University of Bergen
Department of Informatics

Automatic Drum Transcription using Deep Learning

Author: Runar Fosse
Supervisor: Pekka Parviainen

Add correct NT faculty, instead of MAT-NAT

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`
Thursday 5th June, 2025

Contents

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. [28]

Previously, a popular approach to ADT was using signal processing, which later developed into using classical machine learning methods [28]. 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. [30]

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. [28]

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. [30] 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. [26, 25]

displayed good results with both a convolutional, and a convolutional-recurrent neural network. Zehren et al. [30, 31] 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. [6] 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 [18].

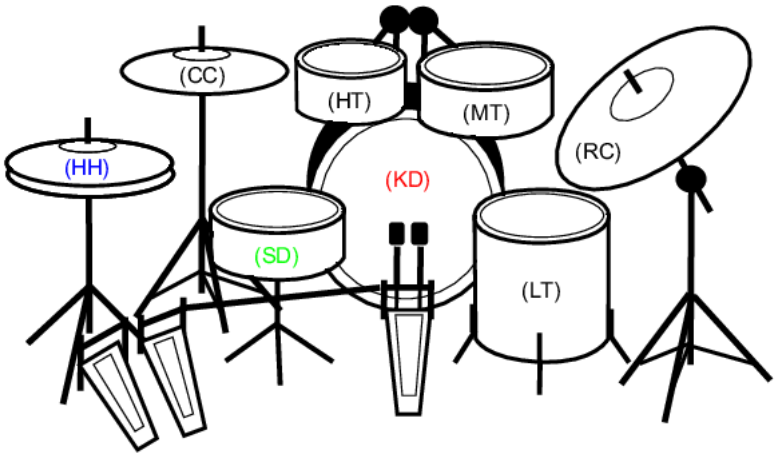

Figure 2.1: Example of the different instruments on the drumset.

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.

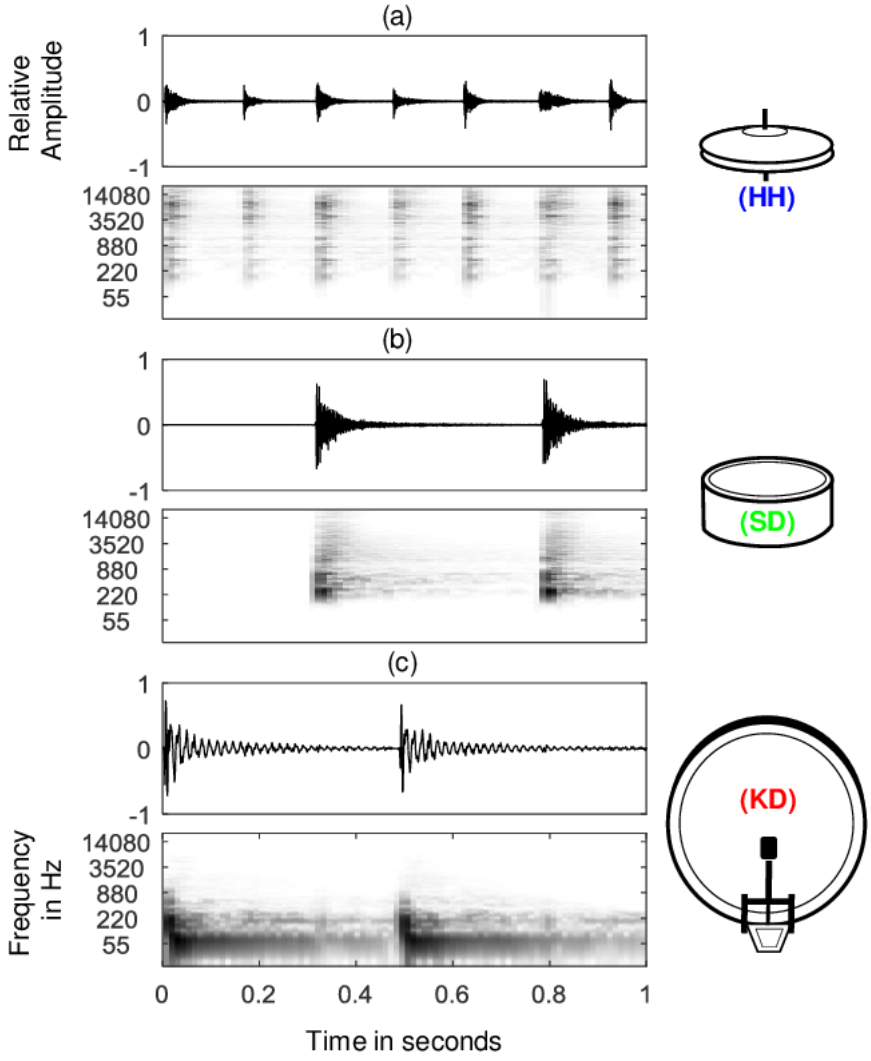

Figure 2.2: Example of the different audible footprint for drum set percussion.

