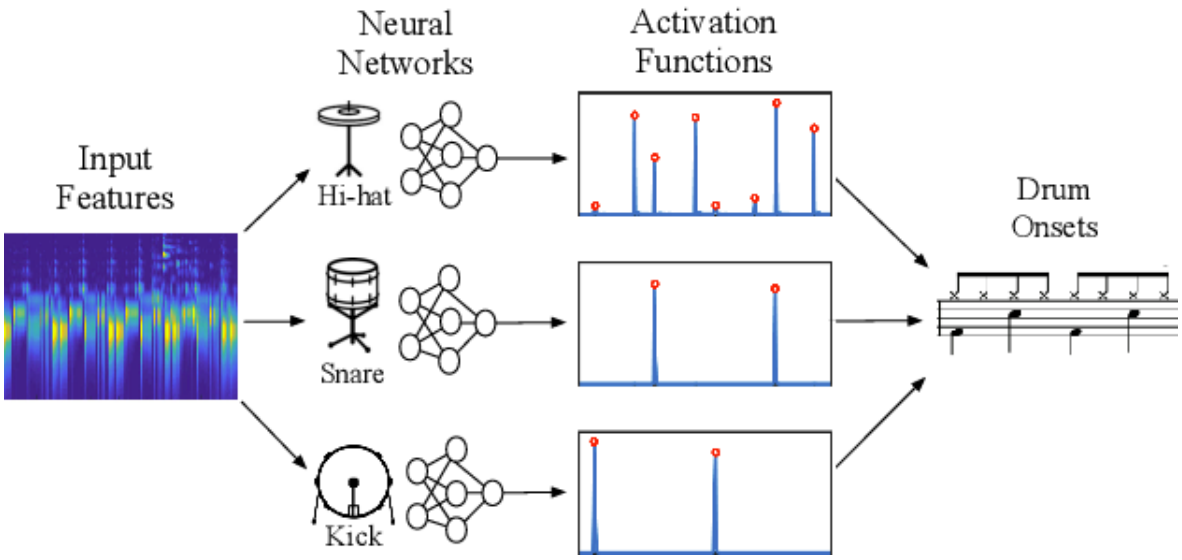

Figure 2.15: Example of a typical ADT pipeline. Given an input spectrogram, the model predicts activation functions for each instrument, which are then quantized into drum onsets and converted into symbolic sheet music [51].

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) [58]. 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 [58]. This approach shifts the evaluation focus from frame-wise 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 naive 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) is 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 kick 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 shows an ADT example with the resulting TPs, FPs, FNs of a sample prediction, and contrasts macro vs. micro aggregation of per-class F1-scores.

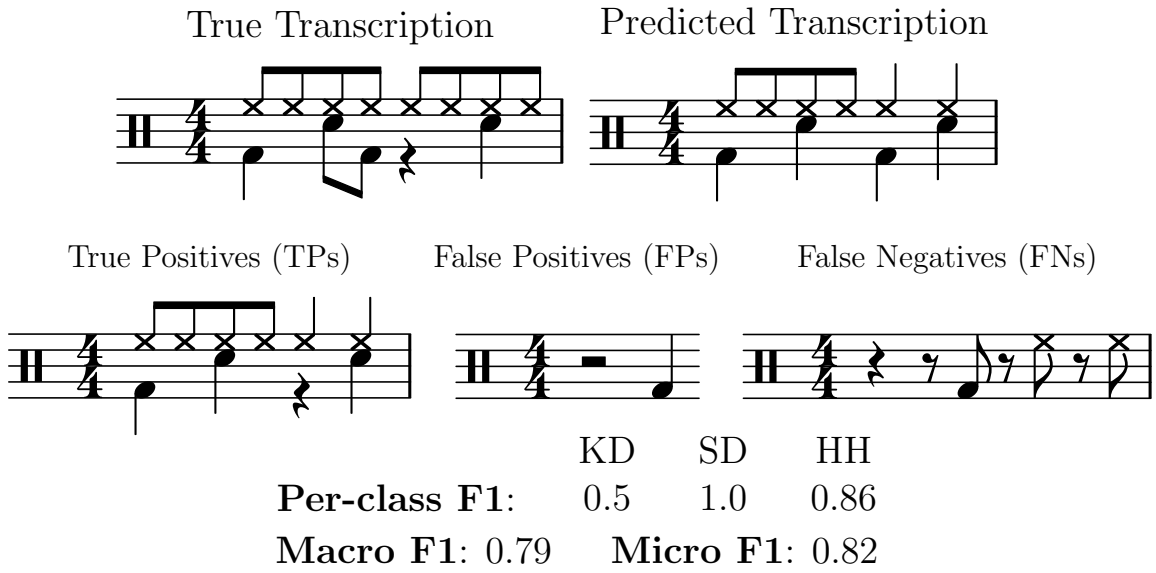

Figure 2.16: Illustration of micro and macro F1-score computation for an example ADT transcription. Symbols such as 𝄽, 𝄾, and 𝄿 denote pauses, and not onsets. Macro F1 reflects the average performance across individual instruments, while micro F1 captures the overall transcription quality across the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

Chapter 3

Architectures

Choosing a 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 accurate, 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 $C = 5$ [66].

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 bidirectional sequential influence is illustrated in Figure 3.1. 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.

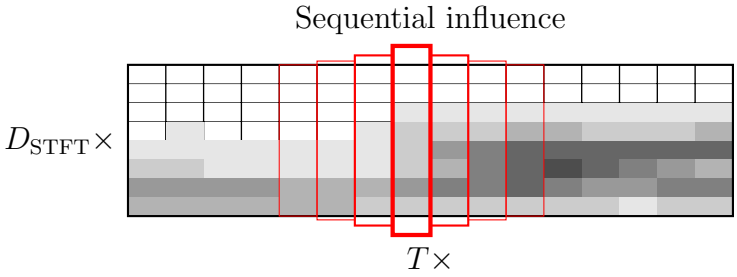

Figure 3.1: Illustration of how bidirectional RNNs include information from surrounding timesteps when predicting the current frame. The background represents a mock spectrogram of shape $(T \times D_{STFT})$, and the red boxes indicate the relative temporal influence on the middle frame. Box height 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. [15] and the Long Short-Term Memory (LSTM) by Hochreiter and Schmidhuber [28].

Both GRUs and LSTMs have been successfully applied to ADT-related tasks [51, 58, 59, 65].

3.1.1 Implementation

I implemented the RNN architecture using a stack of Bidirectional Recurrent Units (BiRUs), followed by a frame-wise 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. The full architecture is visualized in Figure 3.2.

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. These are explicitly visualized in Table 3.1.

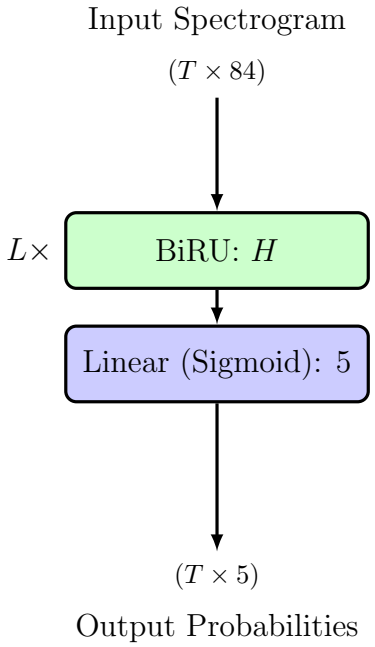

Figure 3.2: Model architecture of the Recurrent Neural Network used in this thesis.

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 (illustrated in Figure 3.3). 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 [59].

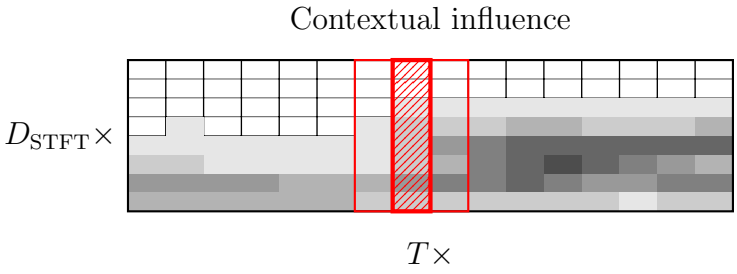

Figure 3.3: Illustration of how CNNs use a fixed-size receptive field to incorporate local context from surrounding time frames. The background represents a mock spectrogram. The red shaded region marks the center time frame being predicted, and the red boxes indicate the receptive field, the neighboring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

3.2.1 Implementation

My CNN architecture 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

patterns. 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 [49].

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. The architecture is visualized in Figure 3.4, with tunable hyperparameters stated in Table 3.2.

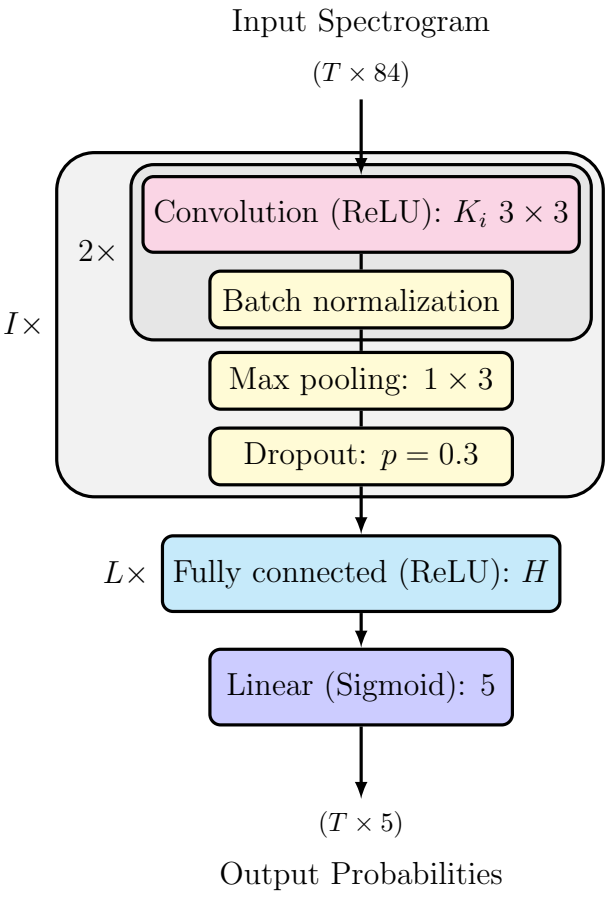

Figure 3.4: Model architecture of the Convolutional Neural Network used in this thesis.

