Transformer-Based Approaches for Automatic Music Transcription

A Thesis

submitted to the designated
by the Assembly
of the Department of Computer Science and Engineering
Examination Committee

by

Christos Zonios

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN DATA AND COMPUTER
SYSTEMS ENGINEERING

WITH SPECIALIZATION
IN DATA SCIENCE AND ENGINEERING

University of Ioannina
School of Engineering
Ioannina 2021

Examining Committee:

- **Aristidis Likas**, Professor, Department of Computer Science and Engineering, University of Ioannina (Supervisor)
- **Konstantinos Blekas**, Professor, Department of Computer Science and Engineering, University of Ioannina
- **Kostas Vlachos**, Assist. Professor, Department of Computer Science & Engineering, University of Ioannina

Acknowledgements

I would like to thank my supervisor, professor Aristidis Likas, for always being helpful, patient and kind, passing on to me the love for scientific research, showing me how to be measured and composed in my approach to new problems, answering every question I had and providing more ideas and solutions than I could possibly ever examine in my time working on this thesis.

A big thanks is also owed to Ioannis Pavlopoulos, who sacrificed many of his coffee breaks to patiently explain complicated NLP concepts to me, and provide analogies, visualizations and alternate viewing angles to difficult problems I faced.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Christos Zonios, M.Sc. in Data and Computer Systems Engineering, Department of Computer Science and Engineering, School of Engineering, University of Ioannina, Greece, 2021.
Transformer-Based Approaches for Automatic Music Transcription.
Advisor: Aristidis Likas, Professor.

Automatic Music Transcription (AMT) is the process of extracting information from audio into some form of music notation. In polyphonic music, this is a very hard problem for computers to solve as it requires significant prior knowledge and understanding of music language and the audio is subject to a multitude of variations in frequencies depending on many factors such as instrument materials, tuning, player performance, recording equipment and others.

Transformers are self-supervised models that have recently showed great promise as they use self-attention in order to learn contextual representations from unlabeled data. They have surpassed state of the art (SOTA) performance in various Speech Recognition (SR), Natural Language Processing (NLP) and Computer Vision tasks.

In this work, we examine transformer-based approaches for performing AMT on piano recordings by learning audio and music language representations. Specifically, we look at the popular SR model `wav2vec2` as a solution to the former and the NLP model BERT in order to perform Music Language Modelling (MusicLM).

We propose a new pre-training approach for MusicLM transformers based on an appropriately defined transcription error correction task. In addition, three novel models for AMT are proposed and studied that appropriately integrate `wav2vec2` and BERT transformers at various stages.

We conclude that a `wav2vec2` encoder model pre-trained on speech audio is not able to surpass SOTA models using mel-scale spectrograms and convolutional network encoders without significant conditioning on music audio.

We show that a BERT transformer pre-trained on natural language has transfer learning potential for MusicLM. We also examine the robustness of such a transformer for performing MusicLM, and find that we are able to achieve interesting results when doing Masked MusicLM and when replacing Recurrent Neural Networks with pre-trained transformers in SOTA models for AMT.

Εκτεταμενη Περιληψη

Χρήστος Ζώνιος, Δ.Μ.Σ. στη Μηχανική Δεδομένων και Υπολογιστικών Συστημάτων, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πολυτεχνική Σχολή, Πανεπιστήμιο Ιωαννίνων, 2021.
Προσεγγίσεις Βασισμένες σε Transformers για Αυτόματη Μετεγγραφή Μουσικής.
Επιβλέπων: Αριστείδης Λύκας, Καθηγητής.

Η Αυτόματη Μετεγγραφή Μουσικής (Automatic Music Transcription, AMT) είναι η διαδικασία εξαγωγής πληροφορίας από ένα σήμα ήχου σε κάποια μορφή μουσικής σημειογραφίας. Στην πολυφωνική μουσική, το AMT είναι ένα δύσκολο πρόβλημα για υπολογιστικά συστήματα καθώς χρειάζεται σημαντική εκ των προτέρων γνώση και κατανόηση της γλώσσας της μουσικής, και το σήμα του ήχου περιέχει πολλές διαφοροποιήσεις στις συχνότητες που περιέχει λόγω διαφόρων συντελεστών όπως τα υλικά του μουσικού οργάνου, το κούρδισμα, την εκτέλεση του κομματιού από τον οργανοπαίκτη, τον εξοπλισμό ηχογράφησης κ.α.

Οι transformers είναι μοντέλα βαθιάς μάθησης τα οποία εκπαιδεύονται με τεχνικές αυτο-εποπτευόμενης μάθησης (self-supervised learning) και χρησιμοποιούν αυτο-προσοχή (self-attention) ώστε να μάθουν αναπαραστάσεις που περιέχουν τα συμφραζόμενα από ακολουθίες δεδομένων χωρίς ετικέτα. Έχουν δείξει ότι ξεπερνούν σε απόδοση τα προηγούμενα state of the art (SOTA) μοντέλα σε πολλά προβλήματα μοντελοποίησης ακολουθιών όπως Αυτόματη Αναγνώριση Φωνής (Speech Recognition, SR), Επεξεργασία Φυσικής Γλώσσας (Natural Language Processing, NLP) και Υπολογιστικής Όρασης (Computer Vision).

Σε αυτή την εργασία εξετάζουμε προσεγγίσεις βασισμένες σε transformers για την υλοποίηση συστημάτων AMT σε ηχογραφήσεις εκτελέσεων μουσικών κομματιων σε πιάνο, μαθαίνοντας αναπαραστάσεις για το σήμα του ήχου και τη γλώσσα της μουσικής. Συγκεκριμένα, χρησιμοποιούμε το δημοφιλές SR μοντέλο wav2vec2 για να εξάγουμε αναπαραστάσεις από το σήμα του ήχου, και το NLP μοντέλο BERT για

να κάνουμε Μοντελοποίηση της Μουσικής Γλώσσας (Music Language Modelling, MusicLM).

Προτείνουμε μια νέα τεχνική προεκπαίδευσης (pre-training) για μοντέλα transformers βασισμένη σε διόρθωση λαθών της μετεγγραφής μουσικής, καθώς και τρία νέα μοντέλα για AMT.

Συμπεραίνουμε πως το μοντέλο wav2vec2 προεκπαιδευμένο σε σήμα ήχου ομιλητικής φύσης δεν καταφέρνει να ξεπεράσει τα καλύτερα μοντέλα που χρησιμοποιούν mel-scale φασματογράμματα και συνελικτικά νευρωνικά δίκτυα, χωρίς να εκπαιδευτεί με σήμα ήχου μουσικών κομματιών.

Δείχνουμε πως ένας BERT transformer προεκπαιδευμένος σε φυσική γλώσσα έχει μεγάλες δυνατότητες μεταφοράς μάθησης σε MusicLM. Εξετάζουμε επίσης την ευρωστία ενός τέτοιου transformer για MusicLM, και βρίσκουμε πως προκύπτουν ενδιαφέροντα αποτελέσματα όταν εφαρμόζουμε Masked MusicLM, καθώς και όταν αντικαθιστούμε τα επαναληπτικά νευρωνικά δίκτυα (Recurrent Neural Networks) με προεκπαιδευμένους transformers στα SOTA μοντέλα για AMT.

Chapter 1

Introduction

1.1 Automatic Music Transcription

Automatic Music Transcription (AMT) [1] is the process of automatically converting an audio signal to a high-level representation of the musical information present in it. When musicians perform transcription of music, they listen to the audio and use some form of music notation to generate a human-readable representation of that audio. One would later be able to use this representation to perform the music, by interpreting this notation. A subfield of *Music Information Retrieval (MIR)*, AMT has been studied extensively due to its applications in musical analysis, teaching of music, annotation and others.

1.1.1 Piano Music

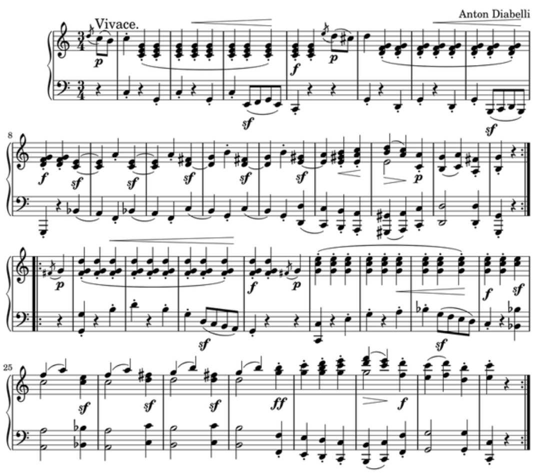
Piano music is comprised of 88 notes. When a piano key is pressed, a small hammer hits three strings, which vibrate. Once hit, they immediately produce a sound (”attack” of the note). The strings keep vibrating and after a while the sound decays slowly, unless the player stops pressing the key and is not using the sustain pedal, in which case the note is quickly silenced. The pitch is considered active from the moment of the attack up to the point where it has decayed below a certain volume threshold, subjective to the transcriber. Music on the piano can range from simple to fairly complex. The music can move slowly or fast (variable tempos), individual notes can be loud or soft (variable dynamics), many can be active at any time (variable melodic and harmonic structures), and bars can be structured in different counting systems (variable time signatures). Piano notes range from very low to very high pitched.

1.1.2 Music Transcription

Music transcription is the process of notating a piece of music by means of listening to a recording or live performance of said piece and deriving, using prior knowledge of music, the musical information present in it in as much detail as possible, so that one could use that notation to recreate the musical performance. Transcription could also mean arranging a piece of music for a different instrument, however when we use the term in this thesis, we are referring to the former definition. A visualization of music transcription is shown in Figure 1.1.

As the piano is an acoustic instrument, and one that needs to be tuned, pressing a key does not necessarily result in the same pitch every time. Furthermore, the frequency (or pitch) of each note is accompanied by different frequencies (called harmonics and disharmonics) due to the materials of the piano, acoustics of the room, vibrations of the strings affecting each other, recording techniques and of course the human touch of the musician. Transcribing a single sequence of non-overlapping notes (called a melody) is somewhat trivial to do since the fundamental frequency of the note is always the most prominent, and the note can be inferred from the frequency. However, in polyphonic music notes can overlap making the task very complex, as harmonics, disharmonics and fundamental frequencies can become mixed together and make it difficult to not only infer how many notes are present at each time but also which ones they are, since a harmonic produced by many pitches might overpower a fundumental of another pitch.

(a) An audio wave



(b) Musical notation (sheet music)

Figure 1.1: The process of transcription extracts the musical information from the sound (left) into human-readable music notation (right)

1.1.3 MIDI

Musical Instrument Digital Interface (MIDI) [2] is a technical standard that describes a communications protocol, digital interface, and electrical connectors that connect a wide variety of electronic musical instruments, computers, and related audio devices for playing, editing and recording music. In this thesis, we will use the term in only the digital interface definition.

MIDI, as a digital interface standard, is analogous to a digital form of music notation (sheet music). In this thesis, the desired output of our model is a matrix that can be directly used with some MIDI interface library to produce a MIDI file containing the exact same notes, starting and ending at the same time, as in the input audio file.

For all models implemented in this work, we adopt a piano roll representation of MIDI information shown in Figure 1.2. A piano roll is a sequence of vectors of size 88, since the piano has 88 keys (notes). Each vector represents a time frame. Hereafter, it will be referred to simply as *"frame"* .

1.1.4 Music Language Modelling

Music Language Modelling (MusicLM) is the musical equivalent of language modelling in *Natural Language Processing (NLP)*. By modelling the language of music,

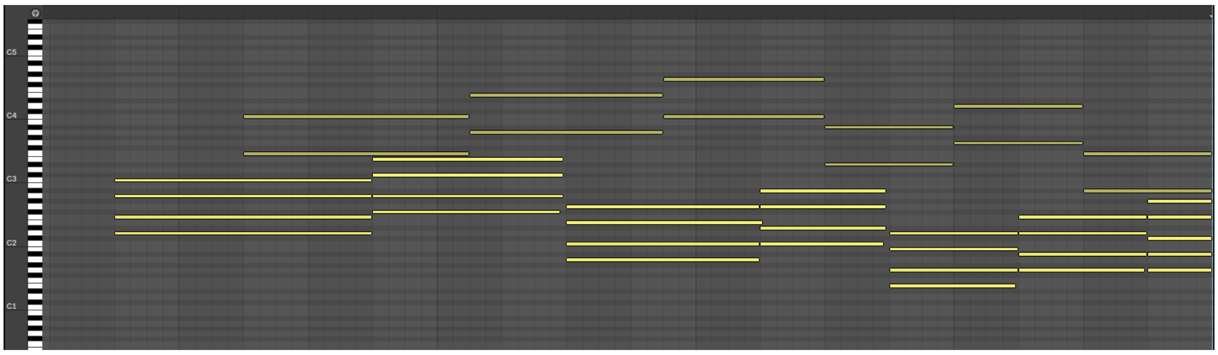


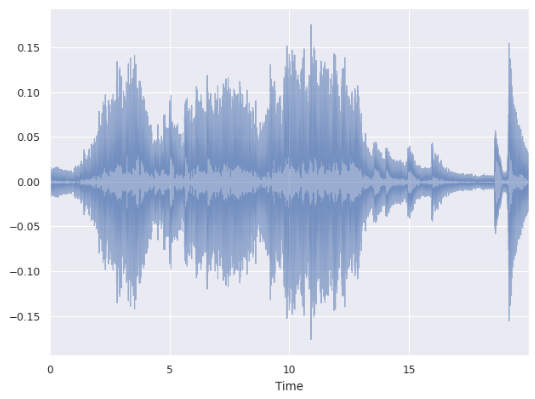
Figure 1.2: Piano composition in MIDI format as represented in a piano roll (picture taken from recording software)

and specifically its temporal, melodic, rhythmic and harmonic structure as well as emergent patterns and repeated passages, we can not only further understand it but also create better representations and abstractions. It is an essential step towards solving various MIR problems. In the context of AMT, better MusicLM allows us to predict more realistic transcriptions [3], improve transcription accuracy and increase the confidence of the model predictions.

