University of Bergen
Department of Informatics

Automatic Drum Transcription using Deep Learning

Author: Runar Fosse
Supervisor: Pekka Parviainen

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

June, 2025

Abstract

Write proper abstract

Lorem ipsum dolor sit amet, his veri singulis necessitatibus ad. Nec insolens periculis ex. Te pro purto eros error, nec alia graeci placerat cu. Hinc volutpat similique no qui, ad labitur mentitum democritum sea. Sale inimicus te eum.

No eros nemore impedit his, per at salutandi eloquentiam, ea semper euismod meliore sea. Mutat scaevola cotidieque cu mel. Eum an convenire tractatos, ei duo nulla molestie, quis hendrerit et vix. In aliquam intellegam philosophia sea. At quo bonorum adipisci. Eros labitur deleniti ius in, sonet congue ius at, pro suas meis habeo no.

Acknowledgements

Est suavitate gubergren referrentur an, ex mea dolor eloquentiam, novum ludus suscipit in nec. Ea mea essent prompta constituam, has ut novum prodesset vulputate. Ad noster electram pri, nec sint accusamus dissentias at. Est ad laoreet fierent invidunt, ut per assueverit conclusionemque. An electram efficiendi mea.

Write proper acknowledgements

`Runar Fosse`
Wednesday 18th June, 2025

Contents

1 Introduction **5**

1.1 Thesis statement 6

2 Background **7**

2.1 Automatic Drum Transcription 7

2.2 The Drum Set 8

2.3 Transcription Task 9

2.4 Audio 10

2.4.1 Fourier Transform 10

2.4.2 Discrete Fourier Transform 11

2.4.3 Nyquist frequency 12

2.4.4 Fast Fourier Transform 13

2.4.5 Short-time Fourier Transform 13

2.4.6 Spectrogram 14

2.4.7 Filters 15

2.5 Transcription 16

2.5.1 Sheet Music 16

2.5.2 MIDI Annotations 16

2.5.3 Activation Functions 17

2.6 Performance Measure 18

2.6.1 Correct Predictions 18

2.6.2 Accuracy 19

2.6.3 F1-score 19

2.6.4 Micro vs. Macro 20

3 Architectures **21**

3.1 Recurrent Neural Network 21

3.1.1 Implementation 22

3.2 Convolutional Neural Network 23

3.2.1 Implementation 23

3.3 Convolutional RNN 24

3.3.1 Implementation 25

3.4 Convolutional Transformer 26

3.4.1 Implementation 26

3.5 Vision Transformer 28

3.5.1 Patch Embedding 28

3.5.2 Architecture Modifications 29

3.5.3 Implementation 29

4 Datasets **31**

4.1 ENST+MDB 31

4.1.1 Splits 32

4.1.2 Mapping 32

4.2 E-GMD 33

4.2.1 Mapping 33

4.3 Slakh 33

4.3.1 Mapping 34

4.4 ADTOF-YT 34

4.4.1 Mapping 35

4.5 SADTP 35

4.5.1 Mapping 36

4.6 Differences 36

5 Methodology **37**

5.1 Data Preparation 37

5.1.1 Audio Files 37

5.1.2 Annotations 38

5.1.3 Splitting and Storing 39

5.2 Preprocessing 39

5.3 Training 40

5.4 Postprocessing 41

5.5 Model Selection 41

5.6 Hyperparameter Tuning 42

5.6.1 Search Strategies 42

5.6.2 Hyperparameters 43

6 Architecture Study **44**

6.1 Methodology 44

6.2 Results 45

6.3 Discussion 46

7 Dataset Study **49**

7.1 Methodology 49

7.2 Results 49

7.3 Discussion 50

8 Conclusion **51**

List of Acronyms and Abbreviations **52**

Bibliography **54**

Chapter 1

Introduction

Within the field of Music Information Retrieval (MIR), the task of Automatic Music Transcription (AMT) is considered to be a challenging research problem. It describes the process of generating a symbolic notation from audio. The majority of instruments are melodic, where key information for transcription would be to discern pitch, onset time, and duration. This stands in contrast to percussive instruments, where instead of pitch and duration one would focus on instrument classification and onset detection. This sets the stage for Automatic Drum Transcription (ADT), which is a subfield of AMT, specifically focusing on drums and percussive instruments. [44]

Previously, a popular approach to ADT was using signal processing, which later developed into using classical machine learning methods [44]. However in later years, deep learning has shown to be quite effective. Therfore, the recent focus of most authors has been to find the best performing deep learning approaches by either; constructing and analysing the best performing model architectures, or by finding datasets which allow models to generalize the best. [46]

Provide a good introduction into the master thesis, mentioning AMT, ADT and why deep learning is suited for such a task.

Also shortly mention how we represent the sound, and the transcriptions. What is/how do we do ADT

1.1 Thesis statement

This leads us to two primary questions. Which deep learning architecture is the best suited for solving a task like this? And, what makes a dataset optimal by making models generalize? These are two of the questions we will try to answer in this thesis.

For the former, we will train different model architectures on different, well-known ADT datasets. Specifically, recurrent neural networks, convolutional neural networks, convolutional-recurrent neural networks, convolutional transformers and, novel to the field of ADT, vision transformers. By comparing their performances we could be able to gauge the one best suited for an ADT task.

For the latter, we will select the best performing model architecture from the first question, and train it over several different combination of the ADT datasets. By performing zero-shot evaluations, we could analyse and figure out what makes a good ADT dataset and how it would supplement a suitable model architecture. Here, I've also introduced a new, small ADT dataset containing drum transcriptions of recent, contemporary musical compositions.

Present the aim of the thesis here. And the questions! How do we train a model capable of solving such a task at a high performing level. More specifically:

Remember the concrete What do we want to figure out.

Chapter 2

Background

2.1 Automatic Drum Transcription

As mentioned, ADT describes the task of transcribing symbolic notation for drums from audio. To be even more descriptive, ADT can be split into further tasks. From least to most complex we have: Drum Sound Classification (DSC), where we classify drum instruments from isolated recordings. Drum Transcription of Drum-only Recordings (DTD), where we transcribe audio containing exclusively drum instruments. Drum Transcription in the Presence of Additional Percussion (DTP), where we transcribe audio containing drum instruments, and additional percussive instruments which the transcription should exclude. Finally, we have Drum Transcription in the Presence of Melodic Instruments (DTM), which describes the task of drum transcription with audio containing both drum, and melodic instruments. [44]

In this thesis, we will focus on the most complex of these, namely DTM. Intuitively, we want to develop a deep learning model which, given input audio, has the ability to detect and classify different drum instrument onsets (events), while selectively ignoring unrelated, melodic instruments.

This task comes with difficulties not seen in the less complex tasks. Zehren et al. [46] describes one example, in where *"melodic and percussive instruments can overlap and mask eachother..., or have similar sounds, thus creating confusion between instruments"*.

Deep learning has shown to be a promising method to solve such a task, and several different approaches have been tried, many with great success. Vogl et al. [42, 41]

displayed good results with both a convolutional, and a convolutional-recurrent neural network. Zehren et al. [46, 47] focused on datasets, showing that the amount of data and quality of data are equally important to get good performance. Most recently, Chang et al. [10] explored an autoregressive, language model approach. This approach explored multi-instrument transcriptions, but their results on ADT were notable.

This reinforces the fact that there still exist many approaches to attempt, which could lead to a general improvement on ADT models.

2.2 The Drum Set

The drum set is a collection of percussive instruments like different drums, cymbals, and possibly different auxillary percussions. A drum set can vary in what it is composed of, however a standard kit usually consists of a snare drum, a bass drum, one or more tom-toms (toms), one or more cymbals (crash and ride), and a hi-hat cymbal [30].

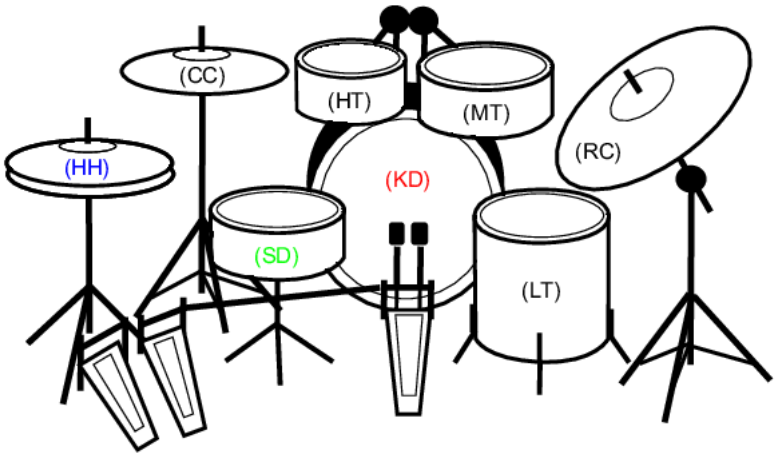

Figure 2.1: Example of the different instruments on the drumset.

As mentioned, percussion like the drum set, stands in contrast to other musical instruments in that the different ways of playing the same instrument often differ a lot in their *"audible footprint"*. The snare drum, bass drum and hi-hat all have quite different timbres, frequency span, volume, and all in all fundamentally are different instruments.