	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 [59, 60, 65].

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 [59, 65]. 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. The full architecture can be seen in Figure 3.5, with its corresponding tunable hyperparameters in Table 3.3.

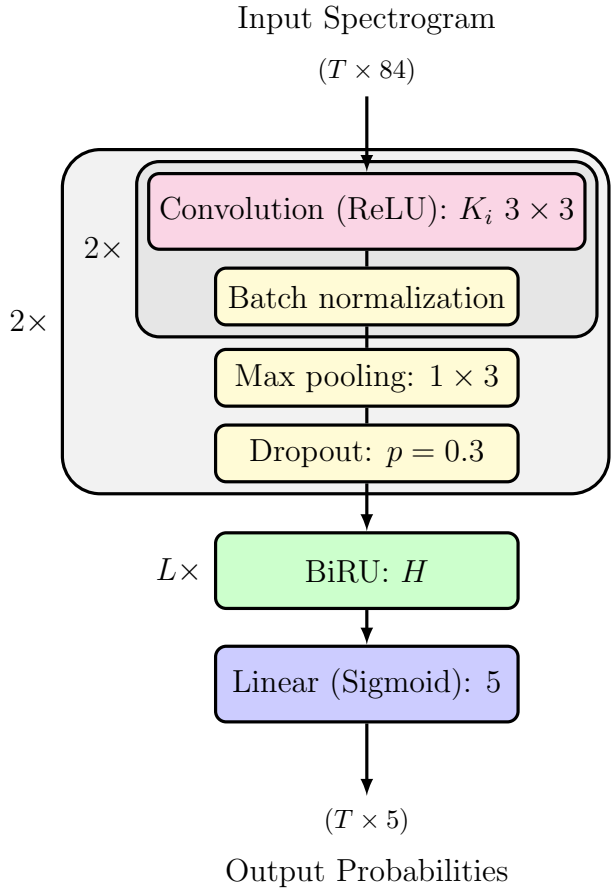

Figure 3.5: Model architecture of the Convolutional Recurrent Neural Network used in this thesis.

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 at combining local time–frequency pattern extraction with temporal modeling, their ability to capture long-range dependencies still remains 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" [57]. 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 (visualized in Figure 3.6).

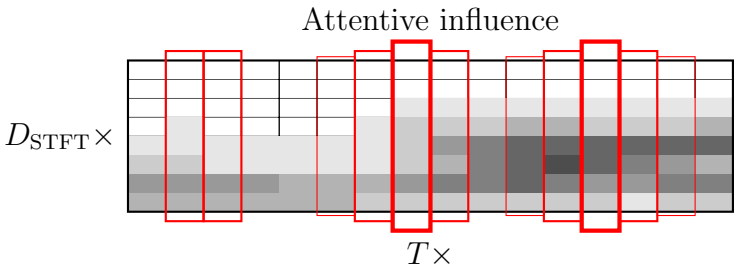

Figure 3.6: Illustration of how attention layers enable influence from time frames across the entire sequence. The background represents a mock spectrogram, and the red boxes indicate the time frames influencing the prediction on the middle frame. Box height reflects the relative influence strength, 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 [31, 21, 65, 14, 66].

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 of 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 blocks with pre-layer normalization, a design shown to improve training stability compared to post-layer normalization [63]. 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 [17, 27].

Finally, a linear output layer computes onset probabilities for each of the five drum instruments. Figure 3.7 illustrates the whole convolutional transformer architecture, with its tunable hyperparameters shown in Table 3.4.

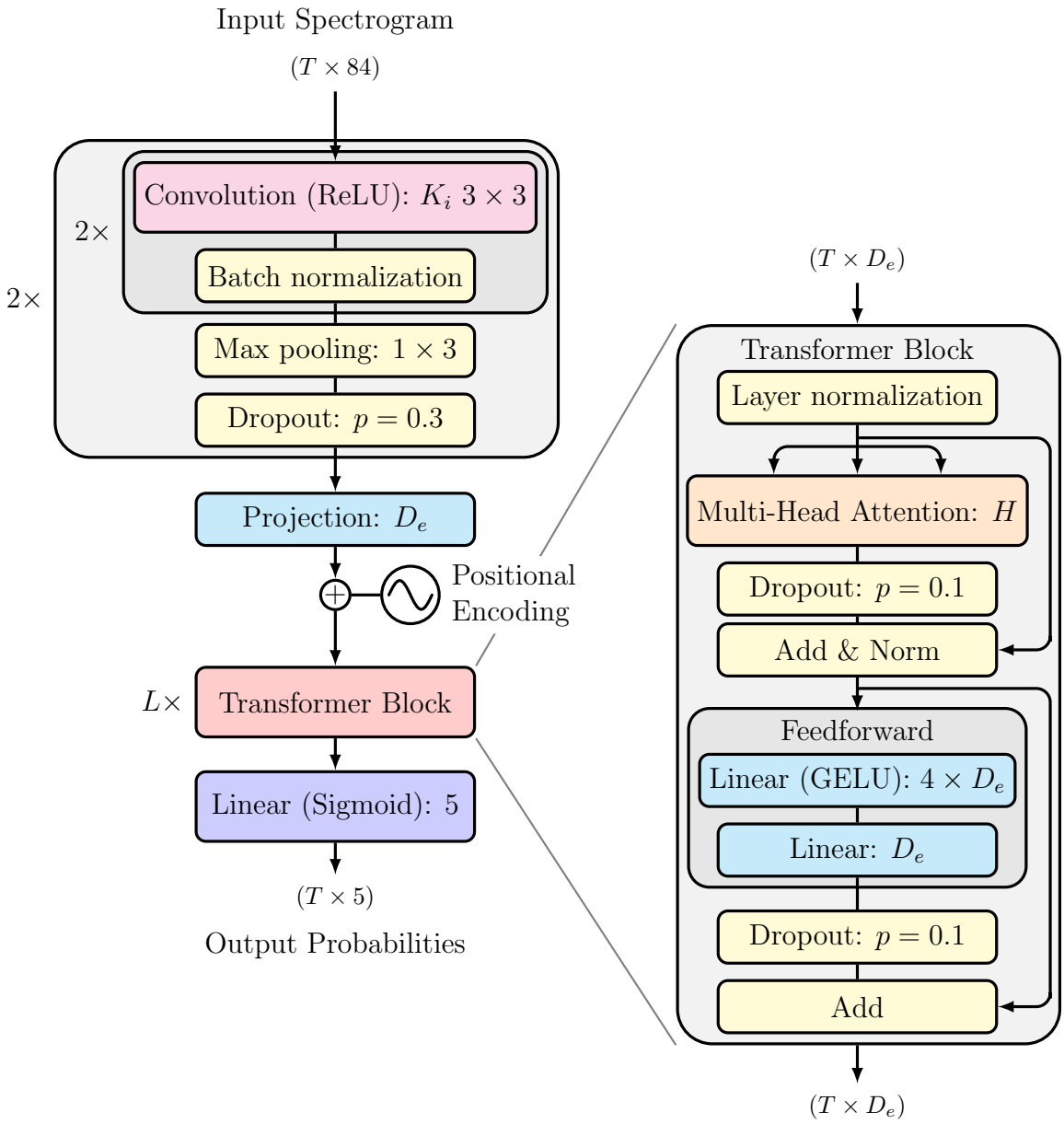

Figure 3.7: Model architecture of the Convolutional Transformer used in this thesis.

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 introduction 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" [18], 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 [18, 25]. 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 pretraining, to reach their full potential. [18].

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. Figure 3.8 shows this process applied to a spectrogram.

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.

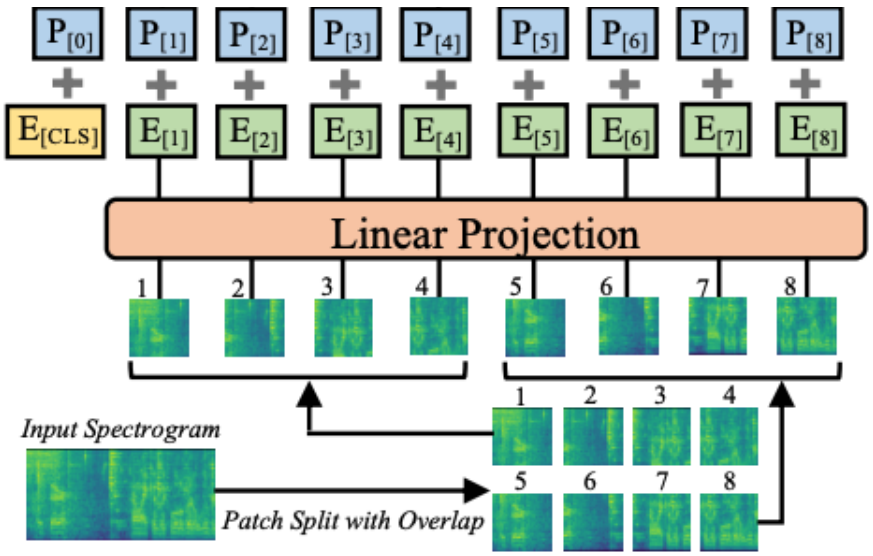

Figure 3.8: Creation of a sequence of patch embeddings from an input spectrogram. The spectrogram is split into overlapping patches, each of which is linearly projected into a one-dimensional embedding vector. A learnable positional embedding is then added to each patch embedding. An initial Class Token ([CLS]) is also included, as is common in classification tasks [25].

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 (as seen in Figure 3.8) is unnecessary and thus omitted [18].

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. This full architecture is illustrated in Figure 3.9, with its corresponding tunable hyperparameters described in Table 3.5.

