Department of Humanities
Institute of Cognitive Science

Master Thesis

Evaluating Jukesbox's Approach to Music Representation on the Basis of Music Classification

A Comparative Study between Neural VQ-VAE-based and Mel-Spectrogram-based Representations of Music

by

Marko Duda
(mduda@uos.de)

April 21, 2022

First Supervisor:
Michael Wittland

Second Supervisor:
Prof. Gordon Pipa

Acknowledgements

I would like to thank Michael Wittland in particular for giving me support and valuable advice in our biweekly meetings.
I would also like to thank Prof. Michael Oehler for fact-checking the discussion of my thesis and for introducing me to the High Performance Computing environment.
Furthermore, I would like to thank Niklas Dettmer in particular for proofreading my thesis and for developing the preprocessing of the spectrograms.
Finally, I would like to thank Paul Liebenow who proofread parts of my thesis.

Abstract

Time-frequency-based representations like mel spectrograms are the dominant audio representation for MIR downstream tasks. This thesis investigates the potential of a new kind of audio representation for music genre recognition (one of the most common downstream MIR tasks). This new audio representation, developed for the state-of-the-art generative music model Jukebox (Dhariwal et al., 2020), discretely encodes music with the help of deep vector quantization. By using an almost identical transformer architecture and the same dataset (Defferrard et al., 2016), the performance of Jukebox's audio representation is compared to mel spectrograms as audio representation. The transformers are pretrained bidirectionally. The findings of this thesis suggest that Jukebox's audio representation is not superior to mel spectrograms (at least when the transformers are pretrained with a rather small dataset of 20k tracks). A possible explanation for this could be that Jukebox's audio representation does not consider the characteristics of human hearing perception enough. Mel spectrograms, on the other hand, are extra designed with the human hearing perception in mind.

Contents

1 Introduction **2**
- 1.1 Current Audio Representation for MIR . . . 2
- 1.2 Deep Vector Quantization - A New Kind of Audio Representation . . . 2
- 1.3 Structure of this Thesis . . . 3

2 Dataset **5**
- 2.1 Content . . . 5
- 2.2 Structure and Split . . . 6
- 2.3 Preprocessing . . . 7
  - 2.3.1 Deep VQ Preprocessing . . . 7
  - 2.3.2 Mel Spectrogram Preprocessing . . . 8

3 Neural Networks **9**
- 3.1 Transformers . . . 9
  - 3.1.1 Encoder-Decoder Architecture . . . 9
  - 3.1.2 Continuous Representation of Symbolic Input . . . 9
  - 3.1.3 Encoder Layer . . . 10
  - 3.1.4 Attention Layer . . . 10
  - 3.1.5 Position-wise Feed-Forward Network . . . 12
  - 3.1.6 Positional Encoding . . . 12
  - 3.1.7 Using Transformers for Downstream Tasks . . . 13
  - 3.1.8 Comparision to RNNs . . . 14
- 3.2 Variational Autoencoders . . . 14
  - 3.2.1 Loss Function . . . 15
  - 3.2.2 Reparametrization Trick . . . 15

4 Related Work **17**
- 4.1 Jukebox . . . 17
  - 4.1.1 Hierarchical Representation of Audio . . . 18
- 4.2 VQ-VAE . . . 19
  - 4.2.1 Inference . . . 19
  - 4.2.2 Loss Function . . . 20
  - 4.2.3 Quality of Reconstruction . . . 21
- 4.3 Wavenet . . . 22
- 4.4 BERT . . . 22
  - 4.4.1 Masked Language Modelling . . . 23
  - 4.4.2 Next Sentence Prediction . . . 24
- 4.5 MusiCoder . . . 24
  - 4.5.1 Overview . . . 24
  - 4.5.2 Pretraining Objectives . . . 26
- 4.6 Time-frequency based Prepossessing Methods . . . 26
  - 4.6.1 Short-time Fourier Transformation (STFT) . . . 26
  - 4.6.2 Mel Spectrogram . . . 28
- 4.7 VQ-VAE Based Approach for Emotion and Theme Recognition . . . 29

5 Methods **30**
5.1 Software and Hardware 30
5.2 Models 31
5.2.1 Pretraining 33
5.3 Experiment and Training Setup 34
5.3.1 Training Hyperparameters 34
5.3.2 Finetuning 36
5.4 Hyperparameters of Mel Spectrogram 36

6 Results **39**
6.1 Detailed Classification Performance Comparison 41
6.2 Pretraining Performance 43

7 Discussion **44**
7.1 Drawbacks of Deep VQ-based Audio Representations 44
7.2 Drawbacks of Fourier-based Audio Representations 45
7.3 Conclusion and Outlook 48

A Software Packages and Class Diagrams **49**

B Scaling Neural Network Training **50**
B.1 Mixed Precision Training 50
B.2 Distributed Training 51

C Environment and Hardware **51**
C.1 SLURM 52
C.2 Hardware 52

Bibliography **54**

List of Figures **57**

List of Tables **60**

1 Introduction

This master thesis evaluates a new kind of audio representation that could be useful for music genre recognition. Using music genre recognition as a performance benchmark, this new audio representation is compared to spectrograms. Spectrograms are currently the established audio representation for music genre recognition (Zhao and Guo, 2021). If this new audio representation outperforms spectrograms at music genre recognition, then it could be concluded that it is useful for other downstream music information retrieval tasks (abbreviated with MIR) as well. Examples of MIR tasks having real-world applications are music annotation and music recommender systems.

1.1 Current Audio Representation for MIR

Music or rather sound is usually stored by encoding the sound's waveform. On CD, music is typically stored as raw audio with a sample rate of 44100 Hz. This means that 30 seconds of music consists of a sequence with over 1 million time steps. Very long input sequences are problematic for machine learning because they imply long-term dependencies. Long-term dependencies are notoriously difficult for machine learning models to handle (Dieleman et al., 2018). To conquer this problem, ML models usually receive spectrograms rather than the sound's waveform as input (Zhao and Guo, 2021). Spectrograms are be explained in Section 4.6 in detail. The basic concept behind spectrograms is to represent the frequencies an audio signal is consisting of rather than the audio signal itself. Spectrograms are significantly shorter than the raw audio signal they are representing.

1.2 Deep Vector Quantization - A New Kind of Audio Representation

There is a new kind of audio representation on the rise: In 2020, Jukebox (Dhariwal et al., 2020), a neural AI model for generating music, was released. Jukebox is the first generative music model that generates convincing results in the raw audio signal domain. Due to the long-term dependencies of raw audio signals, this task can be considered to be inherently difficult. To handle long-term dependencies, Jukebox uses a novel audio representation called deep vector quantization (abbreviated as VQ). Deep VQ is explained in Section 4.2 in detail. Deep VQ consists of a neural network that learns to compress audio to a sequence of so-called codebooks. Codebooks are a fixed-sized set of vectors. Because the set has a fixed size, each vector of this set can be indexed and represented by a token. Thus, deep VQ can represent audio either by a token sequence (1-dimensional token representation) or by codebook vectors sequence (two-dimensional codebook representation). Figure 1 shows a spectrogram and a 2D codebook representation of a rock song. While the spectrogram (Figure 1, left) exhibits a clear structure, the codebook representation (Figure 1, right) shows no clear structure.

However, this does not mean that deep VQ audio representations do not have some structure or pattern that a neural network can identify. Jukebox makes an expressive example that deep VQ is useful for the difficult task of music generation.

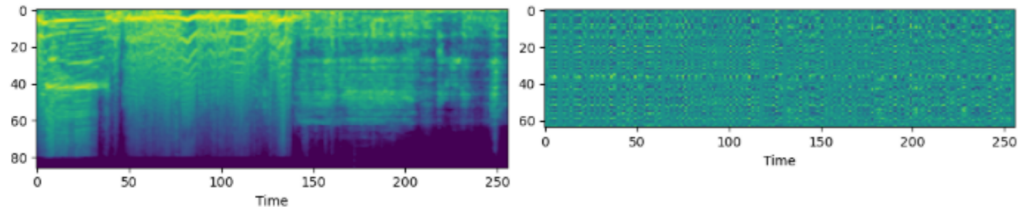


Figure 1: Spectrogram (left) and codebook representation (right) of a rock song . Each column of the codebook representation is a single codebook.

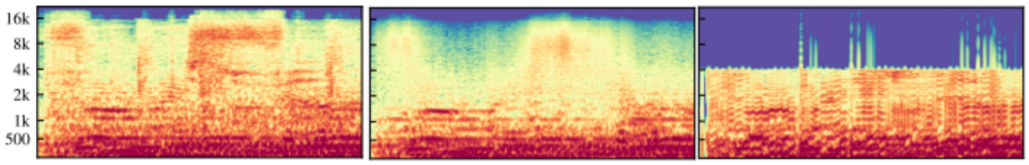


Figure 2: Spectrograms of ground-truth audio signal (left), reconstructed by codebooks (middle) and by Opus codec with bitrate comparable to codebooks (right) (figure taken from Dhariwal et al., 2020, Figure 4 (edited))

Since deep VQ is an audio representation that is even powerful enough for being used for music generation, it could be useful for music information retrieval as well. On the other hand, generating music using spectrograms has not led to convincing results yet (Dieleman). An audio representation that is useful for music generation could be useful for MIR as well.[1] This argument can be made for humans as well: A musician has a deeper understanding of music than somebody who just passively listens to music or as said by the physicist Richard Feynmann: "What I cannot create, I do not understand". This argumentation leads to the research question that this thesis is addressing: **Using music genre recognition as a benchmark task, which audio representation, spectrograms or deep VQ, performs better?** The deep learning architecture called transformer is used for the task of music genre recognition. One transformer is trained with spectrograms, one with deep VQ tokens and one with deep VQ codebooks as input.

Next to rather philosophical motivation for using deep VQ for MIR, there are other, more technical motivations as well: Figure 2 outlines the compression ability of deep VQ tokens compared to the Opus codec. Opus is a modern industry-standard audio codec. Outperforming this standard shows that deep VQ tokens offer a very compressed (or distilled) representation of music. Deep VQ tokens also offer a much more compressed representation of music than spectrograms. Having a very compressed representation of audio might be beneficial for MIR tasks.

1.3 Structure of this Thesis

This thesis is structured in the following way: The 2nd chapter describes the dataset that is used for the music genre recognition experiments as well as the practical/engineering side of the preprocessing. The 3rd chapter describes the neural network architecture transformer used for all experiments of this thesis and the

[1]The definition of MIR usually includes music generation as well. Because it improves the reading flow, MIR is defined to exclude music generation in this thesis.

neural network type VAE which is needed for deep VQ. The 4th chapter describes all works that are related to this thesis. Furthermore, spectrograms are explained in detail in this chapter. The 5th chapter describes the experiment setup developed for answering the research question of this thesis. The name of the deep learning models used for the experiments are Token- and CodebookFormer for the deep VQ based audio representations and SpectroFormer for the spectrograms. The 6th chapter presents the experimental results and chapter 7 tries to find plausible arguments for explaining the experiment results by investigating the conceptual strengths and weaknesses of spectrograms and deep VQ.

2 Dataset

This thesis uses music genre recognition as a benchmark task for evaluating different audio representations. The dataset used for this task is the Free Music Archive (FMA) dataset (Defferrard et al., 2016). This section describes this dataset and the preprocessing.

The Free Music Archive is a free and open library directed by the WFMU. The WFMU is the longest-running freeform radio in the United States. Contrary to normal radio stations, the music choice of freeform radios is controlled by the DJ alone. Genre restrictions and overriding commercial interests do not matter. The FMA dataset was created to boost MIR research on data-heavy ML models such as neural networks. It contains 343 days of Creative Commons-licensed audio. It entails 106,574 tracks from 16,341 artists, arranged in a hierarchical taxonomy of 161 genres. The tracks are available in full-length and high-quality audio. "All tracks are mp3-encoded with a sampling rate of 44,100 Hz, bit rate 320 kbit/s (263 kbit/s on average), and in stereo" (see Defferrard et al., 2016, Section 2.3). The FMA dataset is especially suited for music genre classification. Music genres are subjective by nature. For the FMA dataset, the genres are chosen by the artists themselves.

2.1 Content

The FMA dataset comes in different sizes and with rich metadata. This subsection provides details about the different sizes of the dataset and the different hierarchical levels of the genre taxonomy. Furthermore, it is explained why which size and which level of the genre taxonomy is chosen for the experiments of this thesis.

The FMA dataset comes in 4 different sizes: small (8 000 tracks), medium (25 000 tracks), large (106 574 tracks) and full (106 574 tracks). Except for the full-sized dataset, the other sizes only contain 30-second snippets of a track. When it comes to deep learning, more training data is almost always better. However, the concern of this thesis is to use genre recognition as a performance benchmark not to provide state-of-the-art results for genre recognition. For this task, 30-second snippets should be sufficient. Furthermore, the full-sized dataset needs 879 GB of disk space. Handling such a large dataset requires a lot of additional data engineering effort which would be out of this thesis' scope. The full-sized dataset is therefore not used by this thesis.

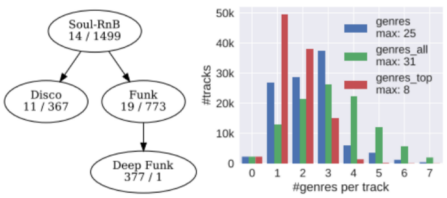


Figure 3: This image is taken from Defferrard et al. (2016). (left) An example of genre hierarchy for the top-level genre Soul-RnB. The left number is the genre id, the right number indicates the number of tracks per genre. Some genres do not occur often: The root-level genre Deep Funk is only represented with a single track within the whole dataset. (right) Number of genres per track.

Figure 3 shows how the hierarchical genre taxonomy is structured. Furthermore, it shows how many tracks there are per genre. Some genres do not occur often. The root-level genre Deep Funk is only represented by a single track for example. Because they cannot generalize well from a single example, using these rare occurring root-level genres as a target is not recommended for deep learning . Therefore, the top-level genre is chosen as the classification target for all experiments of this thesis.

The large-sized dataset provides more than one top-level genre for most tracks, whereas the medium-sized dataset only provides a single genre per track. As already mentioned, the concern of this thesis is to use genre recognition as a performance benchmark for different music representations. Providing state-of-the-art results for genre recognition is not the goal of this thesis. The simpler task of single genre classification is therefore favored over multi-genre classification. Therefore, the medium-sized dataset with the top-level genre as target is used for all experiments of this thesis. Contrary to the small-sized dataset, the medium-sized dataset is not balanced which means that the different music genres do not occur in the same quantity. Figure 4 shows how many tracks there are per genre.

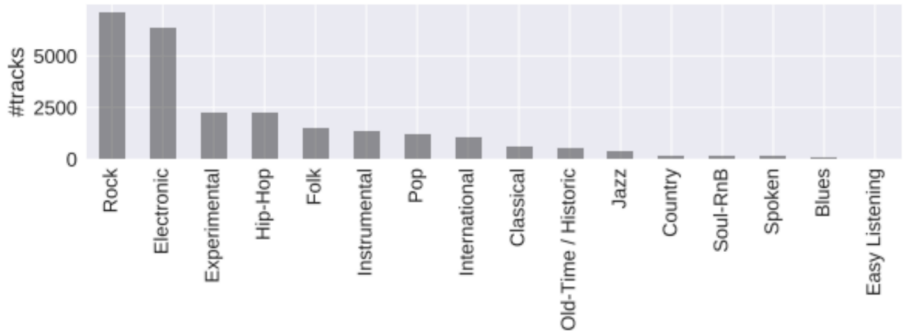


Figure 4: This figure is taken from the FMA dataset paper (Defferrard et al., 2016) and describes how many tracks there are for all 16 root genres on the medium subset (min 21, max 7,103).

2.2 Structure and Split

The dataset is structured into two subfolders. One contains the actual tracks named after their unique track id. The other folder contains several CSV files providing metadata like interpreter, genre, lyrics, etc. about the actual tracks. For music genre classification, the *tracks.csv* file is needed. Besides containing the track id and the top-level genre, the *tracks.csv* file assign each track to either the train, test and validation dataset by an 80%/10%/10% split. Using a predefined split is practical for making the research reproducible. Furthermore, the split satisfy two additional constraints (see Defferrard et al., 2016, Section 2.7):

- It preserves the distribution of tracks per genre which is important for unbalanced datasets.
- Each artist is part of only one set. It has been shown that the use of songs from the same artist in train and test sets leads to over-optimistic accuracy (Flexer, 2007).

Consequently, this thesis adopts the predefined split provided by FMA.

2.3 Preprocessing

Transforming raw audio into spectrograms or deep VQ audio representations requires preprocessing. Because it is technically easy to replace a deep VQ token with its corresponding codebook, both deep VQ audio representations share the same preprocessing procedure. Thus, there is one preprocessing procedure for spectrograms and one for deep VQ. The implementation and data structure of both procedures is described in the following subsections. The codebase of this thesis is made out of two repositories:

1. For deep VQ preprocessing[2]
2. For music classification and spectrogram preprocessing[3]

Because the neural network responsible for deep VQ is taken from Jukebox, the first repository is a fork of the Jukebox repository[4]. The second repository is developed from scratch. Using two separate code repositories avoids conflicting software package dependencies.

2.3.1 Deep VQ Preprocessing

As already mentioned, the neural network responsible for deep VQ is taken from Jukebox (Dhariwal et al., 2020) and is called VQ-VAE (Razavi et al., 2019). How Jukebox and the VQ-VAE work is discussed in Section 4.1 and Section 4.2. This subsection deals with the practical side of using deep VQ for preprocessing. The VQ-VAE is already trained and is only used for inference by this thesis.

The VQ-VAE is embedded into the code basis of Jukebox. It is not documented how to use Jukebox's VQ-VAE for inference. A lot of code exploration was necessary for utilizing the VQ-VAE for inference. The VQ-VAE was trained by Jukebox with a sample rate of 44.1 kHz. Jukebox's VQ-VAE consists of three parts with a compression rate of 8x, 32x, and 128x respectively.[5] Table 1 shows the token sequence length of a 30-second audio snippet compressed by the VQ-VAE. Very long sequence lengths are problematic for ML models. Thus, the highest compression rate of 128 is chosen for preprocessing.