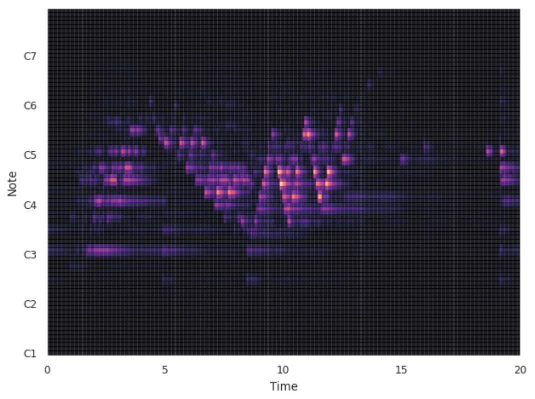
1.2 Audio Representations

Sound, or audio, is a continuous signal representing amplitude over time. When recorded with modern equipment it is usually sampled thousands of times per second. To sample audio, one would use a microphone, a device containing a diaphragm which works like an eardrum and moves according to vibrations in the air, turning those vibrations into an electrical signal. That signal is then optionally passed through a pre-amplifier and turned into a discrete signal through sampling, resulting in a WAVE file. The standard CD quality sampling rate for music audio is 44.1kHz.

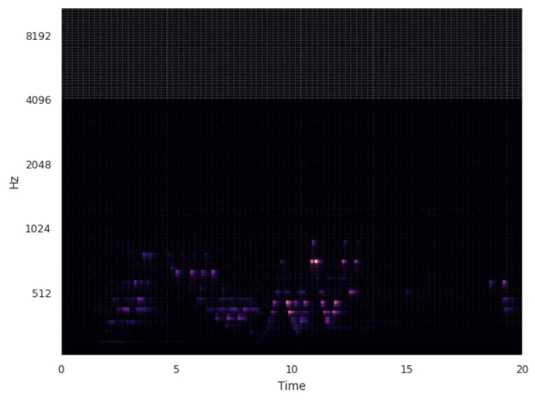
Most approaches for AMT preprocess the audio into a frequency representation ([4], [5], [6], [7] and [8]), using the *Short Time Fourier Transform (STFT)* and scaling the output spectrogram to the *Mel Scale* [9], or the *Constant-Q Transform (CQT)* [10]. Figure 1.3 shows various audio representations.



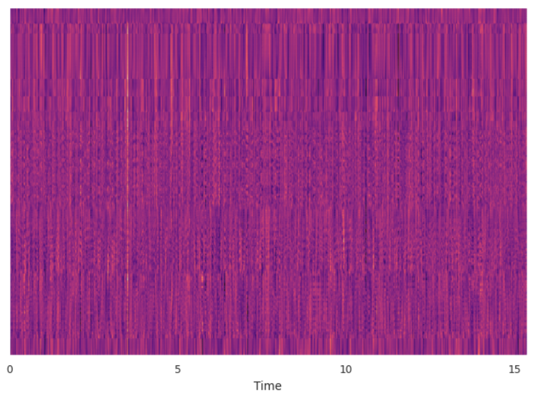
(a) Raw audio wave



(b) CQT of audio



(c) Mel-scale spectrogram of audio



(d) Neural network intermediate features

Figure 1.3: Different audio representations

1.3 Deep Learning

Deep Learning [11] is an area of Machine Learning where neural networks with many layers are used in an attempt to learn a hierarchy of features for solving a problem. Deep neural networks have exhibited near human-like performance in tasks that have been traditionally hard for computers to solve, such as computer vision, speech recognition and automatic translation, even surpassing humans in some cases [12] [13] [14].

1.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [15] [16] have found extreme success in fields such as Computer Vision and Speech Recognition, in some cases of image recognition problems even outperforming human classification [17].

The design of CNNs is used to model the connectivity patterns of neurons in the visual cortex of the brain. Cortical neurons are connected with only a region of neurons, resulting in having a smaller receptive field than the entire visual field. Intuitively, this means that starting from a single pixel, a CNN layer might be trained to detect edges at certain angles, light or dark spots and similar relatively simple features. The next layer might learn more complex patterns, like edges with certan angles and simple shapes. By stacking convolutional layers, we are able to create systems that recognize very complex patterns like different faces, objects and scenes.

In AMT, like Speech Regognition, CNNs are used to extract better intermediate features ([4], [6], [7] and [8]) when fed with spectrogram images, distinguishing between patterns of active notes or even combinations of notes and their respective frequencies.

1.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [18] are used to model temporal sequences. They use their internal state as memory to process sequences of inputs. They have been used extensively in problems such as speech recognition, as well as in AMT. *Long Short-Term Memory (LSTM)* [19] Networks are a type of RNN that mostly negates the vanishing and exploding gradient problems of previous RNN implementations, by using a ”forget gate”, which gives each LSTM cell a small probability allow gradients during backpropagation to flow unchanged through it.

In the context of AMT, LSTMs have been successfully used to improve performance and achieve state of the art (SOTA) performance [4].

RNNs are not feedforward networks, which makes each neuron's output depend on not just the previous layer's outputs, but also outputs of neurons in the same layer. This introduces complexity and does not allow for massively parallel processing as is the case with feedforward networks, slowing down training and inference. Their resulting features are also not particularly interpretable, which is a problem when a system needs to be explainable. In this thesis, we will examine emergent temporal sequence modelling architectures which do not suffer from these flaws.

1.3.3 Autoencoders

The emergence of big data has resulted in new problems and better, larger datasets of various types. However, as data labelling by humans is very time and resource consuming, these datasets cannot be used by traditional supervised learning models. Self-supervised methods can use unlabeled data to learn better representations, called embeddings, of the input data. These embeddings may then be used in place of the input data in various models, requiring much less labeled data for fine-tuning to particular supervised tasks [20].

Autoencoders [21] are self-supervised neural network architectures which learn efficient representations of input data. An autoencoder is given an input and learns to reconstruct the input signal in the output. Intermediate layer outputs can then be used to provide efficient representations of the input data. It also achieves dimensionality reduction, using intermediate layers that have a smaller dimensionality than the input.

1.4 Transformers

Transformers [20] are a class of autoencoders with the addition of an *attention* mechanism, described below, which is used to add positional encodings and learn to weigh the significance of different parts of the input data. Like RNNs, they are used to handle sequential data and produce embeddings that include contextual information of different scales, for example at the character, word, sentence, paragraph and whole text level. Contrary to RNNs, they are massively parallelizable and conceptually simpler, as they are feedforward networks.

Transformer architectures have introduced great improvements in a variety of sequence modelling problems, not only showing robustness in their particular application but also promising potential for transfer learning, by fine-tuning pre-trained models with just one additional layer to achieve SOTA performance in similar applications, using only a fraction of resources such as available data, time and computational power needed by previous approaches [22] [23] [24].

Another advantage of transformers over previous sequence modelling architectures is the ability to inspect the outputs of their attention heads, visualizing the sequence elements that were most important in a prediction [25] which can provide valuable insight into the decision process of the network, leading to explainable AI.

1.4.1 Architecture

Transformers are encoder-decoder models. Specifically, a transformer is comprised of an encoder stack of N encoder layers and a decoder stack of N decoder layers. The encoder stack maps an input sequence of discrete representations $\mathcal{X} : (x_1, x_2, ..., x_n)$ to a sequence of continuous representations $\mathcal{Z} : (z_1, z_2, ..., z_n)$, while the decoder stack generates an output sequence $\mathcal{Y} : (y_1, y_2, ..., y_m)$, one element at a time, at each step i using the encoder representations $\mathcal{Z}$ and previous outputs $(y_1, ..., y_{i-1})$ as additional inputs. In other words, the encoder stack is fed the entire input sequence, while the decoder stack is fed the encoder output as well as the decoder outputs of previous steps.

Each encoder layer is composed of two sub-layers: a multi-head self-attention mechanism, described below, and a position-wise fully connected network, with a residual connection around each of the two sub-layers that is followed by layer normalization. The output of each sublayer is $LayerNorm(x+Sublayer(x))$. All sub-layers have the same dimension d_{model}.

Similarly, each decoder layer is composed of three sub-layers: a masked multi-head self-attention mechanism on the output embedding, then a multi-head self-attention mechanism on the output of the encoder stack, and finally a position-wise fully connected network. All sub-layers employ residual connections like in the encoder stack. The masked multi-head attention module prevents the decoder from attending to subsequent positions, and since the outputs are shifted right, it only has access to output embeddings that are already known.

1.4.2 Self-Attention

Transformers employ an attention mechanism in order to encode sequence element positions and produce representations that include contextual information. Each input element is projected into an embedding (continuous representation) through a linear (fully-connected) layer, producing a ”word embedding”, which is a semantic representation of the input. The element’s position in the sequence is also projected into an embedding of the same size, producing a ”positional embedding”, and the two embeddings are then added or concatenated together to form the input embedding to the next layer.

A self-attention module works by comparing every input embedding with all input embeddings of the sequence, itself included, and reweighing the word embeddings of each word according to their relevance to the meaning of the word in question. This operation is defined using a *query* vector and a *key* and *value* vector pair, mapped to an output. The key, value and query vectors are all representations of the input and are produced using a linear layer for each one, with the embedding as their input.

The key and value vectors for all input embeddings are used at each step, while the query vector is only calculated for the embedding of the current sequence element. This essentially means that the attention score (weight) is computed for all embeddings with regards to the current embedding, for each embedding.

An intuitive way to think about this operation is that it asks the question: "how much does each element in the sequence contribute to the *meaning* of the current element in this context"? In other words, how much *attention* should the network pay to each embedding in order to make a decision about the current embedding? The keys (each embedding of the sequence) and the query (current embedding) answer the question to produce an attention score, and the value of each sequence element is weighted by its corresponding attention score to produce the output for the current embedding.

Mathematically, the output of the self-attention module is defined as the sum of the values weighted by a similarity function of the query and the corresponding key. In the original transformer [20], Scaled Dot-Product Attention is used, which uses a dot-product as the similarity function:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where d_k is the embedding vector size, seq_len is the number of sequence elements, and $Q_{seq\_len \times d_k}$,$K_{seq\_len \times d_k}$ and $V_{seq\_len \times d_v}$ are matrices representing a sequence of queries, keys and values respectively. The scaling factor $\frac{1}{\sqrt{d_k}}$ is used to counteract large dot-products pushing the Softmax function to have small gradients.

Multi-head Attention

Multi-head attention is a technique that allows the model to attend to information from different contexts at different positions in the input sequence. Q, K and V matrices are passed in parallel through h linear layers followed by Scaled Dot-Product Attention layers, and the outputs are then concatenated and passed through another linear layer to project them to the desired dimension:

$$MultiHead(Q, K, V) = concat(head_1, head_2, ..., head_h)W^O$$

where $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$ and W_i^Q, W_i^K, W_i^V and W^O are the weights of the linear layers with inputs Q, K, V and the output of the concatenation $O_{h \times d_v}$ respectively.

1.4.3 `wav2vec2` Model

`wav2vec2` [23] is a transformer-based pre-training framework for self-supervised learning of speech representations. These representations can be used in a variety of speech audio-based supervised learning tasks such as Speech Recognition to achieve and even surpass state-of-the-art methods while using orders of magnitude less labelled data.

The base `wav2vec2` model accepts audio sampled at 16kHz and produces embedding vectors of dimension 768. Each embedding vector has a receptive field of 400 samples which corresponds to 25ms. Embedding vectors are spaced by 320 samples (20ms). We will hereafter refer to this spacing as `hop_length`.

The model is composed of a multi-layer temporal convolutional feature encoder $\mathcal{X} \rightarrow \mathcal{Z}$ that takes in raw audio $\mathcal{X}$ as inputs and outputs latent speech representations $\mathcal{Z}$. Those representations are then fed into a transformer $\mathcal{Z} \rightarrow \mathcal{C}$ that builds representations $\mathcal{C}$ capturing contextual information from the entire input sequence. The outputs of the feature encoder are also discretized using a quantization module $\mathcal{Z} \rightarrow \mathcal{Q}$ and are used as the targets $\mathcal{Q}$ in the self-supervised objective. Self-attention captures dependencies over the entire input sequence of latent representations.

The model is trained using a masking approach, where some time-steps p are randomly sampled to be starting indices, and the following M latent representations are then masked. The quantized representations are not masked. The learning objective is then to predict the correct quantized representations for all masked latent representations.

1.4.4 BERT

Bidirectional Encoder Representations from Transformers (BERT) [22] is a transformer-based pre-training technique for NLP tasks, which achieved state-of-the-art performance on multiple natural language understanding tasks. By design, BERT pre-trains on unlabeled text to learn bidirectional representations by conditioning on both left and right contexts.

The BERT model usually works using sequences of special text representations called `token_ids`. Generally, a tokenizer is trained on a large text corpus of the language it would be applied to in order to be able to process sentences into a series of *tokens* belonging to its learned vocabulary of a standard size. Each token is either a word or a subword, and special tokens are used to represent unknown words, masked words, separators, ends or beginnings of sentences etc. This tokenizer is then used during training and inference to generate those tokens from the input and output sentences.

The input of the BERT model is a sequence of `token_ids`, representing a `sentence`. It produces embedding vectors of dimension 768.

It uses an internal learned lookup table to transform these descrete `token_ids` into continuous vectors, and adds to them a positional embedding and a segmentation embedding to denote which sentence they belog to.

During pre-training with unlabeled data, the BERT model uses two tasks: *Masked Language Modelling* (MLM) and Next Sentence Prediction. For the MLM task, it picks 15% of the input tokens at random. It replaces 80% of them with the `[MASK]` token, 10% of them with a random token, and 10% of them with the unchanged token. It is then asked to predict each picked token using cross-entropy loss. The Next Sentence Prediction task is conceptually simple: for each training example, pick two sentences A and B. 50% of the time, B actually follows A in the text, while the other 50% B is a random sentence from the corpus. The model is tasked to predict whether sentence B actually follows sentence A, with target the binary label `isNext`.

1.4.5 Music Transformers

A few transformer models exist for music tasks, but could not be used in this work for various reasons. However, we decided to include them here for completeness.

The Music Transformer [26] is a model that uses a sparse, MIDI-like representation of music to train a transformer with relative attention, on the task of harmonizing a melody (predicting the rest of the voices). This is unsuitable for an end-to-end AMT system as the tokenization process is not differentiable and loss cannot be propagated back from the music representations. It could theoretically be useful in improving transcriptions of an existing AMT model as an extra decoder step, however the authors have not released a pre-trained checkpoint yet, and we are interested in an end-to-end solution.

Wave2Midi2Wave [27] builds on the Music Transformer by introducing a WaveNet model for generating audio from the symbolic representations. It also uses the Onsets and Frames transcription model [4] to work with raw audio as its input, and shows that an AMT model can be used to provide music representations for pre-training a transformer. The model does not train Onsets and Frames further, as training cannot be done end-to-end for the reason stated above, and the authors did not examine the potential for improving transcription performance using the transformer as a decoder.