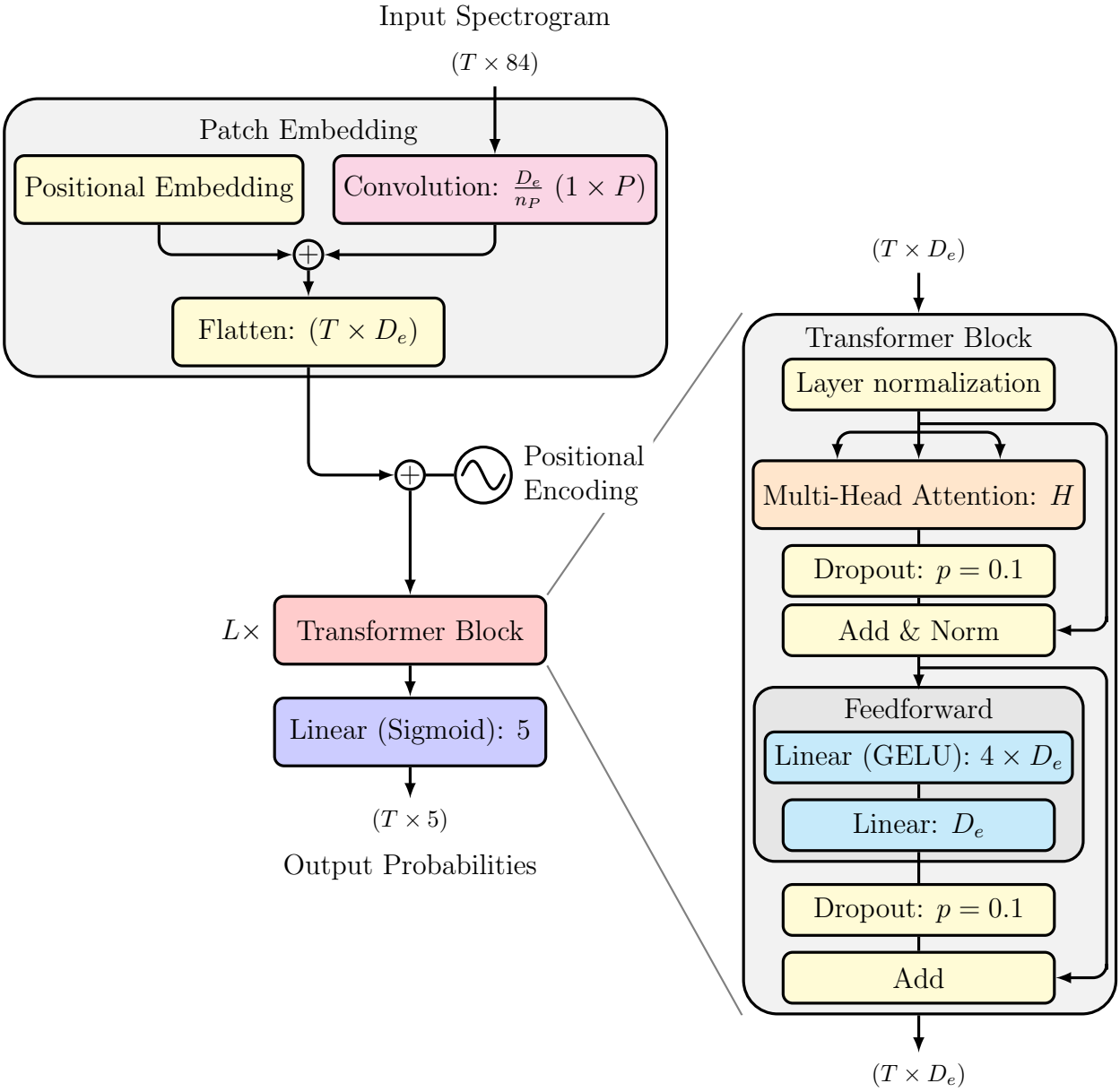

Figure 3.9: Model architecture of the Vision Transformer used in this thesis.

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 [22] is one of the most commonly used ADT datasets [62]. 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 this 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 each performance includes multiple microphone recordings from different spatial positions, I exclusively use the *"wet mix"*, which 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 [23].

The MDB Drums dataset by Southall et al. [52] is another well-known dataset used for ADT. It is built on Bittner et al.'s MedleyDB (MDB) dataset [6], 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 accompaniment. 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 [53].

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, so 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, described in Tables A.1 and A.2.

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 [22, 52].

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 [22]. 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) by Callender et al. [10] is a large ADT dataset consisting of audio recordings from human drum performances, annotated in MIDI format. It expands upon the original Gillick et al.'s Groove MIDI Dataset (GMD) [24], which included only MIDI recordings without audio.

The original dataset was created by recording human performances on a Roland TD-11 electronic drum kit in MIDI format. As it does not contain audio recordings, only MIDI files, it is not directly applicable for ADT tasks. To enable its use for ADT, Callender et al. [10] re-recorded the MIDI sequences in real-time using a Roland TD-17 electronic drum kit within a Digital Audio Workstation (DAW). These re-recordings were rendered using a wide range of soundfonts (often referred to as *drum kits* in the context of electronic drum kits), producing multiple distinctly sounding audio renderings from each original MIDI performance pattern [24, 10].

Unlike the other datasets used in this thesis, E-GMD was not created for a DTM task, but rather for Drum Transcription of Drum-only Recordings (DTD), as all recordings consist of solo, drum-only performances. Additionally, since the data was recorded

from human performances in a semi-manual nature, some errors were introduced during recording, resulting in what the authors describe as *"unusable tracks"* [10]. While the magnitude of these errors is not explicitly quantified, later analysis suggests that up to 20.5% of tracks may contain discrepancies of varying severity [29].

This dataset contains 444.5 hours of audio across 1,059 unique drum performances, resampled into 45,537 MIDI sequences. It is provided with predefined training, validation, and test splits. I accessed the dataset via Zenodo [11, 10].

4.2.1 Mapping

E-GMD is annotated using a Roland electronic drum kit and stored as MIDI files. Each instruments is encoded as a specific MIDI pitch, following the mapping defined by the Roland TD-17 MIDI specifications [24, 10, 2].

In general, MIDI percussion follows the widely adopted *General MIDI* specifications, which defines a standardized pitch-to-instrument mapping. These mappings are extensively documented, for example in Rothstein's "MIDI: A comprehensive introduction" [45].

To ensure correct identification of all instruments in the MIDI transcription, I construct a combined MIDI mapping that uses the Roland TD-17 specification (as referenced for E-GMD) as a base, supplemented with select entries from the General MIDI percussion standard. If the same pitch corresponds to different instruments across these two specifications, I prioritize the Roland mapping. While this approach might introduce redundancies, it ensures that all annotations are accurately parsed [24, 10].

This combined MIDI mapping (seen in Table 4.2) is extensive and includes a wide range of percussive instruments. However, instruments that do not commonly appear in drum set transcriptions or that cannot be logically mapped to the 5-instrument vocabulary are excluded.

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: Mapping from Roland TD-17 and General MIDI percussion pitches to the 5-instrument vocabulary used in this thesis. MIDI pitches that do not have an intuitive mapping to any of the five classes are omitted.

4.3 Slakh

The Slakh (Synthesized Lakh 2100) dataset by Manilow et al. [37] is a synthesized subset of the Lakh Dataset by Raffel [43]. It consists of 2,100 randomly selected tracks from Lakh in which the MIDI files includes at least a piano, bass, guitar, and drums. These MIDI files are rendered and mixed into full audio tracks, which are stored alongside their corresponding original MIDI performances.

Given that the dataset contains full musical mixtures, each with multiple instruments, it was originally designed for AMT tasks. However, since each track includes a drum performance, converting it for ADT is trivial. Therefore, I only utilize the MIDI tracks corresponding to the drum set as our labels for transcription.

The original Slakh dataset was shown to contain data leakage between its different splits. For transcription tasks, it is therefore recommended to use a smaller subset, Slakh2100-redux, in which this issue has been solved. Thus, this is the specific subset I use in this thesis [38].

This dataset contains 115 hours of audio recordings across 1710 different tracks. It is provided with predefined training, validation, and test splits. I accessed it via Zenodo [38].

4.3.1 Mapping

The Slakh dataset stores its annotations in MIDI files, similar to E-GMD. However, unlike E-GMD, it strictly uses the General MIDI percussion mapping. This mapping is nearly identical to the prior combined MIDI mapping, except it excludes MIDI pitches *22*, *26*, and *58*, as shown 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: Mapping from General MIDI percussion pitches to the 5-instrument vocabulary used in this thesis. MIDI pitches that do not have an intuitive mapping to any of the five classes are omitted.

4.4 ADTOF-YT

The ADTOF-YT (Automatic Drums Transcription On Fire YouTube) dataset by Zehren et al. [65] is a large ADT dataset constructed from crowdsourced data, as reflected by the YouTube suffix. Its crowdsourced nature enables the collection of large amounts of non-synthetic, human-performed audio data, but at the cost of reduced control over data quality. Zehren et al. also note that ADTOF-YT appears to be biased toward metal and rock genres [65].

Unlike the other datasets, ADTOF-YT is distributed as prepackaged TensorFlow datasets for each split. Each data point consists of a pair: a logarithmically filtered log-magnitude spectrogram and sequence of instrument onset probabilities (activation functions). The dataset uses a 5-instrument vocabulary and is therefore the least diverse dataset in terms of instruments among those used in this thesis.

In their original paper, "High-Quality and Reproducible Automatic Drum Transcription From Crowdsourced Data", Zehren et al. [65] state that the dataset's spectrograms are stored as log-magnitude, log-frequency spectrograms. However, in the dataset's distribution repository, it is instead claimed that *"The data is available as mel-scale spectrograms."* [64]. To resolve this discrepancy, I manually approximated and reconstructed the original audio waveforms from the provided spectrograms and compared them to known other, known representations. Based on this analysis, I conclude that the dataset is indeed stored as logarithmically filtered log-magnitude spectrograms, not Mel spectrograms. This format matches that used by Vogl et al. [59], and is consistent with the representation used throughout this thesis.

The dataset contains 245 hours of music across 2924 tracks, provided through predefined training, validation, and test splits. The dataset was accessed via Zenodo [64, 65].

4.4.1 Mapping

This dataset is already distributed using a 5-instrument vocabulary that matches the one used in this thesis. Therefore, no remapping of the annotations was necessary, and the dataset was used directly without modifications.

4.5 SADTP

To create a challenging and realistic Out-of-Distribution (OOD) test set, I curated and transcribed a new dataset specifically for this thesis. The SADTP (Small Automatic Drum Transcription Performance) dataset is a novel, small-scale dataset comprising 16 songs with corresponding MIDI transcriptions.

