MSE | MASTER OF SCIENCE IN ENGINEERING

Lucerne University of Applied Sciences and Arts
HOCHSCHULE LUZERN

Master of Science in Engineering
Vertiefungsmodul II

Generating Background Music from Video Game Screenshots

Student: Manuel Jordan
manuel.jordan@stud.hslu.ch

Advisor: Prof. Dr. Thomas Koller
thomas.koller@hslu.ch

Submission Date: 02.02.2020

Declaration of Originality

I hereby declare that I wrote my work independently and did not use any other sources and aids than those indicated. All used text excerpts, quotations or contents of other authors were explicitly marked as such. I hereby declare that this thesis represents my original work and that I have used no other sources except as noted by citations.

All data, tables, figures and text citations which have been reproduced from any other source, including the internet, have been explicitly acknowledged as such.

Place, Date	Signature

Acknowledgements

Hereby I would like to thank all persons who supported me during the work and provided me helpful advice.

Special thanks go to my advisor Prof. Dr. Thomas Koller, who supported and advised me during this time.

Furthermore, I would also like to thank Joel Salzmann and Benjamin Haymond for proofreading parts of my work.

Abstract DE

Im Rahmen des Vertiefungsmoduls II des Master of Science in Engineering beschäftigt sich die vorliegende Arbeit mit der Erzeugung von Hintergrundmusik für Videospiele. In den letzten Jahren wurden große Fortschritte bei der Erzeugung von Bildern, Videos und Text erzielt. Die Erzeugung von Musik hat einige bemerkenswerte Unterschiede zur Erzeugung von Bildern und Videos. Musik ist eine Kunst der Zeit, die ein zeitliches Modell erfordert. Musik besteht in der Regel aus mehreren Instrumenten/Spuren mit einer eigenen zeitlichen Dynamik. In einem Kollektiv entfalten sie sich im Laufe der Zeit in wechselseitiger Abhängigkeit. In mehrstimmiger Musik werden Musiknoten oft zu Akkorden oder Melodien gruppiert.

In dieser Arbeit wurden bestehende Systeme erweitert und kombiniert, die es erlauben, Musik auf Basis eines Bildes zu erzeugen. Zur Vereinfachung wurde der Datensatz auf ein einziges Videospiel reduziert, mit dem die grundsätzliche Machbarkeit bewiesen wird.

Das Model wurde mit 35 Songs und rund 50'000 Screenshots aus dem Nintendo 64 Spiel "The Legend of Zelda: Ocarina of Time" trainiert.

Die Ergebnisse zeigen, dass das Model die Songs zum Teil sehr gut wiedergeben kann. Aufgrund der Überanpassung des Models ist es jedoch nicht in der Lage, zu neuen Screenshots die bis anhin noch unbekannte Musik vorauszusagen.

Durch eine Vergrösserung des Datensatzes, der Ausweitung auf mehrere Videospiele und einer Optimierung des Models sollte es zukünftig möglich sein, dieses Problem zu lösen.

Abstract EN

In the context of the second project work of the Master of Science in Engineering, this thesis deals with the creation of background music for video games. Recent years have seen major progress in generating images, videos and text. Generating music has a few notable differences from generating images and videos. Music is an art of time, requiring a temporal model. Music is usually composed of multiple instruments / tracks with their own temporal dynamics. In a collective they unfold over time interdependently. Musical notes are often grouped into chords, arpeggios or melodies in polyphonic music.

In this work, existing systems were extended and combined, allowing music to be created on the basis of an image. For simplification, the data set was reduced to a single video game to prove the general feasibility.

The model was trained with 35 songs and about 50'000 screenshots from the Nintendo 64 game "The Legend of Zelda: Ocarina of Time".

The results show that the model can reproduce the songs very well in some cases. However, due to the overfitting of the model it is not able to predict the music, which has not been known so far, for new screenshots.

By enlarging the data set, extending it to several video games and optimizing the model, it should be possible to solve this problem in the future.

Acronyms

BLEU bilingual evaluation understudy. 6

CNN convolutional neural network. 6, 9

GAN generative adversarial network. 2, 4, 6, 8

LR learning rate. 18, 27

LSTM long short-term memory. 4, 5

MIDI Musical Instrument Digital Interface. VII, 3, 5–8, 10, 11

PCM pulse-code modulation. 3

RNN recurrent neural network. 6

SeqGAN sequence generative adversarial network. 6

VAE variational autoencoder. 4, 5, 7, 8, 22

Glossary

dungeon a mostly maze-like system of nested rooms and corridors. 3

Keras open-source neural-network library written in Python running on top of TensorFlow. 13, 16, 24

letsplay a video documenting the playthrough of a video game. 10, 12

Contents

Declaration of Originality I

Acknowledgements II

Abstract DE III

Abstract EN IV

Acronyms V

Glossary VI

I. Main Body 1

1. Introduction 2
- 1.1. Motivation . . . 2
- 1.2. Music in Video Games . . . 2
  - 1.2.1. Ambiguity of screenshots . . . 3
  - 1.2.2. Ambiguity of music . . . 3
- 1.3. Music Representation . . . 3
  - 1.3.1. Waveform . . . 3
  - 1.3.2. Musical Instrument Digital Interface (MIDI) . . . 3