Finally, Jukebox [28] is a generative model for music, trained on raw audio. It is built similar to `wav2vec2` as it uses quantized representations of the input audio. It additionally introduces a spectral loss, which encourages the reconstruction to contain the same frequencies as the input by comparing their spectrogram representations. The model performance is impressive, however the top-level prior part of the model, which might be useful in producing audio representations for AMT, has 5 billion parameters and requires very large amounts of GPU memory as well as computing power for pre-training. Theoretically, the top-level prior part could be used and fine-tuned for AMT, but this was unfeasible on our system, therefore it is not used in this work.

1.5 Thesis Objectives

This thesis addresses the emergence of transformer-based approaches in the fields of NLP and Speech Recognition in the context of improving AMT and MusicLM. Specifically, we examine the use of transformers to extract better intermediate features (embeddings) from raw audio, bypassing the usual preprocessing steps of transforming the data to the frequency domain and scaling to some logarithmic scale. Instead, we attempt to learn the best feature extraction process using all of the available information. We also examine text-based autoencoders as a means to introduce improvements in MusicLM, resulting in more realistic transcriptions. As transformers have large potential for transfer learning, this would additionally enable any viable solutions to be used in MusicLM tasks other than AMT.

The research questions in this thesis are:

- can we generate good representations from the raw audio using a transformer, bypassing the need for a preprocessing step and capturing better contextual information?
- can a pre-trained natural language modelling transformer model learn to model music language?
- can we replace RNNs with language modelling transformers, creating more realistic transcriptions?
- finally, can we improve on the SOTA using the aforementioned techniques?

1.6 Thesis Structure

The remainder of this thesis is structured as follows: In Chapter 2 we introduce the approaches used to answer the research questions. In Chapter 3 we provide implementation details for our approaches. Chapter 4 describes the experiments we designed to evaluate the efficacy of the proposed methods. Chapter 5 presents the results of the experiments. Finally, in Chapter 6 we draw some conclusions, discuss our findings, and propose future work directions.

Chapter 2

Transformer-Based Approaches

2.1 Obtaining Better Music Representations

A first approach is to train a `wav2vec2` model from scratch using music audio data, however that is infeasible in the scope of this thesis due to computational resource constraints. Instead, we assume that speech representations adequately transfer to music audio, and fine-tune the model with a transcription head instead.

The first proposed model is conceptually simple: pass a sequence of audio samples directly to a `wav2vec2` model pre-trained on speech audio, acquire the embeddings and add a fully connected layer on top, as shown in Figure 2.1 (a). An input sequence of N audio samples produces a sequence of approximately $\frac{N}{\texttt{hop\_length}}$ embeddings[1]. This leads to two approaches:

1. *sequence-to-one*: The goal of the classifier is to predict the frame right in the middle of this sequence - use a sliding window to predict all frames

[1]The precise number of embeddings is $\frac{N}{\texttt{hop\_length}} + 80$, as the receptive field (400 samples) is not a multiple of the hop_length (320 samples), thus we have a remainder of 80 samples.

2. *sequence-to-sequence*: The goal of the classifier is to predict the frame corresponding to each embedding

Preliminary experiments for this encoder architecture were done using the first approach; however, all other experiments were done using sequence-to-sequence models, as they performed better and required less time to train.

In practice, the difference between the two model architectures is only the top layer, as shown in Figure 2.1. In the sequence-to-one approach, the fully-connected layer takes in the feature representations of all sequence elements and outputs the prediction for a single element. The shape of the top layer is ($\texttt{sequence\_length} \times \texttt{embedding\_dim}$) $\rightarrow$ (88).

Conversely, in the sequence-to-sequence approach, the fully-connected layer takes in the feature representations of a single sequence element and outputs the prediction for that element, for each element of the sequence. This means that the layer is of shape ($\texttt{embedding\_dim}$) $\rightarrow$ (88). The same layer is implicitly repeated for all sequence elements, and predictions for the whole sequence can be calculated in parallel.

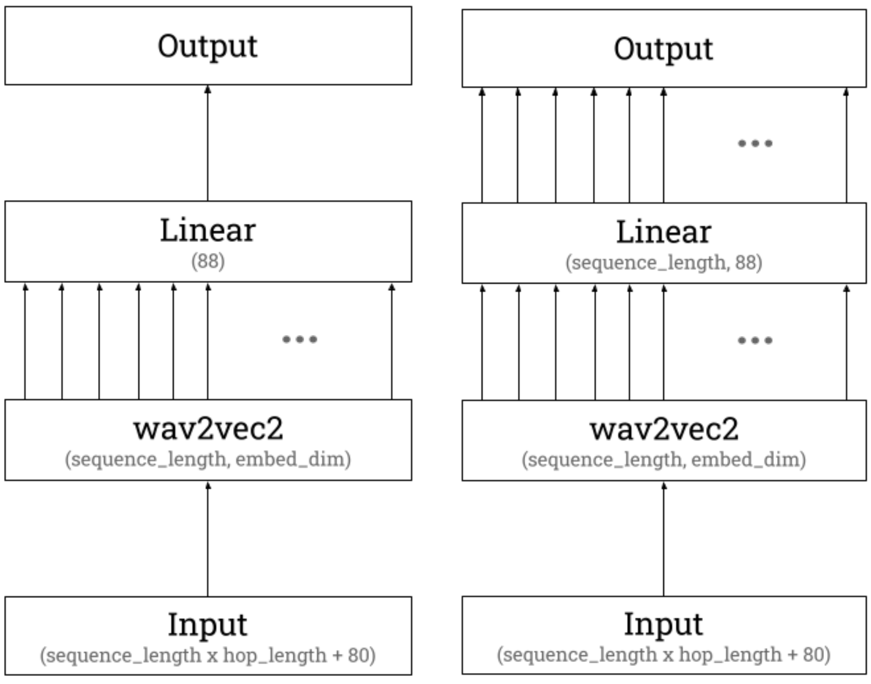
The latter approach is easier to train as the layer has significantly less weights. An added benefit to the sequence-to-sequence approach is that any length sequence can be given, which is not the case with the sequence-to-one approach, as in that case the `sequence_length` must be known and is part of the network architecture.

2.2 Decoding Musical Sequences & Music Language Modelling

2.2.1 Evaluating BERT for Music Language Modelling

In order to evaluate the efficacy of a BERT in the context of improving AMT accuracy by acting as a decoder, we first preprocessed the dataset labels into text representations, and trained a tokenizer using the N most frequent words (each word represents a frame). A DistilBERT [29] model, later replaced with a full BERT model, with a MLM head was subsequently trained on those N most frequent tokens. Both the model performance in the f_1 score and the subjective measures of listening to model masked predictions on music sequences showed a lot of promise for MusicLM.

The problem with this approach is that the string encoding and tokenization steps are not differentiable, and thus the model cannot be trained end-to-end. A simple


Output
Linear
(88)
wav2vec2
(sequence_length, embed_dim)
Input
(sequence_length x hop_length + 80)
Output
Linear
(sequence_length, 88)
wav2vec2
(sequence_length, embed_dim)
Input
(sequence_length x hop_length + 80)

(a) sequence-to-one model (b) sequence-to-sequence model

Figure 2.1: The same model with a different head for two approaches to sequence classification

solution to this problem is to remove the string encoding and tokenization altogether and instead pass the input embeddings directly, bypassing BERT's internal learned lookup tables that transform the input token ids into embedding vectors.

This approach works well and within a few epochs the model achieves an f_1 score higher than 90. After doing this, an end-to-end solution was trained and preliminary testing showed promising results, shown in Section 4.3.

2.2.2 Music-Encoded Pre-Training

Next, we investigate a different pre-training approach for a BERT-like decoder. Since MLM and Next Sentence Prediction are not very suitable for the task of AMT, we introduce a training technique based on the addition of noise to the input data, in order to mimic transcription errors. To our knowledge, this is the first attempt for creating a music transcription error detection/correction model.

The addition of noise is made as follows: for each frame, each active note has a 50% chance of becoming inactive, and each inactive note has a 10% chance of becoming active. This is modelled after transcription errors, which are usually false negatives, as the classifier does not detect a note is active if other notes are active, with higher velocities, in that frame. False positives can also occur, usually when other, similar notes are activated, for example when another note contains harmonics of the same frequency as the false positive. Negative samples are also vastly more common than positive samples. This approach also worked well and the model managed to achieve a high f_1 score within a few epochs.

In order to train the BERT-like decoder, a fully-connected layer was used on top of the BERT network, taking in the embeddings and outputting a sequence of frame predictions. At the base of the network, another fully-connected layer is added in order to transform the input noisy frames into embeddings the same size as the ones BERT uses internally.

Note that BERT is the pre-training method, and we do not use it in this approach, instead using only the model architecture, which is a bidirectional transformer. As we often used a standard BERT checkpoint for NLP tasks as a starting point, we will hereafter refer to this bidirectional transformer as a BERT model, disregarding whether the pre-training approach was used or not.

2.3 Encoder-Decoder Architectures (Wav2Vec2Bert)

An approach to a complete end-to-end model is then simply to combine the encoder model with the decoder model, with an added dropout layer in between to reduce overfitting. A simplified end-to-end model is also used, where the encoder embeddings directly go into the decoder, bypassing the two layers that are needed when training the encoder and decoder parts separately. Figure 2.2 compares the architectures of both models.

We will hereafter refer to this model as ”Wav2Vec2Bert”. The simplified model is used as preliminary experiments showed no loss in performance over the full model while having fewer parameters.

2.4 Onset and Offset Detector Architectures

The *Onsets and Frames* [4] model showed that by adding an onset detector, the model is able to provide more realistic transcriptions and surpass SOTA performance. Cheuk et al. revisited the model with additive attention and showed that the onset locations are the most important feature in the decision process of the model [30]. Thus, our final two models both include both an onset and an offset detector submodule.

2.4.1 Eurydice

The next proposed model is an adaptation of the Onsets and Frames model with the improvements described in [27], with the following changes:

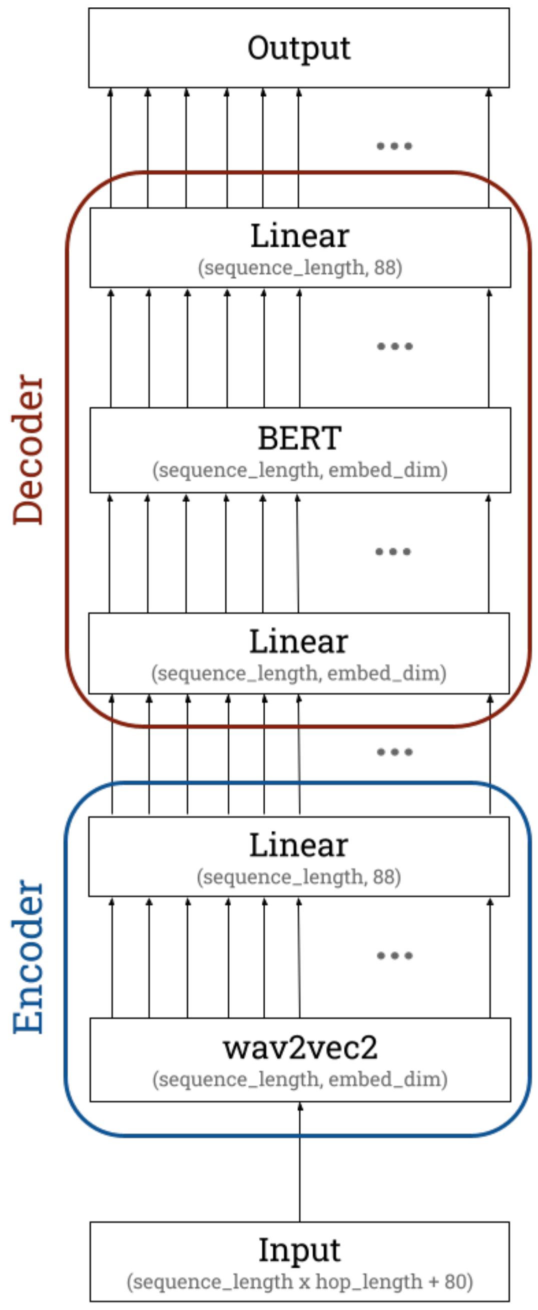
- Each input is a sequence of raw audio instead of a mel-spectrogram
- Convolutional stacks are replaced with a pre-trained `wav2vec2` encoder
- BiLSTM layers are replaced with a pre-trained BERT decoder
- The velocity prediction part of the model is removed

We will hereafter refer to this model as ”Eurydice”. The base Eurydice model is presented in Figure 2.3.

2.4.2 Orpheus

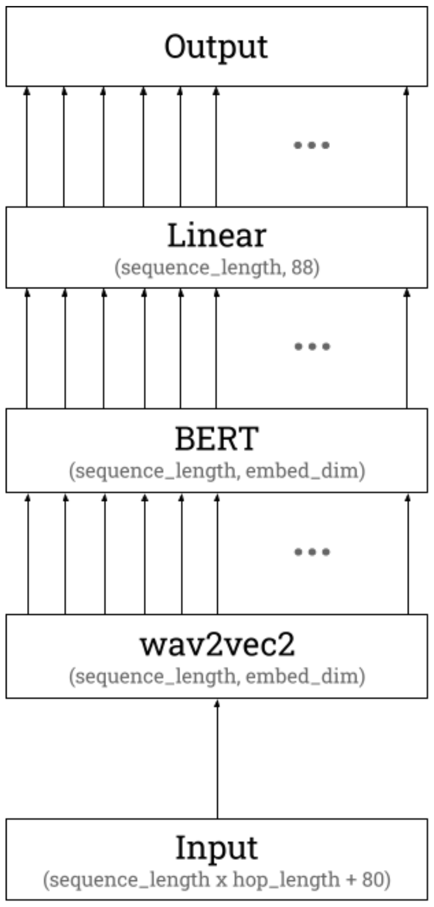
Finally, we propose another *Onsets and Frames*-based model, keeping the preprocessing of the raw audio into a mel-spectrogram and swapping the BiLSTM layers with BERT transformers in an attempt to improve the music language modelling part of the model.

We will hereafter refer to this model as ”Orpheus”. The base Orpheus model is presented in Figure 2.4.