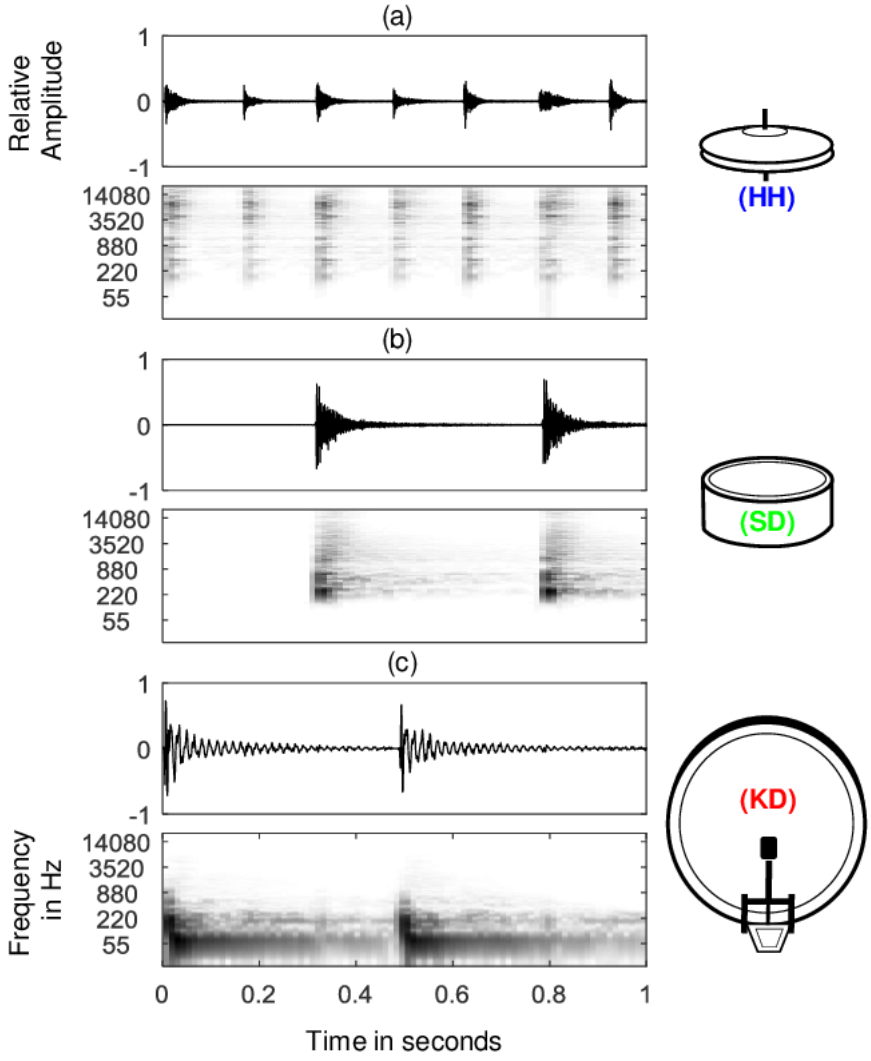

Figure 2.2: Example of the different audible footprint for drum set percussion.

Mention the different drum set instruments. Mention how they all have different musical properties, like frequency, timbre, etc (show waveforms maybe?). Also mention the most fundemental ones, and how Bass, snare and hi-hat are more important than e.g. the mid-tom or something.

2.3 Transcription Task

An introduction to the task, with an image overview over the pipeline (Waveform → Spectrogram → Neural Network → Activation Functions → Transcription) should be done here, and is natural to give the reader a thorough understanding of the task before any other information. UNDERSTANDING THE TASK IS THE MOST IMPORTANT PART!

2.4 Audio

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

Thus sound can be described as the propogation and perception of waves. Mathematically, waves can be studied as signals [9]. To represent these sounds digitally, as *audio*, one can express these waves as a signal, giving rise to the *waveform*. The waveform is a representation of a signal as a graph, and charts the amplitude, or strength of the signal, over time.

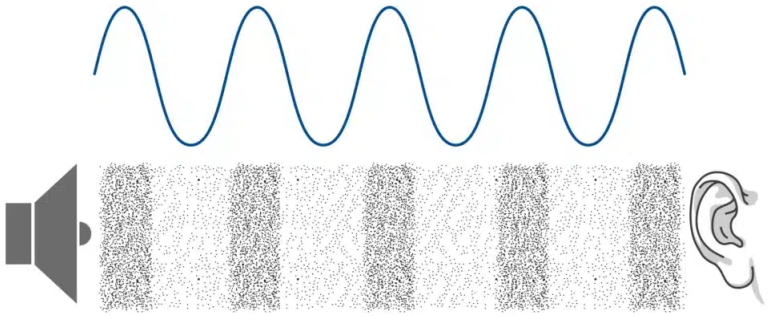

Figure 2.3: Soundwave to waveform relationship

For monophonic sound, this waveform is a one-dimensional representation. Even though this is an excellent way of storing audio digitally, it is very compact. There have been deep learning models working directly with these waveforms, e.g. Oord et al.'s WaveNet [38], however the task of parsing and perceiving such a signal is a complex one.

2.4.1 Fourier Transform

The Fourier Transform is a mathematical transformation which, given a frequency, computes its significance, or intensity, in a given signal. As we've established, audio is

represented as a signal, and we can therefore use this transform to turn this audio signal into frequency space.

The fourier transform is a complex transformation. 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 ξ, resulting in a *complex* number. This number consists of a *real* part and an *imaginary* part. The real part consists of the amplitude of a certain frequency, where as the imaginary part consists of the phase. This information is what allows us to, for a given signal, figure out which frequencies it is made out of and how much each frequency contributes.

By doing such a transform, we turn our temporal data into spectral data. This initively *untangles* our signal into its respective base frequencies. Such an transformation could lessen the complexity of the task, making *understanding* of audio easier.

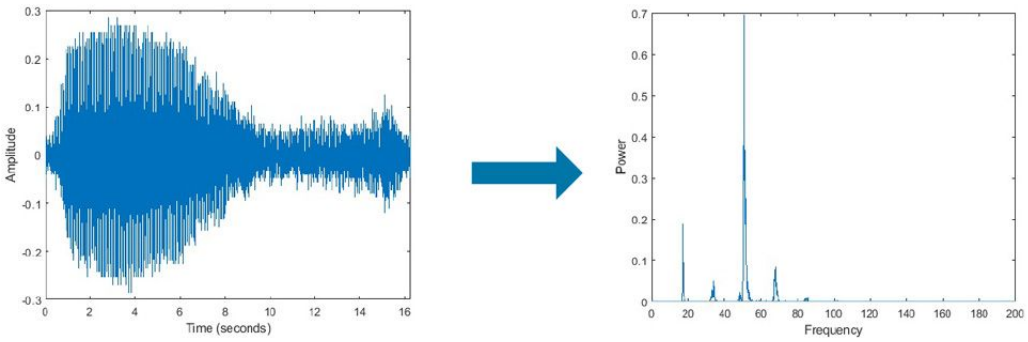

Figure 2.4: Application of a Fourier Transform

Note that the Fourier Transform is invertible, meaning that, given information about each frequency, we can perform a similar integral and reconstruct the original signal. In signal processing, this property is exploited heavily.

2.4.2 Discrete Fourier Transform

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

To extract frequency values from these signals, we instead have to use the Discrete Fourier Tranform (DFT). Intuitively this works as the normal Fourier Transform, but ported to work on discrete-valued 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 the frequency and N the number of discrete samples.

Add a example figure of FT vs DFT

2.4.3 Nyquist frequency

When we discretize a signal, e.g. when going from continuous audio waves in the air to discrete audio signals on a computer, we could lose some information. The discrete representation of the signal is an *approximation* which quality is directly dependent on the sampling rate. The higher the sampling rate, the *closer* we are to the original, continuous signal. However a higher sampling rate comes at the cost of needing to store these signals at a higher precision. A lower sampling rate would need less information stored, but this could also mean a less precise signal approxmation.

Aliasing is the phenomena where new frequencies seem to emerge in undersampled signals. For a given discrete signal, the *Nyquist frequency*, equal to half the sampling rate, is the maximum frequency a signal accurately can represent. Thus to prevent aliasing, one would need to store a signal with a sampling rate of at least double the maximum frequency.

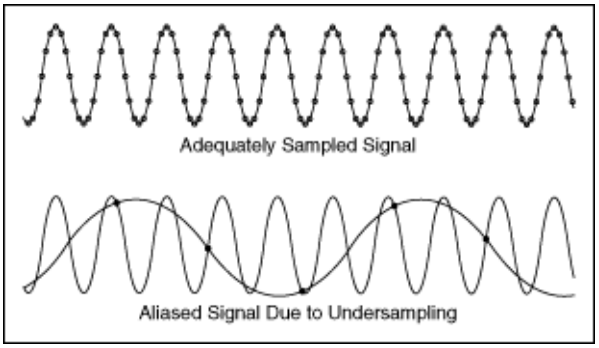

Figure 2.5: Example of aliasing in an undersampled signal.

Regarding the DFT, it here directly follows that the maximum frequency we accurately could extract information about is proportional to the sampling rate of the signal.

2.4.4 Fast Fourier Transform

Keen-eyed computer scientists may have spotted that the DFT runs in $\mathcal{O}(n^2)$ time as we, for every frequency in the range $[0, N]$ have to sum over N different values. In other words, the DFT algorithm scales quite poorly. Take into account that the standard sampling rate for audio is 44.1kHz, i.e. 44100Hz, then we can see that the DFT could be inefficient. [3]

The Fast Fourier Transform (FFT) is an algorithm which solves this problem, and instead computes the DFT of a signal within $\mathcal{O}(n \log n)$ time. Described by Gilbert Strang as *"the most important numerical algorithm of our lifetime"* [37], this practically solves our scaling problem, and allows us to efficiently extract spectral information from a signal regardless of sampling rate.

There exist many different implementations of the FFT. However the Cooley-Tukey algorithm is by far the most used FFT and optimizes calculations through a *divide and conquer* approach, utilizing previous calculations to compute others. [12]

2.4.5 Short-time Fourier Transform

The Fourier Transform comes with some drawbacks, notably how by moving from time space into frequency space, we lose temporal information. For certain tasks this might be sufficient, but the temporal dimension is vital when working with transcriptions and ADT tasks. We've seen how the Fourier Transform computes the frequencies of a signal, but what happens if we had applied the same transform to smaller, *partitions* of a signal.