2. State of the Art 4
- 2.1. Introduction . . . 4
- 2.2. Classifying Variational Autoencoder . . . 4
- 2.3. Hierarchical Variational Autoencoder . . . 5
- 2.4. WaveNet Autoencoder . . . 5
- 2.5. Polyphonic SeqGAN . . . 6
- 2.6. MuseGAN . . . 6
- 2.7. Learning a Latent Space of Multitrack Measures . . . 7
- 2.8. Conclusion . . . 8

3. Methods 9
- 3.1. Concept . . . 9
- 3.2. Training Data . . . 10
  - 3.2.1. Data Acquisition . . . 10
  - 3.2.2. Data Processing . . . 10
- 3.3. Models . . . 12
  - 3.3.1. Type 1 . . . 13
  - 3.3.2. Type 2 . . . 13
  - 3.3.3. Type 3 . . . 14
  - 3.3.4. Type 4 . . . 14
  - 3.3.5. Type 5 . . . 14
  - 3.3.6. Configurations . . . 15

4. Results 17
- 4.1. Training . . . 17
- 4.2. Song Reconstruction . . . 18

4.3. Song Comparison 20

5. Discussion **22**
5.1. Evaluation 22
5.2. Problems 22
5.3. Future Work 22

Bibliography **23**

List of Figures **24**

List of Tables **25**

II. Appendix 26

A. Training **27**
A.1. Training Configurations 27
A.2. Training Data Samples 28

B. Result **29**
B.1. Song Reconstruction 29

Part I.

Main Body

1. Introduction

1.1. Motivation

For decades, researchers have approached the task of algorithmic music generation using computational models [10]. One common approach to this task is to generate samples sequentially using a probability model. It takes a corpus of training data to learn a probability distribution of the most likely notes to be played at each time step.

Recent years have seen major progress in generating images, videos and text, notably using a generative adversarial network (GAN). Along with autoencoders, GANs are nowadays popular models for generating musical sequences. So far, the focus has mostly been on monophonic music.

Recently, the generation of polyphonic music has also moved into focus. Based on the latest findings, this thesis will examine to what extent this can be applied to video games. This work tries to go one step further and will try to generate music in dependence of an image. Thus, for each screenshot from a video game the appropriate background music is supposed to be generated.

1.2. Music in Video Games

The music in video games varies widely and is difficult to generalize. In order to simplify the definition, the focus is moved to a single game, which also serves as a data set later (see chapter 3.2).

For this work, the game "The Legend of Zelda: Ocarina of Time", one of the most popular games on the Nintendo 64 console, was chosen.

Figure 1.1.: Logo of "The Legend of Zelda: Ocarina of Time", Source: Gamepedia [1]

[1] https://gamepedia.cursecdn.com/zelda_gamepedia_en/thumb/3/34/OoT_Black_Logo.png/603px-OoT_Black_Logo.png

With regard to this game, it can be said that music basically depends on three factors:

location: the level or dungeon you are in, the atmosphere

time: day-night cycle, it is quieter at night than during the day

situation: enemy/boss fight, occurrence of an emotional event

1.2.1. Ambiguity of screenshots

A melody cannot always be clearly assigned to a picture. Different themes are possible for the same scene, depending on the situation in the game e.g.:

- low health, short breath, short time
- events
- fights
- in-game music, e.g. someone in the game plays an instrument

This problem can be avoided by focusing only on the main theme of the level and not including the situation-dependent melodies in the data set at all.

1.2.2. Ambiguity of music

The same music can be played at different levels or used several times for different situations. It is also possible that the music is played in the same place, but the place itself varies greatly.

A good model should actually be able to deal with this problem. The prerequisite is a sufficiently large data set.

1.3. Music Representation

1.3.1. Waveform

The most common way to store and play music is by using its waveform. Usually, the waveforms are represented with the pulse-code modulation (PCM), a method to digitally represent sampled analog signals.

1.3.2. Musical Instrument Digital Interface (MIDI)

Standard Musical Instrument Digital Interface (MIDI) Files contain all the MIDI instructions to generate notes, control individuals volumes, select instrument sounds, and even control reverb and other effects. The files are typically created by a "MIDI sequencer" (software or hardware) and then played on some kind of MIDI synthesizer.

Unlike digital audio files (.wav, .aiff, etc.) or even compact discs or cassettes, a MIDI file does not need to capture and store actual sounds. Instead, the MIDI file can be just a list of events which describe the specific steps that a soundcard or other playback device must take to generate certain sounds. This way, MIDI files are much smaller than digital audio files, and the events are also editable, allowing the music to be rearranged, edited, even composed interactively, if desired.

The format also allows tagging the file and the data in the file with copyright notices and other text “meta-events”.

All popular computing platforms can play MIDI files (*.mid) and there are thousands of web sites offering files for sale or even for free. Anyone can make and share a MIDI file, using software that is readily available on smart phones, tablets and computers. [2]

2. State of the Art

2.1. Introduction

Generating music has a few notable differences from generating images and videos. Music is an art of time, requiring a temporal model. Music is usually composed of multiple instruments / tracks with their own temporal dynamics. In a collective they unfold over time interdependently. Musical notes are often grouped into chords, arpeggios or melodies in polyphonic music.