The name *performance* alludes to the transcriptions being recorded live while listening to the audio playback, with only minor post-processing. The recordings were made on a Roland TD-11 electronic drum kit, with MIDI performance data captured in Apple's GarageBand DAW and later extracted as separate MIDI files. This process is similar to that used in E-GMD, as the dataset was also recorded in a semi-manual manner. As a result, it may include slight, human-induced errors such as misaligned onsets or incorrect labels. While the exact magnitude of these errors is unknown, I consider them potentially significant and likely to manifest as a large *transfer gap*.

This dataset stands out as the only one in this thesis not used for training. Its sole purpose is cross-dataset evaluation, providing insight into the generalization ability of models trained on other sources.

The dataset contains 1.08 hours of music across 16 tracks, segmented into 977 non-overlapping 4-second data points (with zero-padding applied when necessary for even partitioning). The dataset is available through GitHub [20].

4.5.1 Mapping

Similar to E-GMD, the SADTP dataset stores its annotations as MIDI files recorded from a Roland TD-11 electronic drum kit (a close analogue to the Roland TD-17 used in E-GMD [10]). It therefore uses the same combined MIDI mapping shown in Table 4.2, making it directly compatible with the 5-instrument vocabulary used in this thesis.

4.6 Summary

Dataset	Duration (h)	Number of tracks	Melodic	Synthetic
ENST+MDB	1.37	87	✓	×
E-GMD	444.5	45,537	×	✓
Slakh	115	1710	✓	✓
ADTOF-YT	245	2924	✓	×
SADTP	1.08	16	✓	×

Table 4.4: Overview of dataset characteristics, including total duration and number of tracks. Datasets marked as *Melodic* contain full musical mixtures and are therefore used for DTM; those without are drum-only and used for DTD. *Synthetic* indicates that the audio is digitally synthesized, often via automatic generation schemes [66].

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 number of sequences in each dataset's train, validation and test splits. All sequences are of length 400, corresponding to 4-second audio clips.

Tables 4.4 and 4.5 provide a structured overview of the datasets used in this thesis. Table 4.4 summarizes key characteristics in a comparable format, while Table 4.5 highlights the substantial differences in dataset sizes. These datasets span a diverse range of properties, which may be beneficial for training ADT models that generalize well across domains.

Chapter 5

Methodology

In this thesis, I conduct two distinct studies. Although they differ in both intention and comparative evaluation methodology, the dataset preparation and model selection pipelines remain identical.

It must be noted that all deep learning models and training procedures in this thesis are implemented using the `PyTorch` framework [40]. PyTorch provides the core tools for constructing the model architectures, managing data pipelines, and conducting training through automatic differentiation and backpropagation.

The complete source code used for model implementation, datasets handling, training, and pipeline, is publicly available in a GitHub repository [19].

5.1 Data Preparation

As mentioned earlier, the datasets used in this thesis are distributed in varying formats. To enable consistent processing, a series of transformations are applied to unify them into PyTorch datasets.

For the ADTOF-YT dataset, this unification is straightforward due to its already preprocessed nature: it only involves converting the stored TensorFlow dataset into a PyTorch format. Otherwise, the data is left unchanged.

5.1.1 Audio Files

The remaining datasets are distributed as either Waveform Audio File Format (`.wav`) or Free Lossless Audio Codec (`.flac`) files. Both formats are loaded using the PyTorch `Torchaudio` library and converted to monophonic format by averaging the left and right channels. If a dataset provides separate audio for drums and accompaniment (as in ENST-Drums), these are additively mixed.

Once a track is loaded as a waveform, zero-padding is added to its end to ensure even partitioning into 4-second segments. Each segment is then transformed into a spectrogram using a Short-time Fourier Transform (STFT) with 2048 FFT bins, and a window length of 2048. The hop length is set to the sampling rate divided by 100, yielding a temporal resolution of 10 ms per frame. Since each segment contains 4 seconds of audio, the resulting spectrograms have a fixed sequence length of $T = 400$ frames.

Next, a filterbank is applied by computing 12 area-normalized, logarithmically spaced filters per octave, centered at 440 Hz and spanning the range from 20 Hz to 20,000 Hz. Applying this filterbank via matrix multiplication results in a logarithmically filtered spectrogram with $D_{\text{STFT}} = 84$ frequency bins. Finally, a logarithmic scale is applied to the spectrogram's magnitude by computing $\log_{10}(x+1)$ for each spectrogram value x. The addition of 1 ensures numerical stability near zero by preventing $\lim_{x\to 0} \log_{10}(x) = -\infty$.

5.1.2 Annotations

The annotations are distributed either as plain text files (`.txt`) or as MIDI files (`.mid` or `.midi`), each requiring a different transformation scheme to convert them into sequences of instrument onset probabilities (activation functions). Although additional metadata such as velocity may be present in specific datasets, I only use the onset time and corresponding drum instrument label.

Datasets that store annotations in text format (ENST-Drums and MDB Drums) follow a similar structure: each onset is listed on a separate line, with the onset time (in seconds) and the instrument ID separated by a space or tab character. To convert these into onset probability sequences, the onset time is first converted to a frame index by converting from seconds to milliseconds, dividing by 10 to match the 10 ms frame resolution, and rounding to the nearest integer. The instrument ID is then mapped to its corresponding class, and the appropriate `(frame, class)` entry is set to 1.

For datasets that provide MIDI annotations, I use the `Partitura` library to parse the MIDI files into arrays of MIDI events called *notes*. Each note contains information such as onset time, pitch, and velocity. To extract drum onsets, I filter for notes with a non-zero velocity. Instrument labels are derived from the MIDI pitch, which is mapped to the corresponding class label. Frame indices are computed in the same manner as for the text-based annotations, and the corresponding `(frame, class)` entries are set to 1.

In addition to these binary onset labels, I apply a *target widening* strategy by assigning a soft label of 0.5 to frames adjacent to each onset. This technique has been shown to mitigate the effects of label sparsity, particularly in beat tracking and transcription tasks [31, 65].

5.1.3 Splitting and Storing

These spectrogram/onset sequence pairs are stored as `TensorDataset` objects in PyTorch, grouped according to each track's respective train, validation, or test split. These are then serialized into PyTorch pickle files (`.pt`). By performing all preprocessing steps in advance, the need for on-the-fly computation during training is minimized, improving runtime efficiency.

5.2 Preprocessing

Although most preprocessing is completed in advance, during the prior dataset preparation step, a few steps are deferred to runtime. Most importantly, each model computes the mean and standard deviation of its training dataset and uses these parameters to standardize input data during both training and inference.

This form of data normalization has been shown to improve the speed and stability of convergence during training, and often results in models generalize better to unseen data. While the specific effects depend on the normalization technique used, there is a broad consensus in machine learning community that normalization is beneficial, which explains its ubiquitous use in state-of-the-art models [30].

Another preprocessing step motivated by ADT-specific methods is the use of infrequency weights, which assign frame-wise loss weights based on the instrument onsets

present at each frame. These weights are precomputed from the training dataset and are calculated, for each instrument, using what Cartwright and Bello refer to as *"the inverse estimated entropy of their event activity distribution"* [12]. While they applied this technique to address sparsity across different tasks, Zehren et al. [65] adapted it to emphasize infrequent instruments.

To compute these weights, I first calculate the probability of an instrument i appearing as $p_i = \frac{n_i}{T}$, where n_i is the number of onsets and T is the number of timesteps. The infrequency weight for instrument i, denoted w_i, is then computed as the inverse entropy:

$$w_i = \left(-p_i \log p_i - (1 - p_i) \log (1 - p_i)\right)^{-1}.$$

Note that this differs from the method by Cartwright and Bello, where the probability is averaged across instruments; here, I compute it per instrument independently [12].

It is worth noting that this entropy-based weighting function is symmetric around $p_i = 0.5$, meaning that weights would decrease again if an instrument appeared very frequently (i.e., in over 50% of all time frames). However, this is not a concern in ADT, as instrument onsets are inherently sparse, particularly at the fine-grained temporal resolution used in this thesis, ensuring that the computed weights emphasize the appropriate classes.

5.3 Training

With the datasets preprocessed and standardized into uniform PyTorch formats, training can proceed using the prepared spectrograms and corresponding onset annotations. This section describes how I implement the training procedure and outlines the loss function and training setup used to optimize model performance for the ADT task.

During training, the datasets are loaded and iterated using PyTorch's `DataLoader` objects, with a batch size of 128. At runtime, additional transformations are applied, such as model-specific input normalization and reshaping to match architectural requirements. All training is performed on an NVIDIA A100 80GB Tensor Core GPU to ensure efficient execution.

As previously discussed, the ADT task can be framed as a sequence labeling problem, where each time frame may contain multiple simultaneous instrument onsets. These are represented as binary targets: 1 if an instrument is active, and 0 otherwise. A natural

loss function for this multi-label setup is binary cross-entropy, which treats each output dimension as an independent probability prediction.

Instead of applying a sigmoid activation function to the model's logits followed by binary cross-entropy, I directly output logits and use PyTorch's `BCEWithLogitsLoss` function, as recommended in the PyTorch documentation. This improves numerical stability by internally applying the log-sum-exp trick, which helps prevent underflow or overflow issues caused by extremely small or extremely large output values.

For the choice of optimizer, I initially considered Adam, which is widely regarded as a strong general-purpose optimizer in deep learning. As Sebastian Ruder notes, *"Adam might be the best overall choice"* [46]. However, despite its popularity, Adam has known limitations, particularly in how it couples its weight decay term within its gradient-based updates [34, 9].

To address this, I instead used AdamW [36], a modified version of Adam that fully decouples weight decay from gradient updates. This decoupling has been shown to improve generalization performance, especially in regularized training scenarios.

During training, I observed that the magnitude of the loss values could vary significantly and occasionally exhibited signs of instability, including exploding gradients. To mitigate this, I applied gradient clipping with a maximum norm of 2, which substantially reduced the risk of these exploding gradients.

Another common addition in ADT literature is the use of a learning rate scheduler. In this thesis, I adopt a scheduler that monitors recent validation loss values and reduces the learning rate by a factor of 5 if no improvement is observed for 5 consecutive epochs. Additionally, I implement early stopping by terminating training if the validation loss does not improve for 15 consecutive epochs [14, 65].

5.4 Postprocessing

After training, the model produces frame-wise activation values rather than discrete predictions. These outputs form a two-dimensional matrix with values on the interval $(0, 1)$, representing the model's estimated confidence that a particular instrument is present at each time frame. While this continuous format is suitable for computing loss during

training, it is less interpretable when evaluating performance or generating final transcriptions. To enable clear evaluation and comparison, additional postprocessing is applied to convert these confidence values into discrete onset predictions.