This leads us to the Short-time Fourier Transform (STFT). By instead of transforming the whole signal, we transform smaller *windows*, we could gain insight into the frequency space while keeping temporal information relatively intact. This turns our data from being one-dimensional into two-dimensional, giving us insight into the intensities of different frequencies, along different timesteps.

Talk more about the partitioning. The window functions applied, and why. Spectral leakage..

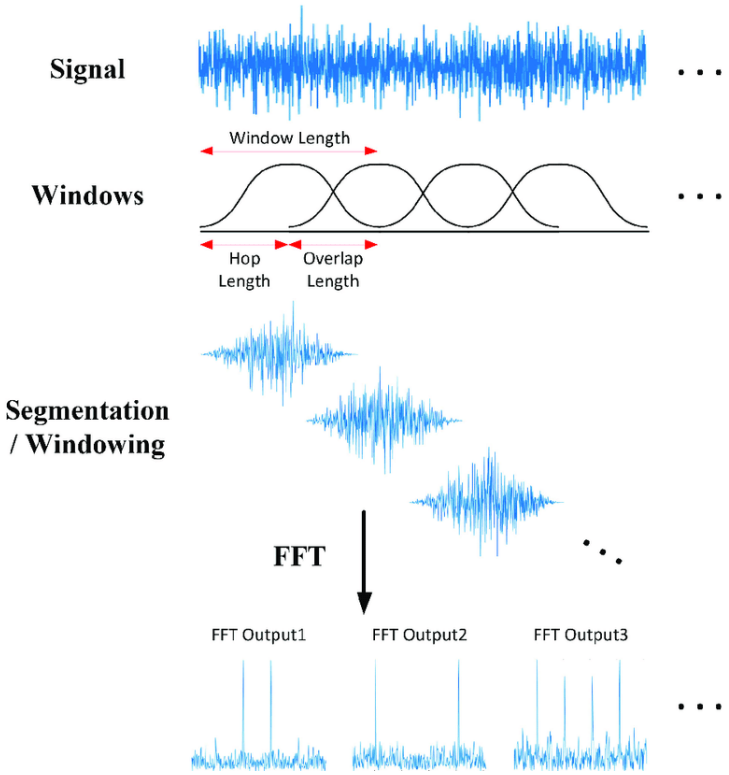

Figure 2.6: Example of the STFT

2.4.6 Spectrogram

The STFT, as the standard Fourier Transform, returns the data as complex values. To turn these into strictly real values without discarding data, we could compute the spectrogram. This is done by squaring the absolute value of each complex number.

This results in a 2-dimensional, real representation of our signal. A representation like this is equivalent to an *image*, but can also still be modelled as a time series. In this way, we've converted our audible information into visual information. Naturally, these spectrograms can be visualized using e.g. a heatmap.

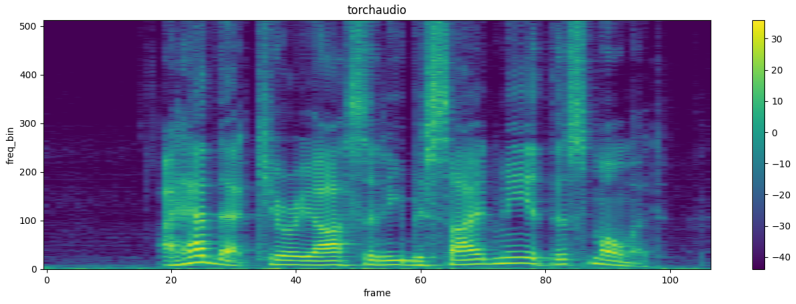

Figure 2.7: Heatmap of an audio spectrogram.

One drawback about the spectrogram is that it contains no information about the phase of the signal it represents. That means it will not be possible to reverse the process and recreate the exact original signal. However, one could try to create an approximation like is done with the Griffin-Lim algorithm [19].

2.4.7 Filters

Signal frequencies and human perception have a special relationship. We humans percieve logarithmic differences in frequencies as a linear difference in pitch, and we tend to be better at distinguishing differences in lower frequencies than higher. E.g., the notes A_2 and B_2 have the same perceptual pitch difference as D_7 and E_7, even though their difference in frequency, $B_2 - A_2 \approx 13.471$Hz and $E_7 - D_7 \approx 287.703$Hz, are vastly different. As the frequency bins in a spectrogram are linearly spaced, this leads to the spectrogram not representing each frequency equally compared to our perception.

To solve this, we can filter the spectrogram into different bins, more suited to represent our perception of sound. This filtering is done by matrix multiplying our spectrogram with a *filterbank*; a matrix representation of different filters.

Mel Spectrograms

The mel scale, presented by Stevens, Volkmann, and Newmann in 1937, is a transformation from the frequency scale to the mel scale. These mels have the property such that a linear difference in mels are percieved as linear differences in pitch. Application of mel-filters result in the *mel spectrogram*, and are widely used when dealing with audio in machine learning, and successful applications have been seen in AMT. [43, 15, 10, 44, 18, 47]

Logarithmic Filters

The mel scale was created to mimic human perception of sound, however within ADT there is a different trend. By instead using logarithmically spaced filters, centered on the note A_4, we get a *logarithmically filtered spectrogram*. Intuitively one could assume this, instead of mimicing human perception, ports the spectrogram into a format preserving musical relationship and information. This seems to be a standard for ADT and has been used extensively by the likes of Vogl et al. [44, 42, 41, 46]

Add a example figure of Spectrogram vs Mel vs Logarithmic

2.5 Transcription

Transcription refers to a process in which we convert information from an audible format, like music, to another medium. This medium then contains a *description* of said audio. As we focus on a musical context, there are a few notable such mediums.

Explain what a transcription is, and what formats they usually are on. Explain what our model is predicting.

2.5.1 Sheet Music

Sheet music is a written transcription using musical notation that, for a given instrument, contains the *recipe* for a musician to play parts of the original recording. This is the standard when it comes to printing arrangements, and is extensively used by musicians.

Sheet music is typically descriptively exhaustive, and could contain information about musical properties like instrument onsets, tempo, velocity, etc.

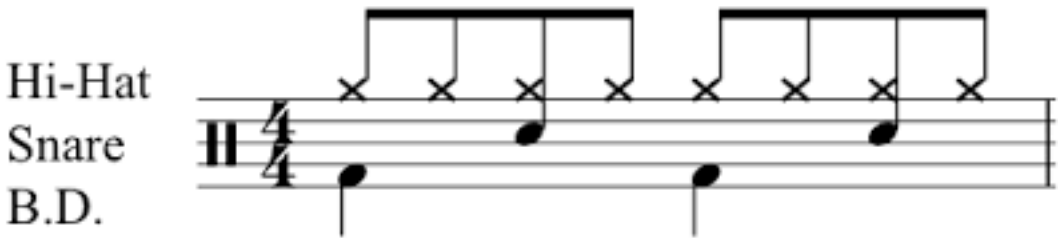

Figure 2.8: Example sheet music for a drumset

2.5.2 MIDI Annotations

Musical Instrument Digital Interface (MIDI) is the industry standard for handling music digitally. It is a binary format, containing sequences of commands that allow digital interfaces to *synthesize* music. As it is binary, it is unreadable to us humans without translating it into another format. When computers play MIDI arrangements, the MIDI sequences are parsed at a constant speed, playing different sounds through *note on*/*note off* events, delayed by time *deltas*. Similar to sheet music, MIDI is also very descriptive.

And one could say that, intuitively, MIDI is to a computer what sheet music is to a musician.

Recently, outputting transcriptions in a MIDI-like format has been attempted in DTM, and has shown to be promising. Utilizing a sequence-to-sequence Natural Language Processing (NLP) approach, Gardner et al. presented MT3 [15], a model inputting spectrograms and outputting MIDI events autoregressively. This format was expanded on by Chang et al.'s YourMT3+ [10], using a Large Language Model (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 2.9: Example MIDI arrangement in a readable format

Accuracy ...

2.5.3 Activation Functions

In machine learning, the task of detecting instrument onsets could be described as a multi-label sequence labeling task. This involves, for each timeframe in a sequence, predicting a probability, or rather confidence value, that a certain instrument onset happens. In the domain of MIR and AMT, it has become common place to describe these confidence distributions as *activation functions*; not to be confused with the general deep learning term, activation functions like ReLU or sigmoid. [35, 42]

This way of frame-level prediction is extensively used within onset detection in ADT and is the approach we will be taking in this thesis.

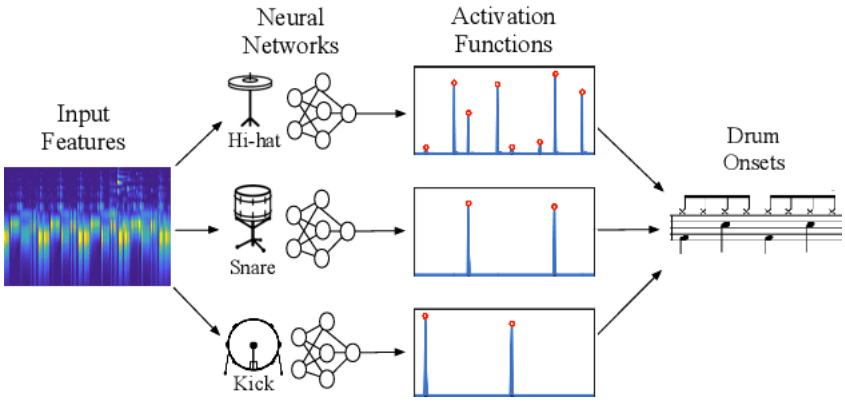

Figure 2.10: Example of ADT activation function output

Peak-picking

When predicting activation functions, we need a separate post-processing step to turn these confidence distributions into onset events. By utilizing a standard *peak-picking* algorithm, we can isolate and enhance peaks in these activation functions, and go from a continuous distribution to a collection of discrete events.