Output
Decoder
Linear
(sequence_length, 88)
BERT
(sequence_length, embed_dim)
Linear
(sequence_length, embed_dim)
Encoder
Linear
(sequence_length, 88)
wav2vec2
(sequence_length, embed_dim)
Input
(sequence_length x hop_length + 80)

(a) Wav2Vec2Bert



Output
Linear
(sequence_length, 88)
BERT
(sequence_length, embed_dim)
wav2vec2
(sequence_length, embed_dim)
Input
(sequence_length x hop_length + 80)

(b) Simplified model

Figure 2.2: Wav2Vec2Bert: an end-to-end model connecting the encoder and decoder

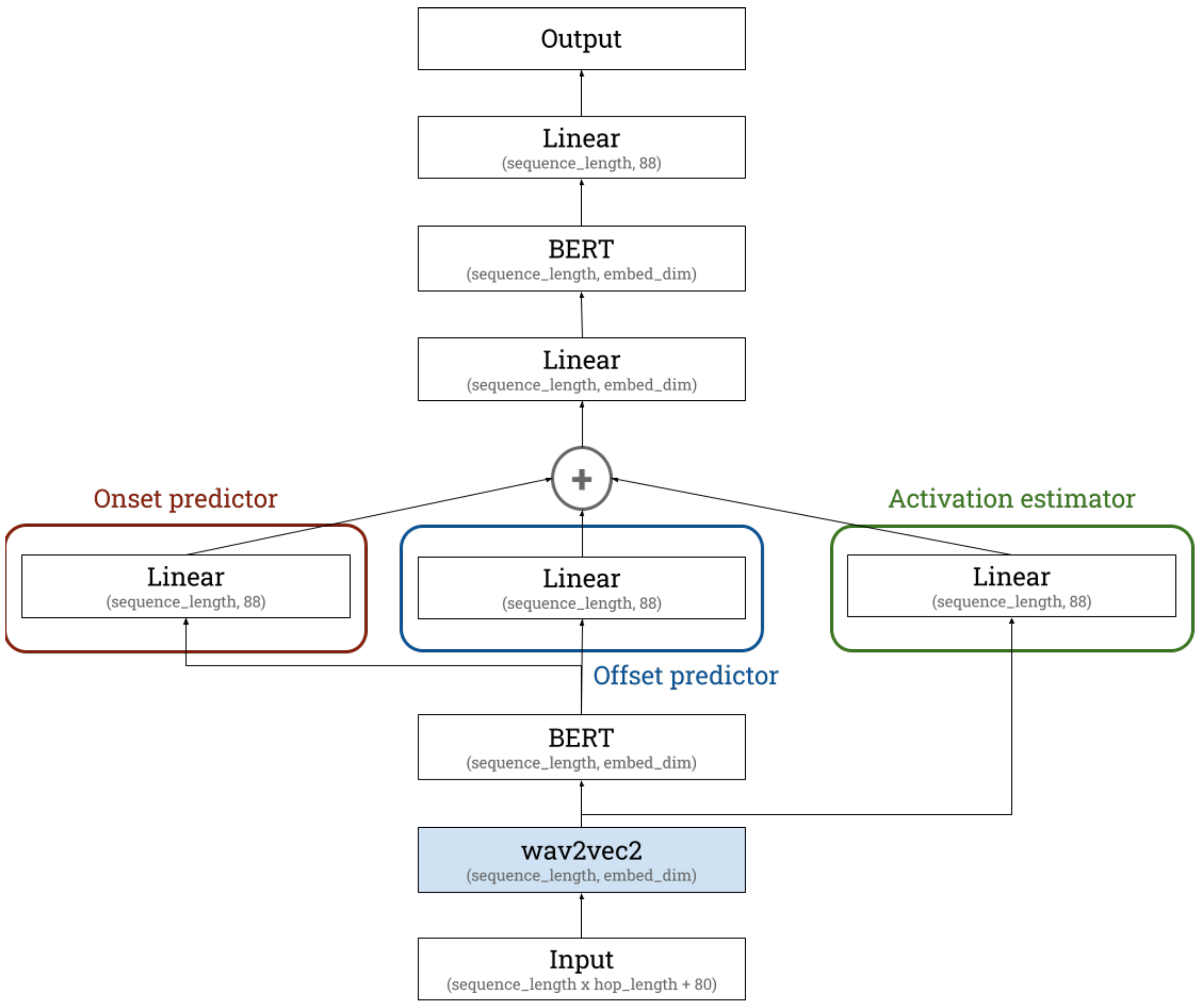


Figure 2.3: Eurydice base model. Modules with frozen parameters are in light blue.

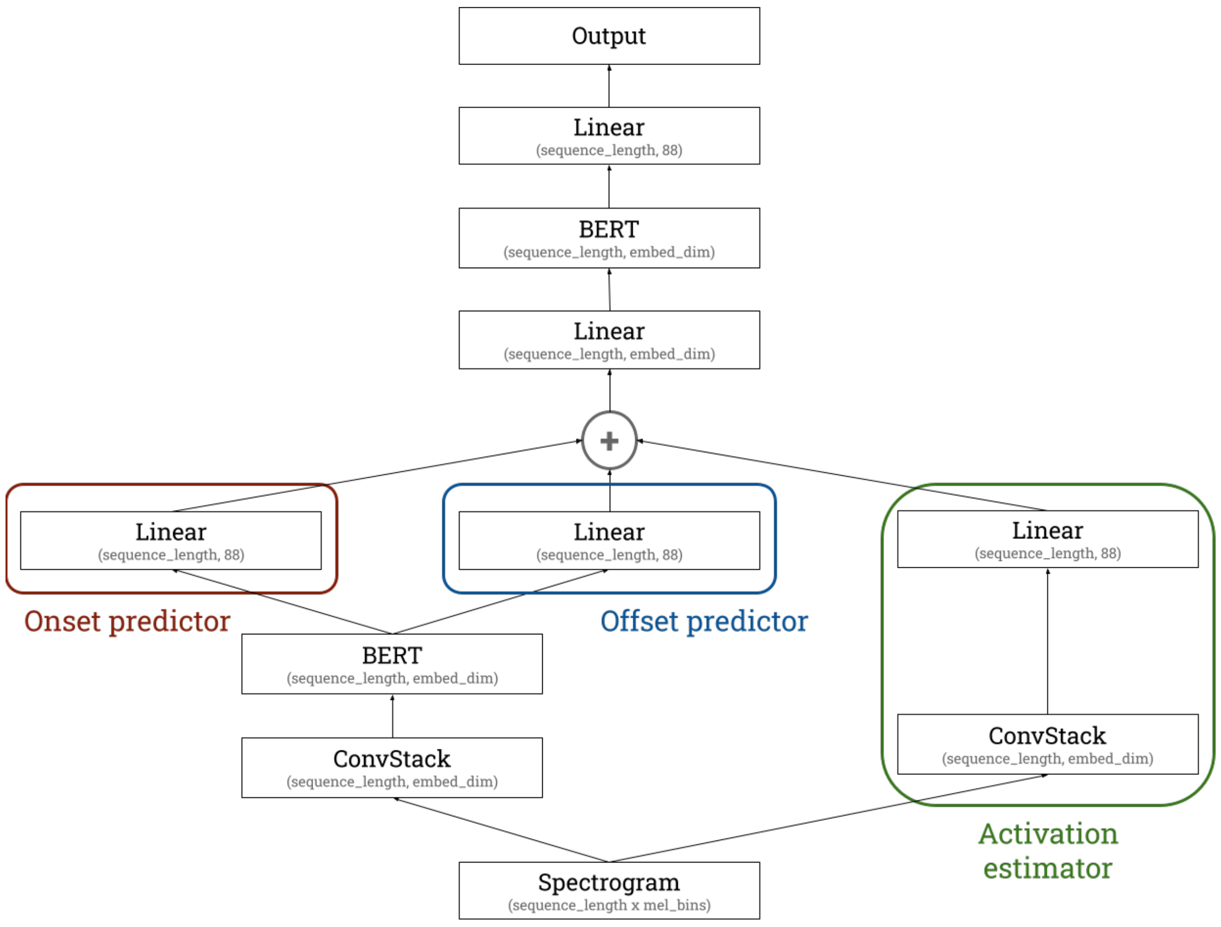


Figure 2.4: Orpheus model

Chapter 3

Implementation

3.1 Dataset and Preprocessing

3.1.1 MAESTRO Dataset

The MAESTRO (MIDI and Audio Edited for Synchronous TRacks and Organization) [27] is a music dataset containing about 200 hours of piano performance audio recording and MIDI pairs. The recordings are real virtuosic piano performances from ten years of International Piano-e-Competition, and they are aligned finely (around 3ms accuracy) with their corresponding MIDI files. The dataset is split by year and then by individual piece. It is part of the Magenta project by Google and freely available.

Audio Files

The file type used for storing uncompressed audio data is called a WAVE file and has the `.wav` extension. For this work, we used a preprocessing script to turn all WAVE files into `.flac` (*Free Lossless Audio Codec*) files to save space. The `librosa` library was used to load the audio from the files, using a sample rate of 16kHz.

MIDI Files

MIDI files (extension `.midi` or `.mid`) contain *events* describing musical events such as a particular note turning on or off at a certain time, with a certain velocity. We use the `pretty_midi` library in order to read that musical information and turn it into a piano roll representation as described in Section 1.1.3.

The piano roll is a tensor of shape $(N_{frames} \times 88)$, where each frame corresponds to `receptive_field` input audio samples and contains the note information for that time frame. A frame is a vector of size 88, each element being either an integer (1 for onset, 2 for inside, 3 for offset) in the case of models with an onset and/or offset detector, or a bit representing whether each piano note was active (1) or not (0) for all other models. Each frame's receptive field is spaced by `hop_length` samples from the next frame.

3.1.2 String Representation and Tokenization

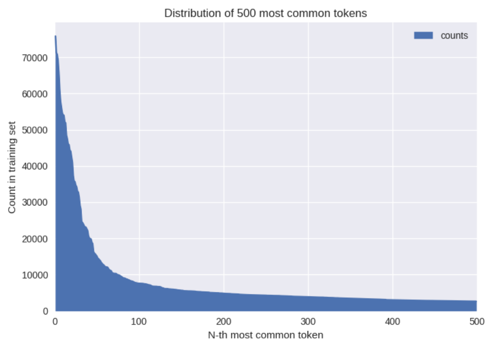
A first approach is to train and use a standard text tokenizer on the musical representations (frames). To that end, we need to define a way to represent frames as text.

A simple way to perform this action is to turn each frame into a string literal. That is, each inactive note is represented as the '`0`' character and each active note is represented by the '`1`' character. A sentence is then a sequence of string representations of frames, separated by a whitespace character (' ').

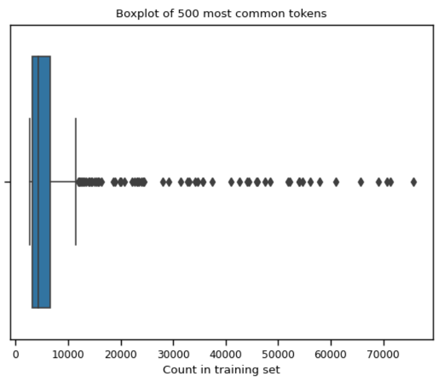
Training a tokenizer on a dataset of such sentences yields words (string representations of frames), subwords (strings representing patterns of active and inactive notes, for example "`10001`" which represents a major third interval) and special tokens such as start and end of sentence, unknown and mask.

A problem we encounter immediately is that the language of piano music, unlike a natural language like English, has a vocabulary too large to fit in memory, specifically 2^{88} possible words or frame states. However, the overwhelming majority of these states are extremely unlikely and will not show up in the dataset.

Figure 3.1 shows the distribution and boxplots for the 500 most common tokens in the training dataset. Note that the most common token, which corresponds to no active notes (musical pause), is not plotted because it is two orders of magnitude more common than the next token. Generally, the first tokens after the musical pause



(a) Distribution of 500 most common training set tokens



(b) Boxplot of 500 most common training set tokens

Figure 3.1: Few tokens make up most of the training dataset. All figures exclude the most common token (musical pause)

are ones that correspond to only one note being played. This means that the dataset is very sparse, with orders of magnitude more negative than positive samples in each frame. The total number of unique tokens in the training set is $885\,768$. For reference, Table 3.1 shows the top 40 most common tokens and their count in the training set.

For our purposes, we pre-train our tokenizer and BERT model using only a subset of the vocabulary consisting of the N-most common tokens. We tried $N \in \{2\,000, 10\,000, 50\,000\}$ but found no difference in the performance of the network.

3.1.3 Introducing Noise to the Dataset

As described in Section 2.2.2, we introduce a new pre-training approach based on music-encoded noise. The implementation of this approach is described in Algorithm 3.1.

3.2 Neural Networks

The huggingface library [31] was used for the implementation of `wav2vec2` and a pre-trained checkpoint for it. It was also used to provide implementations of all BERT and DistilBERT models and tokenizers. We used the ”bert-base-uncased” checkpoint so as to not start training from scratch, as we suspect that natural language has

Table 3.1: Active notes in the most common tokens and token counts

Notes	Count	Notes	Count	Notes	Count	Notes	Count
pause	1616638	(C4)	54002	(D#4)	42551	(E3)	29105
(G4)	75832	(A#4)	53957	(A5)	41016	(G#5)	28082
(C5)	71296	(G5)	52131	(C#4)	37417	(D#3)	24444
(D5)	70794	(E5)	51862	(F#5)	35712	(A#5)	24358
(A4)	69172	(C#5)	48439	(B3)	35624	(G3, G4)	23945
(D4)	65716	(A3)	47464	(F3)	34680	(B5)	23406
(E4)	60953	(D#5)	46045	(A#3)	34218	(A2)	23155
(F4)	57825	(G#4)	45848	(C3)	33032	(F#3)	23074
(F5)	56145	(G3)	44351	(D3)	32731	(C6)	22635
(B4)	54719	(F#4)	44041	(G#3)	31410	(G2)	22169

similarities to music language and the pre-trained checkpoint might contain close-enough representations of those similarities.

3.2.1 Training

Learning rates η for all encoder and decoder models, when trained separately, were constant as shown in Table 3.2, with a reduction if the validation f_1 score plateaued for more than 10 epochs obtained by multiplying the current η with a reduction factor set to 0.1.

For the end-to-end models, after trying various constant learning rates, we instead adopted an adaptive learning rate as in [20], as shown in Figure 3.2, according to the formula:

$$\eta = \texttt{embed\_dim}^{-0.5} \cdot \min(\texttt{step}^{-0.5}, \texttt{step} \cdot \texttt{warmup\_steps}^{-1.5})$$

This approach gave much faster convergence times without sacrificing model performance.