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 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 [5]. 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.

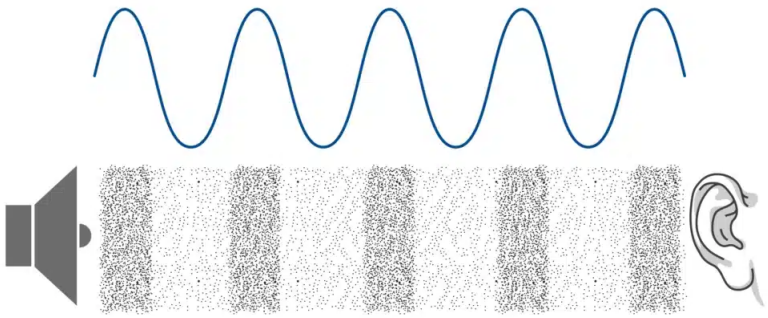

Figure 2.3: Soundwave to waveform relationship

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 [22], however the task of parsing and perceiving such a signal is a complex one.

2.3.1 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.

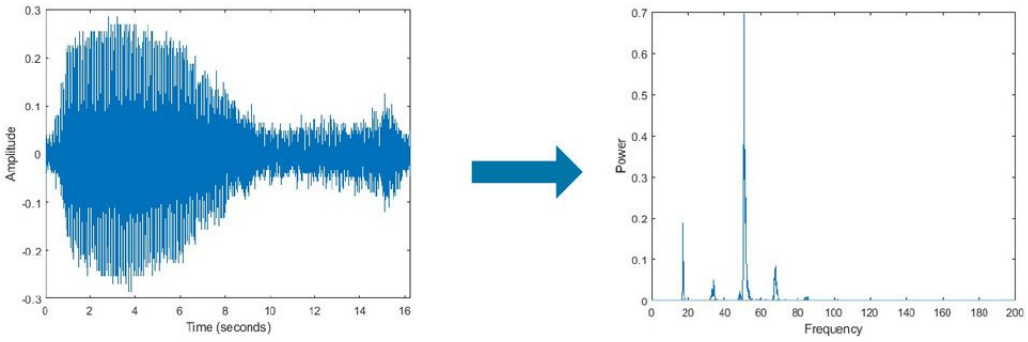

Figure 2.4: Application of a Fourier Transform

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.

2.3.2 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

2.3.3 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.

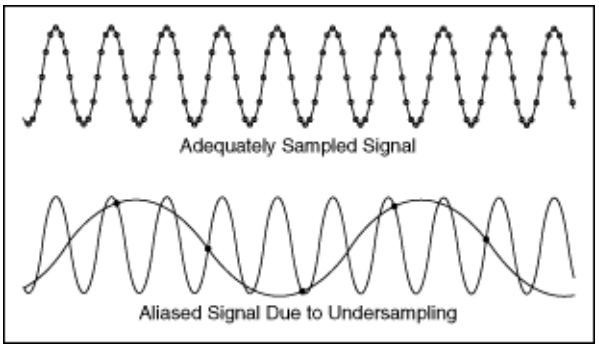

Figure 2.5: Example of aliasing in an undersampled signal.

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.

2.3.4 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. [2]

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"* [21], 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. [8]

2.3.5 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..