The peak-picking algorithm, introduced in its current form by Böck et al. [5], defines that a prediction $\hat{y}_n$ at timeframe n is a *peak* if it fulfills the three conditions:

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

For appropriately trained deep learning models, Vogl et al. [42] showed that the peak-picking parameters which gave the best results were $m = a = w = 2$ and $\delta = 0.1$.

2.6 Performance Measure

2.6.1 Correct Predictions

Our machine learning models predict instrument onset events on a frame-level basis. In other words, are predictions are very granular, and we need some way to decide when a

prediction is correct versus incorrect. In ADT, a standard has become to allow a *tolerance window* where event predictions are correct if they lie within a certain time window, often between 25ms and 50ms. A side effect of this is that, by shifting our focus to predicted events, we lose information about *not* predicting any events [40].

2.6.2 Accuracy

For classification tasks, a standard performance measure would be *accuracy*:

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

Summing up correct predictions, True Positives (TP) and True Negatives (TN), and dividing by total number of predictions, sum of TP, TN, False Positives (FP) and False Negatives (FN), we find a model's probability of having a correct prediction.

This performance measure falls short in that it is very susceptible to imbalanced datasets. In ADT, most timeframes contain no onset, meaning a naïve predictor would get a high accuracy by never predicting any onsets. Another problem with accuracy is that, due to our tolerance window approach we do not have quantities for TN, such that the standard accuracy computation is incomputable.

2.6.3 F1-score

Mentioned above are some of the reasons why *F1-score* has become the typical performance measure within ADT. F1-score combines and tries to maximize two different performance measures, namely *precision*;

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

and *recall*;

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

The precision of a model can tell us how good it is at *hitting* predictions. *Perfect precision* happens when a model has no FP, i.e. never predicting an event where one doesn't happen. Recall is similar, but represents the other end of the stick. It tells us

how good a model is at *not missing* predictions. *Perfect recall* happens when a model has no FN, i.e. never *not* predicting an event where one does happen.

As mentioned, F1-score combines these two measures in an aggregate performance measure by computing their harmonic mean:

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

By maximizing F1, we simultaneously maximize both precision and recall as well, reaping all their benefits.

2.6.4 Micro vs. Macro

There are different ways of computing and combining F1-score on multi-label data. Even though they might seem similar, they fundamentally represent different information, and thus the choice in which one to select is crucial.

Macro F1-score is computed through the arithmetic mean of the classwise computed F1-scores. Finding a model which maximizes this measure would be similar to finding the model which performes best on each of the separate classes, preventing a class from taking priority due to imbalanced datasets. Relating this to ADT, it would mean focusing on transcribing each instrument equally well.

Micro F1-score is computed through finding the F1-score with global TP, FP, FN values. Maximizing this would mean prioritizing classes that occur more frequently in the datasets. Such as in ADT, this would mean focusing on transcribing instruments which appear often, like the snare or base drum, over rarer instruments like the toms (Somewhere in this master thesis we need a quick introduction to the drums, and which piece of the drumset is which).

For ADT, the trend has been to select Micro F1-score as the main performance measure, due to its ability to show a model's *general* performance on musical pieces. We want our model to maximize their ability to transcribe music, not maximize their ability to transcribe each instrument in said music. ADT, prioritizing frequent instruments is relevant. As mentioned previously, the more frequent instruments lay the ground work for the fundamentals, and could be said to be more important than scarcely occuring ones.

Chapter 3

Architectures

Finding a suitable architecture is a vital step in creating a well-performant deep learning model. By leveraging different techniques, we balance introduction of inductive biases and possibility for complexity, which hopefully can help us end up with a generalizing model.

3.1 Recurrent Neural Network

The Recurrent Neural Network (RNN) is a standard architecture when it comes prediction on sequence data. It has been tried and tested, showing promising results for audio tasks.

The fundamental building block for RNNs is the *recurrent unit*. It iterates the whole input sequence, storing information from previous timesteps in a form of memory, through maintenance of a *hidden state*. This can be extended to gaining information about future timesteps by using *bidirectional* versions. In this way, prediction on current timesteps are affected by the information from surrounding timesteps. This is relevant in tasks such as ADT as auditory information usually spreads over several timesteps, e.g. the timbre of an instrument event lingering after onset.

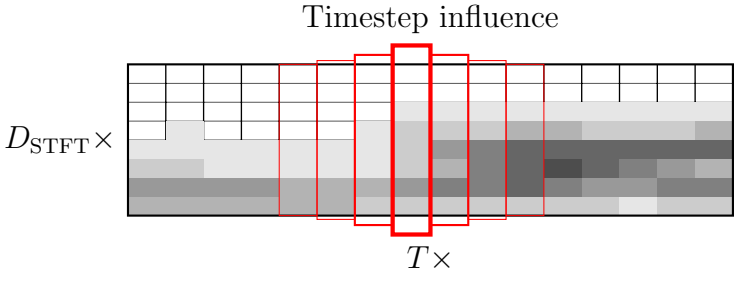

Figure 3.1: Visualization of adjacent timestep's influence for a bidirectional RNN.

However, traditional RNNs suffer from the *vanishing gradient problem* due to a timestep's influence diminishing with distance, making *long range dependencies* harder to learn. Different architectures have been developed to try to overcome these issues, such as the Gated Recurrent Unit (GRU) by Cho et al. [11], and Long Short-Term Memory (LSTM) by Hochreiter and Schmidhuber [22].

It has been shown that GRUs and LSTMs are capable of learning ADT related tasks, and is therefore in interest to comparatively measure how their efficiency stands in regards to other architectures [35, 40, 41, 46].

3.1.1 Implementation

Our RNN architecture consists of several bidirectional recurrent units, ending in a frame-wise linear layer. For the Bidirectional Recurrent Unit (BiRU), we train both a GRU and an LSTM model as hyperparameters, in addition to search over number of layers $L \in \{2, 3, 4, 5, 6\}$ and hidden size $H \in \{72, 144, 288\}$, selecting the one with best performance. At last, we have a linear layer, outputting onset probabilities for each of the drums per timeframe.

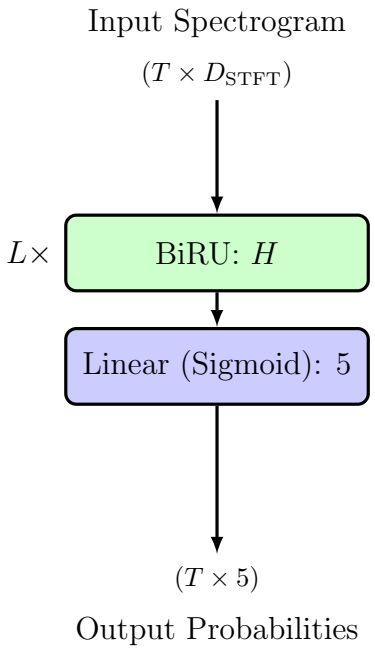

Figure 3.2: RNN architecture structure.

3.2 Convolutional Neural Network

We've mentioned that spectrograms can be treated as images. Therefore it would make sense to try an image focused approach, by utilizing *convolutions*. By applying convolutional layers, each timestep gets access to information around itself, a *context*. These convolutional layers make up the primary building blocks for the Convolutional Neural Network (CNN).

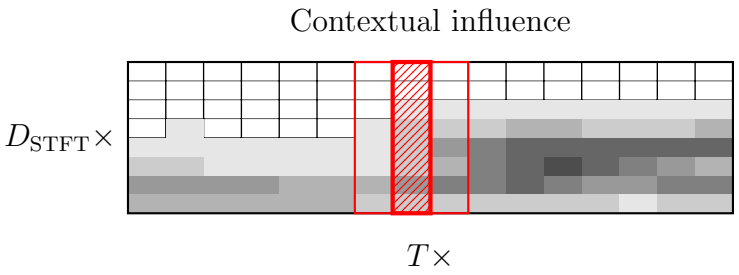

Figure 3.3: Visualization of a timestep's contextual influence with CNNs.

CNNs have been shown to give reasonable performance within ADT. This could be due to contextual information being important for identifying instrument onsets, and making learning easier for our models [41].

3.2.1 Implementation

Our CNN architecure consists of $I \in \{1, 2, 3\}$ initial convolutional blocks. Inside this block, convolutional layers have an increasing number of kernels $C = \{32, 64, 96\}$, intuitively leading to an increase in complexity along with depth. Then we have a varying amount of fully connected layers $L \in \{1, 2, 3, 4\}$, projecting into a latent space sized $H \in \{72, 144, 288, 576\}$. Both the convolutional and fully connected layers are followed by a Rectified Linear Unit (ReLU) activation function. Lastly, an output layer computed each instrument's onset probabilities.

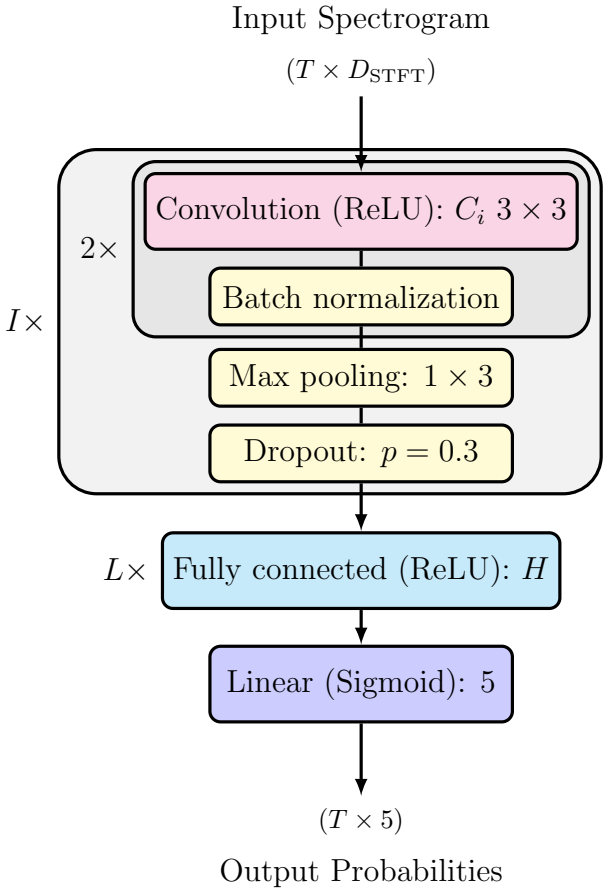

Figure 3.4: CNN architecture structure.