In order to train the final models, two data sampling approaches were considered and tested:

- split each file into non-overlapping sequences of length `sequence_length`, trimming the remainder

Algorithm 3.1 Adding noise to input frames

```
1: for (audio_fname, midi_fname) in performances do
2:     audio ← load(audio_fname, sample_rate)
3:     if len(audio) % (sequence_length * hop_length) > 0 then
4:         audio ← audio[0:len(audio) % (sequence_length * hop_length)]
5:     end if
6:     audio ← audio.reshape(new_shape=(-1, sequence_length * hop_length))
7:     outputs ← parse_midi(midi_fname, len(audio))
8:     outputs ← outputs.reshape(new_shape=(-1, 88))
9:     inputs ← outputs.clone()
10:    for i in range(0, len(outputs)) do
11:        for j in range(0, 88) do
12:            if outputs[j] == 1 and rand() < one_to_zero_probability then
13:                inputs[i][j] ← 0
14:            else if outputs[j] == 0 and rand() < zero_to_one_probability then
15:                inputs[i][j] ← 1
16:            end if
17:        end for
18:    end for
19:    save_tensor_to_file(inputs, inputs_fname)
20:    save_tensor_to_file(outputs, labels_fname)
21: end for
```

- at each epoch, get one sequence of length `sequence_length` at random from each file and use only that

The second approach seems to yield better results as the training and convergence is faster, although the performance of the model did not seem to improve on the first approach.

Weight sharing and parameter freezing

The full Eurydice model is exceptionally large as it contains three `wav2vec2` encoders, three complete BERT transformers, and five additional fully-connected layers. One way to solve this was to freeze the parameters of the `wav2vec2` model encoder as it is already trained, and share it between all three stacks (onset predictor, offset predictor,

Table 3.2: Learning rates for separate training of encoder and decoder models

Model	Submodel	η
Encoder	ConvStack	7.5×10^{-5}
	`wav2vec2`	7.5×10^{-6}
	Linear	1×10^{-5}
Decoder	BERT	5×10^{-5}
	Linear	5×10^{-5}

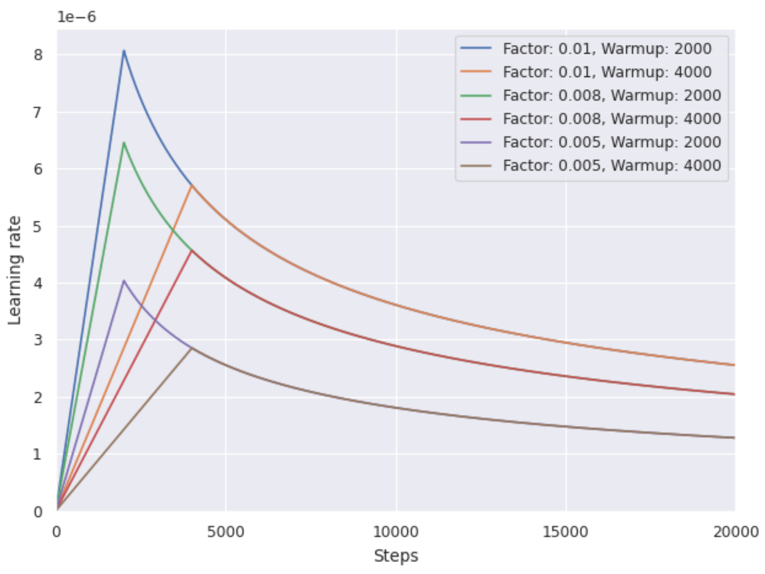


Figure 3.2: Adaptive learning rate using different parameters

note activation estimator). The two BERT transformers in the onset and offset stack can also share weights.

Similarly, for the Orpheus model, in order to avoid having three complete BERT transformers, we simplify our approach by sharing weights between the onset and offset detector stacks, except for the last layer.

3.2.2 Architectures

A multitude of model architectures have been implemented in various phases of this work. In this subsection we describe the most important ones.

Baseline mel-spectrogram model

During preliminary experimentation, a baseline model using mel-spectrogram inputs was established in order to provide a rough comparison between a traditional convolutional stack encoder and the `wav2vec2` encoder.

The architecture of the convolutional stack is presented in Figure 3.3. It consists of three Convolutional layers followed by batch normalization layers, with the last two additionally having max pooling layers. Finally, a fully-connected layer transforms the outputs of the last max pooling layer into the desired model dimension.

The baseline model contains two fully-connected layers as a transcription head.

Baseline `wav2vec2` model

The baseline `wav2vec2` model replaced the entire convolutional stack with a `wav2vec2` encoder. No other changes were made.

Wav2Vec2Bert

The Wav2Vec2Bert model follows a basic encoder-decoder approach; A `wav2vec2` encoder followed by a BERT decoder, with a single layer on top as a transcription head.

Eurydice

For the final Eurydice model, three configurations were implemented: `base`, `large` and `full`.

The `full` model contains no weight sharing, and all submodules are trainable. For the `large` configuration, the `wav2vec2` encoder is shared between modules, and the onset and offset detector subnetworks have a shared `BERT` model. The `base` model is the same as `large` with the modification of having the `wav2vec2` encoder parameters frozen. These configurations are shown in Figures 3.5, 3.4 and 2.3 respectively. The total and trainable model parameters for each configuration are shown in Table 3.3.

3.2.3 Sequences and Sequence Lengths

For all experiments in this work using a BERT decoder, we used a sequence length of 400 or 512 frames, as the BERT decoder can only work with sequences up to

Table 3.3: Different model configurations (m stands for millions)

Model	Total Parameters	Trainable Parameters
Reconstruction of [27]	22m	22m
Wav2Vec2Bert	204m	204m
Eurydice (base)	314m	219m
Eurydice (large)	314m	314m
Eurydice (full)	722m	722m
Orpheus	228m	228m

length 512, which correspond to roughly 8 and 10.24 seconds respectively using the `wav2vec2` encoder, and 12.8 and 16.4 seconds using the ConvStack encoder. For the baseline Onsets and Frames model we used a sequence length of 640 frames which correspond to 20.48 seconds.

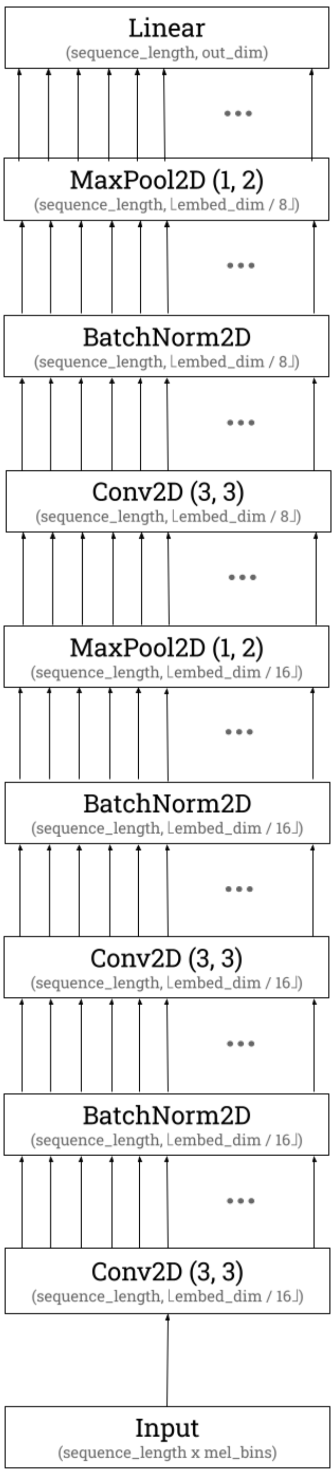

Linear
(sequence_length, out_dim)
MaxPool2D (1, 2)
(sequence_length, ⌊embed_dim / 8⌋)
BatchNorm2D
(sequence_length, ⌊embed_dim / 8⌋)
Conv2D (3, 3)
(sequence_length, ⌊embed_dim / 8⌋)
MaxPool2D (1, 2)
(sequence_length, ⌊embed_dim / 16⌋)
BatchNorm2D
(sequence_length, ⌊embed_dim / 16⌋)
Conv2D (3, 3)
(sequence_length, ⌊embed_dim / 16⌋)
BatchNorm2D
(sequence_length, ⌊embed_dim / 16⌋)
Conv2D (3, 3)
(sequence_length, ⌊embed_dim / 16⌋)
Input
(sequence_length x mel_bins)

Figure 3.3: Convolutional stack used with mel-spectrogram feature inputs

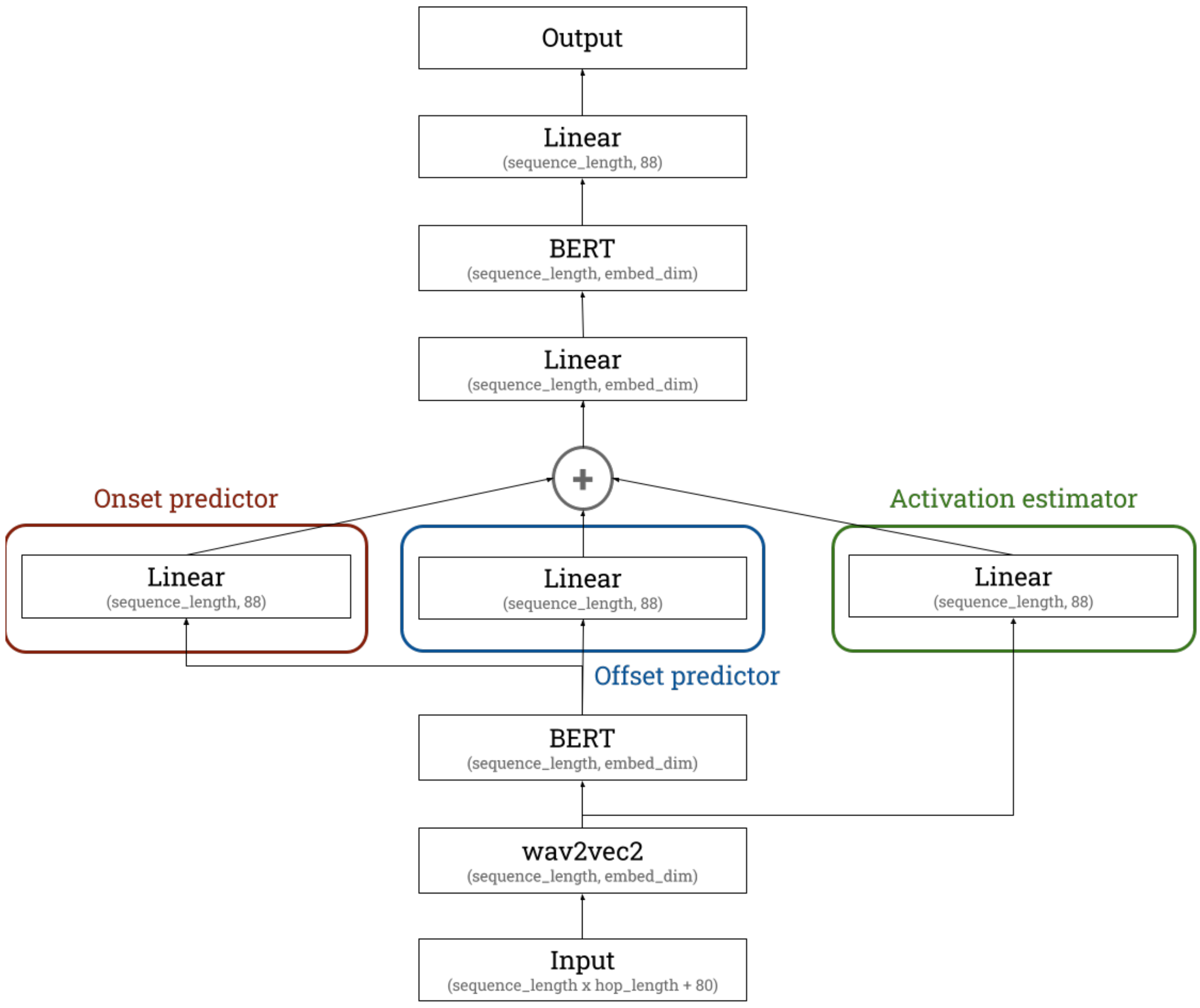


Figure 3.4: Orpheus large model

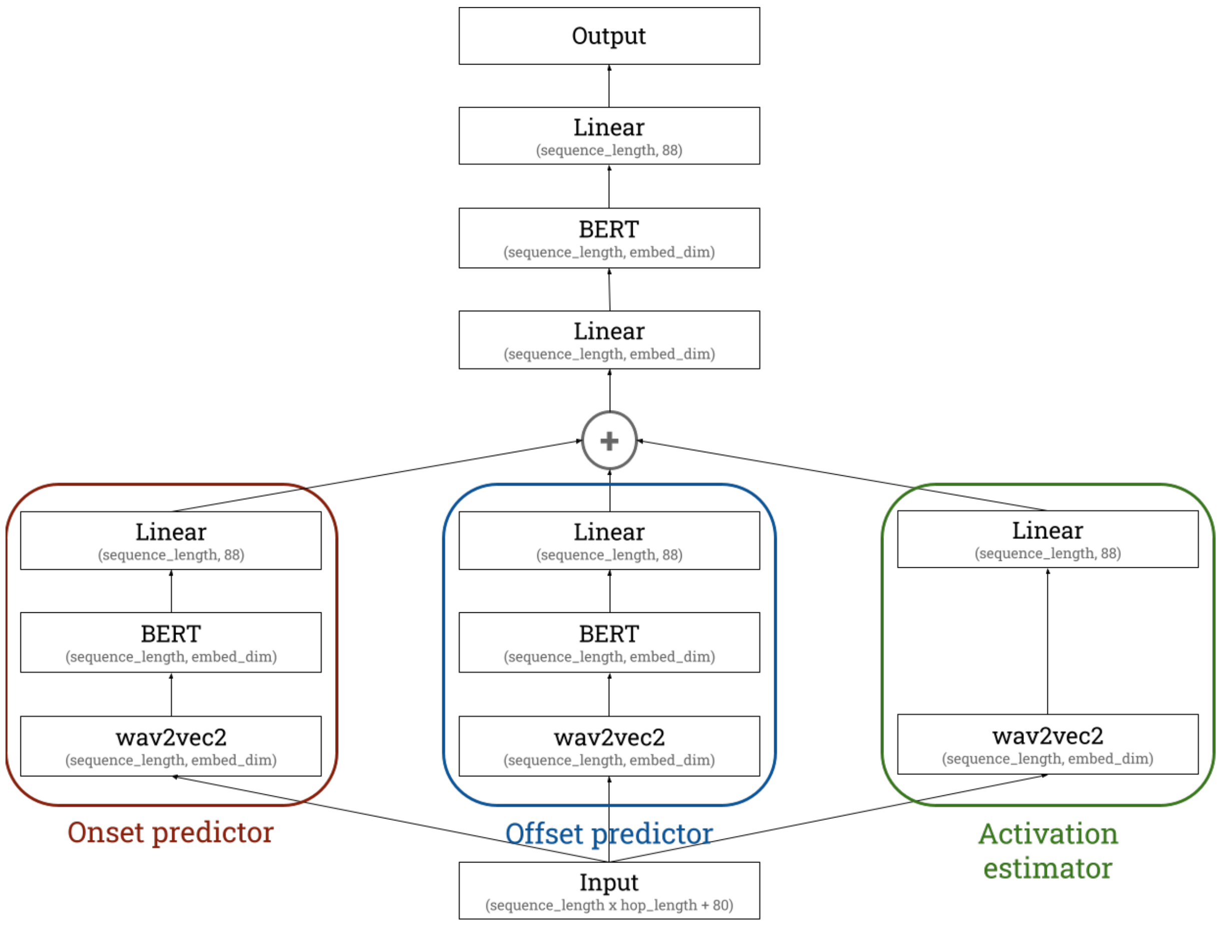

Output
Linear
(sequence_length, 88)
BERT
(sequence_length, embed_dim)
Linear
(sequence_length, embed_dim)
+
Linear
(sequence_length, 88)
BERT
(sequence_length, embed_dim)
wav2vec2
(sequence_length, embed_dim)
Onset predictor
Linear
(sequence_length, 88)
BERT
(sequence_length, embed_dim)
wav2vec2
(sequence_length, embed_dim)
Offset predictor
Linear
(sequence_length, 88)
wav2vec2
(sequence_length, embed_dim)
Activation estimator
Input
(sequence_length x hop_length + 80)