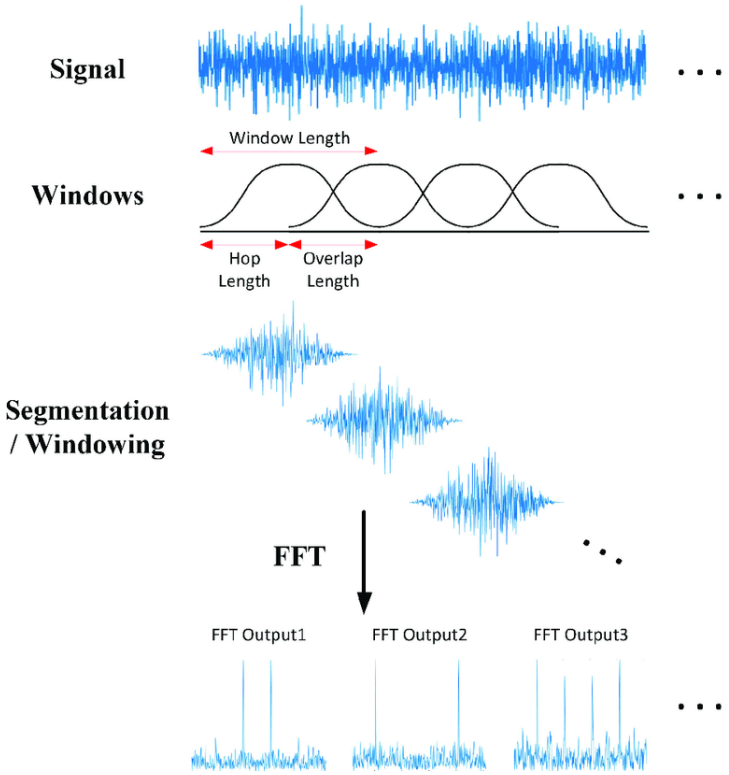

Figure 2.6: Example of the STFT

2.3.6 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*. In this way, we've converted our audible information into visual information. Naturally, these spectrograms can be visualized using e.g. a heatmap.

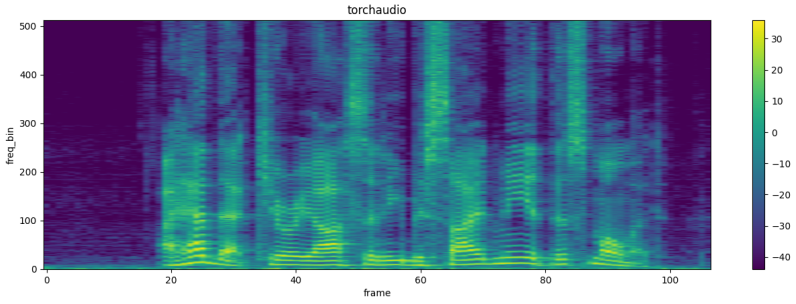

Figure 2.7: Heatmap of an audio spectrogram.

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 [14].

2.3.7 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. [27, 11, 6, 28, 13, 31]

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. [28, 26, 25, 30]

Add a example figure of Spectrogram vs Mel vs Logarithmic

2.4 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.

2.4.1 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.

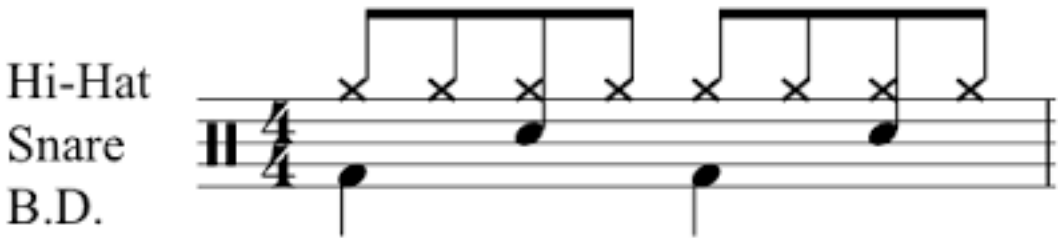

Figure 2.8: Example sheet music for a drumset

2.4.2 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, Garner et al. presented MT3 [11], a model inputting spectrograms and outputting MIDI events autoregressively. This format was expanded on by Chang et al.'s YourMT3+ [6], 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 ...

2.4.3 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. [19, 26]

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.

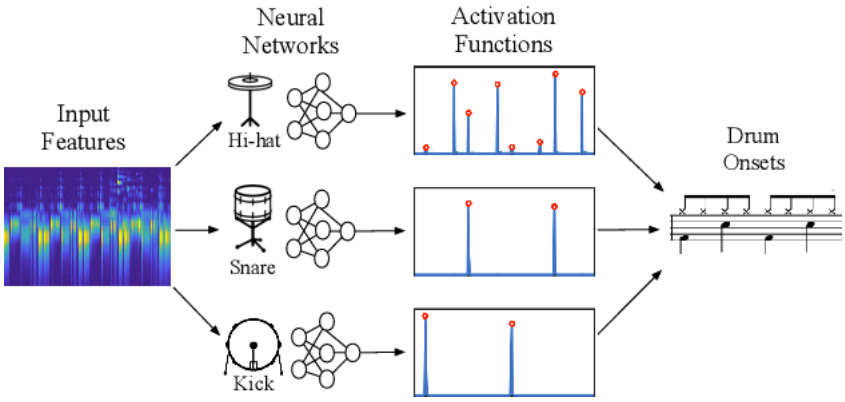

Figure 2.10: Example of ADT activation function output

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. [4], 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. [26] showed that the peak-picking parameters which gave the best results were $m = a = w = 2$ and $\delta = 0.1$.