The VQ-VAE compresses the audio into a sequence of codebooks/tokens. Dhariwal et al. (2020) choose a vocabulary size of 2048 for the codebooks/tokens. The sequence of tokens is represented by a list of integers (e.g. [534, 76, 1003, ...]). Each integer corresponds to a token. These tokens can be considered to be like letters of an alphabet which means that token 0 and token 1 do not have more in common than token 0 and token 300. Just like latter a and b do not have more in common compared to letter a and letter k. Which letters are next to each other in the alphabet is set

[2] `https://github.com/ndettmer/mtdml`
[3] `https://github.com/drduda/TokenMIR`
[4] `https://github.com/openai/jukebox/`
[5] The blog post: `https://openai.com/blog/jukebox/#approach` shows audio samples for each compression rate.

compression rate	sequence length
8	165 375
32	41 344
128	10 336

Table 1: Sequence length for 30 second audio snippets with 44.1kHz sample rate compressed by VQ-VAE.

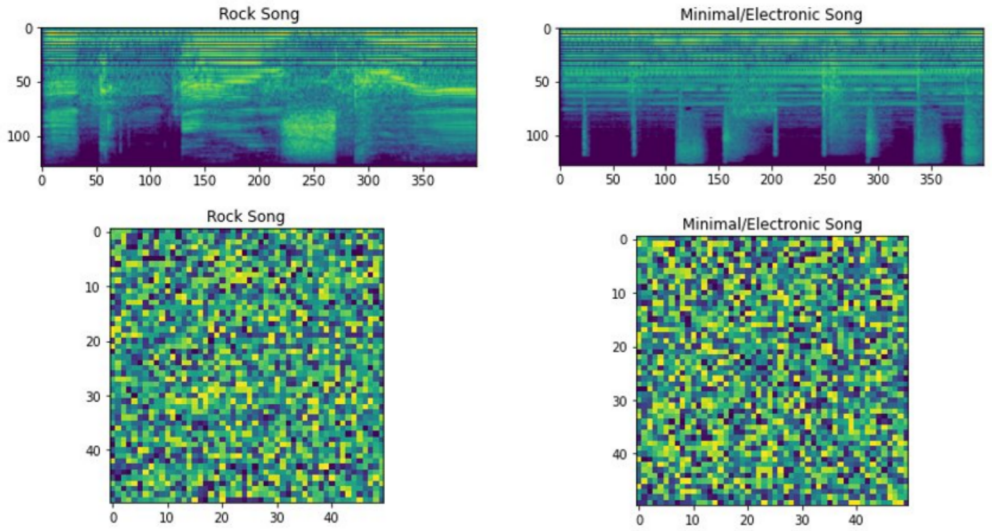


Figure 5: A rock and a electronic/minimal song represented by spectrograms (top) and by a sequence of tokens (bottom). The token sequence is encoded by the VQ-VAE. Each token has its own color. The sequences are listed from left to right and from top to bottom.

arbitrarily. The whole FMA medium-sized dataset is preprocessed by the VQ-VAE and serialized in a two-dimensional NumPy array. Each row of this array corresponds to a song. The medium-sized FMA dataset has a size of 22GB. Compressed into tokens, its size shrinks 400MB. The small size of 400 MB has the advantage that the preprocessed dataset can be directly loaded into the RAM for speeding up training. Dhariwal et al. (2020) states that there are 2048 different symbols. Having a look at the actual tokens confirms that: the smallest token value is 0 and the biggest one is 2047.

2.3.2 Mel Spectrogram Preprocessing

The mel spectrograms are generated with the Python library Librosa. More about mel spectrograms and other time-frequency methods, in general, can be found in Section 4.6. Because the spectrogram dataset is too big to be loaded into the RAM at once. The individual spectrograms are stored on the hard drive so that they do not have to be generated every epoch again. Generating spectrograms requires setting up some hyperparameters. For comparison as direct as possible, the hyperparameters of the spectrogram should be relatable to deep VQ. How to find these relatable hyperparameters is described in Section 5.4.

3 Neural Networks

The following section provides a detailed overview over the deep learning architectures called transformer and variational autoencoder.

3.1 Transformers

Transformer are introduced by the paper "Attention is All You Need" (Vaswani et al., 2017). The transformer architecture is used for all data science experiments of this thesis (see Section 5). Like recurrent neural networks, transformer are originally developed for discrete, sequential input data $X = \{x_i\}_{i=0}^{M}$ like natural language.[6] Unlike recurrent neural networks which have to process the data in its sequential order, transformer can process all elements x of sequence data X at once. For natural language,[7] this means that all words of an input text are processed at once. This parallelization speeds up training and inference time significantly but comes at the cost of having a very high VRAM consumption which only state-of-the-art hardware can handle. To learn the dependency between elements of the sequence data (like the dependency between the words of natural language), transformers have a so-called attention mechanism which is there for learning these dependencies.

3.1.1 Encoder-Decoder Architecture

The standard transformer consists of an encoder (left grey box of Figure 6) and a decoder (right grey box of Figure 6). Using machine translation of natural language as an example task might clear up what the purpose of the encoder-decoder architecture actually is: the encoder encodes an input sentence (e.g. a German sentence) into an internal representation Z. Then, the decoder takes Z and decodes it into the output sentence (e.g. the English translation).

3.1.2 Continuous Representation of Symbolic Input

Neural networks are working with vectors containing continuous numbers. Subsequent, it is described how discrete, symbolic inputs like words of natural language can be used as input for the transformer nonetheless. Each word of an input sentence corresponds to one element x_i of the input sentence X. Each x_i will be projected to its own input embedding vector (see Figure 6). This embedding vector is the continuous representation of the symbolic input. It is initialized randomly, at first, and will be learned during training. Ideally, words that have a similar meaning (like "cute" and "sweet") will have similar embeddings after the training.[8] Onto the embedding, a positional encoding PE telling the transformer the actual position i of x_i will be added up. Details about the positional embedding can be found in

[6] Nowadays, transformers provide state-of-the-art results for a wide range of tasks e.g. image classification (Dosovitskiy et al., 2020)

[7] Although this thesis is not about NLP, discrete tokens will be used as input for several experiments of this thesis. The nature of these discrete token sequences representing music (see Section 4.2) is rather abstract. NLP offers more intuitive access to discrete token sequences. Therefore, natural language will be used as an educational example throughout this section.

[8] This follows the idea of the Distributional Hypothesis. The Distributional Hypothesis states that words that occur in the same context tend to have a similar meaning (Harris, 1954).

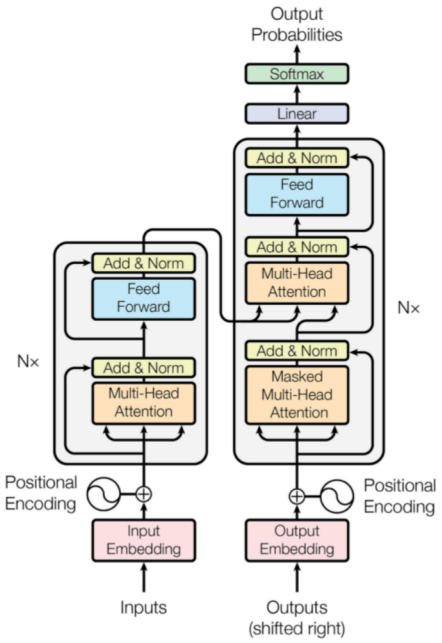


Figure 6: This figure is taken from the paper "Attention is All You Need" (Vaswani et al., 2017) and describes the transformer encoder-decoder-architecture.

Section 3.1.6. The embedding plus the positional encoding gives the final continuous representation $Z^0 = \{z_i\}_{i=0}^{M}$. Z^0 is the input of the transformer encoder and z_i is a vector with dimension d_{model}.

$$z_i^0 = Embedding(x_i) + PE(i) \tag{1}$$

3.1.3 Encoder Layer

Z_0 is fed into the first of N actual transformer encoder layers (see Equation 2). Each of these encoder layers is made up of two sublayers: the multi-head attention layer and a simple feed-forward layer. As visualized in Figure 6, both sublayers are employed around a residual connection (He et al., 2016) followed by layer normalization (Ba et al., 2016). Both the residual connection and the layer normalization are there for preventing the vanishing gradient problem. Neural networks are trained with gradient-based methods. In some cases, the gradient gets vanishingly small and prevents the network from learning effectively.

$$Z^l = EncoderLayer(Z^{l-1}) \qquad l = 1, ..., N \tag{2}$$

3.1.4 Attention Layer

As the name suggests, the multi-head attention layer is the attentive part of the transformer. The multi-head attention layer simply consists of several single-head attention functions whose outputs are concatenated together. To understand multi-head attention, one has to understand single-head attention first. The single-head attention function needs a query q_i, a key k_i and a value v_i vector. Each of them is a learned linear transformation[9] of z_i. The query, key and value are needed for the

[9]Learnt linear transformation is the same as a neural layer without a non-linear activation function.

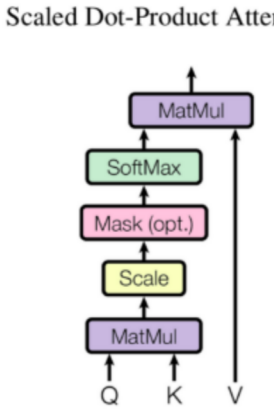


Figure 7: This Figure is also taken from Vaswani et al. (2017) and shows how the attention mechanism works. Since the masking is optional and is not be used in this thesis, it will not be explained any further.

.

so-called attention function. The attention function is computed as follows: the dot product between the query q_i and all keys K normalized by the softmax function is calculated to obtain the weights corresponding to the values V. a_i is then the weighted sum of all values V. In practice, all queries are packed together forming the matrix Q. This enables the transformer to compute all queries simultaneously. This simultaneous computation also enables the transformer to process all elements of a sequence at once. The result is the matrix $A_{unscaled}$ which contains all a_i packed together (see Equation 3).

$$A_{unscaled} = softmax(QK^T)V \tag{3}$$

However, there is a problem with Equation 3. Vaswani et al. (2017) assume that for high dimensional query and key vectors, the dot product grows large in magnitude pushing the softmax function into regions with small gradients. Assuming that the elements of the query and key vector have a mean of 0 and a variance of 1, there dot product qk^T has a mean of zero but a variance of d_k (d_k is the dimension of the query and key vector).[10] This high variance d_k of the dot product is problematic, it can push the softmax into regions with small gradients (see Section 3.2.1 of Vaswani et al. (2017)). In order to counteract this problem, the dot product is divided by $\sqrt{d_k}$, leading to final attention function (see Equation 4). Another way to illustrate Equation 4 is shown in Figure 7.

$$A = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4}$$

As already mentioned, the multi-head attention layer consist of h single attention functions whose outputs are concatenated together. In order to obtain h single attention functions, z_i will be projected to h different query, key and value vectors

[10]Given the basis that the embeddings are initialized from the standard normal distribution and that each sublayer z is normalized by layer normalization, the assumption that the elements of the query and key vector approximately have mean 0 and variance 1 is not "made out of thin air"

(leading to $3h$ learned transformations per encoder layer). The dimension d_k of all these query and key vectors[11] is a fraction of d_{model} as shown in Equation 5).

$$d_k = d_{model}/h \qquad d_{model}\%h = 0 \tag{5}$$

A^{concat} is then obtained by concatenating the matrices from A^1 till A^h together.

3.1.5 Position-wise Feed-Forward Network

The next sublayer of the encoder layer consists of two fully connected feed-forward networks FC which are applied to each position i separately and identically (see Equation 6). Therefore, this fully connected sublayer can also be considered to be a convolution of kernel size 1 (see Vaswani et al., 2017, Section 3.3). The dimension of the in- and output of the feed-forward layer is d_{model}. The inner layer has dimension d_{ff}

$$FFN(a_i^{concat}) = FC(max(0, FC(a_i^{concat}))) \tag{6}$$

3.1.6 Positional Encoding

As already mentioned in Section 3.1.2 Continuous Representation of Symbolic Input, a positional encoding PE is a vector telling the transformer the actual position i of x_i. The positional encoding is added up onto the embedding of x_i (see Equation 1). Since the transformer architecture contains no recurrence and no convolution, the positional encoding is necessary for considering the positional information i of sequence element x_i. Without the positional encoding, the transformer architecture would not consider the order of sequence X at all. There are many choices for positional encodings (see Vaswani et al., 2017, Section 3.5). Vaswani et al. (2017) use sinusoids. Equation 7 shows how the elements (enumerated by dim) of the positional encoding vector are calculated for position i.

The positional encodings should be monotonically increasing. Therefore, i is divided by 10000, so that even large i do not push the sin function beyond the maxima of $sin(\pi/2) = 1$.

$$PE(i, dim) = sin(i/10000^{dim/d_{model}}) \tag{7}$$

The positional encoding of Equation 7 cannot represent any fixed offset k as a linear function. It does not satisfy Equation 8.[12] The attention layer of a transformer layer uses linear transformations to form the query, key and value vectors. Having a positional encoding that is also linear and satisfies Equation 8 would be desirable because then the linear transformations of the attention layer could learn to utilize any fixed offset k.

$$PE(i + k) = PE(i) + T(k) \tag{8}$$

Using functions of sine-cosine pair are known to be linear and satisfy Equation 8. Therefore, the actual final positional encodings are made out of sine cosine-pairs (see

[11] Actually, the value vector can have a different dimension than d_k. For the sake of simplicity, it is assumed that the value vector also has dimension d_k

[12] $PE(i)$ is the vector annotation, whereas $PE(i, dim)$ describes just a single element of the vector.

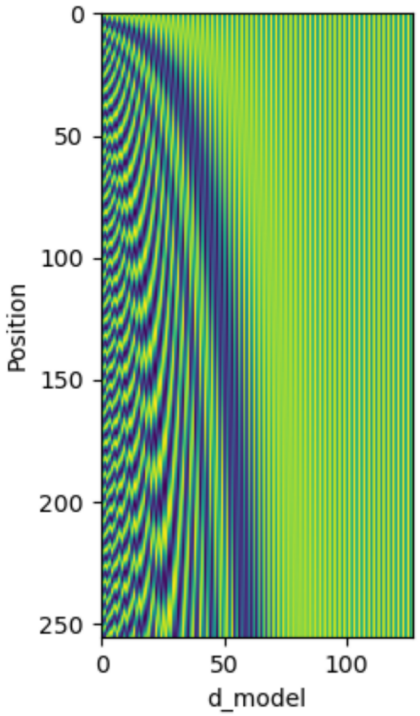


Figure 8: These are the positional encodings. Each row corresponds to a positional encoding vector.

Equations 9). Figure 8 shows a visualization of the final positional encodings.

$$\begin{aligned} PE(i, 2dim) &= sin(i/10000^{2dim/d_{model}}) \\ PE(i, 2dim + 1) &= cos(i/10000^{2dim/d_{model}}) \end{aligned} \tag{9}$$

There is a further favorable property positional encodings do have. Positional encodings of positions close to each other should have a larger dot product than the dot product of positional encodings distant to each other. This way the attention mechanism would favor inputs that are close to the current query q_i. Figure 9 shows the dot product between $PE(i = 256)$ and all other positional encodings. As can be seen, positional encodings close to $PE(i = 256)$ have a higher dot product than distant PE.

3.1.7 Using Transformers for Downstream Tasks

This thesis deals with music genre classification which is a downstream task. Downstream task means that a complicated, high dimensional input space like an audio record is turned into something more simple, low-dimensional like a music genre. For downstream tasks, only the encoder of a transformer is needed. Therefore, the decoder will not be explained any further in this thesis. Subsequent, it is described how the transformer encoder can be utilized for classification or other downstream tasks.

The attention mechanism globally works across the whole sequence. Each internal representation z has access to the whole sequence. Therefore, only the internal

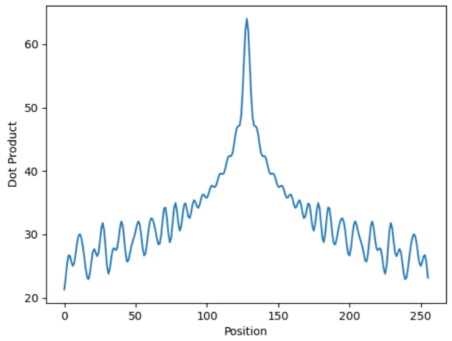


Figure 9: The dot product between $PE(i = 256)$ and all other positional encodings. Close PE have a higher dot product.

representation of the very first[13] position z_0^N is needed for downstream tasks. The other representations of the final layer N are not needed. z_0^N is fed into a so-called classification head. The classification head is usually made up of one or more fully connected neural layers followed by a softmax function.

3.1.8 Comparision to RNNs

Transformers usually give better results than LSTMs for tasks with sequential data. Contrary to LSTMs, transformer can model long-term dependencies as easily as short-term dependencies. Because of the attention mechanism, no matter how distant to elements of a sequence are, the path length which a forward or backward signal has to traverse between both elements is always one: $O(1)$. The attention mechanism gives direct access to all elements of a sequence. On the contrary, LSTMs have a path length of $O(M)$ because they have to process the data in sequence. This immediate access to all sequence elements comes with a cost however. The memory consumption of transformer is $O(M^2)$ with regard to sequence length M. LSTMs have a memory consumption of $O(M)$ with regard to sequence length M (see Vaswani et al., 2017, Table 1). The high memory consumption transformer have is the reason why they can only be trained with state-of-the-art hardware.

3.2 Variational Autoencoders

This section describes the deep learning architecture called variational autoencoder (Kingma and Welling, 2013). Variational autoencoders (abbreviated as VAE) are a subtype of autoencoders. Autoencoders learn to reconstruct the input data. Their input data is also their target data (unsupervised learning). An autoencoder is made up of two parts: an encoder $q(h|x)$ converts the data x into a low dimensional representation[14] h, the decoder $p(x|h)$ learns to reconstruct the input by h. Because of this low dimensional information bottleneck, reconstructing the input data is not a trivial task. What makes autoencoders useful is there ability to compress data (h

[13] The index starts with 0

[14] Low dimensional means here that the dimension of vector h is significantly smaller than the dimension of vector x.

is a compressed version of x). The compressing property of autoencoders is used for preprocessing audio by the deep learning architecture VQ-VAE (see Section 4.2).