Figure 3.5: Orpheus full model

Chapter 4

Experiments

4.1 Performance Metrics

The metrics used to evaluate model performance are those in [27], first described in [32]. The primary evaluation metric is the Note f_1 score. Below are all the metrics used:

- Note precision
- Note recall
- Note f_1 score
- Note with offsets precision
- Note with offsets recall
- Note with offsets f_1 score
- Frame precision

- Frame recall
- Frame f_1 score

4.1.1 Per Note Metrics

A note is evaluated as correct if its onset is detected within ±50ms and its frequency is within 50 cents of the ground truth. In case of evaluating both onset and offset, on top of the above requirements, a note is required to have an offset value within 20% or ±50ms of the ground truth, whichever is larger.

Precision, recall and f_1 score are calculated using the above measurement of a true positive (TP) note. The `mir_eval` Python library is used to provide the scores.

4.1.2 Per Frame Metrics

We calculate frame precision, recall and f_1 score using the standard binary multilabel classification method, where a true positive is a note that is predicted correctly as active (1) at that time frame, over all notes and all frames.

4.2 Experiment Setup

In order to evaluate our models, we first train a complete Onsets and Frames model, extended to introduce the improvements and modifications introduced in [27], for 640,000 steps in our own setup in order to provide an accurate baseline. The performance of the proposed architectures is then evaluated and compared with the baseline.

The models evaluated are the following:

1. Reconstruction of Onsets and Frames (baseline)
2. Wav2Vec2Bert (100,000 steps)
3. Eurydice base (100,000 steps)
4. Orpheus (200,000 steps)

4.3 Preliminary Testing

4.3.1 Evaluating `wav2vec2` for Music Audio Embeddings

As a first step, a preliminary evaluation of the efficacy of the `wav2vec2` model for AMT had to be done. A checkpoint pre-trained on speech audio was used as a starting point, which was provided by the huggingface library. A feature extractor based on mel-spectrograms was also constructed for evaluation, using a convolutional stack as pictured in Figure 3.3, based on the architecture described in [4].

For both encoder architectures, a decoder consisting of one fully-connected layer was used as a transcription head. The task was to predict the middle frame for each input window of frames (sequence-to-one approach). Approximately 10% of the training files were used for the experiments described in this section.

The `wav2vec2` encoder outperformed the convolutional encoder in the frame f_1 score, without the need for a preprocessing step. Specifically, the former achieved an f_1 score of 50 while the latter achieved a score of 44.

Next, we evaluated the `wav2vec2`-based architecture using a sequence-to-sequence approach. This significantly reduced the amount of training time while improving performance in the f_1 score by 3.

Using an RNN decoder, the performance of both architectures improved, however the convolutional encoder slightly outperformed our approach. This presumably happens because the `wav2vec2`-based architecture is harder to train than the much simpler convolutional stack, and the addition of an RNN, which is notoriously hard to train, complicates it further. Perhaps with more data or time we would see different results.

4.3.2 Evaluating BERT for Music Language Modelling

Other than in the final transcription models, a few empirical experiments were conducted to provide a subjective measure of BERT for MusicLM. We show that BERT trained with a MLM task adequately learns to predict musical sequences, even generating subjectively more musically pleasing and interesting sequences than the ones used as a baseline. See 5.1 for a visualization and explanations on the results.

4.4 System Specifications

The hardware system and libraries used to conduct all experiments reported in this work was the following:

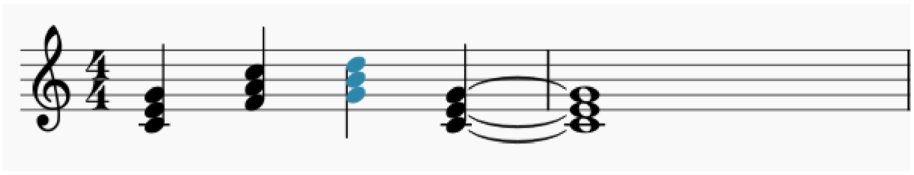
- GPU: 4x NVIDIA GeForce RTX 2080 Ti (1 maximum per experiment)
- CPU: 2x Intel(R) Xeon(R) Bronze 3204 CPU @ 1.90GHz (1 maximum per experiment)
- RAM: 96GB (32 maximum per experiment)
- 72 hours maximum time per experiment
- Python 3.8.5, with libraries:
  - PyTorch 1.8.1
  - Huggingface Transformers 4.5.0
  - MIDO 1.2.9
  - pretty_midi 0.2.8
  - librosa 0.8.0
  - numpy 1.20.2

Chapter 5

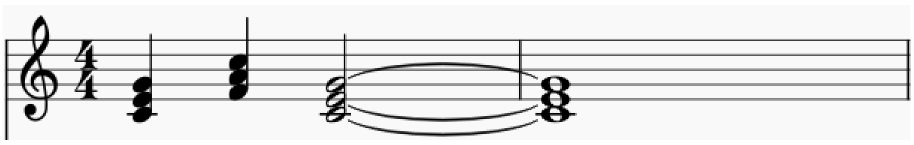
Results

5.1 Music Language Modelling

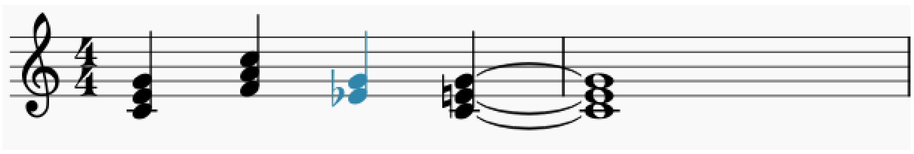
The BERT model trained on the Masked Language Modelling task performed subjectively better than expected in Masked Music Language Modelling (MMLM). Results are presented in Figures 5.1 and 5.2 along with explanations of the underlying harmony as a human musician would attempt to explain it. The model appears to learn musical structure and adequately understand the musical context even in very small sequences as it predicts chords that are justifiable with music harmony.



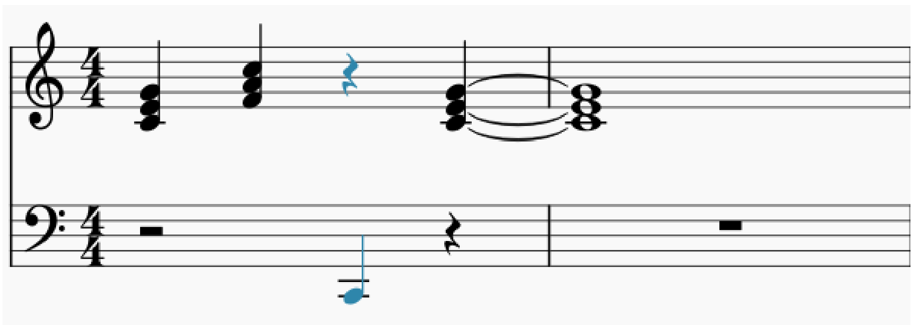
(a) Ground truth (masked notes in blue).



(b) Prediction 1: same as the following chord (imperfect fall). Presumably, this is the top prediction because it puts the I chord in the key of C in a strong position in the measure.

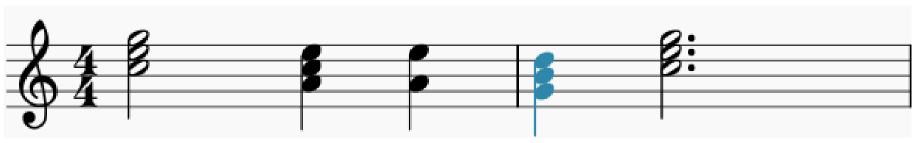


(c) Prediction 2: Eb and G, inferred as a C minor (borrowed) chord in the key of C major, or a suspended dominant chord (V, VII) in the key of E minor that is solved into the VI of the scale.

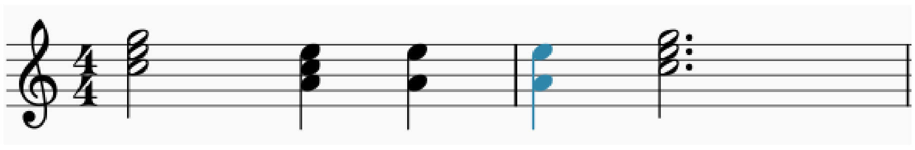


(d) Prediction 3: C (root note in the key of C) in lower register, putting the I chord in a strong position in the measure.

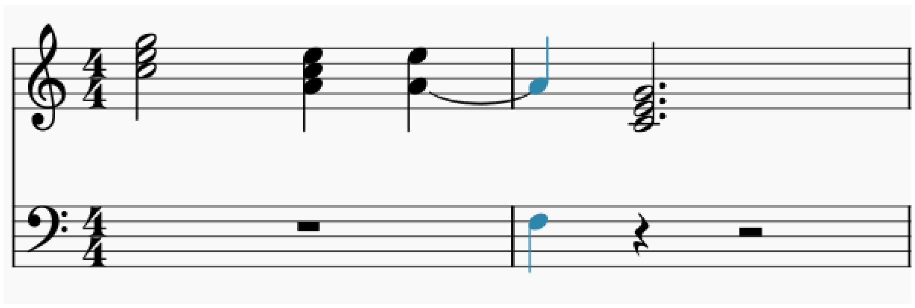
Figure 5.1: MMLM results on masking the V chord of a perfect fall in the key of C. Note that the model has no information on key and can infer different keys and contexts.



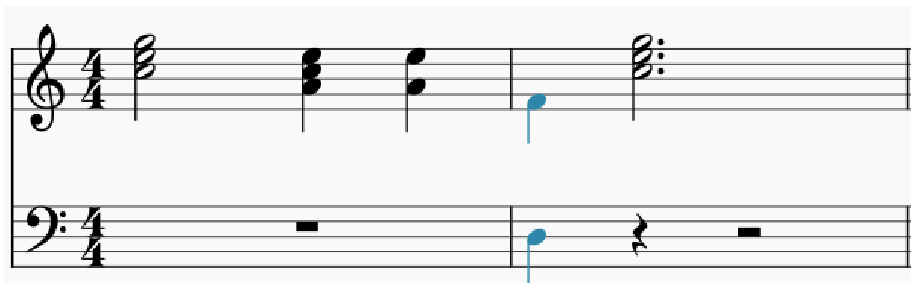
(a) Ground truth (masked notes in blue).



(b) Prediction 1: repeat/continuation of previous chord, the VI of the C major scale with the subtraction of the third. It can also be inferred as the I in the key of A natural minor, as it appears in a strong position of the measure.



(c) Prediction 2: IV or II which are subdominant chords of the C major scale (imperfect fall).



(d) Prediction 3: can be inferred as a II Chord in C major scale (subdominant), or dominant chord (V, VII) of the same key.

Figure 5.2: MMLM results on a musical sequence in the key of C major. Note that the model has no information on key and can infer different keys and contexts.

On the noisy input training task, it produced clean predictions on the test set. Figure 5.3 presents a visualization of the noisy input, prediction and ground truth. In hindsight, this is not surprising as most patterns can be picked out by eye as well.

When initialized with a pre-trained checkpoint from natural language, the BERT model learned and performed better than the randomly initialized model on the task of denoising music input. This might mean that there is at least some transfer learning potential because natural language modelling benefits MusicLM. Figure 5.4 shows the training and validation results, while Table 5.1 shows the test set results of a pretrained versus an uninitialized BERT model. The per-note metrics are very low because of the 50% probability of setting the onset to zero, making it very hard for the model to detect and thus miss it.