For decades, researchers have approached the task of algorithmic music generation using computational models [10]. One common approach to this task is to generate samples sequentially using a probability model. It takes a corpus of training data to learn a probability distribution of the most likely notes to be played at each time step.

Recent years have seen major progress in generating images, videos and text, notably using GANs. Along with autoencoders, GANs are nowadays popular models for generating musical sequences. So far, the focus has mostly been on monophonic music.

This review will subsequently examine six different approaches to generate polyphonic music. All the presented models are based either on a GAN or on a autoencoder.

2.2. Classifying Variational Autoencoder

The first paper [6] introduces the variational autoencoder (VAE), which are popular probabilistic generative model. One drawback of VAEs is that the latent variables cannot be discrete, which makes it difficult to generate data from different modes of a distribution. The researchers propose an extension to existing VAE models to get around this problem.

Jay A Hennig et. al start with preliminary clarifications on polyphonic music and the twelve pitch classes in western music. The key of a piece is the group of these classes that forms the basis of a music composition.

For the model to stay in key during the whole clip the researchers incorporate an additional continuous latent variable. This variable represents the inferred probability of the data belonging to each of d distinct classes (e.g., d is the number of keys). The paper provides comprehensive mathematical definitions of the classifying model.

To generate sequential data the model is extended with a long short-term memory (LSTM).

The evaluation focuses on the key consistency of the generated music. A comparison of different models shows that the classifying VAE generally performs better than standard VAEs.

The researchers admit that they have made the simplifying assumption that the key of a particular musical sequence is fixed over the length of the sequence. In future, they want to include a possibility to predict and apply key changes.

2.3. Hierarchical Variational Autoencoder

In this paper [11] Adam Roberts, Jesse Engel and Douglas Eck from Google AI developed a recurrent VAE trained to reproduce short musical sequences. The resulting model is called “MusicVAE” and is part of the open-source project “Magenta” from Google AI. [1]

They focus their work on three models:

1. 2-bar “loops"
2. 32-bar lead melodies
3. 16-bar “trios" consisting of lead, drums and bass

It should be noted that only the latter is able to produce polyphonic music. But this model is by definition restricted to only three tracks. The individual tracks themselves are not polyphonic. Unlike other models, theirs is able to interpolate between two given samples. An example is provided with a visual representation in the paper.

The autoencoder uses a hierarchical decoder with LSTMs. For the three-track model each instrument has its own LSTM decoder.

The datasets for training were built from publicly-available MIDI files, with a $\frac{4}{4}$ time signature. The bars are quantized to 16 notes. In the appendix of the article, the researchers specify the exact model construction and all required training parameters.

The paper lacks comprehensive evaluation results. Only a few reconstruction accuracies are given. No detailed examination of the quality of the generated clips is provided.

The researchers conclude that their work will enable numerous innovative interactions for musical creativity. They also note that it should be possible to extend the work to model true polyphonic music.

2.4. WaveNet Autoencoder

The next paper [5], also from Google AI, Jesse Engel et. al describe an autoencoder based on WaveNet [14], a generative approach to probabilistic modeling of raw audio. The primary motivation for this approach is to achieve a consistent long-term structure without external conditioning. The researchers claim that it is possible to generate new types of expressive and realistic instrument sounds with their neural network model.

The paper introduces the existing WaveNet model with its current drawback: It needs external conditioning to achieve a long-term structure. The researchers remove this need by embedding the model in an autoencoder. They use a temporal encoder with non-causual dilated convolutions. The model takes a spectrogram as an input, which is encoded in a latent space. The WaveNet acts as a decoder and tries to recreate the original input. The paper provides basic training parameters and describes the exact structure of the model.

The evaluation of the model consists of three tasks:

1. note reconstruction
2. instrument interpolation
3. pitch interpolation

The paper lists a few instrument spectrograms for the reader to compare the original and the reconstruction.

The researchers conclude that their WaveNet autoencoder is a powerful representation for music. They show that their model is able to reconstruct a ten-second piece of polyphonic music, even though it was never trained on more than one note at a time or on clips longer than four seconds. As evidence, they subjectively compare the original and the reconstructed spectrogram.

One open problem are memory constraints, which make the model unable to fully capture global context.

2.5. Polyphonic SeqGAN

In their paper [8] Sang-gil Lee et. al want to extend an existing sequence generative adversarial network (SeqGAN) to incorporate polyphony to the previously monophonic outputs.

In the original work [15] the SeqGAN was proposed as a combination of a recurrent neural network (RNN) and a convolutional neural network (CNN). The RNNs act as a sequence generator and the CNNs as a discriminator that identifies whether a given sequence is a real sample or a generated one.

To use SeqGAN as a polyphonic music generator the researchers developed an efficient representation of a polyphonic MIDI file. They were able to simultaneously capture chords and melodies with dynamic timings. The basic idea behind this representation is to convert the MIDI file into a token sequence. A token combines the duration, the octave of the note, the pitch of the note, the octave of the chord and the pitch of the chord of a time step in a single integer. This sequence can be learned and generated by the SeqGAN.