2.5 Performance Measure

2.5.1 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 [24].

2.5.2 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.

2.5.3 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.

2.5.4 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.

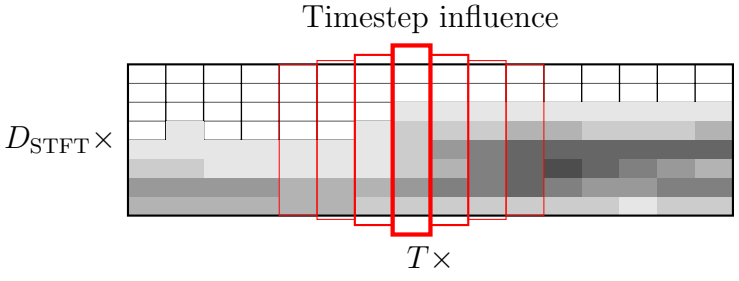

Figure 3.1: Visualization of adjacent timestep's influence for a bidirectional RNN.

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. [7], and Long Short-Term Memory (LSTM) by Hochreiter et al. [17].

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 [19, 24, 25, 30].

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.

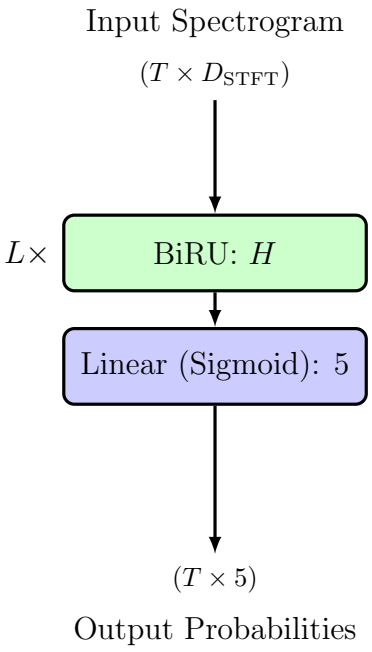

Figure 3.2: RNN architecture structure.

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).

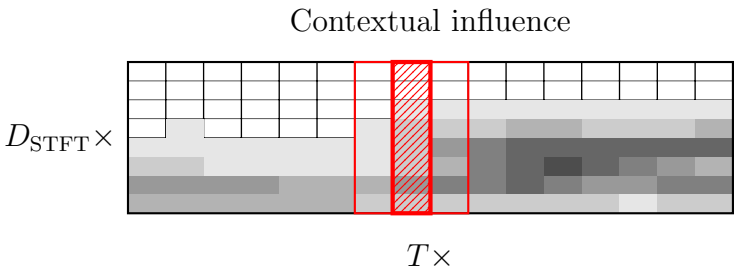

Figure 3.3: Visualization of a timestep's contextual influence with CNNs.

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 [25].

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.