VAEs are a probabilistic take on autoencoders. Instead of mapping input x to a latent vector directly, VAEs map the input x to parameters μ and σ of the normal distribution. The latent vector is then sampled from this normal distribution $h \sim \mathcal{N}(\mu, \sigma)$. This variational approach produces a latent space that is more continuous and structured than the latent space of a normal autoencoder (see Rezende et al., 2014, Section 5.2). The latent space is the set of all latent vectors h from the whole training dataset.

3.2.1 Loss Function

The loss function of a VAE is made of two parts. First, there is the reconstruction loss (see Equation 10). The reconstruction loss ensures that the output resembles the input. The reconstruction loss is logarithmized to increase numerical stability.[15]

$$\mathcal{L}_{Reconstruction(x,h)=log\ p(x|h)} \tag{10}$$

Reconstructing x from a low variant distribution is more easy than reconstructing it from a more variant one because the samples h resemble each other more in low variant distributions. This is why the reconstruction loss incentives the encoder $q(h|x)$ to produce normal distributions with very low variance. This behaviour is not wanted. A more variant distributions lead to a more continuous latent space which in turn avoids overfitting.

Thus, the second part of the loss acts as an incentive to add variance to $q(h|x)$. The second loss part is called regularization loss and it ensures that the probability distribution $q(h|x) = \mathcal{N}(\mu, \sigma)$ is close to the standard normal distribution $\mathcal{N}(0, 1)$. The Kullback-Leibler divergence D_{KL} is an information theoretic criterion for measuring the similarity between two distributions[16]. It is used for the regularization loss (see Equation 11).

$$\mathcal{L}_{Regularization}(x, h) = D_{KL}(q(h|x)||\mathcal{N}(0, 1)) = q(h|x) * log(\frac{q(h|x)}{\mathcal{N}(0, 1)} \tag{11}$$

The final loss can be seen in Equation 12. Since the incentives of both parts of the loss are contrary (one wants to add variance while the other wants to reduce it), both together are figuring out a compromise on how much variance to add.

$$\mathcal{L}(x, h) = \mathcal{L}_{Reconstruction}(x, h) - \mathcal{L}_{Regularization}(x, h) \tag{12}$$

3.2.2 Reparametrization Trick

The latent variable h is sampled from the normal distribution $h \sim N(\mu, \sigma)$. Sampling is a stochastic process. Backpropagation, the algorithm for training neural networks, needs a derivative for every weight of the neural network. However, it is not possible

[15] A logarithm turns a product into a sum. A product of probabilities is much smaller then the sum of probabilities. Very small numbers are difficult to represent by computers due to technical limitations. Hence, sums are numerically more stable than products.

[16] However, the KL divergence is not a distance metric because it is not symmetric $D_{KL}(a||b) \neq D_{KL}(b||a)$

to differentiate through a stochastic process. This means that μ and σ cannot be backpropagated. The reparametrization trick is there for solving this issue by factoring out the stochastic process into the standard normal distribution $N(0, 1)$ (see Equation 13).

$$h = \mathcal{N}(q_\theta(\mu|x), q_\theta(\sigma|x))) = q_\theta(\mu|x) + q_\theta(\sigma|x) \cdot \mathcal{N}(0, 1) \tag{13}$$

When the stochastic process is factored out into the standard normal distribution $N(0, 1)$, it is not dependent on the weights θ of the neural encoder q_θ anymore. The stochastic process can be treated as a constant factor and backpropagating μ and σ is possible (see Equation 14 and Kingma and Welling (2013, Section 2.4)).

$$\frac{\delta}{\delta\theta}(q_\theta(\mu|x) + q_\theta(\sigma|x) \cdot \mathcal{N}(0, 1)) = \frac{\delta}{\delta\theta}(q_\theta(\mu|x)) + \mathcal{N}(0, 1) \cdot \frac{\delta}{\delta\theta}(q_\theta(\sigma|x)) \tag{14}$$

4 Related Work

This section shows all research papers that are relevant for the deep learning models that are used for this thesis' experiments. The name of the deep learning models are Token- and CodebookFormer for the deep VQ based audio representations and SpectroFormer for the transformer receiving spectrograms as input. Furthermore, time-frequency audio representations such as mel spectrograms are explained in this section.

4.1 Jukebox

This subsection is about Jukebox (Dhariwal et al., 2020), the first convincing deep learning model generating music in the raw audio domain.[17] Provided with the genre and artist (and lyrics optionally), Jukebox can generate new songs. Provided with the beginning of a record, Jukebox can continue the record. Jukebox is important for this thesis because it uses a new kind of audio representation. The potential of this new audio representation for music genre recognition is investigated by this thesis. This new audio representation was already introduced in this thesis as deep VQ. The biggest challenge when generating audio in the raw audio domain is the extreme long-range dependency between the elements of the audio sequence. Assuming the standard sampling rate of 44 100 Hz, a 30-second long music record has a sequence length of 1.3 million. Even for modern AI, modeling dependencies between millions of samples is not possible. To conquer this issue, Jukebox uses two techniques:

- A hierarchical, three-leveled representation of audio
- Compressing audio with deep VQ

Hierarchical representation simply means that the music is represented by different temporal resolutions first. When a new song is generated by Jukebox, it is generated in low-resolution and then scaled up to higher resolutions. Since the hierarchical representation is not important for this thesis, it will be explained only shortly by Section 4.1.1.

Jukebox compresses music with deep VQ. The neural network responsible for deep VQ is called Vector Quantization-Variational Autoencoder (abbreviated VQ-VAE). The VQ-VAE models the latent space consisting of the lantent vectors h of a VAE into a distribution of so-called codebook vectors. Because the set of all codebook vectors has a fixed size, each of these codebook vectors can be indexed by a symbolic token. With the help of this symbolic representation, VQ-VAEs achieve an even higher compression rate than VAEs (while still retaining the same audio quality). Since VQ-VAEs are essential for this thesis, they will be explained in detail by Section 4.2.

[17] The official post (`https://openai.com/blog/jukebox/`) offers some generated audio samples.

4.1.1 Hierarchical Representation of Audio

This subsection gives an intuition about the overall working and composition of Jukebox. However, the composition of Jukebox is not further relevant for this thesis. Thus, the information in this subsection is not further relevant for this thesis and is only there for providing the "big picture". As already mentioned, the hierarchical representation of audio is there for addressing the long-range dependency issue. Hierarchical representation simply means that the music is compressed to different temporal resolutions by the VQ-VAE (see Figure 10 (a)). Each of the three hierarchical levels has its own VQ-VAE model. The top level is compressed by a factor of 128 while the bottom level has a compression rate of only 8.

(a) Compression

(b) Generation

Figure 10: The overall structure of Jukebox.

For generating a new song, there is a transformer that receives information about the genre, artist (and song lyrics optionally) and outputs the respective top-level

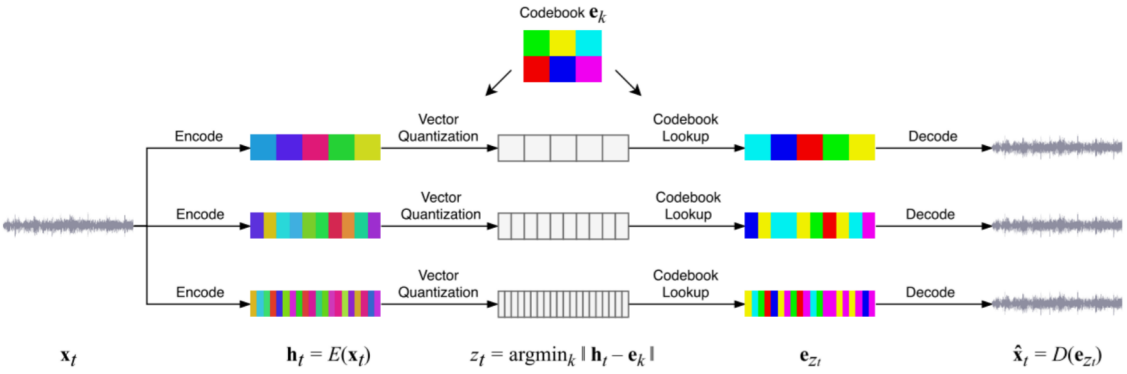


Figure 11: The inference process of the VQ-VAE for all three levels. This is images is taken from the Jukebox (Dhariwal et al., 2020).

encoding. The top-level encoding is then scaled up by another transformer to the middle-level encoding which is then again scaled up to the bottom-level encoding by yet another transformer. All these encodings are made out of token representations of the deep VQ codebook vectors. The bottom level encoding is then decoded to actual audio by the bottom level VQ-VAE (see Figure 10 (b)). Jukebox consists of three independent[18] VQ-VAEs and three independent transformers. It is not an end-to-end trained model. For this thesis, only the compressing part of Jukebox, the VQ-VAE, is relevant.

4.2 VQ-VAE

The neural network responsible for deep VQ is called VQ-VAE. VQ-VAEs (Razavi et al., 2019) are a further development of VAEs. VQ-VAEs model the latent space consisting of the lantent vectors h known from the VAE (see Section 3.2) into a distribution of so-called codebook vectors. One of the codebook vectors is then passed to the decoder. Because the set of codebook vectors has a fixed size, each of these codebook vectors can be represented symbolically by a token. This subsection about the VQ-VAE is structured as follows: first, it is explained how the inference process works. Then, the loss function is explained in detail and finally, the reconstruction quality of the VQ-VAE is evaluated.

4.2.1 Inference

As already mentioned, VQ-VAEs are a further development of VAEs. Like a VAE, they have an encoder that produces a latent vector $h_t = E(x_t)$ with x_t being a snippet from the raw audio. h_t is then compared to the set E of all codebook vectors and the codebook vector e_{z_t} closest to h_t (see Equation 17) is passed to the decoder.

$$z_t = argmin_k |h_t - e_k| \tag{15}$$

The decoder then tries to reconstructs the input $\hat{x}_t = D(e_{z_t})$. Jukebox uses a set of 2048 codebook vectors (see Dhariwal et al., 2020, Section 5.2). The input x_t can therefore be described by one of 2048 tokens. The process of using a codebook

[18] Meaning that no parameters are shared between them.

vector instead of the actual signal (in this case an audio snippet) is called vector quantization (abbreviated VQ). Since Jukebox uses three-leveled hierarchical system for representing a song, it has three independent working VQ-VAEs as can be seen in Figure 11 describing the inference process of the VQ-VAE.

4.2.2 Loss Function

The loss of the VQ-VAE is made out of four terms. The first term, the reconstruction loss (Equation 16) ensures that the reconstruction resembles the input.

$$\mathcal{L}_{Recons} = \frac{1}{T}\sum_{t} ||x_t - D(e_{z_t})||_2^2 \tag{16}$$

Because the VQ-VAE utilizes the nearest neighbor codebook instead of the actual encoding, it is an autoencoder with a discrete bottleneck (see Dhariwal et al., 2020, Section 2.1). Backpropagation is based on derivatives. Thus, a continuous and differentiable function is needed for backpropagation. Therefore, it is not possible to backpropagate through a discrete bottleneck. The VQ-VAE uses a simple trick to address this issue. The latent vector h_t and the codebook e_{z_t} have the same dimensionality and are close together in metric space. Therefore, the gradient of the decoder $\frac{\delta}{\delta e_{z_t}}\mathcal{L}_{Recons}$ is copied and passed to the encoder. By doing this, the encoder can be backpropagated as well. Of course, the gradient $\frac{\delta}{\delta e_{z_t}}\mathcal{L}_{Recons}$ is just an approximation of the actual unknown encoder gradient but experiments using the approximation have shown convincing results (Oord et al., 2017).

The second loss term ensures that the codebook vectors are close to the actual latent vectors h_s (see Equation 17). Thus, h_s acts as a target in this loss term and should not be backpropagated here. The stop gradient operation sg is used for denoting that h_s should not be backpropagated.

$$\mathcal{L}_{Codebook} = \frac{1}{S}\sum_{s} ||sg[h_s] - e_{z_s}||_2^2 \tag{17}$$

The third loss term ensures that the actual latent vectors h_s resemble the codebooks (see Equation 18).

$$\mathcal{L}_{Commit} = \frac{1}{S}\sum_{s} ||h_s - sg[e_{z_s}]||_2^2 \tag{18}$$

Without the loss terms $\mathcal{L}_{Codebook}$ and $\mathcal{L}_{Commit}$, the gradient $\frac{\delta}{\delta e_{z_t}}\mathcal{L}_{Recons}$ would not be a useful approximation of the actual unknown encoder gradient. $\mathcal{L}_{Commit}$ is weighted by $\beta = 0.02$ when adding up the loss terms (see Dhariwal et al., 2020, Appendix B.3. Hyperparameters).

The final loss term is the so-called spectral loss. Without it, the model learns to reconstruct low frequencies only. The spectral loss term encourages the model to match the spectral components of the raw audio (see Dhariwal et al., 2020, Section 3.3).

Contrary to the usual VAE, the KL divergence term is missing in the loss function of the VQ-VAE. Oord et al. (2017) and subsequent papers assume that the prior is a uniform distribution. For a uniform distribution, the KL divergence term is not dependent on the encoder. Thus, is a constant and does not matter for the overall loss function (see Oord et al., 2017, Section 3.2)).

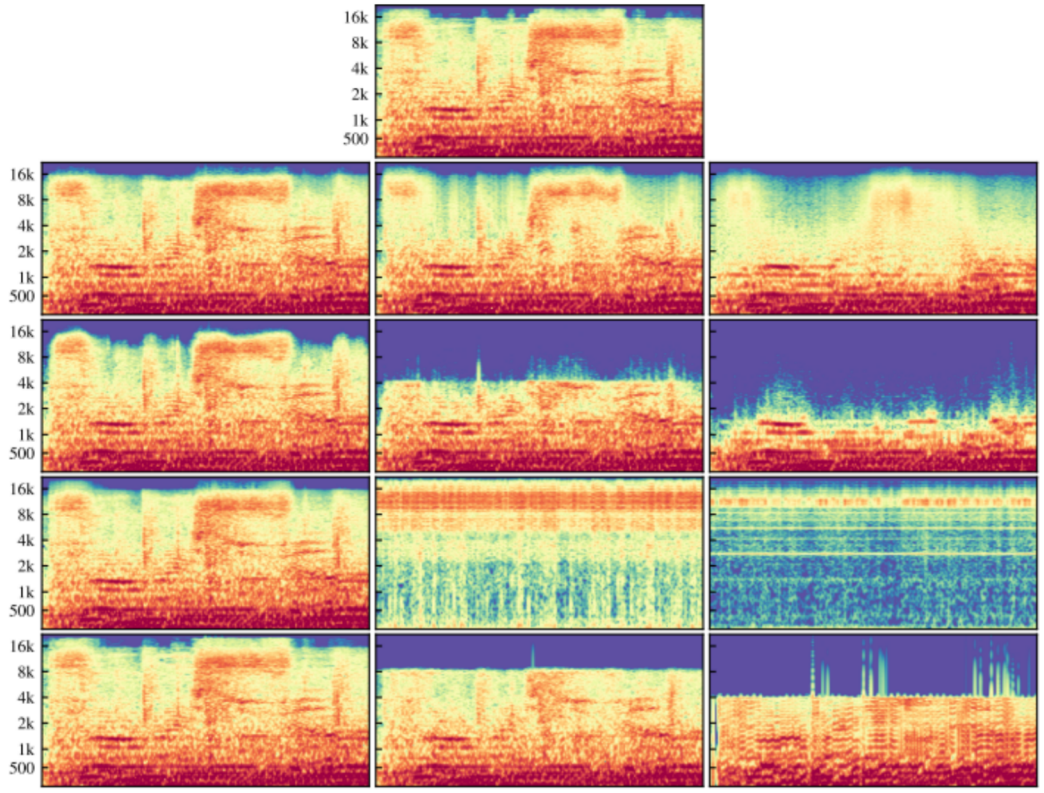
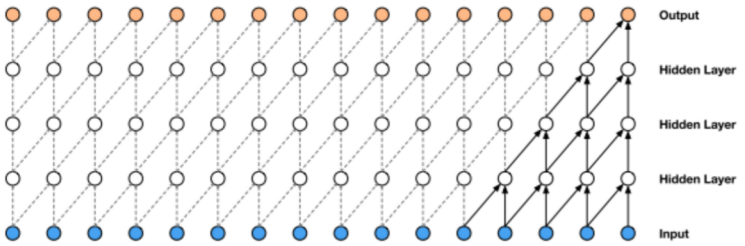


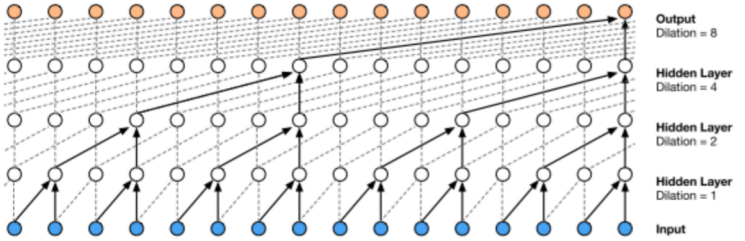
Figure 12: "Comparison of reconstructions from different VQ-VAEs, x-axis is time and y-axis is frequency. The columns from left to right are bottom-, middle-, and top-level reconstructions at hop lengths 8, 32, and 128 respectively, visualized as mel spectrograms. The first row is the ground-truth, and the second row shows the spectrograms of audio outputs from our VQ-VAE. In the third row, we remove the spectral loss, and see that the middle and toplevel lose high-frequency information. In the fourth row, we use a hierarchical VQ-VAE (Razavi et al., 2019) instead of separate auto-encoders [...], and we see the middle and top levels are not used for encoding pertinent information. Finally, the fifth row shows a baseline with the Opus codec that encodes audio at constant bitrates comparable to our VQ-VAE. It also fails to capture higher frequencies and adds noticeable artifacts at the highest level of compression." (image and citation taken from Dhariwal et al., 2020, Figure 4). The Opus codec is a modern and widely used audio format developed 2012.