The authors evaluated the model with the bilingual evaluation understudy (BLEU) score. This metric measures a similarity between the validation set and the generated samples, which is largely used to evaluate the quality of machine translation. In a user study they asked 42 participants to rate seven different sequences from 1 to 5, by responding to three questions: How pleasant is the song? How realistic is the sequence? How interesting is the song?

Despite the fairly good results, the paper points out a drawback caused by the nature of GANs. They often suffer from the mode collapsing problem, where the generator fools the discriminator by creating artifacts rather than realistic samples. This behavior also appeared with the SeqGAN, where generated samples played the same note constantly, which broke the musical coherence.

2.6. MuseGAN

In their paper [4] Hao-Wen Dong et. al propose a GAN-based model. The researchers claim they developed the first model that can generate multi-track, polyphonic music.

The paper focuses on the modeling of multi-track interdependency and the temporal structure. For M tracks the bar generator takes as input four different types of random vectors:

1. an inter-track time-independent random vector
2. an inter-track time-dependent random vector
3. M intra-track time-independent random vectors
4. M intra-track time-dependent random vectors

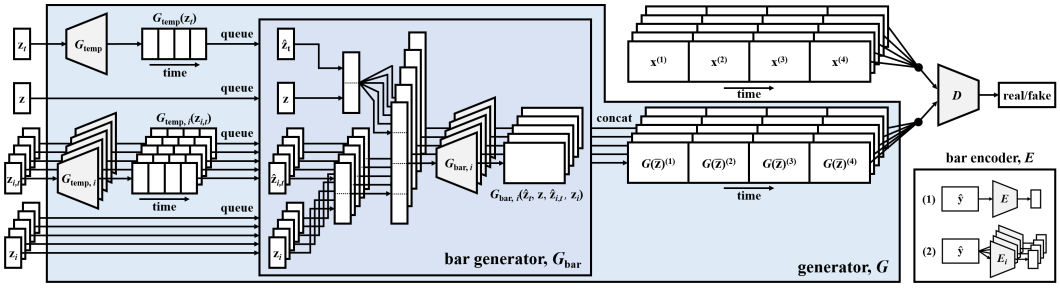


Figure 2.1.: System diagram of the proposed MuseGAN model [4][p.37]

All tracks have private intra-track vectors and share the inter-track vectors, The time-dependent vectors are fed into a temporal structure generator. The output is concatenated with the time-independent vectors and used as input for the M bar generators.

The authors evaluated the model with self-defined objective metrics:

EB ratio of empty bars (in %).

UPC number of used pitch classes per bar (from 0 to 12).

QN ratio of "qualified" notes (in %). The paper considers a note no shorter than three time steps (i.e. a 32th note) as a qualified note.

The results where completed with a user study with 144 participants. 44 of them are rated "pro user", according to a simple questionnaire probing their musical background. The probands rated nine generated clips in terms of whether they 1) have pleasant harmony, 2) have unified rhythm, 3) have clear musical structure, 4) are coherent, and 5) the overall rating.

The researchers conclude that their model is musically and aesthetically still behind the level of human musicians. But the proposed model has a few desirable properties and they hope follow-up research can further improve it.

2.7. Learning a Latent Space of Multitrack Measures

In their paper, Simon, Roberts, Raffel, Engel, Hawthorne and Eck from Google AI present a new way of generating polyphonic music [13]. They use a hierarchical VAE with a recurrent decoder. Their previous work with music generation [11] serves as starting position. The researchers extended their existing model with a new music representation. Like his predecessor, the "multitrack MusicVAE" is aslo part of the open-source project "Magenta". [12]

The model can handle up to 8 tracks with variable numbers of events per track. In contrast to the original system, the new one can play 128 different instruments, limited by the specification of MIDI 1.0. The restriction to a 4/4 time signature is the same as for the predecessor.

Since the model is based on an autoencoder, it is possible to perform these latent space operations:

sampling: generate a sequence with a random input vector

interpolation: synthesize two musical sequences in a semantically meaningful way compared to naively blending the notes together

attribute vector arithmetic: add and subtract attribute vectors (e.g. note density)

chord conditioning: change the harmony while keeping the instrumental choice and rhythmic pattern

The authors claim that their model is the first one capable of generating full multitrack polyphonic sequences with arbitrary instrumentation. They refer to their project page [12] where the reader can listen to some examples or create his own samples. Simon et al conclude that their work will enable numerous innovative interactions for musical creativity.

2.8. Conclusion

These papers are addressed to experts and people with existing knowledge. A basic knowledge of neural networks is required to understand them.

It can be said that there are mainly two approaches for generating polyphonic music: the construction of individual notes resulting in a MIDI file and the the generation of a spectrogram. The former approach is usually based on a recurrent network packed in a generative model like an autoencoder or a GAN. The latter is based on WaveNet, which handles spectrograms. It should be noted that the presented hierarchical VAE [11] is not a truly polyphonic model.

The examined models focus on different aspects: key consistency [6], track / instrument interdependency [4][5], interpolability [11] and note representation [8].