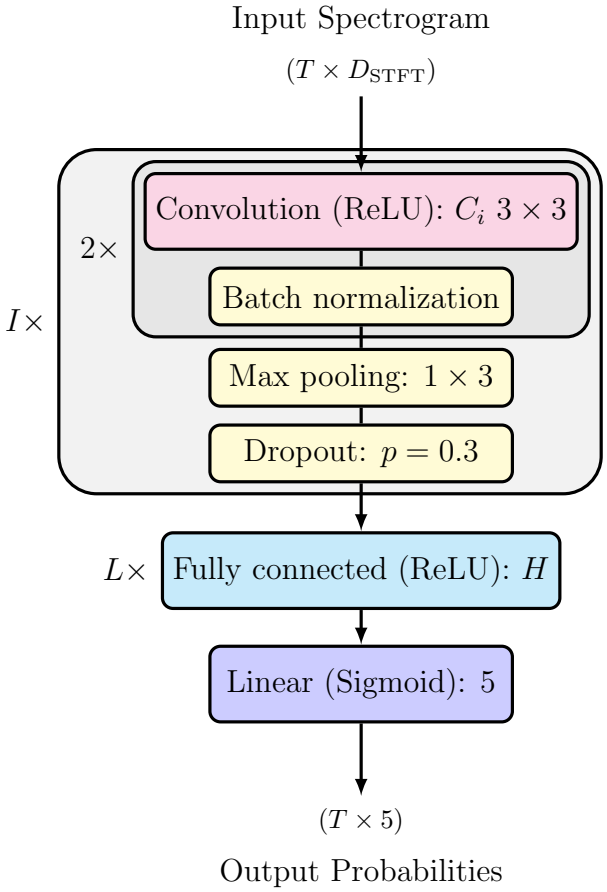

Figure 3.4: CNN architecture structure.

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 [25, 26, 30].

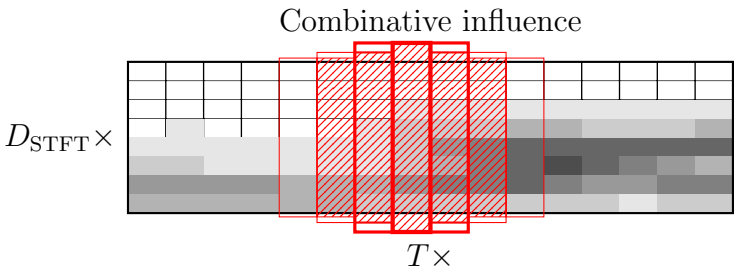

Figure 3.5: Visualization of a timestep's contextual and cross-timestep influence with Convolutional RNNs.

3.3.1 Implementation

We begin with a fixed-size convolutional block with $I = 2$, as used by several ADT authors [25, 30]. 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.

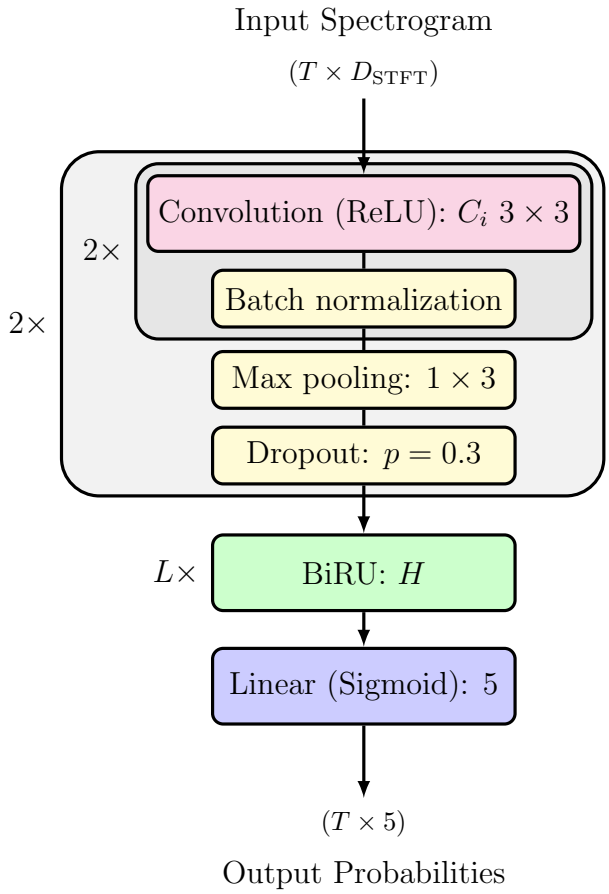

Figure 3.6: Convolutional RNN architecture structure.

3.4 Convolutional Transformer

Google's "Attention Is All You Need" [23] 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.

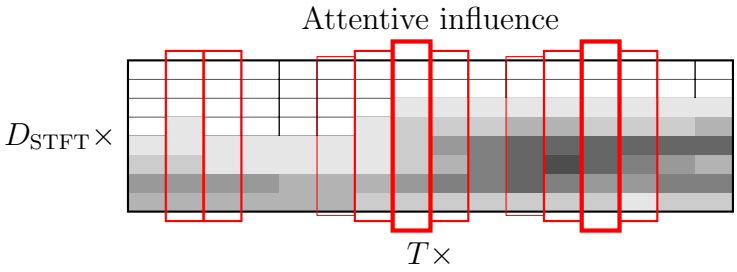

Figure 3.7: Visualization of a timestep's attention to each other timestep in an attention model.