First, I apply a peak-picking algorithm to isolate local maxima in the model's onset confidence values. These peaks intuitively represent frames where the model is most confident that an instrument onset occurs, relative to its surrounding frames [7, 60]. A predicted onset is registered if a peak has a confidence value greater than or equal to 0.5.

To compare predicted onsets with ground-truth annotations, I use a greedy, one-to-one matching procedure. Starting from the beginning of the sequence, a prediction is counted as a True Positive (TP) if it occurs within a 5-frame (50 ms) window of a true onset. Each true onset can be matched to at most one prediction, and vice versa. Predictions outside this tolerance window are considered False Positive (FP), while true onsets without a corresponding prediction in the window are counted as False Negative (FN).

These counts of TP, FP, and FN are then used to compute the micro F1-score, as described earlier.

5.5 Model Selection

For model training and selection, I utilize the RayTune library [35], which provides an efficient and flexible framework for hyperparameter optimization and experiment management. It allows me to define a training function, specify the evaluation metric to optimize, configure hyperparameter search spaces and strategies, as well as manage runtime settings. Through per-epoch reporting, RayTune also handles checkpointing and automatically selects the best-performing model during and after training.

For each experiment, I train 15 different models using RayTune, with the exception of the smallest dataset (ENST+MDB), where 25 models are evaluated due to having a lower computational complexity. Hyperparameters are optimized using Bayesian optimization (described in Section 5.6). Each model is trained for up to 100 epochs, with early stopping applied if the validation loss plateaus.

During each training epoch, I evaluate the model on the corresponding validation set. After all 15 (or 25, in the case of ENST+MDB) models have completed training, I select the one with the highest validation Micro F1-score.

Because most datasets used are pre-split into training, validation, and test sets, I adopt a hold-out validation strategy. This approach is not only standard in ADT research [58, 62, 14], but also widely used throughout the deep learning community. As Raschka notes, *"The holdout method is inarguably the simplest model evaluation technique"* [44], which may help explain its popularity.

Once the best-performing model is selected based on validation performance, I evaluate it on the corresponding test dataset to estimate its generalization performance to unseen data. The final model is then stored along with its learned weights, training configuration, and evaluation metrics.

As mentioned, each selected model is evaluated on the test splits of all datasets. Although test sets are traditionally reserved for a final performance evaluation, providing an estimate of how well a model generalizes to unseen data, they are also used here to compare the effects of different architectures or dataset configurations, depending on the study. This enables the analysis how various controllable factors influence a model's generalization ability.

Importantly, no model selection nor hyperparameter tuning is performed based on test performance, preserving the validity of test performances as an unbiased estimate of generalization.

5.6 Hyperparameter Tuning

5.6.1 Search Strategies

Each model includes several hyperparameters that must be tuned. A thorough, exhaustive approach is grid search, which trains and evaluates every possible combination of hyperparameter values to identify the best-performing configuration. However, the number of combinations grows exponentially with the number of hyperparameters and their value ranges, making grid search computationally infeasible for larger search spaces.

A more efficient alternative is random search, where hyperparameter combinations are sampled randomly from the search space. While this approach improves computational efficiency and has been shown to be surprisingly effective [5], it introduces stochasticity:

a given run may be unlucky, and fail to explore high-performing regions of the search space due to chance, resulting in a suboptimal model.

A strategy that combines the advantages of both grid search and random search is Bayesian optimization. Like random search, it samples hyperparameter configurations from a defined space, but it also uses performance information from previously evaluated models to guide future sampling. This allows the search to progressively focus on more promising regions of the hyperparameter space. In doing so, it retains the computational efficiency of random search while gaining some of the systematic exploration found in grid search [50].

In this thesis, I utilize RayTune's implementation of `OptunaSearch`, which integrates the Optuna optimization framework [3]. Optuna is based on Bayesian optimization and has been shown to be effective in neural network tuning tasks. For example, Shekhar et al. [48] benchmarked several hyperparameter optimization frameworks and noted that *"The performance score of Optuna is the highest for all datasets."*

5.6.2 Hyperparameters

RayTune allows me to define the search space for each hyperparameter in a straightforward and flexible manner. All architecture-specific hyperparameters are sampled from discrete sets using random categorical choices. These sets are carefully constructed to ensure that each architecture operates within a comparable computational budget. This helps prevent any model from over- or underperforming due to discrepancies in parameter count or search space flexibility. Additionally, the optimizer-specific hyperparameters, learning rate and weight decay, are sampled using a logarithmically uniform distribution, enabling exploration across several orders of magnitude. The specific search spaces used in this thesis are listed 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 from predefined values

Table 5.1: Search space definitions for each hyperparameter. Learning rate and weight decay are sampled from log-uniform distributions, while architecture-specific hyperparameters are selected via random choice. The specific values for each architecture are detailed in Chapter 3.

Chapter 6

Architecture Study

The primary objective of this study is to evaluate how well different neural network architectures perform on the task of Automatic Drum Transcription (ADT), with a specific focus on Drum Transcription in the Presence of Melodic Instruments (DTM). The goal is to identify which architectures are most effective for this task, to observe possible common patterns among high-performing models, and to determine whether certain architectural designs consistently underperform. These findings will help assess which architectures are best suited for complex ADT tasks and how architecture choice may influence a model's ability to generalize.

6.1 Methodology

To effectively assess the strengths and weaknesses of each architecture within-domain across all datasets, I conduct an experiment for each architecture/dataset combination. For each combination, I train multiple models (either 15 or 25) and select the best-performing one based on validation performance. The selected model is then evaluated on the test split of the same dataset it was trained on. As a result, all reported performance metrics reflect performance on unseen data drawn from the same distribution as the training set.

Each model is trained and validated exclusively on one dataset, and evaluated only on its corresponding test set. This approach provides insight into each architecture's ability to learn the ADT task and serves as a basis for comparing their generalization performance within-domain.

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: Test micro F1-score for each architecture, trained and evaluated on splits of the same dataset. **Bold** values represent the highest score achieved for that dataset.

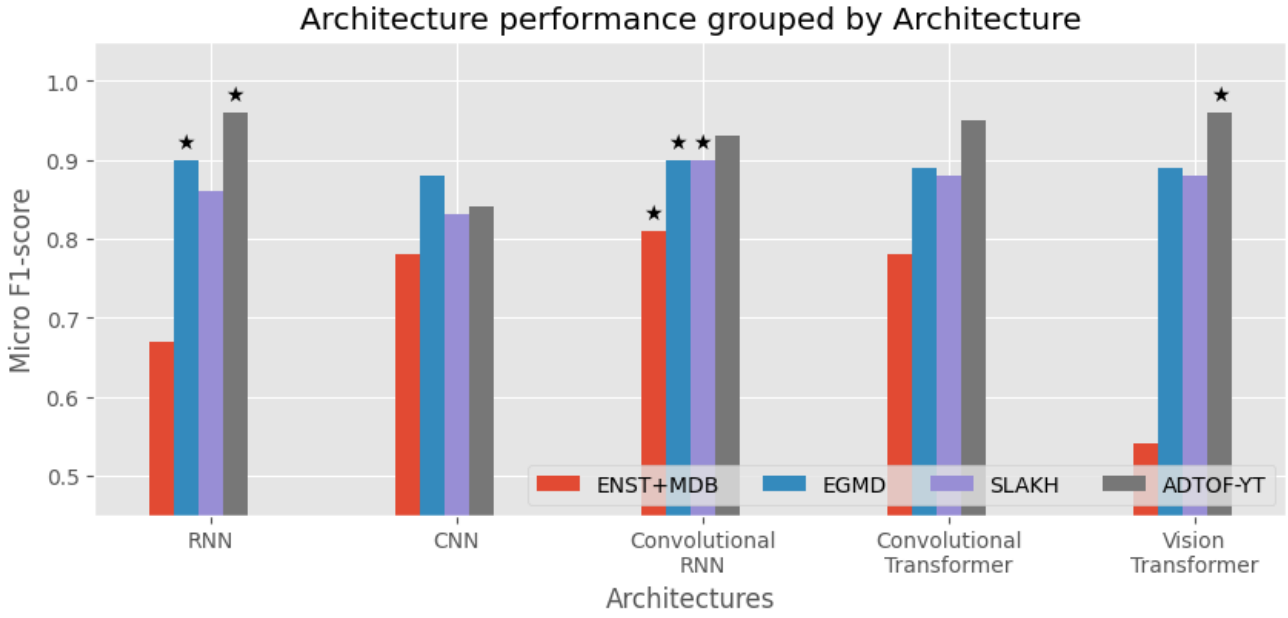

Figure 6.1: Test micro F1-scores per dataset, grouped by architecture. Bars marked with a ($\star$) indicate the best-performing architecture for each dataset.

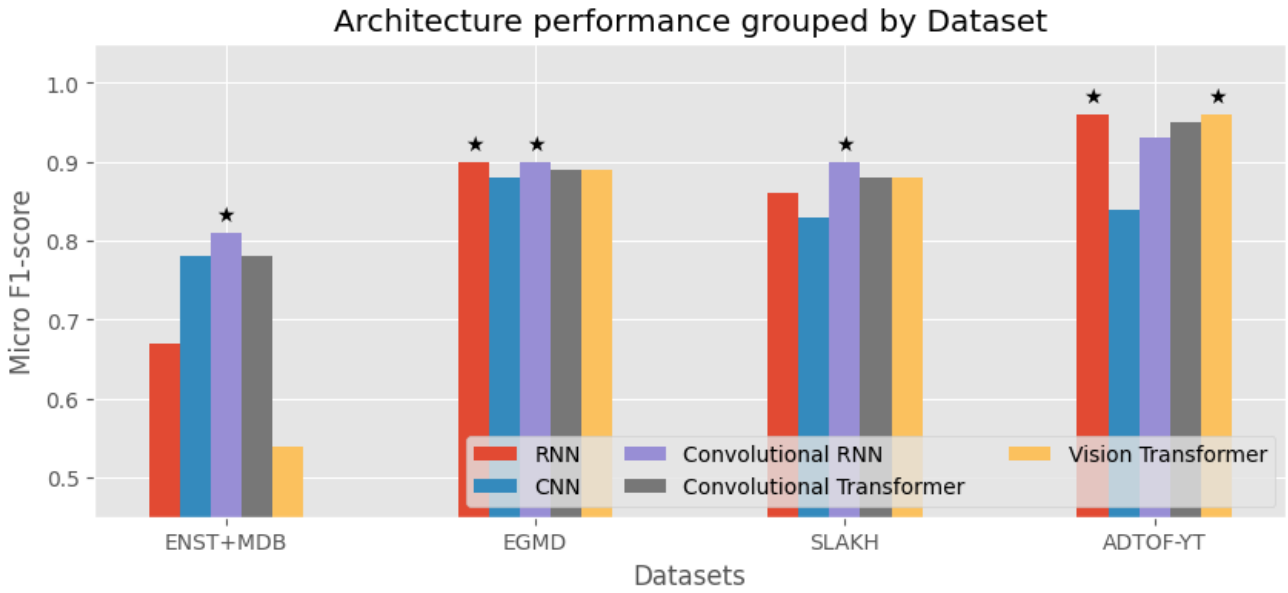

Figure 6.2: Test micro F1-scores per architecture, grouped by dataset. Bars marked with a ($\star$) indicate the best-performing architecture for each dataset.