3.3 Convolutional Recurrent Neural Network

The previous features, recurrent layers and convolutions, are not mutually exclusive. Theoretically they can harmonize together, complementing eachother for easier learning. This results in the Convolutional Recurrent Neural Network (CRNN) architecture.

Intuitively, the CNNs ability to process images like the spectrogram, together with the RNNs ability to understand temporal sequence data should prove beneficial for ADT tasks. And indeed, this combination of cross-timestep memory and contextual data representation has shown to be insightful [41, 42, 46].

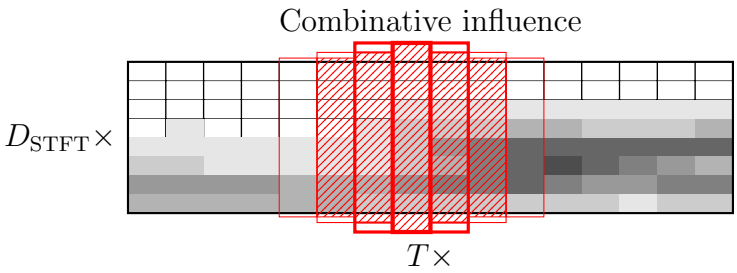

Figure 3.5: Visualization of a timestep's contextual and cross-timestep influence with Convolutional RNNs.

3.3.1 Implementation

We begin with a fixed-size convolutional block with $I = 2$, as used by several ADT authors [41, 46]. Following the CNN it has an increasing number of kernels $C = \{32, 64\}$. We then, similar to the RNN, have a BiRU layer of either a GRU or LSTM, with number of layers $L \in \{2, 3, 4, 5\}$ and hidden size $H \in \{72, 144, 288\}$. Output probabilities are then computed through the final linear layer.

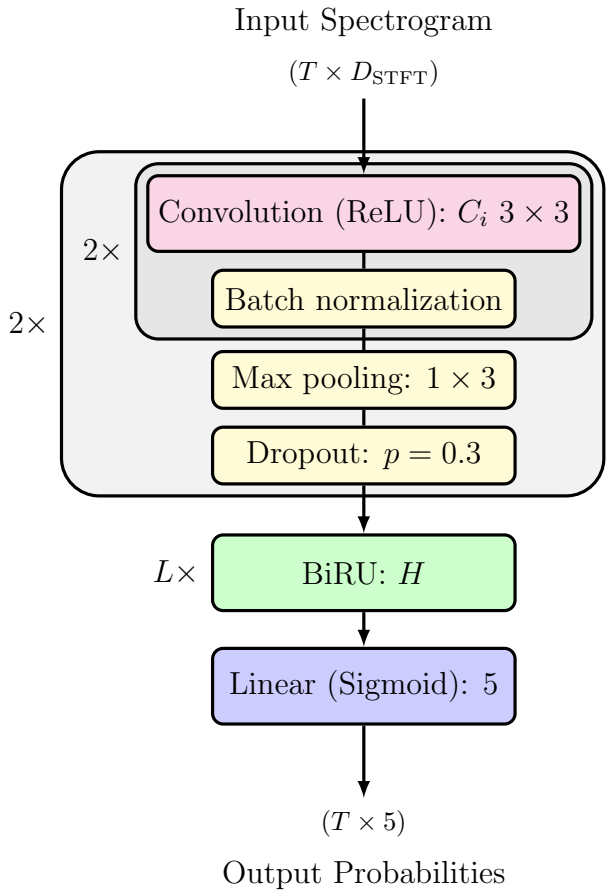

Figure 3.6: Convolutional RNN architecture structure.

3.4 Convolutional Transformer

Google's "Attention Is All You Need" [39] made headway in regards to sequence prediction. It introduced the *Attention* layer, making a model capable of learning to *attend* to different elements in a sequence, and learning the relationship between them. Models dropping the recurrent units in favour of attention blocks are called *transformers*.

As mentioned, the RNN displays difficulty in sustaining long range dependencies through its hidden state, and information further away tends to become attenuated. The attention layer solves this by allowing each element to individually attend to each other element in the sequence separately. Intuitively, it allows each element to *"intelligently"* pick and choose where it wants to look, and what elements it wants to be influenced by. This stands in contrast to the recurrent units, where each element has to learn and predict what information about itself other elements could find useful, and *"remembering"* that, adding it to the hidden state.

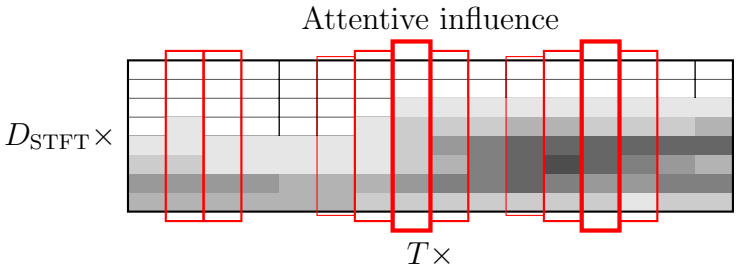

Figure 3.7: Visualization of a timestep's attention to each other timestep in an attention model.

Recently, the attention layers have shown great success in AMT and ADT tasks, in some cases proving superior to the RNN [15, 10, 47].

Simply replacing the recurrent layers with attention blocks could allow our model to reap the reward by increasing its ability to understand sequences, while keeping the previously gotten gains from the convolutional layer. Both inside and outside of ADT, combining convolutional layers with transformers has seemed beneficial [47, 20].

3.4.1 Implementation

Similar to the CRNN, an initial fixed-size convolutional block with $I = 2$ is used, with an increasing number of kernels $C = \{32, 64\}$. Then, we project the resulting latent space

into a separate, lower dimensional embedding space with dimension $D_e \in \{72, 144, 288\}$, before combining it with a sinusoidal positional encoding.

Following this, the model contains $L \in \{2, 4, 6, 8\}$ standard pre-layer normalization attention block, as these recently have been shown to be more stable during learning than the post-layer versions [45]. These attention blocks contain multi-head self-attention layers with $H \in \{2, 4, 6, 8\}$ number of heads. Note that the first layer of the feedforward layer inside these attention blocks uses the Gaussian Error Linear Unit (GELU) activation function due to it being the standard within transformers and for its possible performance improvements over ReLU [13, 21]. Lastly, the linear layer outputs onset probabilities.

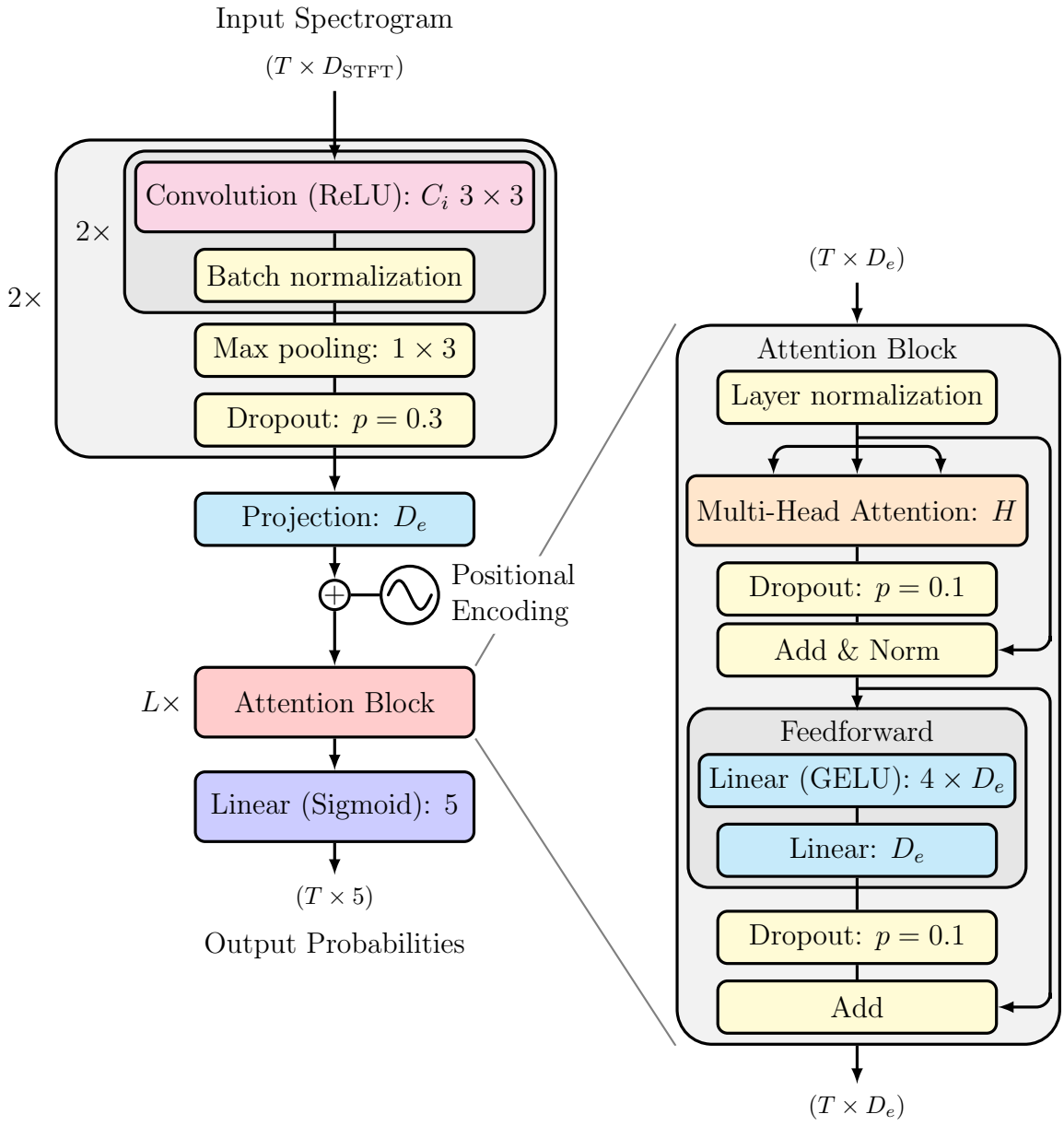

Figure 3.8: Convolutional Transformer architecture structure.