Recently, the attention layers have shown great success in AMT and ADT tasks, in some cases proving superior to the RNN [11, 6, 31].

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 [31, 15].

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 [29]. 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 [9, 16]. Lastly, the linear layer outputs onset probabilities.

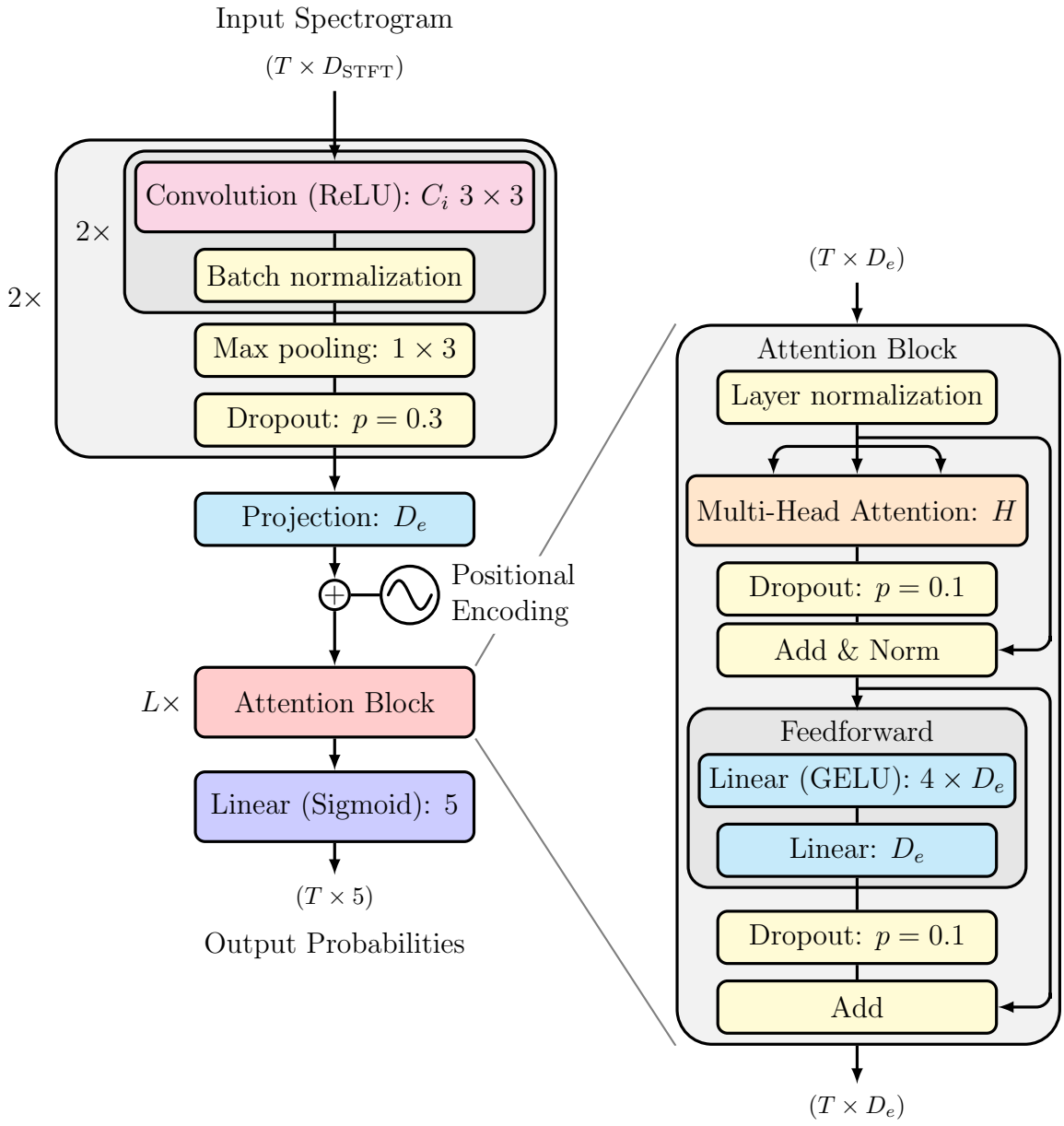

Figure 3.8: Convolutional Transformer architecture structure.

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" [10], 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 [10, 13]. 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 [10].

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.