4.2.3 Quality of Reconstruction

The VQ-VAE's reconstruction quality can be best evaluated by having a look at Figure 12. The caption of Figure 12 explains what each of the respective spectrograms represents. Interestingly, if the Opus codec, a modern and widely used audio format, uses a bitrate that is comparable to the token representation of top-level deep VQ, its reconstruction quality is significantly worse than the respective top-level deep VQ (see Figure 12). The Opus codec is a state of the art industry standard. This comparison illustrates how well the compression of the top-level VQ-VAE works. Having an audio representation with such a great compression ability could be beneficial for music genre recognition.



(a) Non-dilated convolution



(b) Dilated convolution

Figure 13: (a) Normal convolution layers increase their receptive field only linearly with an increasing number of layers (image taken from Oord et al., 2016, Figure 2) (b) The receptive field of dilated convolution kernels increase exponentially with an increasing number of layer (image taken from Oord et al., 2016, Figure 3). That the receptive field goes only to the left side is a particularity of Wavenet. The receptive field of Jukebox's VQ-VAE goes to both sides.

4.3 Wavenet

Wavenet (Oord et al., 2016) is a neural network for generating raw audio signals. It can create relatively realistic-sounding human-like voices (text2speech) and can be used for simple instrumental music generation. Even if Wavenet is not nearly as capable of generating music as Jukebox is. Before Wavenet was introduced, modeling the long-term relationships of raw audio was considered to be not seriously possible (see Dieleman, Section 4.1). Wavenet can therefore be considered to be a ground-breaking work in the realm of raw audio generation. It is also the most cited paper in this area. To address the long-range dependency issue, Oord et al. (2016) use dilated convolutions for increasing the the receptive field of a convolution kernel (see Figure 13 (b)). Normal non-dilated, convolution layers only increase their receptive field size linearly with an increasing number of neural layers (see Figure 13 (a)) whereas the receptive field size of dilated convolution kernels increase exponentially with an increasing number of layers. Inspired by Wavenet, dilated convolution is also used by Jukebox's VQ-VAE for increasing the receptive field.

4.4 BERT

According to the number of citations, BERT (Devlin et al., 2018) is one of the most widely used natural language representation models nowadays. Natural language representation models can be trained on various tasks dealing with natural language like sentiment analysis, spam detection, etc. BERT stands for **B**idirectional **E**ncoder **R**epresentation from **T**ransformers. BERT is relevant for this thesis because the MIR model developed for this thesis is also a Bidirectional Encoder Representation from

Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)
$BERT_{BASE}$	84.4	88.4	86.7	92.7
No NSP	83.9	84.9	86.5	92.6
LTR & No NSP	82.1	84.3	77.5	92.1

Figure 14: Performance comparison of different benchmark finetuning tasks for BERT models pretrained with bidirectional and Next Sentence Prediction (abbreviated NSP), without NSP but bidirectional, and without NSP and left-to-right self supervised training. (figure is taken from Devlin et al., 2018, Table 5).

Transformers. To get a good internal representation of natural language, natural language representation models are usually pretrained. Before training the neural network with the actual task, a pretraining task that is similar to the actual task can be used for training the neural network at first. This can help to boost the performance of the actual task (also called finetuning task), especially when the dataset of the finetuning task is rather small. Language representation models are usually trained self-supervised which means that they have to predict the next word of sentence based on the previous words like predicting the word "is" in the sentence "The car is red" based on the words "The car". This way of self-supervised training is called left-to-right because the left words are used to predict the next word to the right. What makes BERT interesting for this thesis is that it is trained bidirectional. Bidirectional means that both the left and the right words can be considered for predicting a certain word (the word "is" has to be predicted by the input "The car _ red". This bidirectional training leads to a performance boost (see Figure 14).

4.4.1 Masked Language Modelling

Because the output is reconstructed from the masked input, bidirectional self-supervised training can be also called masked language modelling. This subsection is about the details of the masked language modelling training algorithm. The basic concept of the masked language modelling algorithm is simple. Random parts of the input are masked and BERT has to learn to fill these masks. However, there is a problem with this approach as it is stated by Devlin et al. (2018): "Although this allows us to obtain a bidirectional pretrained model, a downside is that we are creating a mismatch between pretraining and fine-tuning, since the [MASK] token does not appear during fine-tuning. To mitigate this, we do not always replace “masked” words with the actual [MASK] token." From the 15% of tokens that were selected, 80% are replaced with [MASK] token, 10% remain the same and the last 10% are replaced with random tokens (see Devlin et al., 2018, Section 3.1)). Devlin et al. (2018) do not even mention in their ablation studies how different benchmark tasks perform without any self-supervised pretraining. Thus, self-supervised pretraining seems to be crucial for good performance. Furthermore, bidirectional self-supervised training performs better than left-to-right (see Figure 14).

4.4.2 Next Sentence Prediction

Masked language modelling focuses on the local context (the adjacent words) of a masked input. To train a model that also considers the global relations of the input, Devlin et al. (2018) use an additional task for pretraining called next sentence prediction (abbreviated NSP). Two natural language sentences, A and B, are put into BERT which then has to predict if B is the actual next sentence that follows A. NSP is useful for understanding the relationship between two sentences which is something that is not directly captured by masked language modelling. As can be seen in Figure 14, NSP is beneficial but not especially crucial for performance. Using NSP additionally during pretraining gives a performance increment of 4%.

4.5 MusiCoder

MusiCoder (Zhao and Guo, 2021) takes the approach of BERT and applies it to music genre recognition. Contrary to BERT, MusiCoder does not use embeddings which are learnt during training (see Section 3.1.2) but features gathered from common audio data preprocessing techniques like mel spectograms, CQT etc. and use these features instead of an embedding. The features of these different preprocessing techniques are concatenated together. In this thesis, a model similar structured to Musicoder called SpectroFormer is trained. SpectroFormer has different hyperparameters than the original MusiCoder and receives only mel spectrograms as input. Mel spectrograms are explained in a Section 4.6.

4.5.1 Overview

A systematic overview of MusiCoder is shown in Figure 15. The features of different preprocessing techniques are concatenated together to time frame t_i (see Figure 16 (a) for the different preprocessing techniques). t_i is projected linearly[19] to a vector of dimensionality d_{model}. d_{model} is the dimensionality of the subsequent transformer which is a usual transformer encoder (as described in Section 3.1.1). The finetuning works the same way as described in Section 3.1.7 "Using Transformers for Downstream Tasks". For the pretraining, frames and channels (corresponding to rows and columns of the input) are masked (see Figure 15). Zhao and Guo (2021) mask several consecutive frames and channels together. As can be seen in Figure 16 (b), pretraining gives a 5% performance increase (measured with ROC-AUC). The ROC curve is a graphical plot illustrating the the ability of a binary classifier as its discrimination threshold is varied. AUC stands for area under curve. The ROC-AUC is therefore the integral of the ROC curve. MusiCoder provides state of the art results for the MTG-Jamendo Dataset audio tagging (Bogdanov et al., 2019) and the GTZAN dataset.

[19]The projection is learnt during training.

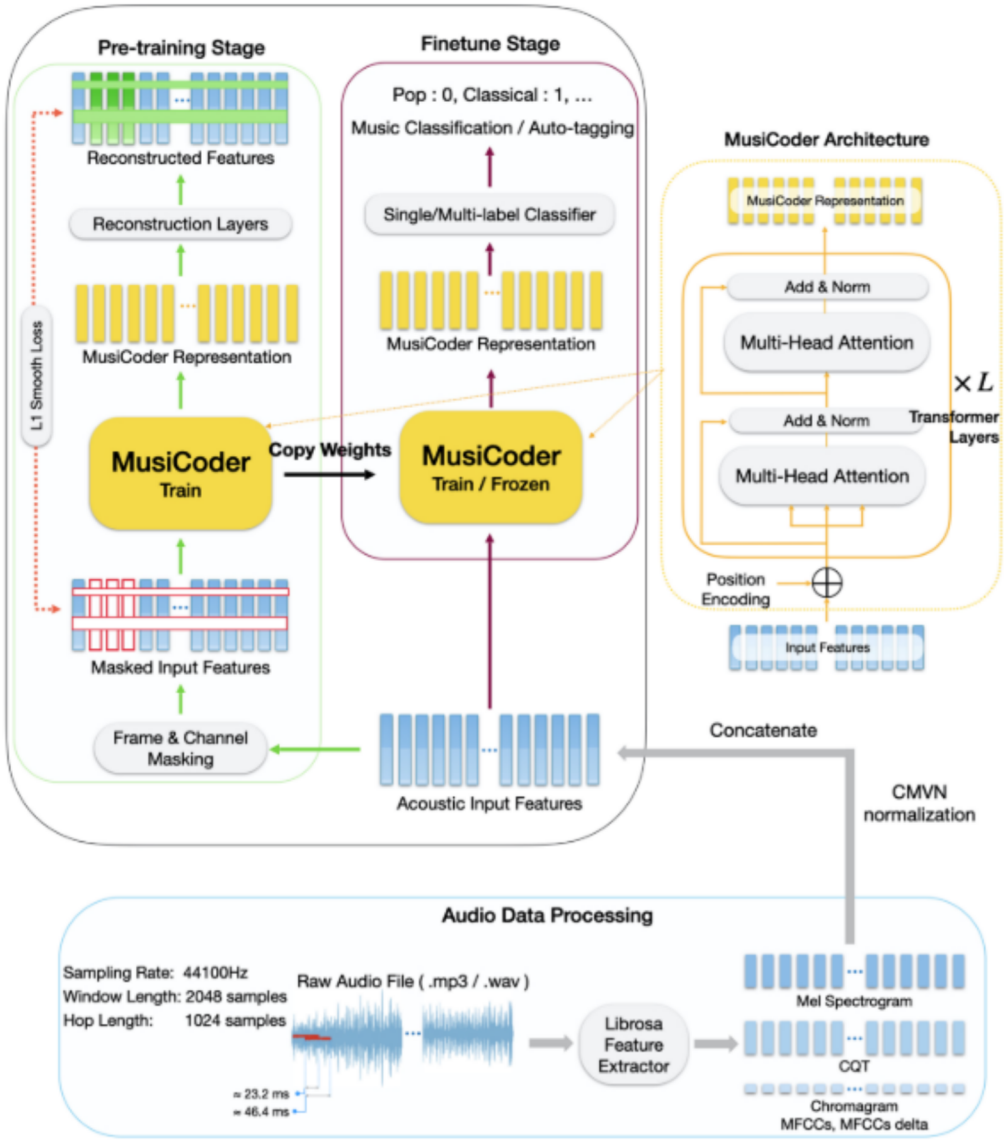


Figure 15: A systematic overview of MusiCoder (image taken from Zhao and Guo, 2021, Figure 1)

Feature	Characteristic	Dimension
Chromagram	Melody, Harmony	12
MFCCs	Pitch	20
MFCCs delta	Pitch	20
Mel-scaled Spectrogram	Raw Waveform	128
Constant-Q Transform	Raw Waveform	144

(a) Acoustic features

Models	ROC-AUC macro	PR-AUC macro
VQ-VAE+CNN [20]	72.07%	10.76%
VGGish [4]	72.58%	10.77%
CRNN [22]	73.80%	11.71%
FA-ResNet [22]	75.75%	14.63%
SampleCNN (reproduced) [28]	76.93%	14.92%
Shake-FA-ResNet [22]	77.17%	14.80%
MusiCoderBase w/o pre-training	77.03%	15.02%
MusiCoderBase with CCM	81.93%	19.49%
MusiCoderBase with CFM	81.38%	19.51%
MusiCoderBase with CFM+CCM	**82.57%**	**20.87%**
MusiCoderLarge with CFM+CCM	**83.82%**	**22.01%**

Figure 16: (a) Acoustic features used by MusiCoder (figure taken from Zhao and Guo, 2021, Table 3). (b) The performance of MusiCoder for the MTG-Jamendo music auto-tagging task dataset measured in ROC-AUC (figure taken from Zhao and Guo, 2021, Table 6). Comparison between no pretraining; channel masking; frame masking; channel and frame masking pretraining.

4.5.2 Pretraining Objectives

As already mentioned, frames and channels (corresponding to rows and columns of the input) are masked for pretraining. The pretraining of SpectroFormer (the spectrogram model trained for this thesis) is inspired by the pretraining objective of MusiCoder.

Frame Masking

15% of frames are masked with varying length of mask. The mask length is sampled from a geometric distribution. The average mask has a length of 3.9 frames (179.6ms of audio). 70% of the mask is replaced with entries having a value of zero. Since each dimension of the input frames is normalized to zero mean. Setting these entries to zero is equivalent to setting them to the mean value. 20% of the mask is replaced with random values while 10% stay the same.

Channel Masking

Only one consecutive block of channels is masked with entries having a value of zero. The length of the mask is chosen from a uniform distribution between zero and the total number of channels. This procedure is performed separately over the log-mel spectrum and the log CQT features.

4.6 Time-frequency based Prepossessing Methods

The SpectroFormer model developed and trained for this thesis receives mel spectrograms as input. Mel spectrograms are time-frequency based audio representations. As the name suggests, time-frequency representations represent an (audio-) signal in the time and the frequency domain. They are a two dimensional representation of an actual one dimensional (audio-)signal. Because mel spectrograms are specifically designed to consider the characteristics of human hearing perception, they are one of the most widely used time-frequency based audio representation for MIR. This section explains the underlying concept behind mel spectrograms. Because mel spectrograms are based upon another time-frequency representation called short-time Fourier transformation, one has to understand how short-time Fourier transformations work before trying to understand mel spectrograms.

4.6.1 Short-time Fourier Transformation (STFT)

The short-time Fourier transformation is a basic time-frequency representation. To understand what a time-frequency representation exactly is, one has to understand what a frequency representation is at first. The idea behind frequency representations is that any (audio-)signal, now matter how complicated it appears, can be decomposed into its frequencies. Each frequency is expressed as a sine wave. The signal can be recreated by summing up these frequencies which the signal is made of. To decompose the signal into its frequencies, the so-called Fourier transformation is used. The Fourier transformation is a mathematical method. Explaining how it exactly works is beyond the scope of this thesis.[20] For this thesis, it is sufficient to know that the Fourier transformation is responsible for decomposing a signal into its frequencies. Furthermore, it is important to know that a frequency representation is

[20]This video provides a good and intuitive explanation of how the Fourier transformation exactly works: `www.youtube.com/watch?v=spUNpyF58BY`

actually made of complex numbers $a + bi$. The real part a represents the magnitude that a certain frequency f has, whereas the imaginary part b represents the phase shift of f (Dieleman). For most cases (including this thesis), the phase is discarded since it is not very informative.[21]

Another important thing to know about the Fourier transformation is that it assumes signals to be stationary. A stationary signal is a signal that does not change over time. However music is not a stationary audio signal. Music consists of melodies, refrain and strophes. It changes over time. An intuitive solution to this problem is to chop the signal into so-called frames (sometimes also called windows) and compute the Fourier transformation on these frames. Actually, this is what the short-time Fourier transformation does. It choppes the signal into frames and computes the Fourier transformation on these frames. This frames have a short-time period (usually something between 10ms and 100ms). For such short-time periods, the signal is assumed to not change dramatically. It is **quasi-stationary** (Velardo, b). Chopping the signal into short time frames, calculating the frequency representation of these frames and stacking the frequency representations together gives a spectrogram (see Figure 17). Because a spectrogram shows how a signal changes over time, it is a time-frequency representation.

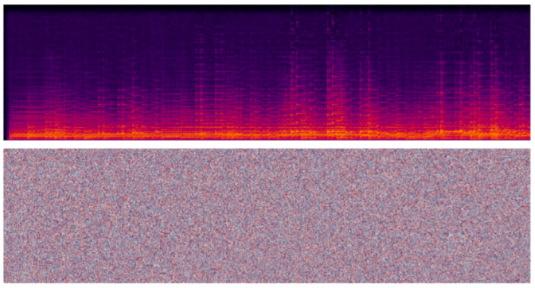


Figure 17: Top: magnitude spectrogram of a piano recording. Bottom: the corresponding phase spectrogram. The y scale represents time and the x scale represents the frequencies

Spectrograms have two important hyperparameters: the frame size n_{fft} specifying how many samples are inside a frame and the hop size. The hop size specifies how many samples the start of a new frame is away from the start of a previous frame. When the hop size is smaller than the frame size, the frames are overlapping. When choosing the frame size, one has to consider the so-called time-frequency trade between high temporal and high frequency resolution:

- A smaller frame size n_{fft} leads to a higher temporal resolution. The higher the temporal resolution, the better the ability to discriminate impulses that are closely spaced in time.
- A bigger frame size n_{fft} leads to a higher frequency resolution. The higher the frequency resolution, the better the ability to discriminate tones that are closely spaced in frequency.

[21] There is an iterative algorithm called Griffin-Lim which can estimate the phase of a spectrogram using the frequencies' magnitude only.