3.5 Vision Transformer

Introducing the Transformer raises a question. Would it be possible for an architecture to be comprised of solely attention layers, removing convolutions and breaking this need for a mixed convolutional-transformer architecture. This question was answered by Google's "An Image Is Worth 16x16 Words" [14], introducing the Vision Transformer.

The Vision Transformer was created to tackle image recognition tasks, though it has been applied to audio classification, displaying great performance on both [14, 18]. However, application of the Vision Transformer on an ADT task is a novel approach.

It is worth to note that Vision Transformers have been shown to display excellent performance, however they usually need more significantly more data than other architectures to function optimally [14].

3.5.1 Patch Embedding

A key component of the Vision Transformer is the creation of a patch embedding. First, we split the input image into different non-overlapping patches, and flatten them. These patches are linearly projected into a latent space, and a positional encoding is added to retain positional information. This resulting sequence of patches is what is referred to as a patch embedding.

We usually say that patch embeddings eliminate the use of convolutions in the Vision Transformer. However, the actual implementation of splitting the image into patches and linearly projecting each patch are usually implemented with a single 2D convolutional layer. Note that it is a linear projection, meaning the convolutional layer is absent of an activation function.

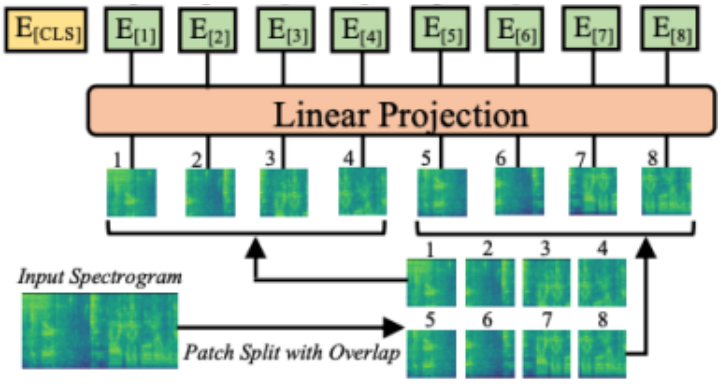

Figure 3.9: The creation of a patch embedding from an input spectrogram.

3.5.2 Architecture Modifications

Originally, the Vision Transformer has been used on classification tasks where the output is not a sequence. ADT is a sequence labeling task, and requires that the output is a sequence, where the time dimension matches the size of the input.

This can be solved by treating a group of patches from the patch embedding together as a timeframe, as long as we ensure that there the number of timeframes are a factor of the number of patches. Then the output of the Vision Transformer could be construed to match our intended output sequence.

3.5.3 Implementation

Initally, we transform the input spectrogram into a $(T \times D_e)$ patch embedding. The Convolutional layer splits the spectrogram into $(T \times D_e/P \times P)$ different patches. Added to these are a $(D_e/P \times P)$ learnable positional embedding, providing positional information to each patch. These are then permuted and flattened, transforming them into our final patch embedding.

Afterwards, we combine it with a sinusoidal positional encoding, and successively apply L attention blocks with number of heads H, identical in structure with those used in the Convolutional Transformer. Lastly, the linear layer outputs onset probabilities.

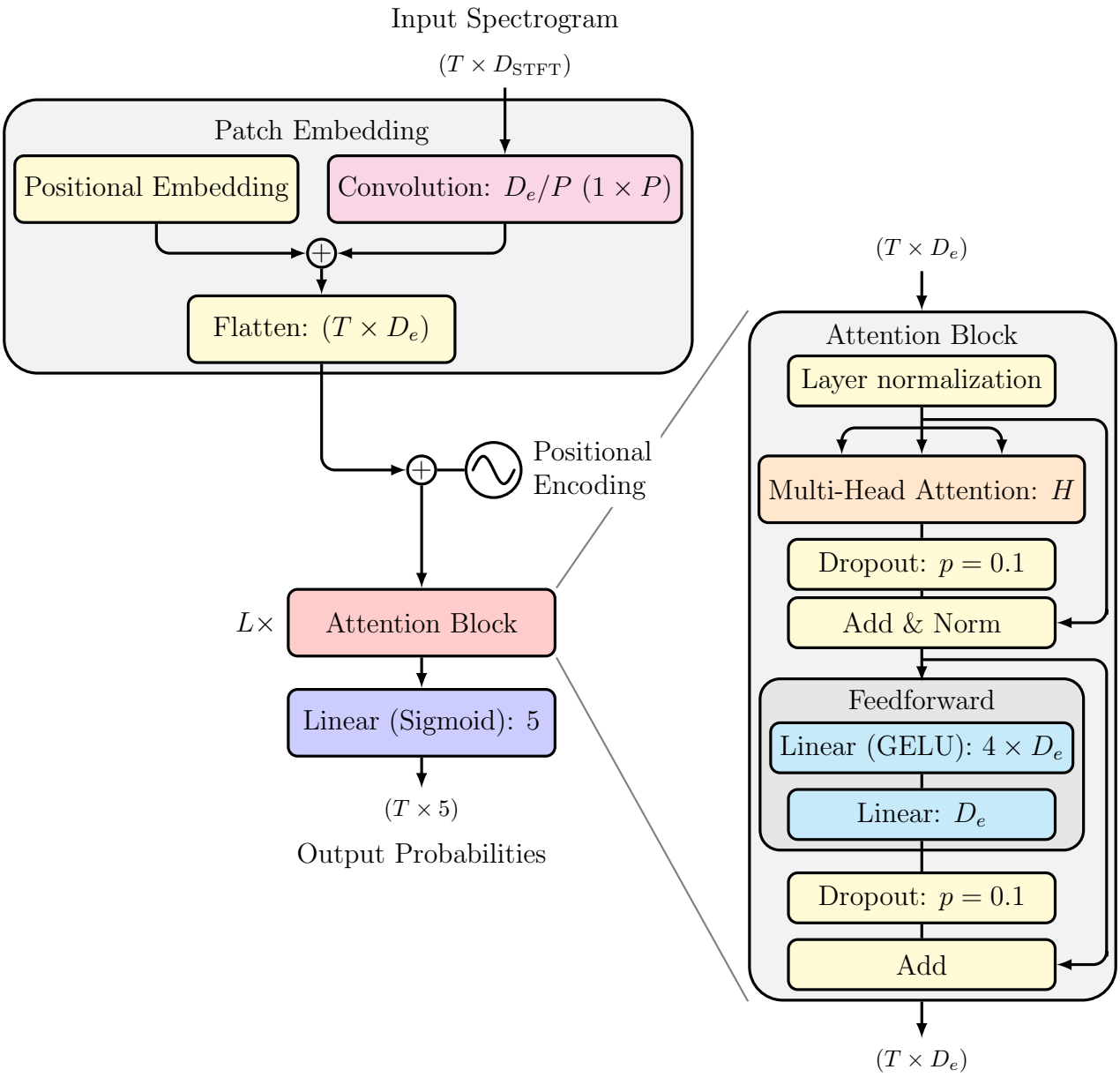

Figure 3.10: Vision Transformer architecture structure.

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 3.1: The different hyperparameters tuned to train the Vision Transformer.

Chapter 4

Datasets

4.1 ENST+MDB

The ENST-Drums dataset by Gillet and Richard [16] has been one of the most commonly used ADT datasets [44]. It features thorougly annotated drum samples by three drummers over different musical genres. Most of the tracks contain drum-only recordings, except the *minus-one* subset, which is played together with a music accompaniement. As this thesis attends to DTM tasks, we isolate our focus to this subset of tracks.

As this dataset contains separate audio files for performance and accompaniement, we additively combine them to create a singular respective mixture track. There exist many recordings from differently placed microphones of a single performance, however in this thesis, we solely selected the *"wet mix"* due to it being a combined recording of all other microphone recordings, and its *mix* showing resembelance of a polished performance track.

ENST-Drums contains 1.02 hours of music over 64 tracks.

Another well-known MedleyDB Drums dataset, from Southall et al. [36]. This dataset is built on top of Bittner et al.'s MedleyDB (MDB) dataset [4], but re-annotated and specialized for ADT related tasks. This dataset is, similar to ENST-Drums, also split into different stem tracks, such as isolated drum recordings and accompaniement. However, they also contain already-mixed *full mix* tracks, which are the ones we use in this thesis.

MDB-Drums contains 0.35 hours of music over 23 tracks.

Both of these datasets distributes their audio in waveform files, and their annotations in text files. Annotations are formatted by onset time and instrument label. They are also relatively small, and contain thorough, real, annotated data. Due to these similarities, in this thesis they are combined together into a slightly larger ENST+MDB dataset.

In total, this dataset contains 1.37 hours of music over 87 tracks.

4.1.1 Splits

These two datasets do not have predefined train/validation/test splits, such that we decided to construct our own splits. From ENST-Drums, *drummer1* and *drummer2* make our training split. The remaining drummer, *drummer3* is split in half, each for validation and test respectively. From MDB-Drums we do not have different explicit drummers, but instead split on specific genres. The explicit splits are given below, in table 4.1.

Explain how this dataset gets train/val/test split. Give explicit titles for split as well. Here we've only given a text/explained split, however not the actual information. Probably do this in an appenix!