Table 5.1: BERT model performance on music denoising task with and without checkpoint start

	Frame metrics			Note metrics		
Checkpoint	Precision	Recall	F1	Precision	Recall	F1
`bert-base-uncased`	$\mathbf{93.32 \pm 2.1}$	$\mathbf{88.83 \pm 3.2}$	$\mathbf{91.01 \pm 2.6}$	$\mathbf{53.83 \pm 3.7}$	$\mathbf{39.84 \pm 5.3}$	$\mathbf{45.58 \pm 4.1}$
None	91.42 ± 2.5	73.80 ± 5.7	81.60 ± 4.4	12.96 ± 2.8	17.86 ± 3.0	14.88 ± 2.5

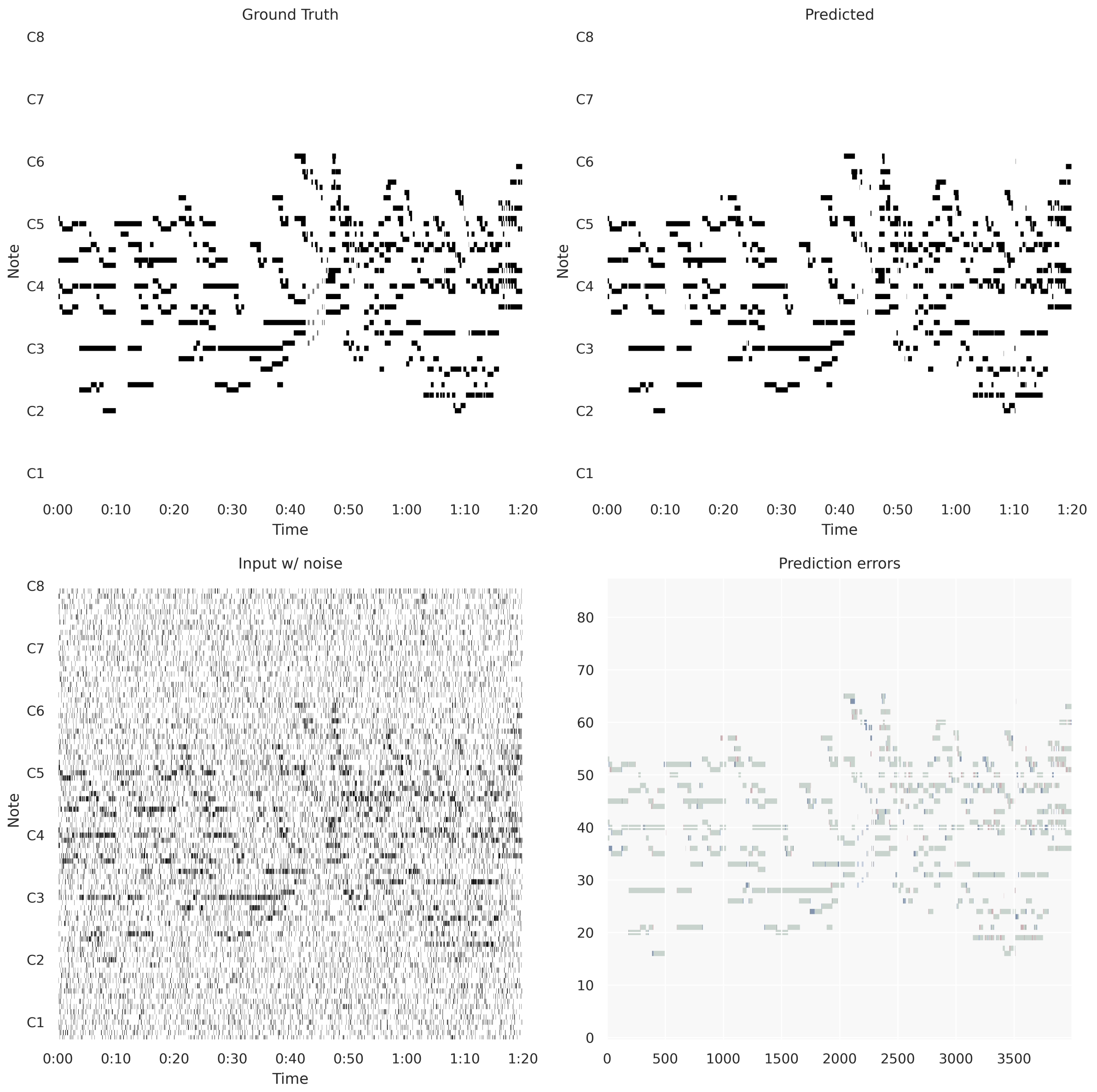
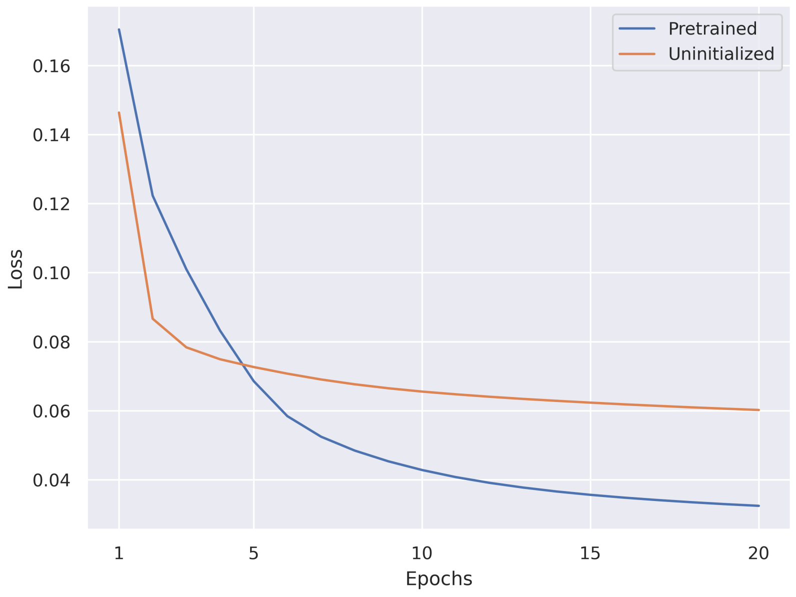
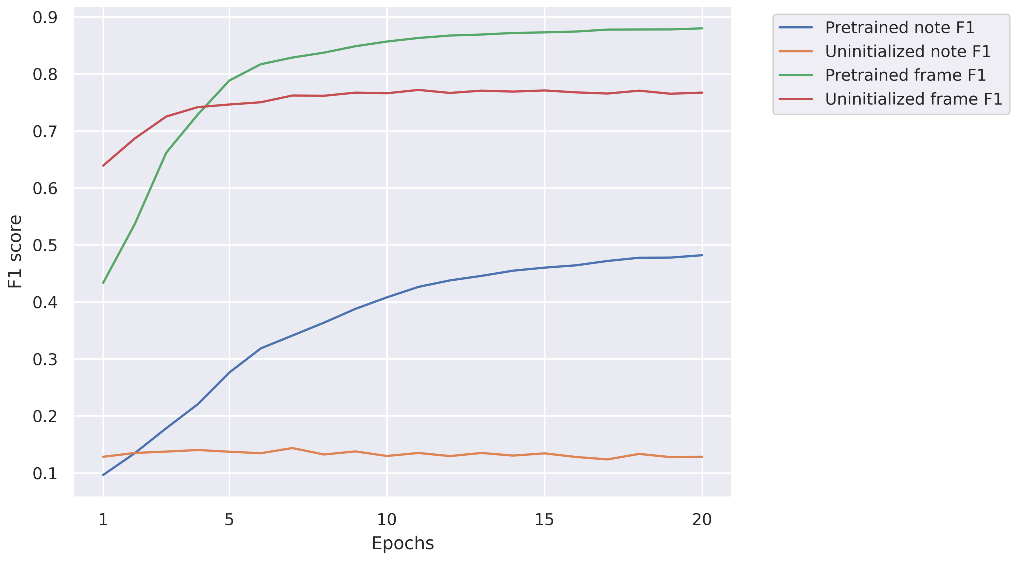


Figure 5.3: BERT pre-training results: using a noisy input, our model learns to correct the errors and output a more realistic transcription (Legend for bottom right picture: Green: TP, Red: FP, Blue: FN)



(a) Training loss



(b) Validation F1 scores

Figure 5.4: Comparison of training a pretrained on natural language versus an uninitialized BERT model for music denoising

5.2 Automatic Music Transcription

We were able to reproduce, within around 1.5 points in the F1 scores, the results reported in [27], and used that model as a baseline.

Table 5.2 shows the experimental results. Figures 5.5, 5.6 and 5.7 show visualizations of predictions and transcription errors of the baseline model, Eurydice (`base`) and Orpheus respectively. We could not train the `large` and `full` versions of Eurydice as the large number of parameters prohibited training in a reasonable amount of time in our system, due to very small batch sizes or in the case of the `full` model, not enough GPU memory.

Our Wav2Vec2Bert model performed poorly compared to the baseline. While it was able to achieve a reasonable Note f_1 score, it was not particularly accurate in predicting frames correctly.

The Eurydice base model improved on all metrics except, somewhat counterintuitively, Note f_1. The addition of an onset detector significantly raised the precision score in all situations, but also lowered the recall somewhat except in the Note with Offset case. Our explanation for this is that the onset detection problem is significantly easier than the frame estimation problem, and the model was too large to effectively learn to adequately solve the latter in the given time and with the given dataset.

Finally, our Orpheus model improved on our previous models in all metrics, but still failed to outperform the baseline model, except in the Note precision metric. However, it came reasonably close to the SOTA model, being around 6 to 10 points lower in the f_1 scores in all situations. It might have the capacity to reach or even exceed the performance of the SOTA model, however we expect it to need either more hyperparameter tuning, time, or data.

Given the above results, we conclude that the `wav2vec2` encoder, pre-trained on speech audio, cannot compete with the mel-spectrogram and convolutional stack encoder of the baseline. This was not unexpected, as the information in speech audio does not contain as many changes in pitch, has different tempo and beat patterns, and speech audio is usually not polyphonic. These differences are problematic as music transcription relies on this information, and `wav2vec2` would have no incentive to capture them when pre-trained with speech audio. We expect that with enough pre-training on music audio, perhaps with some added, appropriately designed objectives, this result might be different.

The BERT-like decoder appears to be on par with the Bi-LSTM networks of the baseline, even though it did not manage to achieve their performance. We expect that it simply needs more training and hyperparameter tuning in order to achieve better results, as it did not manage to converge in the allocated time. Another positive attribute of the BERT-like decoder is the fact that it can be pre-trained separately, whereas the RNNs cannot, and more importantly that it can be used in other MIR and MusicLM problems, due to the transfer learning potential of transformers, while the RNNs have not shown that property.

Table 5.2: Model performance on MAESTRO test set

	Frame			Note			Note w/ offset		
Model	P	R	F1	P	R	F1	P	R	F1
Hawthorne et al.[27]	92.11	**88.41**	**90.15**	98.27	**92.61**	**95.32**	82.95	**78.24**	**80.50**
Reproduction of [27]	**93.65**	84.67	88.89	98.68	89.38	93.77	**84.02**	76.13	79.86
Wav2Vec2Bert	69.94	60.16	64.22	83.26	75.38	78.74	50.14	45.03	47.23
Eurydice (base)	88.52	59.73	70.47	95.58	64.07	75.74	68.52	46.69	54.86
Orpheus	91.18	76.50	82.97	**98.83**	80.87	88.77	77.22	63.21	69.38

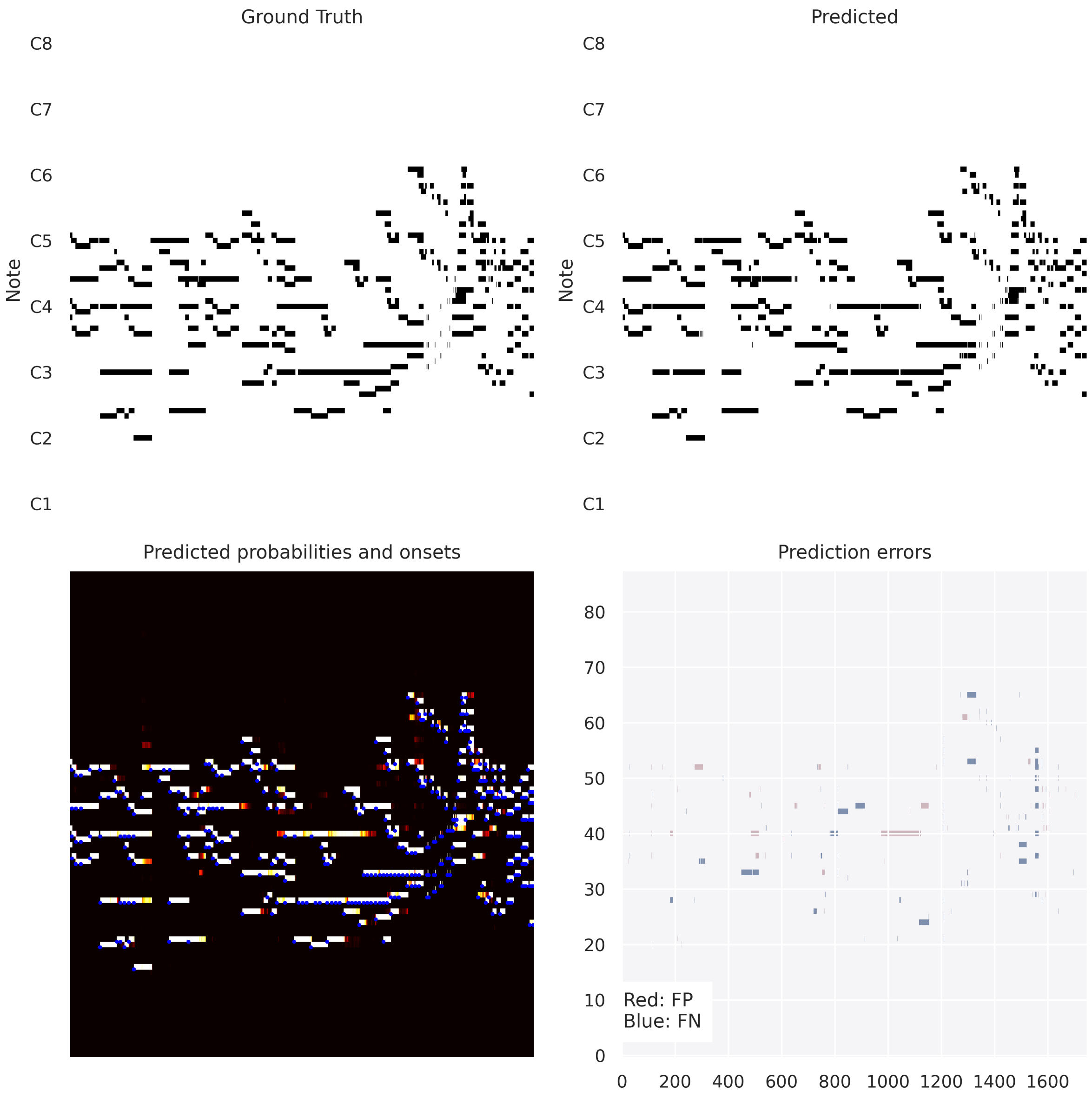


Figure 5.5: Piano roll visualizations of our reconstruction of the Onsets and Frames model predictions on a part from the piece ”Fantasy in F-sharp Minor, Op. 28” by composer Felix Mendelssohn