A problem that all of these papers have is that generated music cannot be thoroughly evaluated in practice. The researchers can show for every model the reconstruction accuracy, but the quality of newly generated music is partially based on subjective perception. Both GAN-based model where credibly evaluated with a user study.

The most promising of all examined models is the multitrack VAE from Google AI [13]. Unfortunately it is only able to create 2 second long samples. There are other multitrack models, which generate samples more than 10 seconds long. The biggest competitor is probably MuseGan [4]. Both models use a latent space shared across multiple tracks to handle interdependencies between instruments. However, in MuseGAN, the set of instruments is a fixed quintet comprising bass, drums, guitar, piano, and strings. Simon et al's model can handle any arbitrary instrument combination. In contrast to MuseGAN it allows latent space operations and can represent or manipulate preexisting music.

These points are the greatest strengths of multitrack VAE. The interactive examples provide a taste of what is possible with this model. This is yet another step forward towards a generator for truly polyphonic, multi-instrumental music which will be indistinguishable from man-made compositions. It will not take long before Google AI will expand the model, so that it can generate longer sequences and handle more than 8 instruments at a time.

3. Methods

3.1. Concept

The task of the thesis can be divided into two subproblems: extract useful information from an image and generate music from this information.

The multitrack MusicVAE is used as the basic model for music production, as it is the most promising of all examined models (see chapter 2.8).

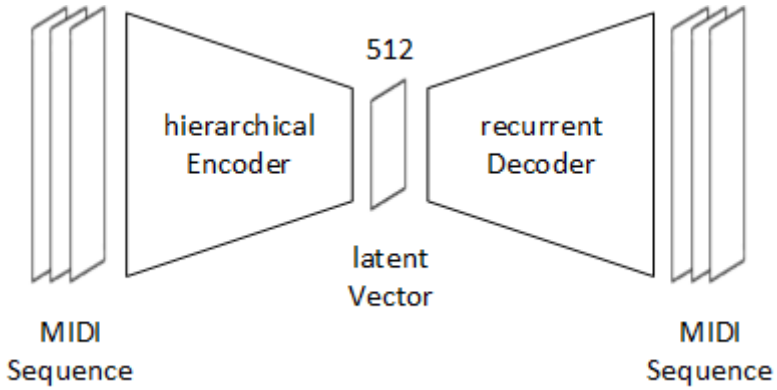


Figure 3.1.: Schematic view of the MusicVAE

Theoretically this model could be trained from scratch, but Google AI already offers pre-trained checkpoints for the different MusicVAE configurations [1]. It turns out that a pre-trained model can reproduce the data set compiled for this work very well (see chapter 3.2.1). Therefore further training of the MusicVAE model is not necessary.

Since MusicVAE generates music from a latent, 512-dimensional vector, an upstream model must generate a matching latent vector from a screenshot that corresponds to the desired music. Thus the task of the thesis can be reduced to a regression problem with 512 output variables. For these tasks a pre-trained CNN can be used, which generates a 512-dimensional vector using a customized fully connected network. The complete model is shown in figure 3.2.

[1] `download.magenta.tensorflow.org/models/music_vae/multitrack/*`

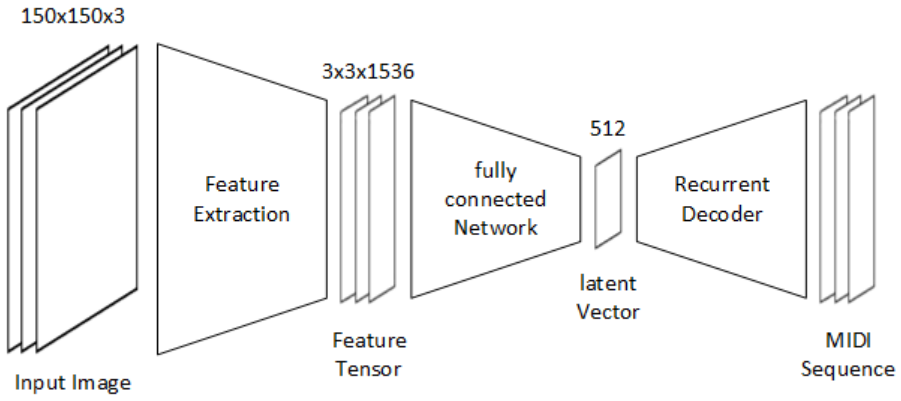


Figure 3.2.: Schematic view of the entire network

3.2. Training Data

To train the neural network, a data set must be created that maps a 512-dimensional vector to each input image. These vectors can be generated using the MusicVAE encoder. The acquisition and preparation of training data is explained below.

3.2.1. Data Acquisition

The model used requires MIDI files. In a first step, as many songs as possible were collected in MIDI format for the selected game "The Legend of Zelda: Ocarina of Time". Two providers were identified, each offering a collection of tens of thousands of MIDI files for many video games on different platforms:

- KHInsider [7]
- VGMusic [9]

This thesis focuses only on one game, but in the future much more powerful models could be trained using the existing data of these providers.