The frequency resolution can be quantified by the number of bins n_{bin} that the frequency representation has (see Equation 19).[22]

$$n_{bin} = n_{fft}/2 + 1 \tag{19}$$

4.6.2 Mel Spectrogram

Humans perceive pitch in an almost logarithmic manner. The perceived difference between a 100Hz and 200Hz tone is bigger than the perceived difference between a 1100Hz and 1200Hz tone. Mel spectrograms take the spectrogram of the short-time Fourier transformation and convert it to a spectrogram that consider this characteristic of human hearing perception. As the name suggests, mel spectrograms represent the pitch in the measurement unit mel. The mel scale is designed such that equal distances on the mel scale have the same "perceptual difference". The mel scale was deduced by empirical perception experiments (Velardo, a). Equation 20 provides the formula for converting hertz to mel. Mel is linked to hertz such that 1000Hz equals 1000mel.

$$m = 2595 * log(1 + \frac{f}{500} \tag{20}$$

To get a mel spectrogram, the following steps are executed:

1. Compute the short-time Fourier transformation spectrogram
2. Convert amplitude to decibels. Decibels are a logarithmic representation of loudness (loudness is logarithmically perceived by humans).
3. Convert the frequency from hertz to mel scale

Step 1 and 2 are also needed for ordinary Fourier transformation spectrograms. Only the third step is unique for mel spectrograms. The third step can be subdivided into three more steps:

1. Choose number of mel bands
2. Construct mel filter banks
3. Apply mel filter banks to spectrogram

The mel bands can be considered to be bins on the mel scale. The number of mel bands is a hyperparameter. Its value is usually chosen to be something between 40 and 128. All mel bands together form the mel bank. To get the mel bank, the following steps are needed:

1. Convert the lowest and highest possible frequency that can be represented to mel
2. Create mel bands. Mel bands are equally spaced points on the mel scale
3. Convert points back to hertz

[22] Actually, n_{bin} is actually equal to the number of samples n_{fft} but the frequency output is symmetric. Thus, only one half of the bins plus one additional bin for the frequency zero need to be considered.

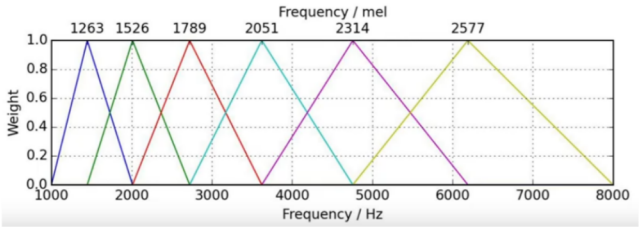


Figure 18: Mel bands are triangular filters

4. Round to nearest frequency bin on spectrogram

5. Create triangular filters

Actually, mel bands are not bins but triangular filter as can be seen in Figure 18. This is how mel spectrograms are calculated. Because music is based on human perception, mel spectrograms are usually the better time-frequency representation for MIR compared to ordinary Fourier transformation spectrograms. (Velardo, a).

4.7 VQ-VAE Based Approach for Emotion and Theme Recognition

There is already a paper called "VQ-VAE Based Approach for Emotion and Theme Recognition" (Hung et al., 2019) that uses a VQ-VAE for preprocessing audio for MIR. However, Hung et al. (2019) use a VQ-VAE for compressing a mel spectrogram. Hung et al. (2019) do not directly compress audio by a VQ-VAE as it is done by this thesis.

5 Methods

The research question of this thesis is to find out whether there is an advantage of using an VQ-VAE for preprocessing audio instead of using conventional audio preprocessing methods like mel spectrograms. This section is about the experimental setup that tries to answer that research question. Furthermore, the deep learning models and details about the code implementations[23] belonging to that experimental setup are presented. The experimental setup is structured as follows: The genre classification performance on the FMA medium dataset is compared between a transformer that receives mel spectrograms as input and another transformer which receives VQ-VAE generated tokens as input and yet another transformer that receives the VQ-VAE generated codebooks as inputs. The transformer using the mel spectrograms is called **SpectroFormer**, the transformer for the VQ-VAE generated tokens is called **TokenFormer** and the transformer for the codebooks is called **CodebookFormer** consequently. This section is structured as follows: first, the software and hardware used for experimenting are introduced. Then, all three models, SpectroFormer, TokenFormer and CodebookFormer, are presented in detail and finally, the experiment and training setup is brought up.

5.1 Software and Hardware

PyTorch Lightning is used for conducting the experiments of this thesis. It is an open-source framework offering a high-level API for PyTorch. PyTorch Lightning is developed for decoupling the science code from boilerplate engineering code. The decoupling has the following advantages:

- Models become hardware agnostic
- Code is clear to read
- Fewer bugs because Lightning handles boilerplate engineering code

In this thesis, a lot of different setups will be trained (pretraining or without pretraining, spectrograms or tokens or codebooks as input). Thus, it makes sense to structure the research code (the code developed for this thesis is the research code) as modular as possible to avoid redundancy. Details about the research code and corresponding class diagrams can be found in Appendix A.

The High-Performance-Computing-Cluster of Universität Osnabrück (see Appendix C.2) is used for training the transformer models. Training transformers require very strong hardware with a lot of VRAM. The NVidia GPUs used by the High-Performance-Computing-Cluster can utilize modern scaling techniques (see Section B) for almost doubling the effective VRAM size compared to using a standard single GPU setting. This enables the transformers to utilize almost 160 GB of VRAM which is sufficient for good training performance.

[23] `https://github.com/drduda/TokenMIR`

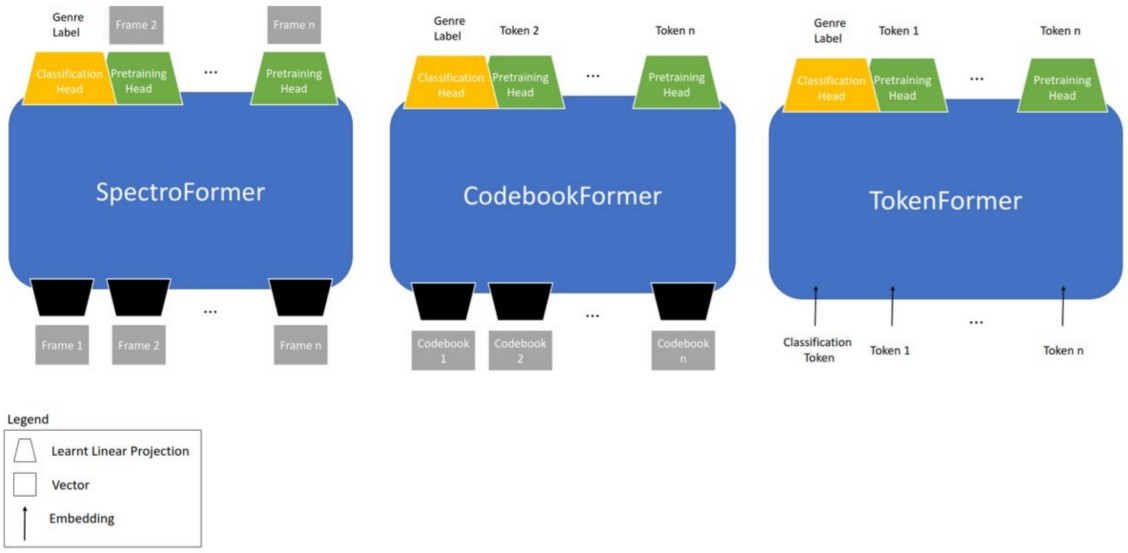


Figure 19: A systematic overview over all three transformer models of this thesis. Although the pretraining happens before the classification, both are shown in this figure as if they are happening simultaneously. The legend explains what the corresponding symbol means. Trapezoids having the same color are actually referring to the same linear projection applied to different inputs.

5.2 Models

All three models, TokenFormer, CodebookFormer and SpectroFormer, are transformer encoders and can be pretrained. The only difference between the models is the kind of input they receive. Like BERT, TokenFormer receives discrete tokens as input (see Section 3.1.2 and 4.4), whereas SpectroFormer receives spectrogram frames as continuous vector inputs like MusiCoder (see Section 4.5). Thus, BERT acts as a "role model" for TokenFormer and MusiCoder acts as a "role model" for SpectroFormer. CodebookFormer's architecture can be considered to be hybrid of both these models: It receives continuous input vectors like SpectroFormer but predicts tokens during its pretraining in the manner of TokenFormer. Figure 19 shows a systematic overview over all three models. The reason why the classification head only receives the first output of the transformer encoder is explained in Section 3.1.7 "Using Transformers for Downstream Tasks".

Token-, Codebook- and SpectroFormer share the same hyperparameters. To find adequate hyperparameters, the hyperparameters of their "role models" BERT and MusiCoder are considered (see Table 2). When it comes to their architectural hyperparameters, BERT and MusiCoder have a lot of hyperparameters in common. One main difference is that BERT has substantially more layers than MusiCoder. Because Spectro-, Codebook- and TokenFormer are receiving preprocessed audio and not natural language as input, they are going to have 4 layers like MusiCoder. All architectural hyperparameters of this thesis models can be seen in Table 3.

Hyperparameter	BERT	MusiCoder
Sequence Length	512	1600
Batch Size	256	64
Sequence Elements/Batch	128.00	102.400
Training Steps	1.000k	200k
Max. Learning Rate	1e-4	4e-4
Warmup Steps	10.000	8.000
Learning Rate Decay	Linear	Exponential
β_1/β_2	0.9/0.999	0.9/0.999
d_{model}	768	768
n_{layers}	12	4
n_{heads}	12	12
$d_{feedforward}$	3072	?
Dropout	0.1	0.1
Sample Length	-	35s
Frame Size	-	2048 (46ms)
Hop Size	-	1024

Table 2: All relevant hyperparameters of MusiCoder and BERT. For both models, the smaller *BASE* size is presented. The question mark indicates that the hyperparameter exists but could not be found in the corresponding paper.

Architecture Hp.	Spectro-, Codebook- and TokenFormer
d_{model}	768
n_{layers}	4
n_{heads}	12
$d_{feedforward}$	3072

Table 3: Architecture hyperparameters of Spectro-, Code- and TokenFormer

5.2.1 Pretraining

Like their "role models", Token-, Code- and SpectroFormer are pretrained in a self-supervised fashion. Listening 1 shows the pseudocode for pretraining Token-, Codebook or TokenFormer. Subsequently, it is described how the pretraining of these models works compared to the pretraining of their "role models".

Listing 1: Pseudocode for the pretraining of TokenFormer.

```
def train_step(x):
    y = copy(x)
    y = stop_gradient(y)

    # selected_elements is a boolean array describing which
    # elements should be predicted
    x, selected_elements = mask_input(x)

    # Forward step
    y_hat = model(x)

    # Loss remains an array
    loss = loss_function(y_hat, y)
    # By multiplying two arrays, loss is reduced to single number
    loss = mean(loss * selected_elements)

    return loss
```

TokenFormer
The pretraining of TokenFormer works exactly like the Masked Language Modelling of BERT (see Section 4.4.1) but with 30% instead of 15% of tokens chosen for prediction. From these 30% of tokens, 80% are replaced with the [MASK] token, 10% remain the same and the last 10% are replaced with random tokens. Because token prediction is a classification task, cross-entropy (see Equation 21) is used as a loss function. y_c is the true probability and $\hat{y_c}$ the predicted probability for class C. $\hat{y}$ and y are vectors that entail the probabilities for all classes.

$$CrossEntropy(\hat{y}, y) = -\sum_{c}^{C} y_c * log(\hat{y_c}) \tag{21}$$

CodebookFormer The pretraining of CodebookFormer works like the pretraining of TokenFormer. Instead of using the [MASK] token, all entries of the corresponding codebooks are masked with zero. Codebooks that are chosen for prediction should be predicted by their corresponding token (see Figure 19).
SpectroFormer
The pretraining of SpectroFormer is a simplified version of MusiCoder's pretraining (see Section 4.5.2). 25% of rows and columns of the mel spectrogram are masked which is approximately 30% of the total area masked.[24] Contrary to Zhao and Guo (2021), row masks do all have the same length. The average length of MusiCoder's

[24]The formula for calculating the total area $\frac{1}{4} + (\frac{1}{4})^2) = 0.3125$

row mask is 3.87 (∼ 180ms), SpectroFormer uses a static row mask length of 42 (∼ 120ms). SpectroFormer uses a higher temporal resolution which is the reason why the duration of masked audio is shorter, although its mask length is longer than the one of MusiCoder. Like Zhao and Guo (2021), 70% of the masked rows are set to a value of zero, and 20% of masked rows are set to random values coming from the standard normal distribution. The remaining 10% of masked rows are unchanged. In the manner of MusiCoder, the channel mask is one big block. Because the input of SpectroFormer is continuous, the pretraining of SpectroFormer is a regression task. The Huber loss is a popular loss function for regression. It is used for the pretraining of SpectroFormer. "The Huber loss (see Equation 22) is a robust L1 loss that is less sensitive to outliers" (see Zhao and Guo, 2021, Section 3.3). y is an entry of the mel spectrogram and $\hat{y}$ is the corresponding predicted value.

$$Huber(\hat{y}, y) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & if\ |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2} & otherwise \end{cases} \tag{22}$$

5.3 Experiment and Training Setup

This section is about the experiment and training setup. In total, there are 6 different setups: SpectroFormer pretrained, SpectroFormer not pretrained, CodebookFormer pretrained, CodebookFormer not pretrained, TokenFormer pretrained and TokenFormer not pretrained. First, the training hyperparameters for the pretraining and ordinary classification are presented. "Ordinary" means that no pretraining before the classification training takes place. Then, the finetuning is explained. At last, the hyperparameters specific to mel spectrograms are described. Whenever possible, all three models share the same hyperparameters.

5.3.1 Training Hyperparameters

The pretraining and ordinary classification training share the same hyperparameters. A lot of training hyperparameters affect the VRAM usage (see Appendix C for details about the available VRAM). The input sequence length is one of these hyperparameters. The VRAM consumption is quadratic with regards to the input sequence length. This makes the input sequence length the hyperparameter that has the biggest effect on VRAM usage. The number of weights a transformer has is not dependent on the input sequence length, however. This is because all elements of the input sequence are using the same weights of the transformer for inference. Multiplying the batch size with the input sequence length gives the value sequence elements/batch (spoken sequence elements per batch). Because all sequence elements are training the same weights of the transformer, the value sequence elements per batch gives a more detailed overview as a training hyperparameter than the batch size alone. The bigger sequence elements per batch the more effective the training is. Next to this, a rather long sequence length means that a greater snippet of the audio is used for genre classification which is beneficial for the performance as well. Thus, the input sequence length is chosen to be as big as possible with regards to VRAM size. The other training hyperparameters are derived from BERT or MusiCoder (see Table 2) and can be seen in Table 4. It is important to notice that compared to MusiCoder; Spectro-, Codebook and TokenFormer are trained with much less data.

Training Hp.	Spectro-, Codebook- and TokenFormer
Sequence Length	1344
Respective Sample Length	3.9s
Batch Size	64
Sequence Elements/Batch	86k
Training Steps	156k
Max Learning Rate	4e-4
Warmup Steps	8000
Learning Rate Decay	Exponential
β_1/β_2	0.9/0.999
Frame Size	512/480/480 (11ms)
Hop Size	128
Mel Bins (Preprocesssing)	86 / - / -
Sample Rate	44.1kHz
Row Mask Length (Pretraining)	42 / - / -
Column Block Mask Length (Pretraining)	21 / - / -
Masked Input (Pretraining)	30%

Table 4: Training hyperparameters concerning the training of Spectro-, Codebook- and TokenFormer.

Musicoder is pretrained with 152k songs, whereas Spectro-, Codebook-, TokenFormer are (pre-)trained with a dataset of roughly 20k songs.

Transformers are usually trained with a learning rate schedule. Learning rate schedules change the learning rate dependent on the current training step. All three models of this thesis use the same learning rate schedule as Zhao and Guo (2021) and Vaswani et al. (2017) shown in Equation 23 and Figure 20.

$$lrate = d_{model}^{-0.5} * min(step^{-0.5}, step * warmupsteps^{-1.5}) \tag{23}$$

The distribution of tracks per genre of the FMA medium dataset is imbalanced (see Figure 4). For example, there are a lot more rock than jazz tracks in the FMA medium dataset. In the worst case, this would mean that the models would classify every track as rock music simply because rock music is the most occurring class. To

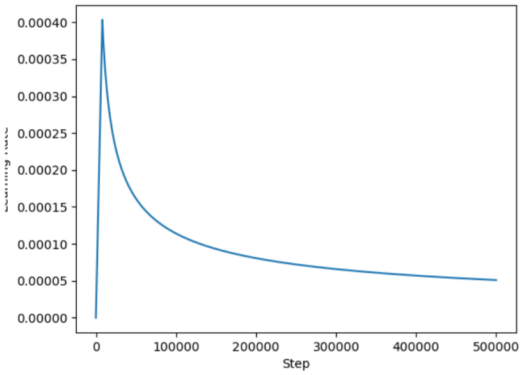


Figure 20: The learning rate schedule used for Token-, Codebook- and SpectroFormer.

Finetuning Hp.	Candidate Value
Batch Size	16, 32
Static Learning Rate	2e-5, 5e-5
Epochs	40

Table 5: For the finetuning, an exhaustive search over the set of these hyperparameters is performed.

prevent this, class weighting is used. When the loss is calculated, class weighting weights minor classes heavier than major classes. By doing this, class weighting establishes a balanced loss function in which each class has a relatively equal share of the overall loss.

5.3.2 Finetuning

The pretrained models are used for finetuning. A static, non-changing learning rate is used during finetuning. An exhaustive search on the sets of parameters of Table 5 is performed. The configuration that performed the best on the validation set is selected. All the other training hyperparameters remain the same as before.

5.4 Hyperparameters of Mel Spectrogram

SpectroFormer receives mel spectrograms as input. Mel spectrograms have hyperparameters on their own. To compare the performance of all three -Former models, the hyperparameters of the mel spectrograms should be relatable to the tokens and codebooks generated by the VQ-VAE. Since the VQ-VAE is already pretrained, its hop size and frame size are fixed. The hop size of the VQ-VAE (also called compression rate) is 128, its frame size is 480 (see Dhariwal et al., 2020, B.1. Music VQ-VAE). Consequently, the spectrograms have a hop size of 128 as well.

For spectrograms, it is advised to take a frame size which is a power of two. Thus, the frame size of the mel spectrogram is 512 and not 480 (see Table 4). The audio analysis Python library Librosa recommends a frame size of 93ms for music information retrieval and 23ms for speech recognition (lib). The frame size of the VQ-VAE and SpectroFormer correspond to only 11ms of audio. This considerably lower frame size might be beneficial for generating music but could be a disadvantage for MIR. Therefore, it could be unfair to compare the music genre classification performance of TokenFormer to an established model like MusiCoder which uses a frame size of 46ms. This is why only a comparison between SpectroFormer and the other models of this thesis makes sense.