ALSO VERIFY THAT SPLITS ARE AS WRITTEN. JUST TO BE SURE!

Split	ENST-Drums	MDB-Drums
Train	**"107_minus-one_salsa_sticks"**	**"MusicDelta_Punk"**
Validation	Number of heads	{2, 4, 6, 8}
Test	Number of layers	{2, 4, 6, 8, 10}

Table 4.1: The different tracks used for each respective train/validation/test split for this thesis.

4.1.2 Mapping

Maybe (or maybe not) explain the mapping from this to 5-instrument mapping. Also put this under a Appendix.

4.2 E-GMD

The Expanded Groove MIDI Dataset (E-GMD) from Callender et al. [7] is a large ADT dataset consisting of audio recordings from human drum performances annotated in MIDI. It is an expansion of Gillick et al.'s Groove MIDI Dataset (GMD) [17].

GMD was created through recording human performances in MIDI format through a Roland TD-11 electronic drumkit. This dataset does not contain audio recordings, only MIDI files, such that it is not applicable for ADT. To expand this dataset to be used with ADT, Callender et al. re-recorded the MIDI sequences on a Roland TD-17 electronic drumkit in real-time on a Digital Audio Workstation (DAW). These re-recordings were done over a large amount of differing soundfonts (confusingly also called *drumkits* on electronic drumkits), synthesizing several differently sounding audio recordings from a single MIDI performance [17, 7].

In contrast to the other dataset's utilized in this thesis, this dataset is not created for a DTM task, but rather a DTD task. This is due to all recordings containing solo, drum-only performances. In addition, as this data is recorded from human performances in a semi-manual nature, there exist some errors from the recording process *"that resulted in unusable tracks."* [7]. The magnitude of these errors are not stated, however other authors propose that it might be as high as 20.5% of the tracks with varying amounts of discrepancies [23].

This dataset contains 444.5 hours of audio recordings, from 1,059 unique drum performances resampled to 45,537 MIDI sequences.

4.2.1 Mapping

Maybe (or maybe not) explain the mapping from this to 5-instrument mapping. Also put this under a Appendix.

4.3 Slakh

The Synthesized Lakh 2100 (Slakh) Dataset from Manilow et al. [29] is a synthesized version of a subset of the Lakh Dataset from Raffel [31], that subset being a 2100 randomly

selected tracks from Lakh where the MIDI files contain at least a piano, bass, guitar, and drums. These MIDI files are rendered and mixed into combined audio files, stored together with their respective original MIDI performances.

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

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

This dataset contains 115 hours of audio recordings over 1709 different songs.

4.3.1 Mapping

Maybe (or maybe not) explain the mapping from this to 5-instrument mapping. Also put this under a Appendix.

4.4 ADTOF-YT

The Automatic Drums Transcription On Fire YouTube (ADTOF-YT) dataset from Zehren et al. [46] is a large ADT dataset containing crowdsourced data, hence the YouTube suffix. Due to the crowdsourced nature of this dataset, it is possible to utilize large amounts of non-synthetic, human data, but with the tradeoff in which we can not guarantee its quality.

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

Note that their sourcecode and dataset distributor informs us that the dataset contain mel-scale spectrograms, and not logarithmically filtered ones as we previousy proposed. However, through thorough code investigation and approximate reconstruction of original waveforms we hypothesize and conclude that this is a mistake. Is this enough justification? Should I provide more information? If so, what exactly?

The dataset contains 245 hours of music over 2924 tracks.

4.4.1 Mapping

Due to this dataset being distributed in a preprocessed nature, no re-mapping has been done, and the dataset is used "as is".

4.5 SADTP

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

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

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

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

4.5.1 Mapping

Maybe (or maybe not) explain the mapping from this to 5-instrument mapping. Also put this under a Appendix.

4.6 Differences

Provide a table with information neatly gathered in a table. Such information should be size (total duration), vocabulary, real or synthetic data (human/non human), number of tracks. Should be like e.g. [46, 7] or others.

Chapter 5

Methodology

Though we perform 2 different studies, with 2 different intentions, the dataset preparation and model selection pipeline remains the same.

5.1 Data Preparation

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

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

5.1.1 Audio Files

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

After each track is loaded into a waveform, a zero-padding is added to the end of each sequence allowing for even partitioning into 4 second partitions. Then we turn the track

into a spectrogram with 2048 fft's, and a window length of 2048. By keeping the hop length equal to the sampling rate divided by 100, the resulting spectrogram's timesteps represent a 10ms window of the original waveform.

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

5.1.2 Annotations

The annotations are either distributed in specific formats as text files (with the suffix *.txt*), or in MIDI files (with the suffix *.mid* or *.midi*), each one having a different transformation into a sequence of instrument onset probabilities.

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

The data given in MIDI format, the annotations are parsed using the library `Partitura`, and loads the information into an array of MIDI events called *notes*. These *notes* contain information for each event, importantly time, pitch, and velocity. These events are very thorough, but strict instrument onsets can be isolated by restricting our view to notes with a non-zero velocity. Instruments are denoted by the event's pitch, and a mapping is done from each pitch to a respective class. The time is converted identically to the loading of the text annotations, turning them into timeframe indices. At last, we also here set each specific registered onset (timeframe, class) cell to 1.

In addition to this, we apply a *target widening* step, setting values in timeframes adjacent to an instrument onset with a lower weight, equal to 0.5. These additional neighbouring *soft labels* have shown to be benifical in countering sparsity in our labels following multiple works on beat transcription [25, 46].

5.1.3 Splitting and Storing

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

5.2 Preprocessing

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

Data normalization like this has been shown to increase the speed and stability of convergence during training and, in summary, producing models which better generalize to unseen data. Although the specific benefits depend on the normalization technique used, the general consensus is that normalization in itself is beneficial in machine learning, hence their ubiquitous use in state-of-the-art models [24].

Another preprocessing step motivated by ADT specific methods is the use of infrequency weights, which framewise weighs the loss based on the instrument onsets that are present at each frame. These weights are precomputed from the training dataset, and are, for each instrument, computed by what Cartwright and Bello call *"the inverse estimated entropy of their event activity distribution"* [8]. Although they apply this to account for sparsity in data along different tasks, Zehren et al. [46] applied it to give more weight to infrequent instruments.

These weights are computed by calculating the probability of an instrument i appearing $p_i = \frac{n_i}{T}$ as the total number of onsets n_i divided by the total number of timesteps T. With this probability we compute its inverse entropy, giving us our final weights $w_i = (-p_i \log p_i - (1 - p_i) \log 1 - p_i)^{-1}$. Note that our probability computation differs from the work of Cartwright and Bello, as we do not divide by the number of instruments [8].

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

5.3 Training

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

With the choice by what to use, Adam was considered. This is an optimizer so ubiquitously used within the field of deep learning that when questioned with what optimizer to generally use authors like Sebastian Ruder state that *".., Adam might be the best overall choice* [33]. However contrary to this, the Adam optimizer has been shown to contain some issues, like its coupling of the weight decay term inside its gradient-based updates. Due to this, the choice instead fell on AdamW, a modified Adam implementation decoupling weight decay in whole from the gradient-based updates, and displaying a better ability to generalize. [26, 6, 28]

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

Another addition which is frequenct in other ADT works is the use of a learning rate scheduler. A learning rate scheduler keeps track of recent validation loss values, and if the minimum loss achieved plateaus (meaning it stop decreasing) for a certain number of epochs, the learning rate gets reducing by a given factor. In this thesis we reduce the learning rate by a factor of 5 if we observe 5 epochs of plateauing, with no improvement to our minimal validation loss [10, 46]. We also keep track of the general count of epochs since validation loss last improvement and perform an early stop if we ever observe 15 epochs without improvement.

5.4 Postprocessing

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

First, we apply the aforementioned peak picking algorithm to isolate peaks in the model's onset confidence, intuitively being frames where the model is most confident in an instrument onset happening [5, 42]. Afterwards, we count a predicted onset if the given peak has a value larger or equal to 0.5. From this, it is trivial to compare predicted onsets with actual onsets, by greedily iterating our output sequence from the beginning, counting a prediction as a true positive if it happens within a 5 frame (50 ms) interval of a true onset, false positive if it happens outside such an interval, or false negative if a true onset happens with no prediction within said interval.

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

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

5.5 Model Selection

For model training and selection we utilize the RayTune library [27]. It simplifies the training of models by allowing us to input a training function, which metric to optimize for, hyperparameter spaces and search strategy, and additional configs. This simplifies our training, and through per-epoch reporting, it handles both checkpointing of model weights and best performing model selection for us.

Through RayTune, we train 15 different models (25 solely for the smallest dataset ENST+MDB), with hyperparameters tuned through bayesian optimization (see the next section 5.6). Each model is also ran for at most 100 epochs. As mentioned, we perform an

early stop if performance stops increasing. We utilize PyTorch's dataloaders for iterating the datasets, with a batch size of 128. And using the AdamW optimizer, as previously mentioned.

During each epoch of training, we evaluate the model on the training dataset's corresponding validation data. After training, the model with the highest Micro F1-score is selected. Due to the pre-split nature of most of our datasets, we utilize hold-out validation, with separate train, validation and test datasets. This is not only very common within ADT [40, 44, 10], but throughout the whole field of deep learning. Raschka mentions that *"The holdout method is inarguably the simplest model evaluation technique"* [32], and might be one of the reasons for its popularity. Then, we estimate its performance on unseen data through evaluating it on the test dataset. At last, said model is stored together with its corresponding weights, training config, and metrics.

5.6 Hyperparameter Tuning

5.6.1 Search Strategies

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

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

We utilize RayTune's implementation of `OptunaSearch`, a hyperparameter searching regime which uses Optuna, an automatic hyperparameter optimization software framework based on bayesian optimization [2]. It has shown to be efficient in finding a good tradeoff between computation and performance, with authors like Shekhar et al. performing benchmarks on neural networks and showiung that *"The performance score of Optuna is the highest for all datasets"* [34].