The source for the screenshots is a letsplay, taken from the youtuber ZorZelda [16], which provides about 10 hours of video material of the whole game. When choosing the letsplay it was made sure that there are no additional overlays on the video and that the gameplay is straight forward, so that no stationary images are created.

3.2.2. Data Processing

For the data set, the frames from the video must be mapped to the respective latent vector of the corresponding music. The rough process is shown in figure 3.3.

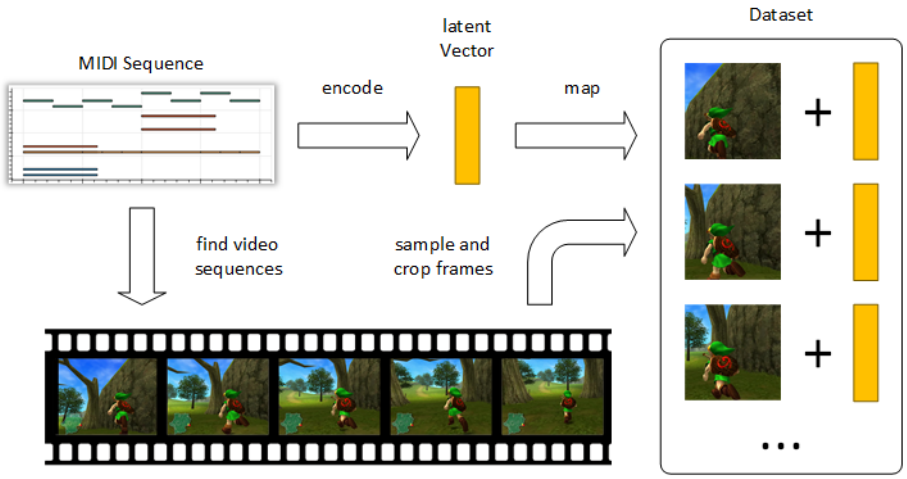
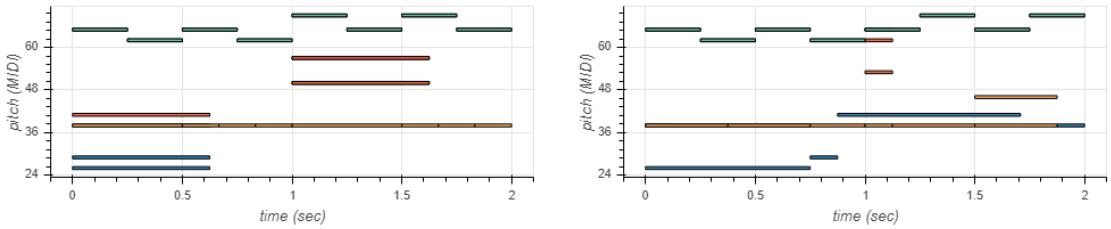


Figure 3.3.: Schematic view of the dataset creation

At first all MIDI files must be divided into bars, because the MusicVAE processes only single bars. The Magenta framework of Google AI offers a suitable function for the MusicVAE. In this process some songs are sorted out that MusicVAE cannot handle. From all parsed bars one per song must be selected manually. Normally the first bar of the chorus or the most recognizable melody is always selected.

A total of 35 songs were prepared in this way. Afterwards the latent vector was generated for each song with the encoder of the multitrack MusicVAE. For comparison purposes each vector was decoded again. A list of all songs with an sample frame can be found in the appendix under chapter A.2.

The reconstructed songs were compared with the original ones to make sure that no too big losses occurred. No negative outliers were found. Figure 3.4 shows an example of the comparison of the original and the reconstruction of a song.



(a) original (b) reconstruction

Figure 3.4.: Reconstruction of the song “Bolero of Fire”

To ensure the multivariate normal distribution of the latent space, the distribution of all $35*512 = 17'920$ latent variables of the latent vectors was calculated (see figure 3.5).

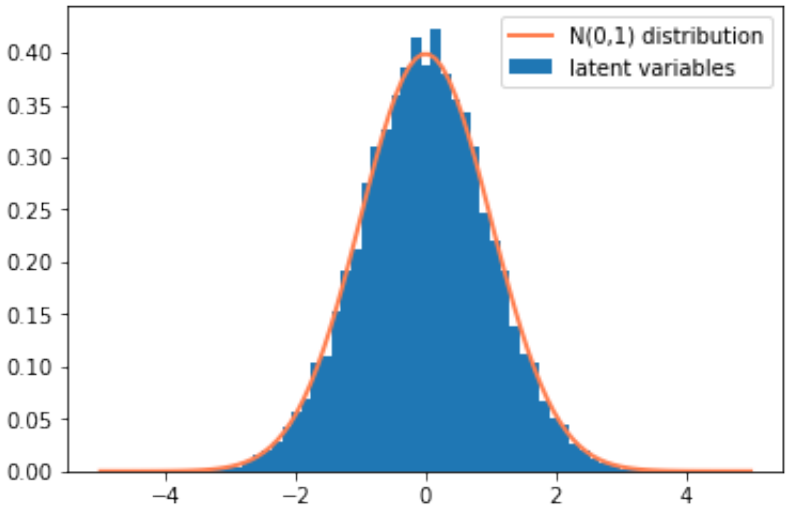


Figure 3.5.: Distribution of latent variables

The videos of the letsplay must be cut in parts and assigned to the corresponding songs. For each song frames are extracted from the segments. It is made sure that each song is assigned at least 500 and at most 5000 frames. Due to the length of the segments, the frequency of the extracted frames varies between 3.5fps and 25fps (thus each frame). In total nearly 50000 frames were extracted.