The VQ-VAE weights the sound in the middle of a frame more heavily than the sound on the edges of the frame. The reason for this is that Jukebox's VQ-VAE uses residual connections (He et al., 2016) which means that a layer gets as input the output of multiple previous layers instead of the output from the last layer only. The residual architecture of the VQ-VAE can be "represented using directed acyclic computation graphs, where the set of nodes L represents the layers and the set of edges E encodes the connections between the layers" (see Araujo et al., 2019, Section: Arbitrary computation graphs). Several unique paths are going through this graph.

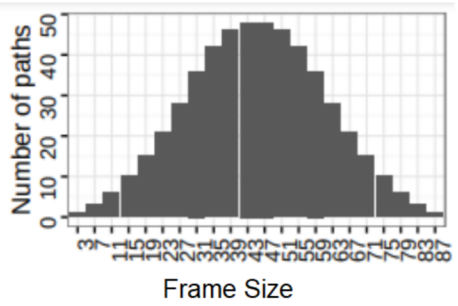


Figure 21: This image (taken from Li et al., 2017, Figure 2) shows the frame size of a neural network made out of residual blocks only. This means each layer is connected to all previous layers. n residual blocks have a collection of 2^n unique paths. The figure shows the number of unique paths per frame size for this network.

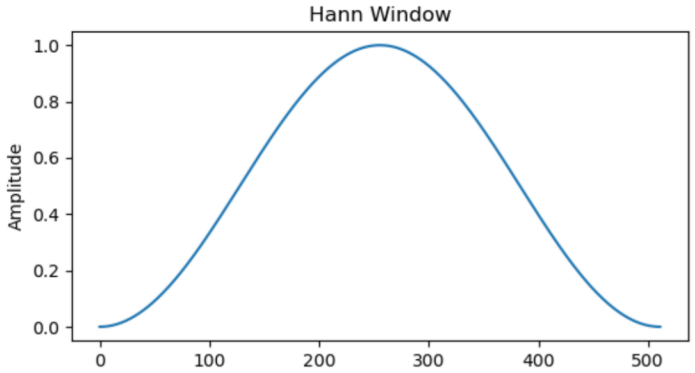


Figure 22: The Hann function is used for smoothing the frames of the Mel Spectrograms.

To illustrate what this means for the frame size, a CNN which consists of residual blocks only is considered as an example. Figure 21 shows the number of unique paths per frame size for such a network. The more unique paths share the same frame size the more the network favors this frame size. The frame size (also called receptive field) of a residual network cannot be stated by a single number but by a distribution. The so-called **effective frame size** of the CNN of Figure 21 follows a normal distribution (bell-shaped curve).

From Figure 21, it can be concluded that the effective frame size of the VQ-VAE follows a bell-shaped curve as well. Thus, the audio-signal in the middle of the frame is weighted the most. To provide a better comparison between VQ-VAE generated tokens or codebooks and mel spectrograms, the frames of the mel spectrogram are smoothed with an also bell-shaped curve. This curve is called Hann function (see Figure 22).

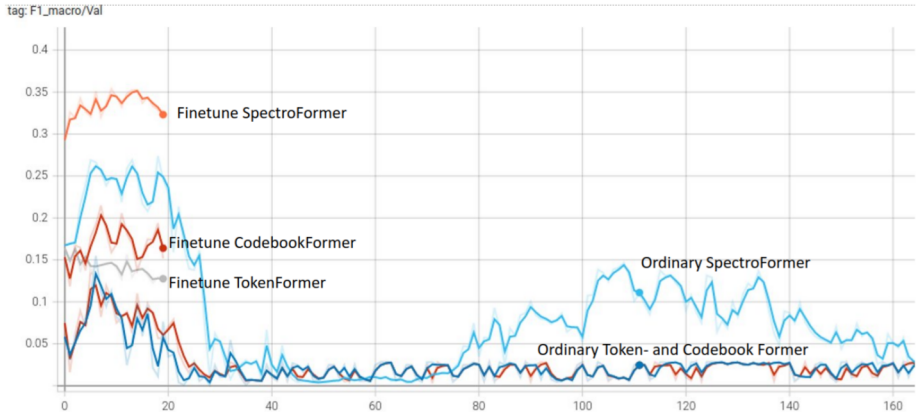
The only remaining hyperparameter needed for generating mel spectrograms is the number of mel bands. The number of mel bands is linked to the frequency resolution. Using more mel bands than the frequency resolution provides would result in a messy upscaling of the mel bands. Using less mel bands than the frequency resolution is providing would mean that there is a downscaling for no particular reason. Therefore, the number of mel bands is chosen to be equal to the frequency resolution. Equation 24 shows how the frequency resolution is calculated for mel

spectrograms. Following Equation 24, 86 mel bands are used.

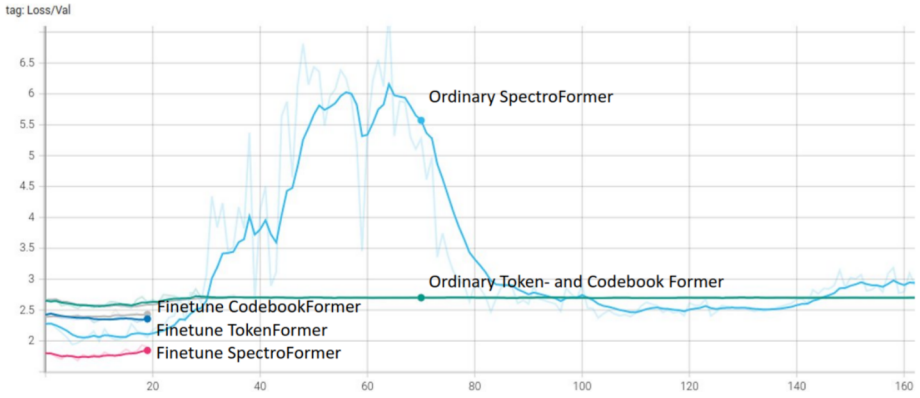
$$\begin{aligned} freq_{bins} &= frame\ size/2 \\ freq_{max} &= sample\ rate/2 \\ freq_{res} &= \frac{freq_{max}}{freq_{bins}} = 86.3 \end{aligned} \tag{24}$$

6 Results

The overall results can be best described with Figure 23 showing the performance of all classification experiments on the validation set. The classification performance in Figure 23 is measured with the F1 score. The F1 score is the harmonic mean of precision and recall (see Equation 25).



(a) Classification performance on the validation set. The y scale describes the macro F1 scale.



(b) Loss on the validaiton set.

Figure 23: Performance of all classification experiments. The x axis numerates the epochs. The x scale is cut out after 160 epochs because no significant changes occur afterwards.

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{25}$$

Precision and recall can be best explained with Figure 24. Concretely, the macro averaged F1 score is used. Macro averaged means that the F1-score is calculated for each class separately and then the F1-scores of all classes are averaged in an unweighted manner. Thus, major and minor classes have the same influence on the macro averaged F1-score.

If the model is just randomly guessing the genre, the probability of being right is $\frac{1}{16}$ (because there are 16 genres). Based on Equation 25, the F1 score for such a scenario would be 0.11. Thus, the baseline F1 score for this thesis experiments is 0.11. A model performing worse than this is not better than random guessing. For all three models, the finetuning setup with a learning rate of $2e-5$ and batch size 16 performs

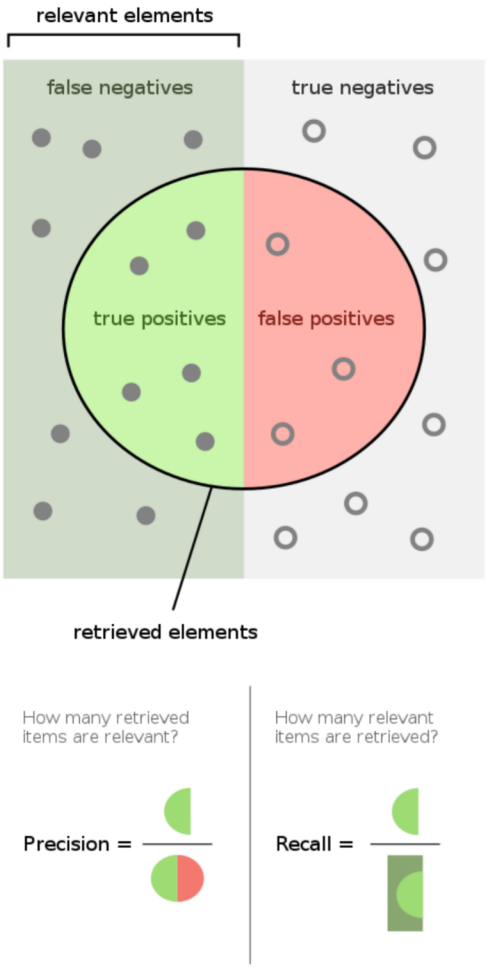


Figure 24: The following illustration provides a intuitive explanation for precision and recall (Figure taken from Wikipedia, 2021)

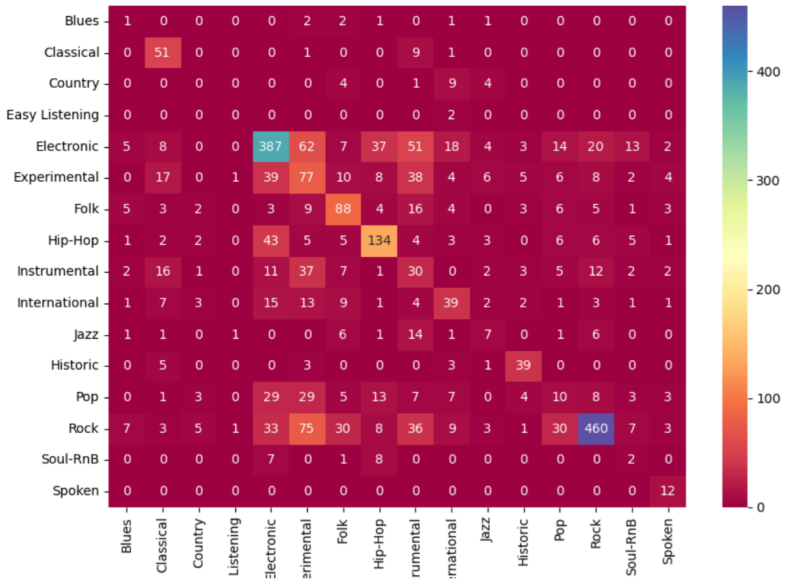
Experiment Setup	F1 Train	F1 Test	Train Loss	Test Loss
Finetune SpectroFormer	0.43	0.34	1.4	1.7
Ordinary SpectroFormer	0.26	0.25	2.0	2.0
Finetune CodebookFormer	0.11	0.17	2.6	2.3
Ordinary CodebookFormer	0.07	0.10	2.7	2.6
Finetune TokenFormer	0.13	0.12	2.5	2.4
Ordinary TokenFormer	0.07	0.09	2.7	2.6

Table 6: Performance after 10 epochs measured with the macro-averaged F1 score.

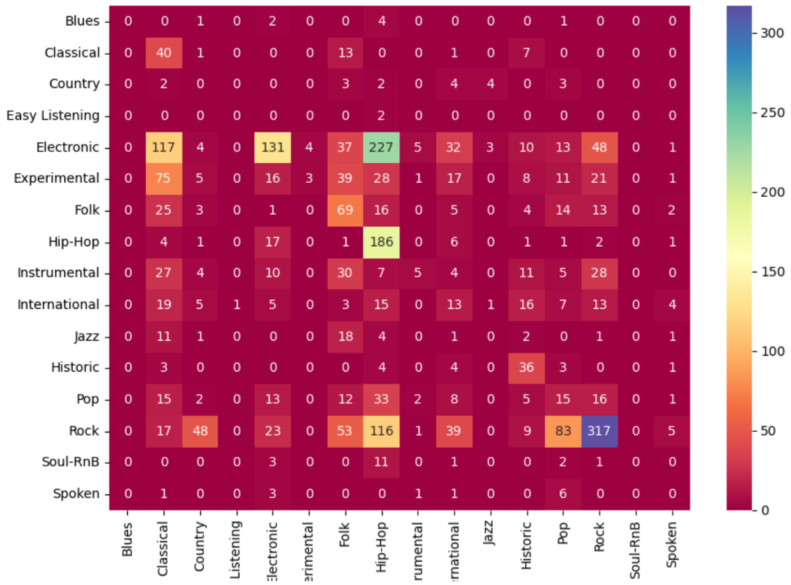
best and is shown in Figure 23 consequently. The other finetuning setups have a slightly worse performance for all three models. All three model perform better when pretrained. SpectroFormer outperforms the other two models significantly, especially when its pretrained. **Therefore, the research hypothesis of this thesis that using deep VQ generated codebooks or tokens as audio representations is superior to mel spectrograms needs to be denied** (at least for this thesis experiment setup).

6.1 Detailed Classification Performance Comparison

No matter if pretrained or not, the overall best results were achieved after just 10 training epochs. Table 6 shows classification performance and loss after 10 epochs for all 6 setups. The only models that clearly performs better than the baseline F1 score of 0.11 are both SpectroFormer models. The finetuned Codebook- and TokenFormer are only slightly above the baseline performance. Figure 25 shows the confusion matrix for the finetuned Spectro- and CodebookFormer offering a more detailed overview over the classification. The confusion matrix shows clearly that the finetuned CodebookFormer is not performing well. Interestingly, all ordinary setups do show similar results on the train and validation dataset. They neither over- nor underfit the training data. However, the best performing model, the finetuned SpectroFormer, tends to overfit significantly, whereas CodebookFormer tends to underfit slightly (see Figure 26).



(a) SpectroFormer finetuned



(b) CodebookFormer finetuned

Figure 25: Confusion matrix on the validation dataset. The left column represents the ground truth, the row on the bottom represents the predicted genre.

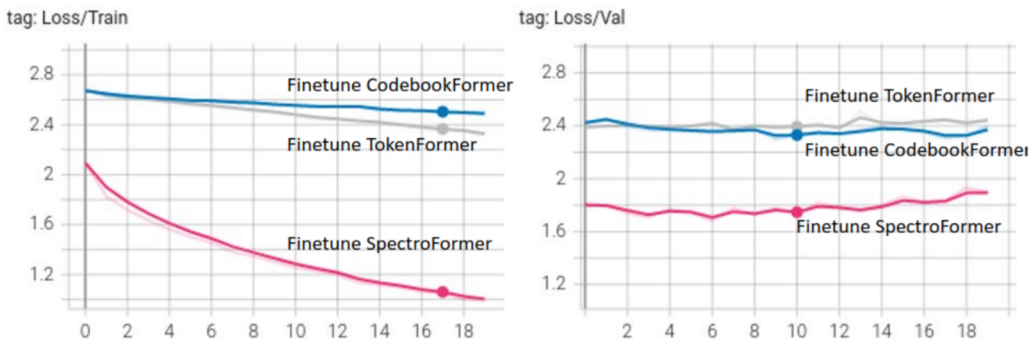


Figure 26: Loss comparison between train and validation set for the finetuned models.

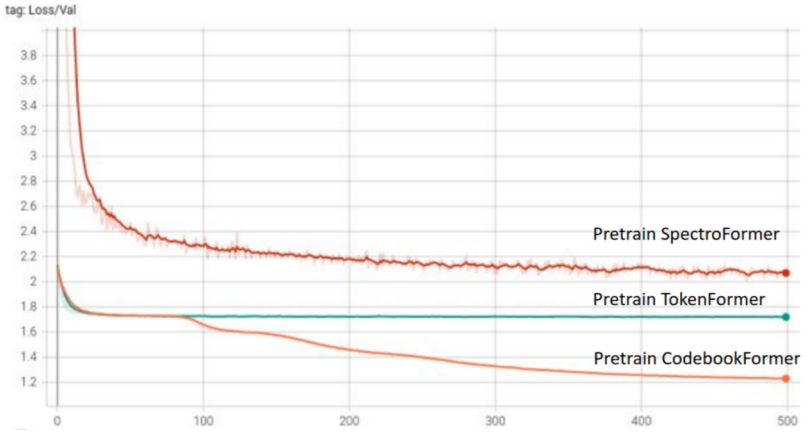


Figure 27: Pretraining loss for all three models

6.2 Pretraining Performance

Figure 27 shows the pretraining loss for all three models on the validation set. For all three models, the pretraining train and validation loss does not significantly differ. The models neither over- nor underfit during pretraining. Although the pretrained SpectroFormer has the best classification performance during finetuning, its pretraining loss remains higher than the other models losses over all epochs. However, it needs to be mentioned here that the pretraining of SpectroFormer is about regressing elements of mel spectrogram matrices whereas Token- and CodebookFormers' pretraining is about classifying the correct token. Because they are trained on different tasks, the pretraining loss of SpectroFormer is not directly comparable to the other models. On the other hand, Codebook- and TokenFormer are trained on the exact same task. Their pretraining losses are directly comparable. Considering the loss of the last epoch, CodebookFormer performs 30% better during pretraining than TokenFormer. During finetuning, however, CodebookFormer achieves almost the same classification performance as TokenFormer (see Figure 23). **The 30% better performance during pretraining is not transmitted to the classification task**.

Although not really recognizable on Figure 27, the TokenFormer pretraining loss is decreasing up until the 300th epoch. The SpectroFormer loss is also decreasing up until the 300 epoch. The CodebookFormer loss is decreasing until the very end of the training. Interestingly, the CodebookFormer loss decreases in rapid shifts. This rather untypical behaviour for a deep learning loss curve could indicate that CodebookFormers' loss surface is highly non-convex (Dirac). Because of this highly untypical loss curve, the pretraining of CodebookFormer is repeated with a different weight initialization. The loss curve of the repeated experiments looks almost like first loss curve.

7 Discussion

This thesis started with the following research question: Using music genre recognition as a benchmark task which audio representation mel spectrograms or deep VQ-based audio representations perform better. Since this research question is empirical, it depends on the experiment setup which in turn depends on various hyperparameters and design choices. Therefore, the research question cannot be answered in an absolute manner. However, the results of this thesis clearly indicate that mel spectrograms significantly outperform deep VQ-based audio representations. For the classification performance, it does not really matter whether tokens or codebooks are used as input. Both Token- and CodebookFormer are almost similarly inferior to mel spectrograms and perform only slightly over baseline.