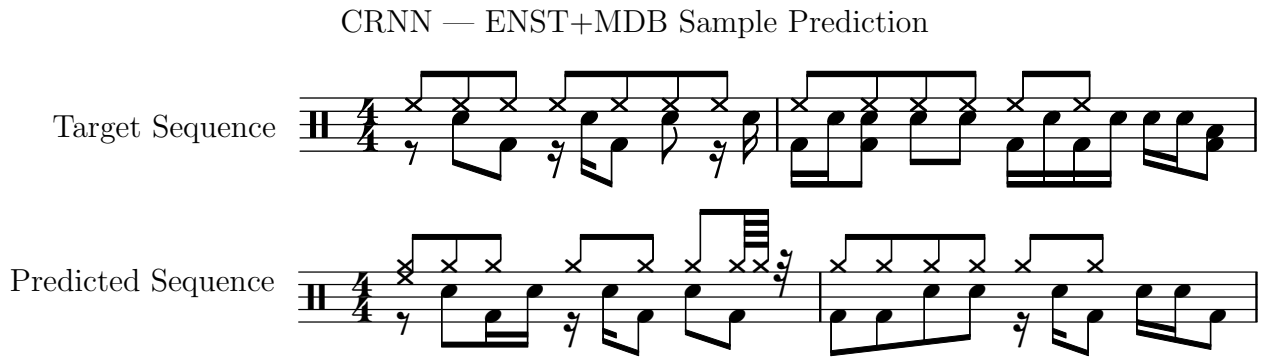

Figure 6.3: Prediction example from the Convolutional Recurrent Neural Network (CRNN) on a randomly selected sequence from the ENST+MDB test split. Both sequences were manually converted from activation function format into drum staff notation. This is a complex drum pattern, but the transcription is generally accurate. Most onsets are correctly placed and labeled, although several False Positives (FPs) and False Negatives (FNs) occur.

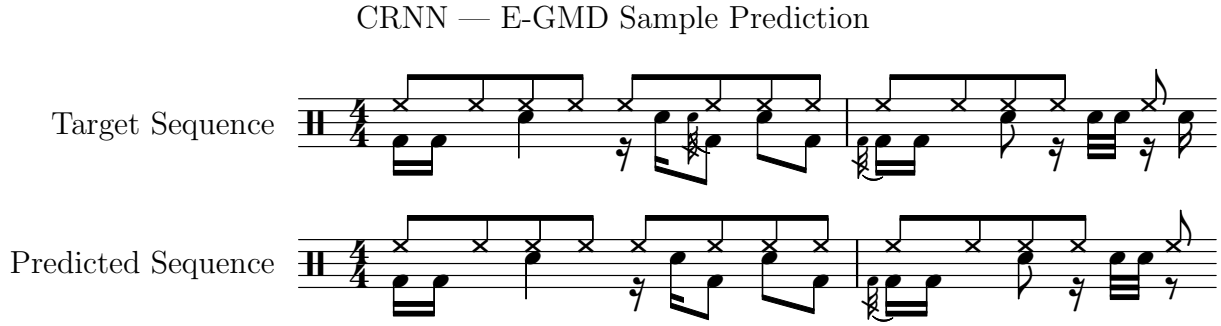

Figure 6.4: Prediction example from the Convolutional Recurrent Neural Network (CRNN) on a randomly selected sequence from the E-GMD test split. Both sequences were manually converted from activation function format into drum staff notation. The drum pattern is moderately complex, but the model transcribed it with high accuracy. Only two False Negatives (FNs) are present, and the prediction aligns closely with the target sequence. Note that E-GMD is a DTD dataset without melodic accompaniment, which may make the transcription task slightly easier.

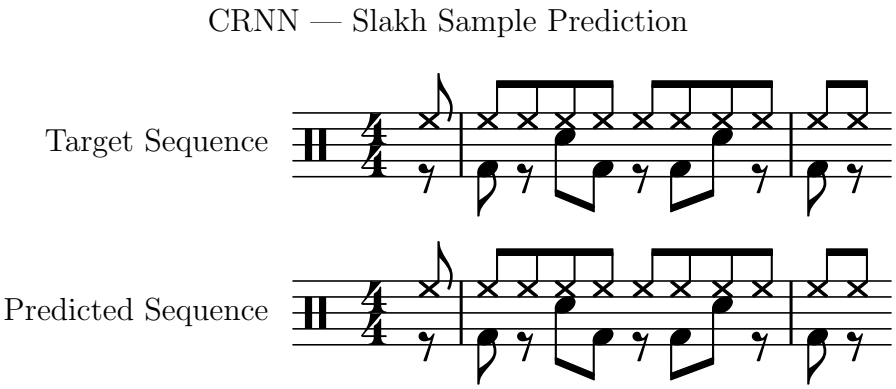

Figure 6.5: Prediction example from the Convolutional Recurrent Neural Network (CRNN) on a randomly selected sequence from the Slakh test split. Both sequences were manually converted from activation function format into drum staff notation. The drum pattern is relatively simple, and the model produces a perfect transcription with all onsets correctly placed and labeled.

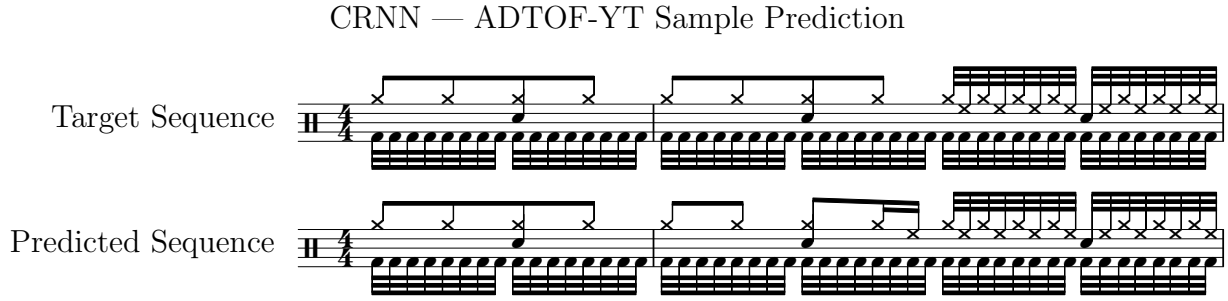

Figure 6.6: Prediction example from the Convolutional Recurrent Neural Network (CRNN) on a randomly selected sequence from the ADTOF-YT test split. Both sequences were manually converted from activation function format into drum staff notation. The pattern is dense, originating from a metal track, yet the model performs remarkably well. Only a single False Positive (FP) is present, with all other onsets accurately transcribed.

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 architecture that consistently outperforms the others for ADT. Instead, performance varies depending on the characteristics of each dataset and the inductive biases inherent to each architectural design.

Firstly, while multiple architectures perform well across different datasets, none emerge as universally superior. No single architecture consistently outperforms the others on all the datasets, several architectures exhibit similarly strong results depending on the dataset.

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 performs strongly, but not exceptionally, on the fourth (ADTOF-YT). The consistency of its high performance across datasets with differing characteristics suggests that it is capable of handling a wide variety of ADT tasks. This robustness appears to hold relatively independently of dataset size or complexity.

One possible explanation for this is the architectural synergy between the convolutional and recurrent layers. The convolutional blocks provide a strong spatial feature extraction over local time–frequency patches, while the recurrent layers enable short-range temporal modeling. Together, these inductive biases may make the CRNN especially well-suited for the challenges of ADT.

The Convolutional Neural Network (CNN) demonstrates moderate performance across all datasets. It achieves adequate performance on the smallest dataset, ENST+MDB, achieving the second-highest F1-score. However, on E-GMD, Slakh and ADTOF-YT, it yields the lowest scores among the architectures, though the relative performance gap in E-GMD is minor.

In terms of absolute F1-scores, the CNN performs reasonably well across all datasets. Nonetheless, it consistently trails behind the other architectures, particularly on the complex and larger datasets like Slakh and ADTOF-YT. The results suggest that the CNN is a less optimal choice for ADT, except perhaps in scenarios with limited data or when applied on simpler tasks, such as DTD or Drum Sound Classification (DSC).

More broadly, the results underscore the importance of explicitly modeling temporal dependencies in ADT. Relying solely on the inductive biases provided by convolutional layers, such as spatially-local feature extraction, does not appear to offer sufficient flexibility to perform optimally across complex ADT tasks.

The importance of temporal modeling is further supported by the performance of the Recurrent Neural Network. The RNN performs surprisingly well across all datasets except the smallest, ENST+MDB. In contrast to the CNN, it demonstrates relatively poor performance on ENST+MDB, but achieves strong results on the remaining three datasets, tying for the highest F1-score on both E-GMD and ADTOF-YT. The RNN's lower performance on ENST+MDB suggests that its inductive biases are weaker than those of convolutional layers, requiring more data to effectively learn the task. Notably, while it matches the CRNN on E-GMD, it outperforms it on ADTOF-YT. This could indicate that the inductive biases introduced by convolutional layers, although powerful, may become overly dominant in certain scenarios, potentially limiting performance. Overall, the RNN proves to be a viable architecture for ADT, particularly when sufficient training data is available.