Each frame is cut out rectangularly to remove the graphic overlays (see figure 3.6). For the training all frames are scaled to 150x150 pixels.

(a) original frame with cropping indication

(b) cropped frame

Figure 3.6.: Rectangular frame cropping

Each frame is mapped with the corresponding latent vector. After shuffling, the complete data set is divided into training, validation and test set at a ratio of 80/10/10.

3.3. Models

In this chapter the previously outlined model is defined more precisely. The focus lies on the first half of the model, shown in figure 3.7.

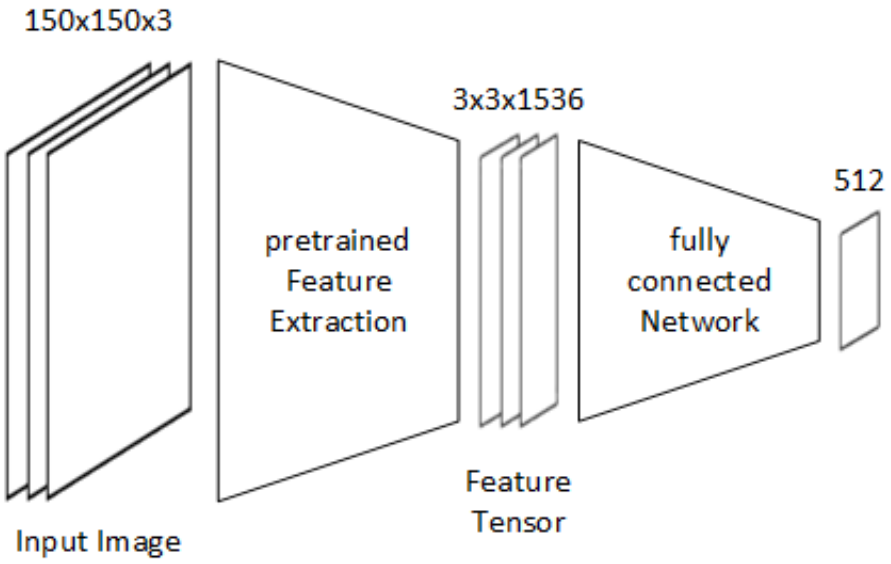


Figure 3.7.: Schematic view of the base network

As feature extractor the InceptionResNetV2 [2] is used. It is the best performing model provided by the Keras API. With an input tensor of (150x150x3) the feature extractor provides a feature tensor of (3x3x1536).

In the following five different network topologies are listed, which map the feature tensor to the (512x1x1) latent vector.

3.3.1. Type 1

Type 1 is the simplest model. It directly connects the flattened tensor with the output layer.

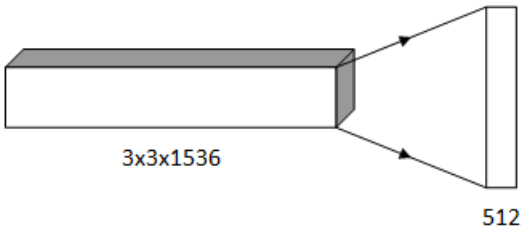


Figure 3.8.: Model Type 1

3.3.2. Type 2

Type 2 extends the Type 1 model by a hidden layer. Due to the size of the input tensor the number of parameters is enormous.

[2] https://keras.io/applications/#inceptionresnetv2

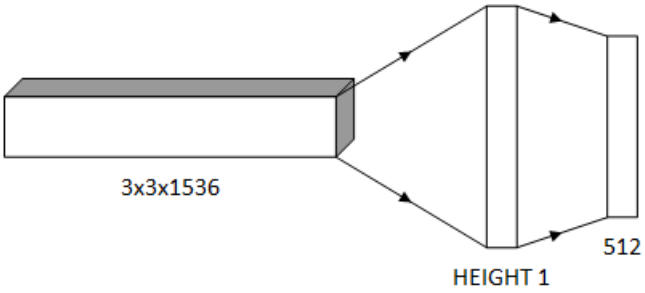


Figure 3.9.: Model Type 2

3.3.3. Type 3

Type 3 extends the Type 1 model by a convolutional layer. A 1x1 convolution is performed to reduce the input tensor and thus the number of parameters.

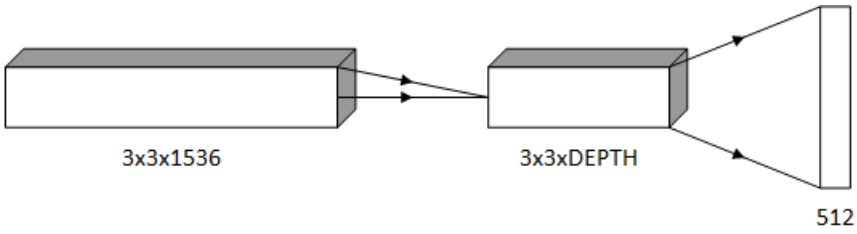


Figure 3.10.: Model Type 3

3.3.4. Type 4

Type 4 extends the Type 3 model by a hidden layer. In comparison to Type 2 it has much less parameters thanks to the 1x1 convolution.