How much do I need to cite here?

5.6.2 Hyperparameters

RayTune allows one to easily declare the search space for each of the hyperparameters. Every architecture-specific hyperparameter is chosen based on a random choice. In addition to this, we also tune the optimizer-specific learning rate and weight decay using a logarithmically uniform random sample. The specific spaces can be found below, in Table 5.1.

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

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

Chapter 6

Architecture Study

This study's main purpose is to figure out how suited each architecture is for ADT, more specifically DTM tasks. This could help us figure out which architecture is superior for ADT, if there are any similarities between architectures who perform similarly well, or if there are any architectures who perform poorly.

6.1 Methodology

We perform hyperparameter tuning and model selection to train a separate model for each architecture over each dataset. At last we test the model on each dataset's respective test split. As a result, we are left with performance measures on unseen data from the same distribution as those each model was train on. This will give us a good intuition into each architecture's ability to learn the task of ADT and could help us estimate their generalization ability.

6.2 Results

Architecture	ENST+MDB	E-GMD	Slakh	ADTOF-YT
Recurrent Neural Network	0.6682	0.889	0.864	**0.9635**
Convolutional Neural Network	0.7797	0.8744	0.8318	0.844
Convolutional Recurrent Neural Network	**0.8132**	**0.8935**	**0.8959**	0.9333
Convolutional Transformer	0.776	0.8831	0.8826	0.9535
Vision Transformer	0.5426	0.8779	0.879	**0.9635**

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

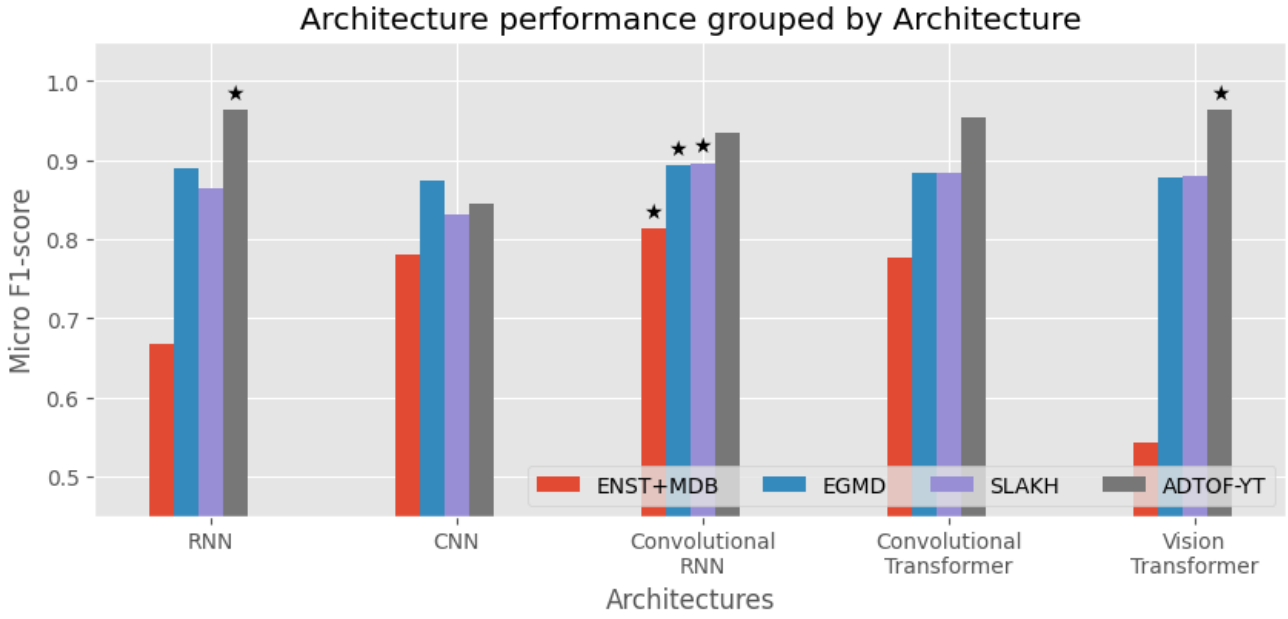

Figure 6.1: Comparison of Micro F1-scores for each dataset across the different architectures. Bars marked with a ($\star$) indicate the best performing architecture for each respective dataset.

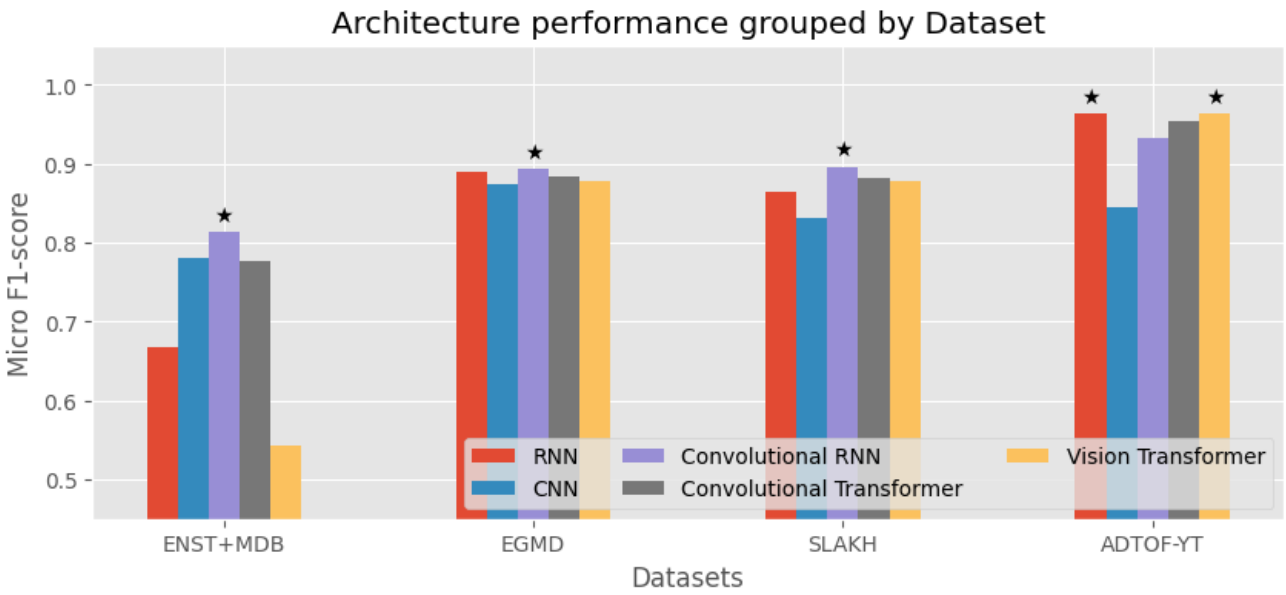

Figure 6.2: Comparison of Micro F1-scores for each architectures across the different dataset. Bars marked with a ($\star$) indicate the best performing architecture for each respective dataset.

6.3 Discussion

The architecture study's results are summarized in Table 6.1 and visualized in Figures 6.1 and 6.2. From these results we can compare and discuss how each model performs, and speculate where the sources of their performances lie.

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

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

convolutions and recurrent units, allowing for both initial spatial feature extraction as well as short-range temporal modelling.

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

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

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

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

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

All in all, most of the architectures display a sufficient ability to learn the ADT task well. However, due to being the best performing architecture on all but one of the datasets, we conclude that the convolutional recurrent neural network currently is the generally best performing architecture for ADT.

Should I mention interesting future steps here or in the final conclusion?

Chapter 7

Dataset Study

7.1 Methodology

This study's main goal train the selected architecture from the last study over different combinations of ADT datasets, while zero-shot testing on the remaining ones (+ SADTP, the novel dataset introduced in this thesis) and figure out if there exists any combinations of datasets which outperforms the other on the majority, making the model generalize better. This could answer the question; "Does training on combinations of dataset lead to generalization?", and "What type of data makes our model generalize the best when it comes to ADT related tasks?". For this study as well; some kind of introduction to the study more or less.

7.2 Results

Display a table of results, micro-F1 (or maybe just micro-F1): Non zero-shot tests are grayed out, best zero-shot test are bolded.

	Dataset 1	Dataset 2	Dataset 3	Dataset 4
Dataset 1	0.5	0.3		
Dataset 2	0.4	0.8		
Dataset 1+2	0.8	0.7		
Dataset 3	**0.9**	0.6		
Dataset 1+2+3	0.95	0.8		
Dataset 1+4	0.95	**0.75**		
Dataset 1+2+3+4	0.95	0.82		

7.3 Discussion

Chapter 8

Conclusion

This is the conclusion.

List of Acronyms and Abbreviations

ADT Automatic Drum Transcription.

AMT Automatic Music Transcription.

BiRU Bidirectional Recurrent Unit.

CNN Convolutional Neural Network.

CRNN Convolutional Recurrent Neural Network.

DAW Digital Audio Workstation.

DFT Discrete Fourier Tranform.

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 Negatives.

FP False Positives.

GELU Gaussian Error Linear Unit.

GRU Gated Recurrent Unit.

LLM Large Language Model.

LSTM Long Short-Term Memory.

MIDI Musical Instrument Digital Interface.

MIR Music Information Retrieval.

NLP Natural Language Processing.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

STFT Short-time Fourier Transform.

TN True Negatives.

TP True Positives.

Bibliography

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

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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`.

[12] 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`.

[13] 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/`.

[14] 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`.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

[20] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented transformer for speech recognition, 2020.
URL: https://arxiv.org/abs/2005.08100.

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

[22] 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.

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

[24] 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.

[25] 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.

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

[27] 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`.

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

[29] 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.

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

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

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

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

[34] 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.

[35] 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.

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

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

[38] 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.

[39] 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.

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

[41] 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.

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

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

[44] 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.

[45] 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.

[46] 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.

[47] 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.