The convolutional transformer exhibits performance that generally falls between the RNN and CRNN across most datasets. It consistently achieves relatively high results, ranking just below the CRNN on ENST+MDB, E-GMD, and Slakh, while outperforming it on ADTOF-YT. These results suggest that transformer-based models provide sufficient temporal modeling capacity to perform well on ADT tasks. However, the findings also hint that the long-range dependencies more easily captured by attention mechanisms may be less critical than the short-range temporal aggregation offered by recurrent layers in many ADT settings. Still, the convolutional transformer shows strong, consistent performance and can be considered a viable choice for ADT tasks.

Finally, the Vision Transformer (ViT) exhibits a performance trend similar to the RNN. On the smallest dataset, ENST+MDB, it performs notably poorly, yielding the lowest F1-score among all architectures. On E-GMD and Slakh, it achieves competitive, though not leading, results. Its performance is identical to that of the convolutional transformer. On ADTOF-YT, it performs exceptionally well, matching the highest F1-score achieved by the RNN.

These results reinforce earlier observations about the role of architectural inductive bias. The ViT's omission of convolutional layers appears to require a larger volume of high-quality data to perform well. Its underperformance on ENST+MDB further suggests that the inductive bias of attention-based architectures may be even weaker than that of recurrent layers, making them more dependent on large datasets to generalize effectively.

This interpretation aligns with findings from existing literature, where Vision Transformers are often shown to require an extensive amount of training data, or pretraining on large-scale dataset to achieve optimal performance [18]. Future work involving such pretraining could provide a more accurate estimate of the ViT's true potential in ADT.

As shown earlier, the Convolutional Recurrent Neural Network (CRNN) achieves the highest F1-score on most of the datasets. While a high F1-score is a useful aggregate metric, it does not fully capture the qualitative nature of the model's predictions. To provide further insight, Figures 6.3, 6.4, 6.5, and 6.6 present example predictions made by the CRNN on randomly selected sequences from the test split of each dataset. These visualizations offer a more intuitive understanding of model behavior on unseen data and further support our findings.

The ENST+MDB sample prediction (Figure 6.3) is generally accurate, with many onsets correctly identified. However, it also contains several False Positives (FPs) and False Negatives (FNs). Notably, the model consistently misclassifies the Hi-Hat (HH) onsets as Crash and Ride (CC+RC), and missed occasional fast, closely spaced Snare Drum (SD) onsets. Given the small size of ENST+MDB and the complexity of this particular sequence, the prediction remains reasonable overall.

In contrast, the three other sample predictions are remarkably accurate. Two are nearly flawless, while the third is entirely correct. Across these three examples, only a single FP and two FNs are observed. These samples also span a range of difficulty: the Slakh transcription (Figure 6.5) is relatively simple, E-GMD (Figure 6.4) is moderately

complex, and ADTOF-YT (Figure 6.6) is highly dense with a moderate rhythmic intricacy. The CRNN's ability to handle this spectrum of complexity while maintaining a high prediction accuracy, is notable.

Although each model was trained and evaluated within its own dataset domain, these examples provide helpful intuition into how the CRNN's high F1-scores translate to real-world transcription performance.

It is important to acknowledge the potential sources of variability that may influence the results of this study. Factors such as random weight initialization, stochastic hyperparameter sampling, numerical precision limits, and dataset-specific variability (e.g., differences in splits or noise artifacts) can all impact model performance. However, several design choices help mitigate these effects. Training a substantial number of models per experiment, using batch sizes that average out gradient noise, and relying on predefined, independently curated train, validation and test splits all contribute to stabilizing the evaluation. These measures increase the likelihood that the observed performance differences primarily reflect the influence of architecture choice, rather than random variation.

Taking this into account, my results strongly suggest that the Convolutional Recurrent Neural Network (CRNN) is the most effective architecture for ADT across the datasets studied. It achieved the highest micro F1-scores on most benchmarks and produced accurate transcriptions in qualitative visualizations. That said, further research should continue to explore the scalability of both Recurrent Neural Networks (RNNs) and the Vision Transformer (ViT), particularly how they perform on larger datasets and, for the latter, in conjunction with pretraining schemes. Such investigations may offer deeper insight into these models' full potential for ADT tasks.

Chapter 7

Dataset Study

The primary objective of this study is to investigate how different datasets, and their unique characteristics, influence model performance, both within-domain and under Out-of-Distribution (OOD) conditions. This insight can help guide how existing datasets are best utilized or combined, and how future datasets might be intelligently constructed to improve model performance and generalization in Automatic Drum Transcription (ADT) and Drum Transcription in the Presence of Melodic Instruments (DTM).

To isolate the effect of dataset composition, all models in this study use the Convolutional Recurrent Neural Network (CRNN) architecture, identified in the previous chapter as the most effective for ADT.

7.1 Methodology

I train CRNN models on various combinations of the four main datasets: ENST+MDB, E-GMD, Slakh, and ADTOF-YT. Dataset combinations are treated as unions, with one training epoch corresponding to a full pass over all included datasets. Each trained model is then evaluated not only on the test split(s) of the dataset(s) it was trained on, but also on all other dataset test sets, including the unseen SADTP dataset, to measure OOD generalization.

To strike a balance between coverage and feasibility, I strategically select 10 dataset combinations. These are chosen to provide meaningful insight into how different dataset properties affect generalization, while avoiding redundant experiments with diminishing returns.

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: Micro F1-scores for a Convolutional Recurrent Neural Network trained on various dataset combinations and evaluated across all datasets. **Bold** values represent the highest score achieved for that dataset. Underlined values represent the highest score achieved in Out-of-Distribution (OOD) evaluation for that dataset. Cells shaded in light blue indicate OOD evaluations.

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 [10]	0.77/×*	0.83	×	×
MT3 (mixture) [21]	×	×	0.76	×
YPTF.MoE+M [14]	0.87/×**	×	0.85	×
ADTOF-RGW + ADTOF-YT [65]	0.78/0.81***	×	×	0.85

Table 7.2: Micro F1-scores for selected Convolutional Recurrent Neural Network models from this dataset study, compared with the best-performing models reported in related literature. **Bold** values represent the highest score achieved for that dataset. Cells shaded in light blue indicate OOD evaluations.
(*) Callender et al. evaluate only on ENST-Drums, using isolated drum stems in a DTD setting and a different test split [10].
(**) Chang et al. evaluate only ENST-Drums, also using a different test split [14].
(***) Zehren et al. report separate scores for ENST-Drums and MDB Drums using different test splits [65].

7.3 Discussion

The results from the dataset study, summarized in Table 7.1 reveal important insights into how dataset composition affects model generalization. Notably, these findings demon-

strate that strategically combining ADT datasets can significantly improve both within-domain and Out-of-Distribution (OOD) generalization performance.

Firstly, it is evident that each dataset yields strong performance on test splits drawn from the same distribution as its training data, but substantially lower performance under OOD conditions. In other words, all models exhibit a noticeable *transfer gap*, a decline in performance when evaluated outside their training domain. This behaviour suggests that the datasets differ sufficiently in their characteristics to introduce meaningful domain shifts, which in turn hinder generalization.

This generalization penalty appears relatively consistent across most datasets, with the exception of models trained on E-GMD (discussed in the next paragraph). Most of the best-performing OOD results fall within a micro F1-score range of 0.6 to 0.7, underscoring the difficulty of cross-domain generalization in ADT. A notable outlier is the ENST+MDB test set, where the model trained on Slakh+ADTOF-YT achieves a remarkably high OOD F1-score of 0.86, nearly matching its best within-domain score of 0.88.

I hypothesize that this unusually small transfer gap can be attributed to the exceptional annotation quality of ENST+MDB. Unlike the other datasets, which may contain alignment errors or inconsistencies, ENST+MDB consists of highly accurate and precisely aligned DTM annotations. This likely makes it easier for models trained on other datasets to demonstrate their learned capabilities when evaluated on ENST+MDB, whereas noisier test sets might obscure such strengths. As a result, performance on ENST+MDB may reflect model competence more transparently than other datasets.

Another important observation concerns the E-GMD dataset. Unlike the others, E-GMD is a Drum Transcription of Drum-only Recordings (DTD) dataset containing only isolated drum stems, without any melodic accompaniment. Despite its large size, models solely trained on E-GMD generalize poorly to other datasets. In fact, it performs worst across all OOD evaluations, with F1-scores as low as 0.28 when evaluated on SADTP. This highlights a significant transfer gap between DTD and DTM tasks, suggesting that training on isolated drum tracks does not readily transfer to full musical mixtures.

However, it is worth noting that E-GMD achieves the highest within-domain performance when evaluated on its own test set. This contrast reinforces the distinction between DTD and DTM as separate problem formulations. It also suggests that models trained on one task may not be directly applicable to the other.

Building on this observation, we can consider this reverse relationship, how well models trained on DTM datasets perform on the DTD task. Interestingly, the OOD performance of models trained on DTM datasets when evaluated on E-GMD is comparable to that their performance on other unseen DTM datasets. This supports the hypothesis that DTM may encompass DTD in terms of task complexity. In other words, while DTM poses an additional challenge due to the presence of melodic accompaniment, it may equip models with more generalizable skills, allowing them to perform reasonably well on DTD task without explicit exposure.

I argue that this asymmetry stems from the structural relationship between the two tasks. DTD can be seen as a strict subset of DTM, since DTM datasets often include passages where drums play in isolation (e.g., drum solos), whereas DTD datasets never include accompaniment. A model trained on DTM can thus learn to transcribe drums both in isolation and in mixture, while a model trained solely on DTD lacks exposure to the contextual variability needed for generalization to DTM. This may explain why DTM-trained models exhibit some degree of zero-shot generalization to DTD, whereas the reverse does not hold.

One result that merits further discussion is the observed correlation between high performance on ADTOF-YT and high OOD performance on SADTP. Models that perform well on the ADTOF-YT test split also tend to achieve relatively high F1-scores when evaluated on SADTP. Given the crowdsourced nature of ADTOF-YT and the fact that SADTP consists of real-time performed contemporary music tracks, there may be a degree of distributional overlap that explains this trend.

However, I hypothesize that generalization ability does not arise from direct overlap in audio content, but rather from the heterogeneity inherent in the ADTOF-YT dataset. Because it was constructed through crowdsourcing, ADTOF-YT includes recordings from a broad range of sources, audio quality, and playing styles. This diversity likely results in a more varied training distribution, one that better captures the domain complexity found in real-world, user-generated datasets like SADTP.