This section is structured as follows: In the first subsection, the conceptual reasons for the failure of deep VQ-based audio representations are investigated. Although mel spectrograms seem to be a good audio representation for music genre classification, the second subsection investigates if mel spectrograms and other time-frequency based representations do have conceptual drawbacks as well.

7.1 Drawbacks of Deep VQ-based Audio Representations

As mentioned in the introduction of this thesis, spectrograms exhibit a clear, interpretable representation whereas deep VQ-based representations do not show such a structure (see Figure 1). The reason for this difference is simple. Spectrograms are based on the Fourier transformation. Fourier transformations are linear (Karlsruhe). Linear transformations have the quality of being easily understandable and interpretable. A transformation T has to fulfill Equation 26 to be linear.

$$T(x_1 + x_2) = T(x_1) + T(x_2) \tag{26}$$

For Fourier transformations, Equation 26 can be explained by a simple example: If two audio-signals x_1 and x_2 are played simultaneously, they are basically summed up from a mathematical perspective. It does not matter whether the Fourier transformation is applied only once to the summed up signal (left side of Equation 26) or to each signal individually and the frequency representations are summed up (right side of Equation 26).

Due to the non-linear activation functions, deep learning is non-linear. Both the encoder and decoder of the VQ-VAE do not fulfill Equation 26. Non-linearity gives neural networks their expressive power but it does also means that small changes to the input can lead to big differences in the output (Ng). It also means that the loss surface of neural networks is non-convex which is part of the reason why neural networks need so much data for training.

Although SpectroFormer itself is a neural network and therefore non-linear, the spectrograms it receive are a linear representation of audio-signals. Token- and CodebookFormer receive non-linear representations of audio. Having a non-linear representation of audio as input might make the task of genre classification more difficult compared to using a linear audio representation such as spectrograms. This could also mean that a non-linear representation of audio is more data hungry.

Although their paper is about music generation, the assumption that Token- and CodebookFormer could be more data hungry than SpectroFormer is partly supported

by Engel et al. (2020): The VQ-VAE used by this thesis is part of the generative music model Jukebox. Jukebox is a so-called **waveform model**. This means that it generates music by modeling the waveform (although it does so with the help of the VQ-VAE). "The bias of the natural world is to vibrate. [...] Accordingly, human hearing has evolved to be highly sensitive to **phase-coherent oscillation**"[25] (see Engel et al., 2020, Introduction). By modelling the waveform directly, waveform models are not biased towards oscillation. Jukebox could be trained with non-oscillating time series data like stock prices as well. Without the oscillation bias, waveform models are much more data hungry and require larger networks (see Engel et al., 2020, Section 1.1). This is also true for Jukebox. The VQ-VAE of Jukebox is trained with 1.2 million songs (see Dhariwal et al., 2020, Section 5.1). On the other hand, the transformers trained by this thesis only used a dataset of 20k songs. The Fourier transformation decomposes a signal into its frequencies. These frequencies are nothing else than simple oscillations. This means that the Fourier transformation assumes that signals are oscillatory. It is biased towards oscillations. This means that spectrograms and SpectroFormer are biased towards oscillation as well.

The VQ-VAE is trained with 1.2 million songs. By being trained with such a big dataset, the VQ-VAE probably takes oscillation into account after being trained (biased by the dataset). Deep VQ-based representations can therefore be considered to be biased towards oscillation as well. The question is whether this bias is so strong that Token- and CodebookFormer do not need to be trained with a bigger dataset than SpectroFormer. This question could only be answered empirically. However, training a neural network with over 1 million songs is out of the scope of this master thesis.

Finally, there is another thing that is considered by spectrograms and ignored by deep VQ-based audio representations: Different audio signals can be perceived to be exactly the same by humans. Signal 1 and 2 illustrated by Figure 28 are such signals. They are perceived the same by humans. Figure 28 is made by a Jupyter Notebook that is available online.[26] This notebook also provides hearing samples. As can be seen by their waveforms, signal 1 is just a mirrored version of signal 2. The mel spectrogram for both signals looks the same whereas their corresponding token representation looks different. **Because the VQ-VAE generated tokens represent two signals sounding identical for humans differently, it can be concluded that machine learning models using deep VQ-based audio representations need more data than machine learning models that use spectrograms.** In contrast to the non-linearity and the oscillatory biases hypothesis, this hypothesis is the simplest one. According to Occam's razor, simple hypotheses are more likely to be true than complicated ones.

7.2 Drawbacks of Fourier-based Audio Representations

So far, only the drawbacks of the deep VQ-based audio representations were discussed. However, Fourier-based representations like mel spectrograms do have some drawbacks as well.

[25] Phase-coherent means that the relative phase between the frequencies a signal is made of does not change over time.

[26] `https://github.com/ndettmer/mtdml/blob/master/representation_comparison.ipynb`

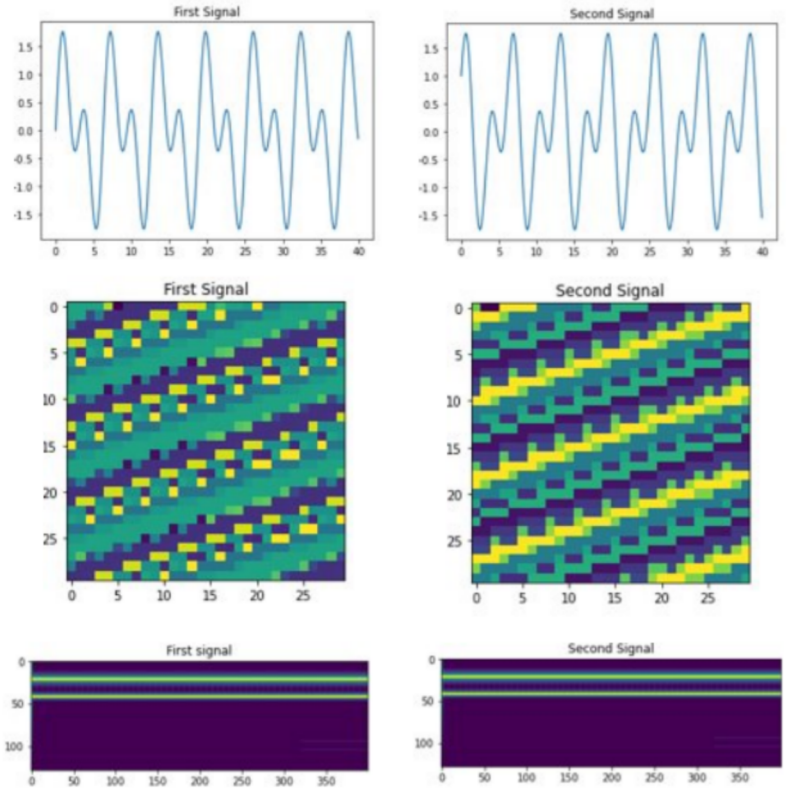


Figure 28: Two identical sounding yet different audio-signals illustrated with different representations. The first row represents the audio-signals as raw waveforms. As can be seen, Signal 1 is a mirrored version of Signal 2. The second row represents them by the token sequence encoded by the VQ-VAE. Each token has its own color. The token sequences are listed from left to right and from top to bottom. The last row illustrates them as spectrograms.

One of these drawbacks is called spectral leakage. In theory, the Fourier transformation is based on an infinite long signal. Of course, real signals and their corresponding frames are always finite. Spectral leakage describes the fact that when the Fourier transformation is applied to a finite long observation frame, there are frequencies in the frequency representation that are not actually in the signal. Window functions like the Hann window (see Figure 22) reduce spectral leakage but cannot make the effect disappear completely. Figure 29 shows the frequency representation of signal 1 from Figure 28. Although signal 1 only entails two frequencies, its frequency representation indicates that it is made out of much more frequencies. The cause for this is the leakage effect. Figure 28 shows the mel spectrogram of signal 1 and 2. The two sharp lines represent both frequencies, signal 1 and 2 are actually made of. The leakage effect does not seem to have a big impact on the mel spectrograms. Because of the hanning window (see Section 4.6), the leakage effect can be reduced dramatically. Thus, it can be concluded that the leakage effect does not matter much for music genre classification and other MIR tasks.

Another possible drawback of Fourier-based representations has to do with the so-called phase. The phase ϕ describes how much a frequency f represented by a periodic wave is shifted from its origin. The phase is described by an angle ranging

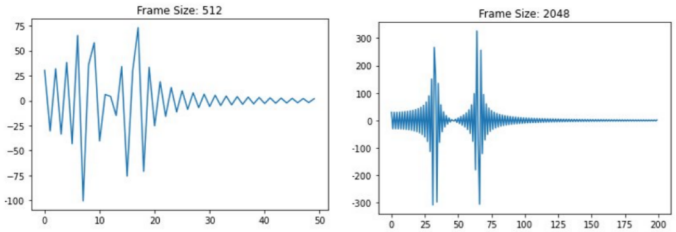


Figure 29: Frequency representation for signal 1 of Figure 28 with different frame lengths for illustrating the leakage effect.

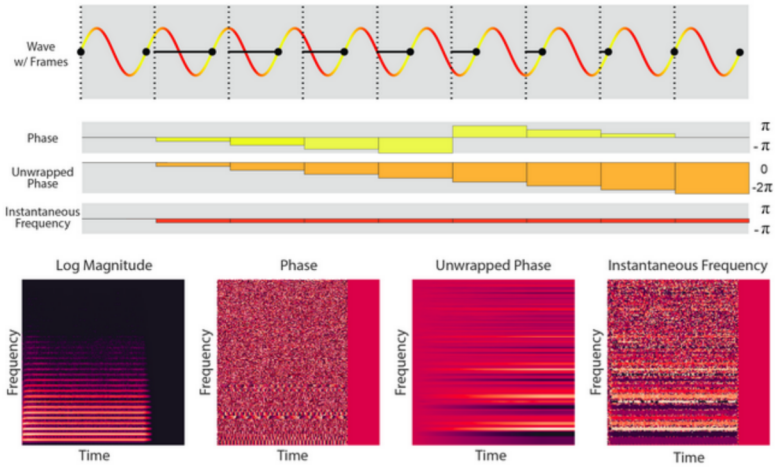


Figure 30: Different representations of the phase.

between 0 and 360 degrees. Usually, a spectrogram shows only the magnitude of the frequencies a signal is made of. The phase is another output of the Fourier transformation and can be represented by a spectrogram as well (see the box second from left at the bottom of Figure 30). Because the phase appears random, it is usually discarded for MIR tasks. For music generation, however, the phase is highly important. Because of its apparent randomness, it is not really possible to model the with machine learning, however (Dieleman). By modeling the waveform directly, Jukebox and other generative models can bypass modeling the phase itself.

The phase can be transformed into the so-called instantaneous phase. "The instantaneous phase captures how much the phase of a signal moves from one spectrogram frame to the next. For harmonic sounds, this quantity is expected to be constant over time, as the phase rotates at a constant velocity" (see Dieleman, 5.2 GANSynth). The instantaneous phase spectrogram does not look random like the normal phase spectrogram (at least for harmonic sounds). How the instantaneous phase is calculated is not further explained by this thesis. The instantaneous phase is only illustrated here as an example to show that the phase can entail useful information, too. GANSynth (Engel et al., 2019) uses the instantaneous phase for generating music. Since the instantaneous phase is only useful for strictly harmonic sounds, GANSynth's performance and application domains are very limited compared to Jukebox, however.

The drawbacks of Fourier-based representations are the reason spectrograms are usually not used for music generation. For MIR, however, the drawbacks of

Fourier-based representations are usually considered to be not significant. This does not necessarily mean that an audio representation that does not has the drawbacks of Fourier-based methods would not be beneficial for MIR.

7.3 Conclusion and Outlook

The discussion section can be summarized as follows: Each kind of audio representation has its advantages and drawbacks. The severity of a drawback depends on the use case. Fourier-based representations seem to be better for downstream MIR tasks, deep VQ-based representations and waveforms themselves seem to be better for music generation. This thesis tried to challenge this assumption and used deep VQ-based representations for music genre recognition which is the most common MIR task. However, the findings of this thesis rather support than challenge the assumption that waveform-based audio representations like deep VQ are not well suited for MIR. Especially Figure 28 shows that deep VQ-based audio representations are not considering the characteristics of human hearing perception enough. Since Token- and CodebookFormer (the models trained with deep VQ-based audio representations) are pretrained with 70 times less data than Jukebox, repeating the pretraining of with more much data might lead to a significant better performance. The assumption that much more data is needed for pretraining might be also supported by the findings of Castellon et al. (2021).

Castellon et al. (2021) successfully repurpose Jukebox for various MIR tasks. They do so by "probing" Jukebox's transformer encoder. "Probing" means that they take the activations of the encoder and feed it in some other machine learning model (a simple linear model in this case). The findings of Castellon et al. (2021) are a strong sign that Jukebox does learn representations of music that are useful for downstream MIR tasks. It could be assumed that Jukebox's transformer encoder does learn useful representations of music because it is trained with 70 times more data than the models of this thesis. If Token- and CodebookFormer would be pretrained with much more data then they might perform better during finetuning with small-sized finetuning datasets.

A Software Packages and Class Diagrams

The deep learning framework PyTorch (version 1.9.1) is used together with PyTorch Lightning (version 1.4.9). PyTorch Lightning provides a high-level API for PyTorch and decouples the science code from boilerplate engineering code. The science code (the source code developed during this thesis is an example of science code) is structured in a class called `LightningModule` which defines a complete machine learning system that contains the code for the neural network together with the loss function. The train loop, validation loop and various other things are also functionalities of the `LightningModule` that are already provided by Lightning.

The source code of this thesis is structured as follows: the different neural networks are coded in the module `architures.py` (see Figure 31) and are then passed over to the adequate `LightningModule` (see Figure 32).

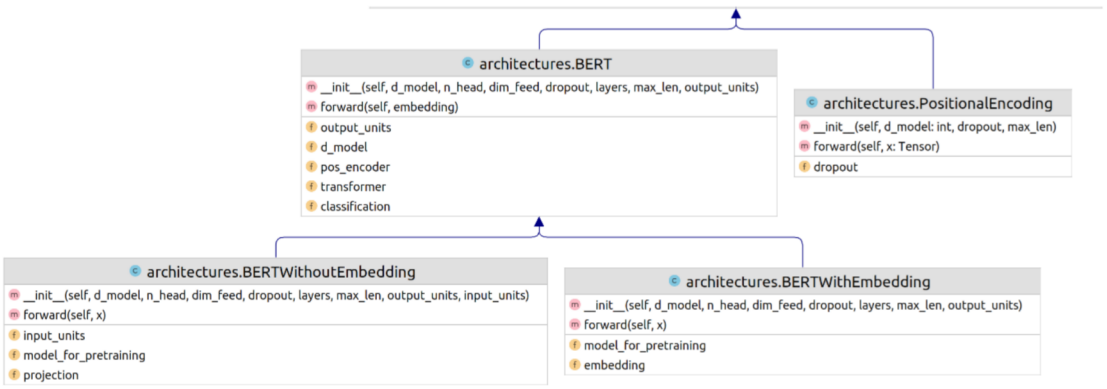


Figure 31: The class diagrams for the different neural networks. They are all child classes of the PyTorch `torch.nn.Module` class for neural networks. `BERTWithEmbedding` is for BERT4Music and `BERTWithoutEmbedding` is for SpectroFormer.

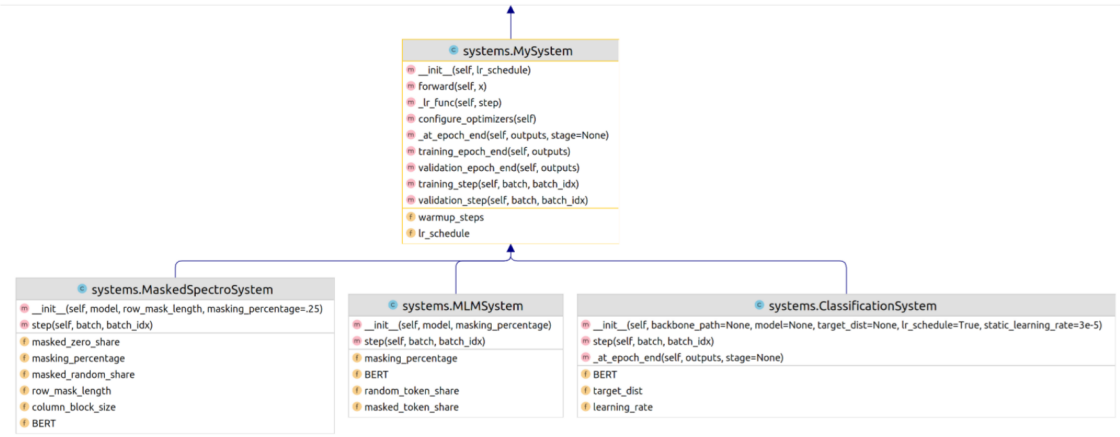


Figure 32: The class diagram for the different machine learning systems. When initialized, they are handed over the neural network that they will train. They are all child classes of `LightningModule`.

B Scaling Neural Network Training

Because transformers VRAM usage is quadratic with regards to the input sequence length, special scaling techniques are usually necessary for training transformers successfully. This section describes two popular techniques for scaling neural network training: Mixed precision and multi-GPU training. These techniques are also used for training Spectro-, Token-, CodebookFormer.