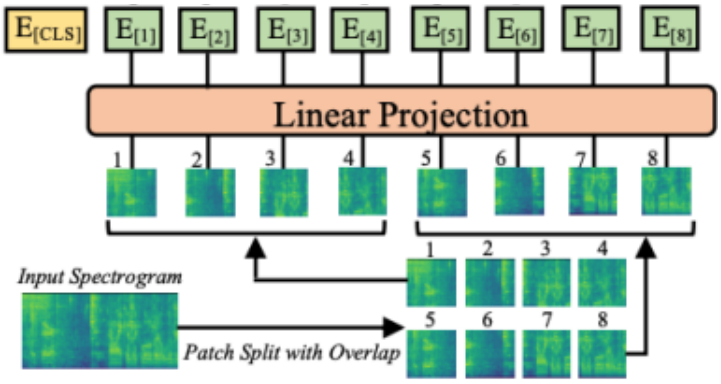

Figure 3.9: The creation of a patch embedding from an input spectrogram.

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.

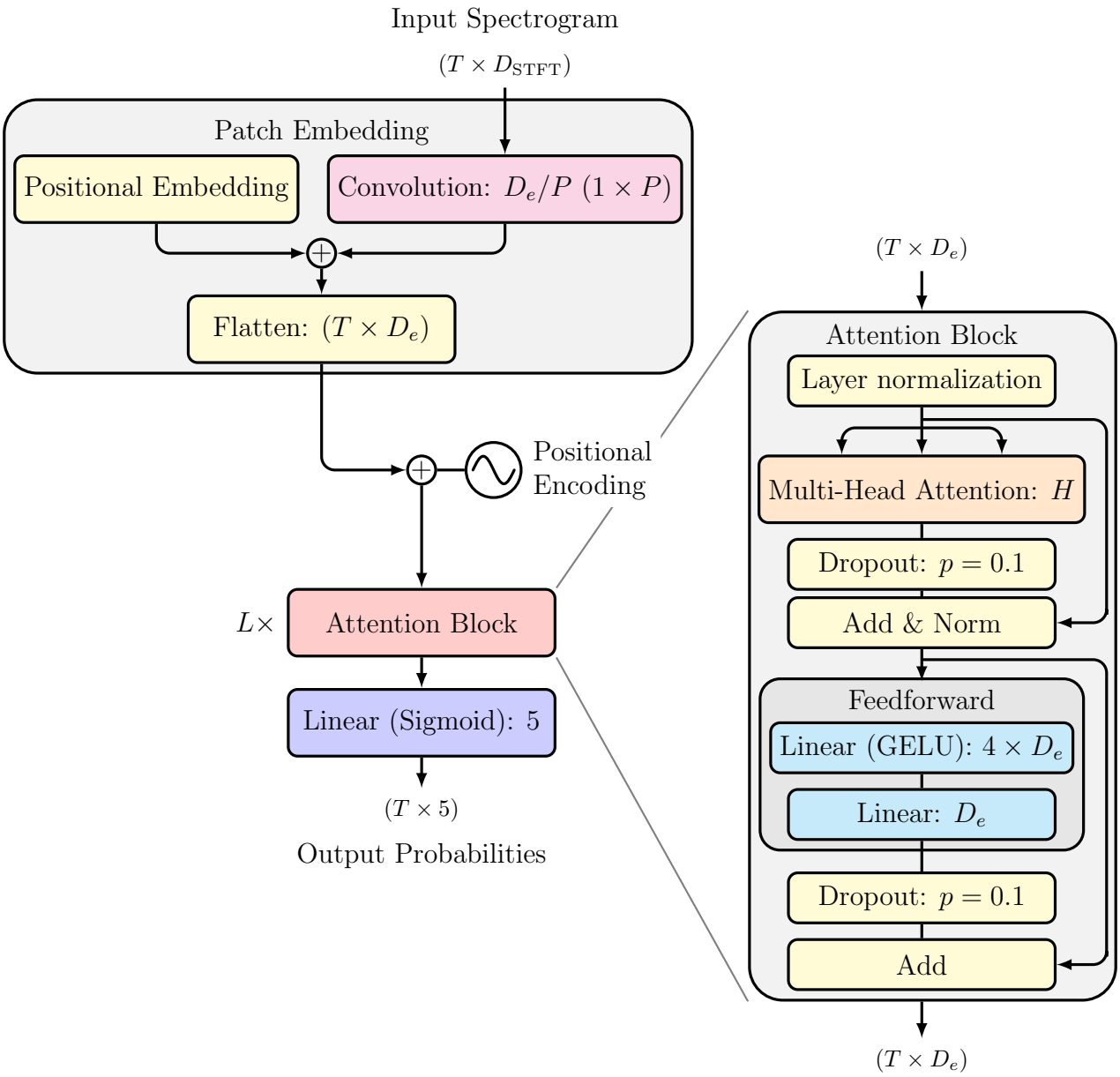

Figure 3.10: Vision Transformer architecture structure.