By contrast, the other datasets used in this study are significantly more homogeneous. ENST+MDB consists of a small group of professional drummers recorded in controlled, studio environments. E-GMD and Slakh are both fully synthesized, lacking the variability of live performances or real acoustic conditions. This domain uniformity likely limits their ability to generalize, especially when transitioning from synthetic or curated domains to more realistic ones.

This interpretation aligns with prior findings on the so-called synthetic-to-real transfer gap, a well-documented challenge in both ADT and other fields [66]. Based on these observations, I argue that dataset heterogeneity, particularly when achieved through crowdsourcing, is an invaluable asset for improving model generalization. Providing that data quality is maintained, crowdsourcing offers a practical and effective means of capturing real-world variability, and should be prioritized in the construction of future ADT and DTM datasets.

An important observation is that the best-performing models in this study, across nearly all DTM datasets, are those trained on combinations of multiple datasets rather than on a single dataset. For example, in Table 7.1, expanding ADTOF-YT with Slakh increases its within-domain micro F1-score from 0.93 to 0.97. Similarly, expanding ENST+MDB with both Slakh and ADTOF-YT leads to a more significant within-domain performance increase from 0.81 to 0.88. The exceptions are the E-GMD and Slakh datasets, where the highest micro F1-scores are achieved when trained solely on the respective dataset. This suggests that, in many cases, expanding the training data to include additional datasets enhances within-domain performance.

This finding might seem counter-intuitive: adding data from other domains introduces distributional variation, which one might expect to dilute the relevance of the training data for the target domain. However, the results show that this dilution does not necessarily harm performance, but often improves it. Broader training data appears to help the model learn more generalizable features, even when evaluated on its source domain.

This benefit, however, is not universal. In some cases, performance declines when additional datasets are introduced. For example, the ADTOF-YT dataset achieves its highest micro F1-score (0.97) when trained jointly with Slakh, but performance drops to 0.94 when ENST+MDB is added, and further to 0.93 when E-GMD is included. This suggests that expanding the dataset distribution can occasionally reduce on-domain performance, especially when the added data differs strongly in characteristics.

Interestingly, the inclusion of datasets of less complexity, such as DTD tasks through E-GMD, does not substantially impair OOD generalization. While models trained on DTM datasets may see a slight drop in within-domain accuracy when augmented with DTD data, their performance on other domains remains relatively stable. This indicates that heterogeneity in training data can still be beneficial for generalization, even when the added data stems from a simpler task formulation.

Secondly, take a look at the model trained on the largest dataset combination, using all available datasets. While it does not achieve the highest micro F1-score on any single test set (except for Slakh, where it matches the top result), it performs consistently well across all evaluations. This supports the idea that increasing dataset size tends to improve a model's ability to generalize, in line with findings from prior work [65].

Based on this, I suggest that future ADT datasets should aim to include a wide variety of examples, as broader training data often helps models generalize better. This also reinforces my earlier argument about the value of crowdsourced data, highlighting how diversity in the training domain can lead to more robust performance. That said, if the goal is to maximize performance in a specific domain, it might still be beneficial to tailor the training data more carefully to match the intended use case.

Lastly, in Table 7.2, I compare my best-performing models with results reported in other literature. There isn't much prior work that evaluated these specific datasets, so this comparison offers some useful perspective.

Gardner et al.'s MT3 [21] and Chang et al.'s YPTF.MOE+M [14] are general-purpose AMT models designed to transcribe multiple instruments beyond just drums. Both report results using the Onset F1-score, which is equivalent to the micro F1-score used in this thesis. Zehren et al.'s ADTOF-RGW + ADTOF-YT model [65] is more directly comparable, as it is trained specifically for DTM.

Across E-GMD, Slakh, and ADTOF-YT, my models outperform these previous approaches by a clear margin. The Slakh+ADTOF-YT model in particular stands out, reaching an F1-score of 0.97 on ADTOF-YT. For ENST+MDB, the comparison is less straightforward due to the different dataset splits and setups, often keeping ENST-Drums and MDB Drums both disjoint datasets and in-full, using the whole dataset as a test split. However, disregarding that, my best-performing model, ENST+MDB+Slakh+ADTOF-YT, can be said to seem comparable in ENST+MDB performance to that of Chang et al.'s YPTF.MOE+M [14].

Altogether, I argue that these results support the claim that combining well-selected datasets with a suitable architecture leads to models that perform at or above the current state of the art within ADT and DTM. Comparing my models to those in related work helps put both their performance and mine into perspective, and demonstrates that the dataset choices and training strategies I've employed make a meaningful impact.

As in the previous study, it is important to acknowledge the potential sources of variability that may influence these results. Factors such as random weight initialization,

stochastic hyperparameter selection, and dataset-specific artifacts can all impact model performance. However, the methodological choices applied, training multiple models per experiment, using consistent data splits, and averaging over batches, help reduce the influence of such randomness.

It should also be noted that all models in this study use the Convolutional Recurrent Neural Network (CRNN) architecture, which was selected based on its performance in the previous architecture study. While this choice was made to isolate the effect of dataset composition on an architecture proven effective for the ADT task, it inevitably introduces a form of architectural bias. Different architectures may interact with the datasets in different ways, potentially leading to different generalization behaviours. Nonetheless, given the CRNN's strong and consistent performance across datasets, I argue that the results observed here still provide meaningful insight into how dataset choices influence performance and generalization in ADT.

In summary, I conclude that the results strongly support the hypothesis that combining datasets with diverse characteristics, particularly larger, more varied, and ideally crowdsourced datasets, significantly improves model performance on ADT tasks. When the domain and complexity of the training data match the task at hand, models not only perform better within-domain but also generalize more effectively to Out-of-Distribution (OOD) data. Paired with a well-suited architecture, such as the CRNN, this approach yields models that compete with or exceed the current state of the art. Finally, these findings also suggest a hierarchical relationship between ADT tasks: models trained on DTM data generalize reasonably well to DTD, but not necessarily the other way around.

Chapter 8

Conclusion

This thesis set out to investigate how to achieve optimal generalization in Automatic Drum Transcription (ADT), with a particular focus on Drum Transcription in the Presence of Melodic Instruments (DTM), both within-domain and in Out-of-Distribution (OOD) evaluation. To address this, I explored how different deep learning architectures, both established and novel, and varying dataset compositions affect model performance across a diverse set of datasets. The goal was to understand how specific design choices in architecture and data influence a model's ability to generalize to both familiar and unseen input.

In the first study, I trained and compared five deep learning architectures: a Recurrent Neural Network (RNN), Convolutional Neural Network (CNN), Convolutional Recurrent Neural Network (CRNN), Convolutional Transformer, and Vision Transformer (ViT). Each was trained and evaluated on separate datasets to assess within-domain generalization. The Convolutional Recurrent Neural Network emerged as the most effective overall, achieving the highest micro F1-scores on three out of four datasets. However, the Recurrent Neural Network and Vision Transformer each achieved top performance on the remaining dataset, suggesting that these architectures may scale well with larger or more diverse training data and warrant further investigation.

In the second study, I explored how combining datasets with different characteristics influences a model's ability to generalize. The results showed that dataset combinations substantially improve both within-domain and Out-of-Distribution (OOD) performance. Comparison with related literature further demonstrated that, when paired with a suitable ADT architecture, these dataset choices can produce models with performance that surpasses the current state of the art.

That said, the results also emphasize the importance of careful dataset construction. While increasing the amount of data generally improves performance, the specific properties of that data matter. In particular, I highlighted the value of covering a wide domain and the benefits of crowdsourced datasets, which offer a heterogeneity that improves generalization.

Finally, the findings suggest a hierarchical relationship between different ADT tasks. More complex tasks, such as DTM, appear to encompass simpler ones like DTD, enabling models trained on the former to generalize well to the latter. However, the reverse does not hold, as models trained on simpler tasks struggle to generalize to more complex ones.

Reflecting back on the questions raised in the introduction:

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

   The Convolutional Recurrent Neural Network consistently achieves strong performance across a variety of ADT and DTM datasets. It stands out as the best-performing architecture for learning the ADT task and for generalizing within-domain.

2. **What makes an ADT dataset optimal for training models to generalize on DTM?**

   Training on large, DTM-specific datasets that span a wide range of musical styles and characteristics, especially those derived from crowdsourced sources, is key to improving generalization. Additionally, aligning the complexity of the dataset with that of the task helps ensure that the models learn appropriate representations and generalize effectively to Out-of-Distribution (OOD) data.

This thesis contributes not only by comparing different architectural choices for DTM tasks, but also by evaluating how various ADT datasets affect model generalization, both on existing public datasets and on a novel dataset I composed and transcribed specifically for this thesis, SADTP [20]. These contributions offer valuable insight into ADT, emphasizing the importance of both architecture selection and thoughtful dataset construction.

Future work should expand on this research by conducting more rigorous architectural analyses on larger datasets. In particular, further investigation is needed into how

Recurrent Neural Networks (RNNs) and Vision Transformers (ViTs) scale with increased training data, and how pretraining a ViT affects its final performance. This should also be extended to include evaluating the models' ability to generalize in OOD settings.

Lastly, this thesis demonstrates that deep learning models can achieve strong generalization performance on ADT and DTM tasks through deliberate choices in both architecture and dataset, and provides a clear direction for future research in the field of Automatic Drum Transcription.

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 Transform.

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] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The journal of machine learning research*, 13(1):281–305, 2012.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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/.

[18] 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.

[19] Runar Fosse. Automatic drum transcription: Optimizing generalization through architectural and dataset choices, 2025.
URL: https://github.com/RunarFosse/master-thesis.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.

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

[28] 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`.

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

[30] 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.

[31] 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.

[32] 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.

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

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

[35] 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`.

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

[37] 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.

[38] 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.

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

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

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

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

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

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

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

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

[47] 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.

[48] 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.

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

[50] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[51] 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`.

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

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

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

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

[56] 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`.

[57] 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`.

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

[59] 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.

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

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

[62] 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.

[63] 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.

[64] Mickael Zehren, Marco Alunno, and Paolo Bientinesi. Adtof datasets, November 2023.
URL: https://doi.org/10.5281/zenodo.10084511. Accessed: 23.10.2024.

[65] 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.

[66] 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.