B.1 Mixed Precision Training

Neural Networks are usually trained with GPUs which work with the single precision floating point format. This mean that each floating point is presented by 32 bits (FP32). It has been shown that training parts of the neural networks with half precision floating points (FP16) instead, leads to equal performance which makes sense considering the "approximative nature" of deep learning Micikevicius (2020). Combining single with half bit precision is called mixed-precision training. NVidida has developed special GPUs which can utilize Mixed-Precision training. Mixed-precision training has the following benefits:

- Decrease the required amount of memory by almost a factor of 2
- Shorten the training or inference time

Since July 2020, mixed-precision training is an integrated functionality of PyTorch.[27] Because mixed-precision is so new to PyTorch, it does not always work as expected. For mixed-precision to work properly with this thesis' models, the epsilon values of the Adam optimizer and layer normalization needed to be increased. The default epsilon values are too small to be represented by 16bit precision.[28]

[27] `https://pytorch.org/blog/accelerating-training-on-nvidia-gpus-with-pytorch-automatic-mixed`

[28] See PyTorch issue dealing with NaN loss when using 16bit precision (`https://github.com/pytorch/pytorch/issues/40497`)

B.2 Distributed Training

The environment used by this thesis has 4 GPUs (see Section C.2). Because transformers benefit from very strong hardware, it makes sense to use all these GPUs together. Using multiple GPUs for training a neural network is called distributed training. In general, there are two high-level kinds of distributed training:

- Data parallel: The data is split across the GPUs but each GPU has its own instance of the neural network
- Model parallel: Different parts of the neural network are split across the GPUs

Because data parallel is conceptually more simple, model parallel is usually only used when the neural network does not fit inside a single GPU which is not the case for this thesis' models. Consequently, distributed data parallel[29] is used as training strategy.

Distributed data parallel works as follows:

1. Each GPU across each node gets its own process.
2. Each GPU gets visibility into a subset of the overall dataset. It will only ever see that subset.
3. Each process inits the model.
4. Each process performs a full forward and backward pass in parallel.
5. The gradients are synced and averaged across all processes.
6. Each process updates its optimizer.

Because each process inits the model, the source code of the model needs to be "pickleable". "Pickleable" means that the software object defining the neural network is serializable. Functional programming functionalities of Python like lambda functions are not "pickleable" and need to be replaced. With 4 GPUs, distributed data parallel scales the effective VRAM size by a factor of 4. Together with mixed precision, the VRAM size is almost eightfold.

C Environment and Hardware

The environment in which the neural network training was performed is the High-Performance-Computing (HPC)-Cluster from Universität Osnabrück.[30] HPC runs with Linux and SLURM as a resource manager. HPC can only be accessed over the command line interface and the VPN of Universität Osnabrück. Conda is used as package management system. Every publication using HPC needs to state its HPC project number. This thesis has the **project number 456666331**.

[29] `https://pytorch-lightning.readthedocs.io/en/1.4.9/advanced/multi_gpu.html#distributed-data-parallel`

[30] `https://www.rz.uni-osnabrueck.de/Dienste/HPC/index.htm`

C.1 SLURM

Many different people work with the same supercomputer. Therefore, the resources of that computer need to be managed. In the case of HPC, resource sharing is managed by the software SLURM. A supercomputer is a set of compute nodes (a node is a single computer). SLURM separates these nodes into different partitions. In the case of HPC, only the home partition (dedicated to resource sharing and not to job execution) has access to the internet. Because all partitions share the same file system for the same user, packages can be downloaded and installed in the home partition and can still be used in another partition.

Listing 2: Shell Script for SLURM

```
#!/bin/bash
#SBATCH --job-name=token
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --ntasks-per-node=1
#SBATCH --partition=hpc3_gpu

source ~/.bashrc
conda activate TokenMIR

srun python train.py classify_from_tokens
--ds_path=~/data/fma_medium_tokens --batch_size=64
--token_sequence_length=1344 --epochs=500 --d_model=768
--n_head=12 --dim_feed=3072 --dropout=0.1 --layers=4
--gpus=1 --precision=16 --name=example
```

C.2 Hardware

The partition used by this thesis is called *hpc3_gpu*. It consists of only one node. 4 NVidia A100 GPUs are built inside this node. Each has 82 GB VRAM and a performance of 19.5 Teraflops for 32bit precision. For Tensor Floats, the performance increases up to 156 Teraflops (32bit precision) or 312 Terraflops (16bit precision).[31] The GPUs run with Cuda (version 11.4) and use the NVidia Collective Communication Library (NCCL) for multi-GPU and multi-node communication.

[31] As a comparison, the gaming console PS5 has 10.3 Terraflops 32bit precision.

Using SLURM

Marko Duda edited this page on 4 Jan · 8 revisions

Here you can find some useful terminal commands while using SLURM!

Connect with SLURM

Make sure the University VPN is turned on!

```
ssh USERNAME@hpc1.rz.uos.de
```

Download TokenDataset from GDrive

```
wget --load-cookies /tmp/cookies.txt "https://docs.google.com/uc?export=download&confirm=$(wget --quiet
--save-cookies /tmp/cookies.txt --keep-session-cookies --no-check-certificate 'https://docs.google.com
/uc?export=download&id=1-DwEf9Z7B5GOH8wyCZs1x9d-eqTEPpqa' -O- | sed -rn 's/.*confirm=([0-9A-Za-z_]+).*
/\1\n/p')&id=1-DwEf9Z7B5GOH8wyCZs1x9d-eqTEPpqa" -O medium_ds.zip && rm -rf /tmp/cookies.txt
```

Interactive GPU Cluster

```
srun -p hpc3_gpu --pty bash
```

Configuration

Do this before you execute a job with `sbatch JOB.sh` !

SLURM with Librosa

```
source ~/.bashrc
```

```
module load libsndfile
```

```
conda activate CONDA_ENV
```

Folder structure

Make sure your data is stored under the path `~/data`

Backbone

When you want to finetune a pretrained model, make sure to copy the model to the correct path before:

```
cp SOURCE ~/backbone/backbone.ckpt
```

Logging

Copy the `tb_log` folder to your local machine with

```
scp -r USERNAME@hpc1.rz.uos.de:PATH_TO_TOKEN_MIR_REPO/tb_log ~/.
```

Create a Jupyter Notebook on your local machine with Tensorboard to look at 'tb_log'

Figure 33: Wiki page made for this thesis on how to use SLURM with this thesis' research code. `https://github.com/drduda/TokenMIR/wiki/Using-SLURM`

Bibliography

Librosa short-time fourier transform. URL `http://librosa.org/doc/main/generated/librosa.stft.html`.

André Araujo, Wade Norris, and Jack Sim. Computing receptive fields of convolutional neural networks. *Distill*, 2019. doi: 10.23915/distill.00021. https://distill.pub/2019/computing-receptive-fields.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dmitry Bogdanov, Minz Won, Philip Tovstogan, Alastair Porter, and Xavier Serra. The mtg-jamendo dataset for automatic music tagging. 2019.

Rodrigo Castellon, Chris Donahue, and Percy Liang. Codified audio language modeling learns useful representations for music information retrieval. *arXiv preprint arXiv:2107.05677*, 2021.

Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. Fma: A dataset for music analysis. *arXiv preprint arXiv:1612.01840*, 2016.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Sander Dieleman. Generating music in the waveform domain. `https://benanne.github.io/2020/03/24/audio-generation.html`. Accessed: 2021-07-14.

Sander Dieleman, Aaron van den Oord, and Karen Simonyan. The challenge of realistic music generation: modelling raw audio at scale. *Advances in Neural Information Processing Systems*, 31, 2018.

Leo Dirac. Geometric intuition for training neural networks. URL `https://www.youtube.com/watch?v=Z_MA8CWKxFU`. Accessed: 2022-03-04.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Jesse Engel, Kumar Krishna Agrawal, Shuo Chen, Ishaan Gulrajani, Chris Donahue, and Adam Roberts. Gansynth: Adversarial neural audio synthesis. *arXiv preprint arXiv:1902.08710*, 2019.

Jesse Engel, Lamtharn Hantrakul, Chenjie Gu, and Adam Roberts. Ddsp: Differentiable digital signal processing. *arXiv preprint arXiv:2001.04643*, 2020.

Arthur Flexer. A closer look on artist filters for musical genre classification. *World*, 19(122):16–17, 2007.

Zellig S. Harris. Distributional structure. *iWORD/i*, 10(2-3):146–162, August 1954. doi: 10.1080/00437956.1954.11659520. URL https://doi.org/10.1080/00437956.1954.11659520.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Hsiao-Tzu Hung, Yu-Hua Chen, Maximilian Mayerl, Michael Vötter, Eva Zangerle, and Yi-Hsuan Yang. Mediaeval 2019 emotion and theme recognition task: A vq-vae based approach. In *MediaEval*, 2019.

Hochschule Karlsruhe. Rechenregeln der fourier-transformation - linearität. URL https://www.eit.hs-karlsruhe.de/mesysto/teil-a-zeitkontinuierliche-signale-und-systeme/spektrum-eines-signals/rechenregeln-der-fourier-transformation/linearitaet.html. Accessed: 2022-03-07.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Wenqi Li, Guotai Wang, Lucas Fidon, Sebastien Ourselin, M. Jorge Cardoso, and Tom Vercauteren. On the compactness, efficiency, and representation of 3d convolutional networks: Brain parcellation as a pretext task. *Information Processing in Medical Imaging*, page 348–360, 2017. ISSN 1611-3349. doi: 10.1007/978-3-319-59050-9_28. URL http://dx.doi.org/10.1007/978-3-319-59050-9_28.

Paulius Micikevicius. Mixed-precision training of deep neural networks, Aug 2020. URL https://developer.nvidia.com/blog/mixed-precision-training-deep-neural-networks/.

Andrew Ng. Why non-linear activation functions (c1w3l07). URL https://www.youtube.com/watch?v=NkOv_k7r6no.

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. *arXiv preprint arXiv:1609.03499*, 2016.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

Ali Razavi, Aaron van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, pages 14866–14876, 2019.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

Valerio Velardo. Mel spectrograms explained easily, a. URL `https://www.youtube.com/watch?v=9GHCiiDLHQ4&list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0&index=18`. Accessed: 2022-02-09.

Valerio Velardo. Short-time fourier transform explained easily, b. URL `https://www.youtube.com/watch?v=-Yxj3yfvY-4&list=PL-wATfeyAMNqIee7cH3q1bh4QJFAaeNv0&index=16`. Accessed: 2022-02-09.

Wikipedia. F-score — Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=F-score&oldid=1054927460`, 2021. [Online; accessed 26-November-2021].

Yilun Zhao and Jia Guo. Musicoder: A universal music-acoustic encoder based on transformer. In *International Conference on Multimedia Modeling*, pages 417–429. Springer, 2021.

List of Figures

1 Spectrogram (left) and codebook representation (right) of a rock song 3
2 Spectrograms of ground-truth audio signal (left), reconstructed by codebooks (middle) and by Opus codec with bitrate comparable to codebooks (right) (figure taken from Dhariwal et al., 2020, Figure 4 (edited)) 3
3 This image is taken from Defferrard et al. (2016). (left) An example of genre hierarchy for the top-level genre Soul-RnB. The left number is the genre id, the right number indicates the number of tracks per genre. Some genres do not occur often: The root-level genre Deep Funk is only represented with a single track within the whole dataset. 5
4 This figure is taken from the FMA dataset paper (Defferrard et al., 2016) and describes how many tracks there are for all 16 root genres on the medium subset (min 21, max 7,103). 6
5 A rock and a electronic/minimal song represented by spectrograms (top) and by a sequence of tokens (bottom). The token sequence is encoded by the VQ-VAE. Each token has its own color. The sequences are listed from left to right and from top to bottom. 8
6 This figure is taken from the paper "Attention is All You Need" (Vaswani et al., 2017) and describes the transformer encoder-decoder-architecture. 10
7 This Figure is also taken from Vaswani et al. (2017) and shows how the attention mechanism works. Since the masking is optional and is not be used in this thesis, it will not be explained any further. 11
8 These are the positional encodings. Each row corresponds to a positional encoding vector. 13
9 The dot product between $PE(i = 256)$ and all other positional encodings. Close PE have a higher dot product. 14
10 The overall structure of Jukebox. 18
11 The inference process of the VQ-VAE for all three levels. This is images is taken from the Jukebox (Dhariwal et al., 2020). 19

12 "Comparison of reconstructions from different VQ-VAEs, x-axis is time and y-axis is frequency. The columns from left to right are bottom-, middle-, and top-level reconstructions at hop lengths 8, 32, and 128 respectively, visualized as mel spectrograms. The first row is the ground-truth, and the second row shows the spectrograms of audio outputs from our VQ-VAE. In the third row, we remove the spectral loss, and see that the middle and toplevel lose high-frequency information. In the fourth row, we use a hierarchical VQ-VAE (Razavi et al., 2019) instead of separate auto-encoders [...], and we see the middle and top levels are not used for encoding pertinent information. Finally, the fifth row shows a baseline with the Opus codec that encodes audio at constant bitrates comparable to our VQ-VAE. It also fails to capture higher frequencies and adds noticeable artifacts at the highest level of compression." (image and citation taken from Dhariwal et al., 2020, Figure 4). The Opus codec is a modern and widely used audio format developed 2012. 21
13 (a) Normal convolution layers increase their receptive field only linearly with an increasing number of layers (image taken from Oord et al., 2016, Figure 2) (b) The receptive field of dilated convolution kernels increase exponentially with an increasing number of layer (image taken from Oord et al., 2016, Figure 3). That the receptive field goes only to the left side is a particularity of Wavenet. The receptive field of Jukebox's VQ-VAE goes to both sides. 22
14 Performance comparison of different benchmark finetuning tasks for BERT models pretrained with bidirectional and Next Sentence Prediction (abbreviated NSP), without NSP but bidirectional, and without NSP and left-to-right self supervised training. (figure is taken from Devlin et al., 2018, Table 5). 23
15 A systematic overview of MusiCoder (image taken from Zhao and Guo, 2021, Figure 1) 25
16 (a) Acoustic features used by MusiCoder (figure taken from Zhao and Guo, 2021, Table 3). (b) The performance of MusiCoder for the MTG-Jamendo music auto-tagging task dataset measured in ROC-AUC (figure taken from Zhao and Guo, 2021, Table 6). Comparison between no pretraining; channel masking; frame masking; channel and frame masking pretraining. 25
17 Top: magnitude spectrogram of a piano recording. Bottom: the corresponding phase spectrogram. The y scale represents time and the x scale represents the frequencies 27
18 Mel bands are triangular filters 29
19 A systematic overview over all three transformer models of this thesis. Although the pretraining happens before the classification, both are shown in this figure as if they are happening simultaneously. The legend explains what the corresponding symbol means. Trapezoids having the same color are actually referring to the same linear projection applied to different inputs. 31

20 The learning rate schedule used for Token-, Codebook- and SpectroFormer. . . . 35
21 This image (taken from Li et al., 2017, Figure 2) shows the frame size of a neural network made out of residual blocks only. This means each layer is connected to all previous layers. n residual blocks have a collection of 2^n unique paths. The figure shows the number of unique paths per frame size for this network. . . . 37
22 The Hann function is used for smoothing the frames of the Mel Spectrograms. . . . 37
23 Performance of all classification experiments. The x axis numerates the epochs. The x scale is cut out after 160 epochs because no significant changes occur afterwards. . . . 39
24 The following illustration provides a intuitive explanation for precision and recall (Figure taken from Wikipedia, 2021) . . . 40
25 Confusion matrix on the validation dataset. The left column represents the ground truth, the row on the bottom represents the predicted genre. 42
26 Loss comparison between train and validation set for the finetuned models. . . . 42
27 Pretraining loss for all three models . . . 43
28 Two identical sounding yet different audio-signals illustrated with different representations. The first row represents the audio-signals as raw waveforms. As can be seen, Signal 1 is a mirrored version of Signal 2. The second row represents them by the token sequence encoded by the VQ-VAE. Each token has its own color. The token sequences are listed from left to right and from top to bottom. The last row illustrates them as spectrograms. . . . 46
29 Frequency representation for signal 1 of Figure 28 with different frame lengths for illustrating the leakage effect. . . . 47
30 Different representations of the phase. . . . 47
31 The class diagrams for the different neural networks. They are all child classes of the PyTorch `torch.nn.Module` class for neural networks. `BERTWithEmbedding` is for BERT4Music and `BERTWithoutEmbedding` is for SpectroFormer. . . . 49
32 The class diagram for the different machine learning systems. When initialized, they are handed over the neural network that they will train. They are all child classes of `LightningModule`. . . . 50
33 Wiki page made for this thesis on how to use SLURM with this thesis' research code. `https://github.com/drduda/TokenMIR/wiki/Using-SLURM` . . . 53

List of Tables

1 Sequence length for 30 second audio snippets with 44.1kHz sample rate compressed by VQ-VAE. 8
2 All relevant hyperparameters of MusiCoder and BERT. For both models, the smaller *BASE* size is presented. The question mark indicates that the hyperparameter exists but could not be found in the corresponding paper. 32
3 Architecture hyperparameters of Spectro-, Code- and TokenFormer . 32
4 Training hyperparameters concerning the training of Spectro-, Codebook- and TokenFormer. 35
5 For the finetuning, an exhaustive search over the set of these hyperparameters is performed. 36
6 Performance after 10 epochs measured with the macro-averaged F1 score. 41

Nomenclature

BERT **B**idirectional **E**ncoder **R**epresentation from **T**ransformers; see Section 4.4

Codebooks The encodings of the VQ-VAE are represented by a set of codebook vectors.

Deep VQ audio representations Codebooks and tokens generated by the VQ-VAE

Fourier-based audio representations Time-frequency representations like spectrograms and mel spectrogram

MIR Music Information Retrieval; includes all kinds of downstream tasks that extract information from music like music genre classification, music sentiment analysis, and music recommender systems. The definition of MIR usually includes music generation as well. Because it improves the reading flow, MIR is defined to exclude music generation in this thesis

NLP Natural Language Processing

NSP Next Sentence Prediction; a pretraining task of BERT

Token Because the set of codebook vectors has a fixed size, each of these codebook vectors can be represented symbolically by a token

VQ-VAE Vector Quantization-Variational Autoencoder; the part of Jukebox responsible for compressing audio (see Section 4.2)

VRAM Video RAM; working memory of the GPU

Declaration of Authorship

I, Marko Duda, hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university. Furthermore, I certify that the digital version of this thesis (send per mail on April 21st, 2022) match exactly the printed one.

.....

signature

.....

city, date