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 et al. [12] has been one of the most commonly used ADT datasets [28]. 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.

Explain how many samples + total duration of this dataset.

Another well-known MedleyDB Drums dataset, from Southall et al. [20]. This dataset is built on top of Bittner et al.'s MedleyDB dataset [3], 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.

Explain how many samples + total duration of this dataset too.

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.

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.

4.2 EMG-D

4.3 Slakh

4.4 ADTOF-YT

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, and extracting them to separate MIDI files.

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).

Chapter 5

Methodology

5.1 Task

The specific task we are trying to solve is one of drum instrument onset detection. For each timestep of an input spectrogram, we want to predict the probability of a drum onset happening.

Move this piece to background and expand ADT information there?

5.2 Pipeline

5.2.1 Preprocessing

Now explain what we do to the data before prediction. Explain the preprocessing steps we do afterwards (normalization, etc).

And explain how we preprocess the labels (target widening, etc.).

5.2.2 Training

Mention the loss function used, and why we use this (BCEWithLogitsLoss).

Mention the computation of infrequency weights, i.e. how they are computed, why they are computed, the intuition into how they will help us...

5.2.3 Postprocessing

Mention how model outputs a "confidence in event happening" distribution, which we want to discretize into events. I.e. explain Vogl's peak picking algorithm [26].

5.3 Experiments

Here we mention the setups for each of the experiments.

Mention that we use RayTune to train, with PyTorch models. Mention that we only used RayTune's FIFOScheduler, and how for random search / grid search we used their built in parameter space functionality.

Mention that every single experiment was trained for at most 100 epochs, with a early stopping if validation loss didn't decrease within 10 epochs. Mention the learning rate scheduler, where we reduce the learning rate by a factor of 5 if the model hasn't improved in the last 3 epochs.

Mention that we perform early stopping on the validation loss (and why, like the smooth nature, overfitting prevention, etc.), where as we store the best performing model based on the validation F1-score (due to this representing overall prediction performance).

Mention that every experiment is model selected using hold-out validation, and best model is chosen based on micro F1-score.

Chapter 6

Architecture Study

6.1 Methodology

This study's main goal train different architectures over different ADT datasets, and figure out if there exists any architecture which outperforms the other on the majority. This could answer the question; "Is there an architecture which is superiour when it comes to ADT related tasks?". For example. Some kind of introduction to the study more or less.

6.2 Results

Display a table of results, class-wise and micro-F1: Best architecture per dataset is bolded.

	ENST+MDB	E-GMD	Slakh	ADTOF-YT
Recurrent		0.8892	0.8502	0.9498
Convolutional		0.8657	0.8305	0.8290
Convolutional Recurrent		**0.8911**	**0.8914**	0.9455
Convolutional Transformer		0.8852	0.8828	0.9533
Vision Transformer		0.8818	0.8755	**0.9560**

Display the results in a barplot, to easily capture well-performing models.

6.3 Discussion

As we can see...

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.

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

[3] Rachel Bittner, Justin Salamon, Mike Tierney, Matthias Mauch, Chris Cannam, and Juan Pablo Bello. Medleydb sample, October 2014.
URL: https://doi.org/10.5281/zenodo.1438309.

[4] 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.

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

[6] Sungkyun Chang, Emmanouil Benetos, Holger Kirchhoff, and Simon Dixon. Yourmt3+: Multi-instrument music transcription with enhanced transformer architectures and cross-dataset stem augmentation, 2024.
URL: https://arxiv.org/abs/2407.04822.

[7] 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.

[8] 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.

[9] 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/.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] 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.

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

[17] 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.

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

[19] 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.

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

[21] Gilbert Strang. Wavelet transforms versus fourier transforms, 1993.
URL: https://arxiv.org/abs/math/9304214.

[22] 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.

[23] 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.

[24] Richard Vogl, Matthias Dorfer, and Peter Knees. Recurrent neural networks for drum transcription. 08 2016.

[25] 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.

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

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.