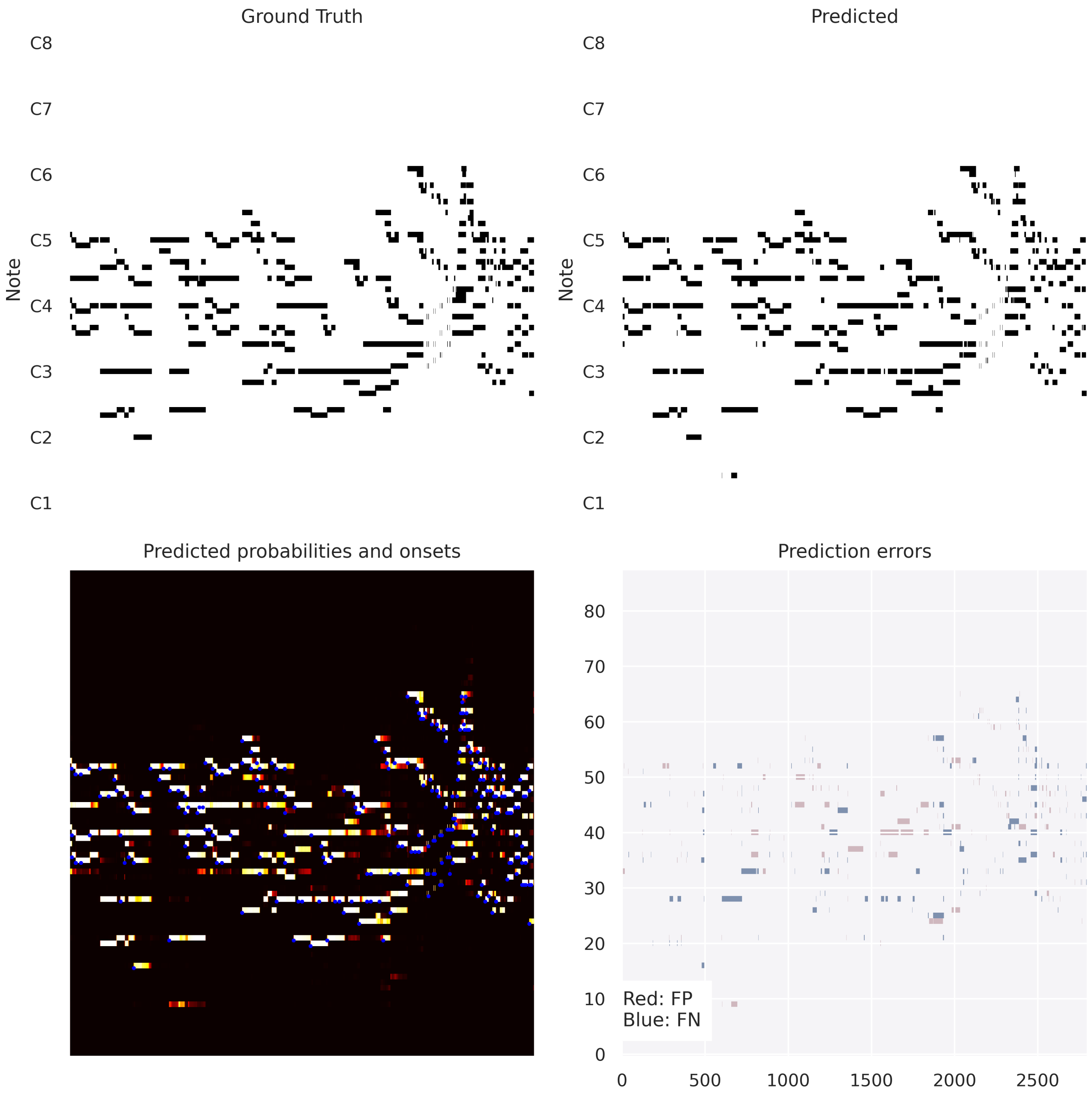


Figure 5.6: Piano roll visualizations of our Eurydice base model predictions on a part from the piece ”Fantasy in F-sharp Minor, Op. 28” by composer Felix Mendelssohn

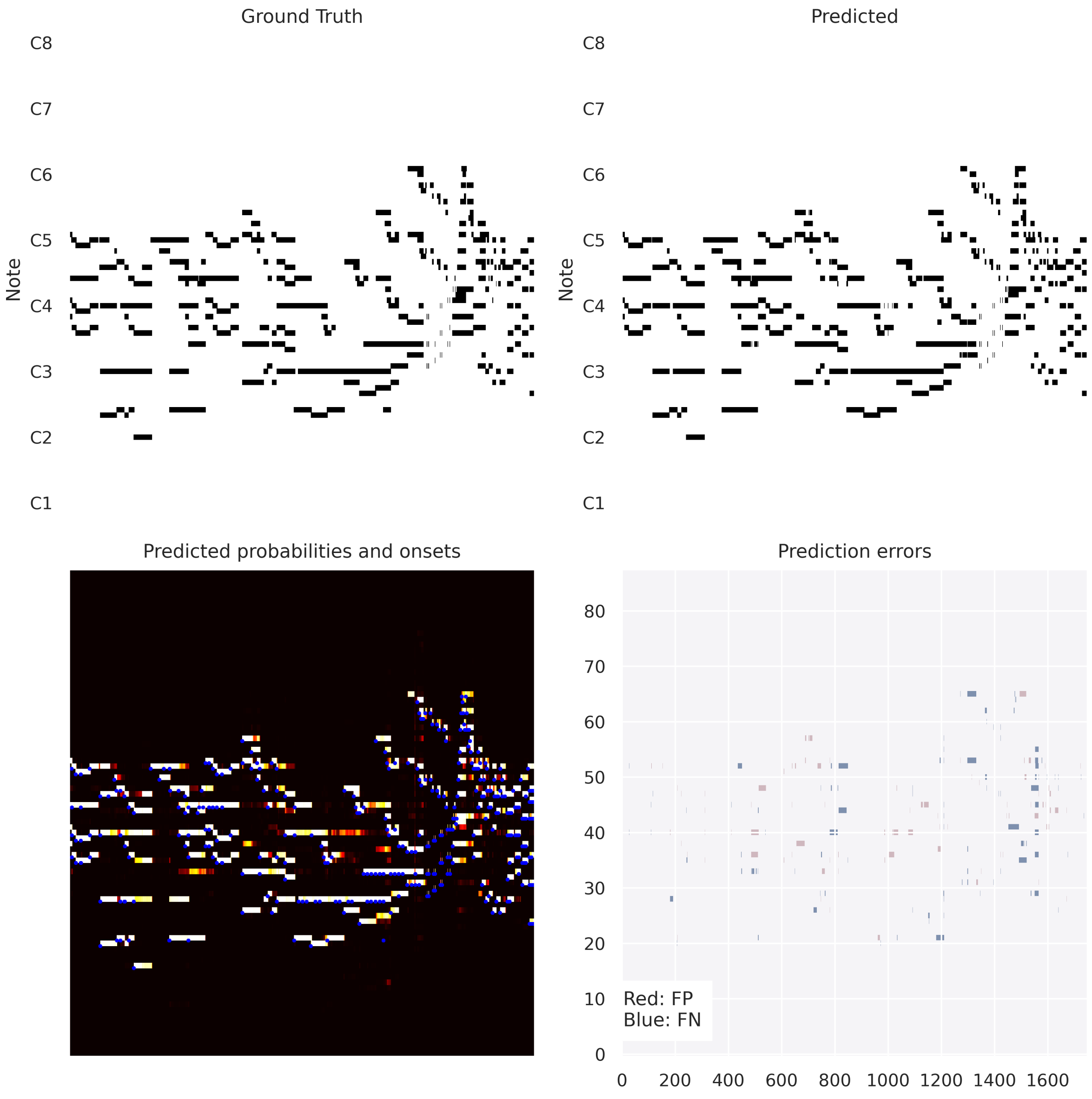


Figure 5.7: Piano roll visualizations of our Orpheus model predictions on a part from the piece "Fantasy in F-sharp Minor, Op. 28" by composer Felix Mendelssohn

Chapter 6

Conclusions and Future Work

In this work, we show that a `wav2vec2` encoder pre-trained on speech audio alone produces embeddings that cannot reach the performance of a convolutional stack encoder using mel-spectrogram features in the task of Automatic Music Transcription.

We also investigate and show that a BERT model is suitable for performing Music Language Modelling, and there is evidence of transfer learning potential between natural and music language.

When replacing BiLSTMs in the state of the art Automatic Music Transcription model with a BERT transformer, we find that the model is able to learn context representations from the data and approach state of the art transcription accuracy.

Finally, we were not able to surpass state of the art performance using transformers. This might be because of our limited resources for running experiments leading to sub-par hyperparameter search and having to simplify all models in order to fit them in GPU memory.

These limitations are not caused by the models we describe, and we are hopeful that future work will overcome them and achieve performance greater than the current state of the art, enabling the transfer learning and explainability benefits of transformers to advance related Music Information Retrieval fields and helping solve various Music Language Modelling problems.

Progress in the field of Automatic Music Transcription would allow the development of musical tools for music editing, synthesis and teaching. Furthermore, advancement of Music Language Modelling would initiate a bloom in automatic music analysis, synthesis, and possibly enhance our own understanding of musical language and structure.

The `wav2vec2` model could be pre-trained on music audio instead of speech, allowing it to learn better representations for music-specific tasks. This would require large amounts of unlabeled raw music audio, which can be gathered using web-scraping techniques, using multiple music datasets, and using data augmentation techniques. A spectral loss could also be added like in the Jukebox model.

Furthermore, the proposed decoder pre-training approach could be augmented by intelligently designing input noise to more closely mimic transcription errors. For example, more of the onsets could be left intact as transcription models usually are able to accurately predict them, but more noise could be added towards the offset side of the note, or when onsets occur on other notes (and so existing notes might get briefly ”buried” in the new frequencies).

Musical sequences can have simultaneously very short (e.g. fast note alternations) and long term (e.g. repeating melodic patterns or choruses) structures. The BERT transformer used in this work can only work with sequences up to length 512, which correspond to roughly 10.24 seconds using the `wav2vec2` encoder and 16.384 seconds using mel-spectrogram inputs. An approach such as the Longformer [33] could model longer sequences, capturing more contextual information.

A novel music notation tokenization method, perhaps based on the representations used in the Music Transformer, would enable pre-training standard NLP models such as BERT without compromises and hacks. Such a method is, in our opinion, the key to unlocking the power of transformer architectures and enabling new approaches for various MusicLM tasks.

Bibliography

[1] E. Benetos, S. Dixon, Z. Duan, and S. Ewert, “Automatic music transcription: An overview,” *IEEE Signal Processing Magazine*, vol. 36, no. 1, pp. 20–30, 2018.

[2] T. M. Association. (2020, 02) The midi association. [Online]. Available: https://www.midi.org/

[3] J. W. Kim and J. P. Bello, “Adversarial learning for improved onsets and frames music transcription,” *arXiv preprint arXiv:1906.08512*, 2019.

[4] C. Hawthorne, E. Elsen, J. Song, A. Roberts, I. Simon, C. Raffel, J. Engel, S. Oore, and D. Eck, “Onsets and frames: Dual-objective piano transcription,” *arXiv preprint arXiv:1710.11153*, 2017.

[5] S. Sigtia, E. Benetos, S. Cherla, T. Weyde, A. Garcez, and S. Dixon, “Rnn-based music language models for improving automatic music transcription,” 2014.

[6] J. Sleep, “Automatic music transcription with convolutional neural networks using intuitive filter shapes,” 2017.

[7] S. Sigtia, E. Benetos, and S. Dixon, “An end-to-end neural network for polyphonic piano music transcription,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 24, no. 5, pp. 927–939, 2016.

[8] J. M. Ender, *Neural Networks for Automatic Polyphonic Piano Music Transcription*. University of Colorado Colorado Springs, 2018.

[9] S. S. Stevens and J. Volkmann, “The relation of pitch to frequency: A revised scale,” *The American Journal of Psychology*, vol. 53, no. 3, pp. 329–353, 1940.

[10] J. C. Brown, “Calculation of a constant q spectral transform,” *The Journal of the Acoustical Society of America*, vol. 89, no. 1, pp. 425–434, 1991.

[11] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[13] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[14] Y. M. Assael, B. Shillingford, S. Whiteson, and N. De Freitas, “Lipnet: End-to-end sentence-level lipreading,” *arXiv preprint arXiv:1611.01599*, 2016.

[15] K. Fukushima and S. Miyake, “Neocognitron: Self-organizing network capable of position-invariant recognition of patterns,” in *Proc. 5th Int. Conf. Pattern Recognition*, vol. 1, 1980, pp. 459–461.

[16] Y. LeCun *et al.*, “Lenet-5, convolutional neural networks,” *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[17] J. Wu, Y. Yu, C. Huang, and K. Yu, “Deep multiple instance learning for image classification and auto-annotation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3460–3469.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[19] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[21] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, “Autoencoder for words,” *Neurocomputing*, vol. 139, pp. 84–96, 2014.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[23] A. Baevski, H. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *arXiv preprint arXiv:2006.11477*, 2020.

[24] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[25] J. Vig, "A multiscale visualization of attention in the transformer model," 2019.

[26] C.-Z. A. Huang, A. Vaswani, J. Uszkoreit, N. Shazeer, I. Simon, C. Hawthorne, A. M. Dai, M. D. Hoffman, M. Dinculescu, and D. Eck, "Music transformer," *arXiv preprint arXiv:1809.04281*, 2018.

[27] C. Hawthorne, A. Stasyuk, A. Roberts, I. Simon, C.-Z. A. Huang, S. Dieleman, E. Elsen, J. Engel, and D. Eck, "Enabling factorized piano music modeling and generation with the maestro dataset," *arXiv preprint arXiv:1810.12247*, 2018.

[28] P. Dhariwal, H. Jun, C. Payne, J. W. Kim, A. Radford, and I. Sutskever, "Jukebox: A generative model for music," *arXiv preprint arXiv:2005.00341*, 2020.

[29] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[30] K. W. Cheuk, Y.-J. Luo, E. Benetos, and D. Herremans, "Revisiting the onsets and frames model with additive attention," *arXiv preprint arXiv:2104.06607*, 2021.

[31] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for

Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[32] J. J. Salamon *et al.*, “Melody extraction from polyphonic music signals,” Ph.D. dissertation, Universitat Pompeu Fabra, 2013.

[33] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The long-document transformer,” *arXiv preprint arXiv:2004.05150*, 2020.

Index

Author's Publications

Zonios, Christos, and Vasileios Tenentes. ”Energy Efficient Speech Command Recognition for Private Smart Home IoT Applications.” *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2021.

Short Biography

Christos Zonios received a B.Sc. degree in Computer Science from the department of Computer Science and Engineering, University of Ioannina, Greece in 2020. He is currently a M.Sc. student in Data and Computer Systems Engineering, specialized in Data Science and Engineering in the University of Ioannina, Greece.

His research interests include Sequence and Language Modelling, Deep Learning, Neuromorphic Computing, Music Information Retrieval, Near-threshold Computing and Intelligent IoT Systems.