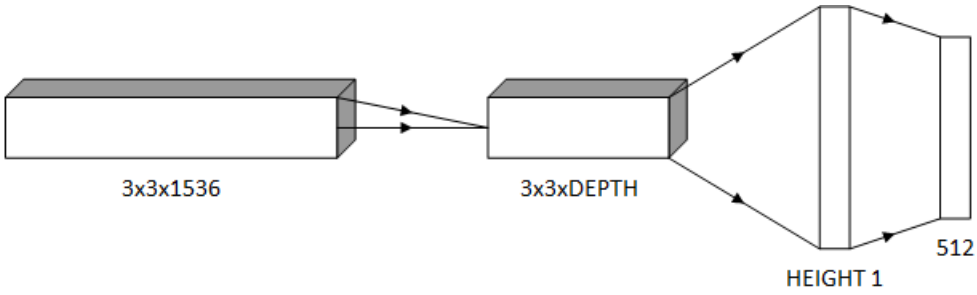


Figure 3.11.: Model Type 4

3.3.5. Type 5

Type 5 extends the Type 4 model by a further hidden layer and is therefore the deepest of all topologies presented.

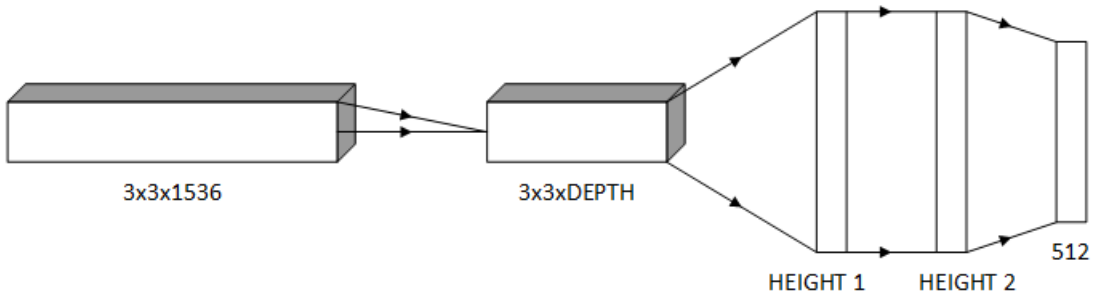


Figure 3.12.: Model Type 5

3.3.6. Configurations

The presented model types can be parameterized differently. The table 3.1 shows the respective parameters.

	Input Layer	Convolutional Layer	Dense Layer	Output Layer
Type 1	3x3x1536	-	-	512
Type 2	3x3x1536	-	HEIGHT 1	512
Type 3	3x3x1536	3x3xDEPTH	-	512
Type 4	3x3x1536	3x3xDEPTH	HEIGHT 1	512
Type 5	3x3x1536	3x3xDEPTH	HEIGHT 1, HEIGHT 2	512

Table 3.1.: Architecture of all base models

Based on the basic types, 11 different configurations were defined. The table 3.2 lists all configurations with the corresponding parameters. Figure 3.13 shows a code snippet with the definition of the models in python.

All hidden layers are assigned the tangent hyperbolicus as activation function. The output layer must be activated linearly, so that an N(0,1) normal distribution of the output variables is possible.

```
SHAPE = (IMG_WIDTH,IMG_HEIGHT,IMG_CHANNEL)
conv_base = InceptionResNetV2(include_top=False, weights='imagenet',
    input_shape=SHAPE)
conv_base.trainable = False

model = tf.keras.Sequential()
model.add(conv_base)

#Convolutional Layer [OPTIONAL]
model.add(layers.Conv2D(DEPTH, (1,1), activation='tanh'))

#Flatten
model.add(layers.Flatten())

#Hidden Layer [OPTIONAL]
model.add(layers.Dense(HEIGHT1, activation='tanh'))
model.add(layers.Dense(HEIGHT2, activation='tanh'))

#Output vector
model.add(layers.Dense(512, activation='linear'))

#Configure a model for mean-squared error regression.
model.compile(optimizer=tf.train.AdamOptimizer(LEARNIG_RATE), loss='mse')
```

Figure 3.13.: Definition of the models in python using Keras

Name	Base Model	DEPTH	HEIGHT 1	HEIGHT 2	# Params
0	Type 1	-	-	-	7'078'400
1024	Type 2	-	1024	-	14'681'600
2048	Type 2	-	2048	-	29'362'688
60c	Type 3	60	-	-	369'212
100c	Type 3	100	-	-	615'012
200c	Type 3	200	-	-	1'229'512
60c-512	Type 4	60	512	-	631'868
100c-768	Type 4	100	768	-	1'239'396
200c-1024	Type 4	200	1024	-	2'676'424
60c-512-512	Type 5	60	512	512	894'524
100c-768-768	Type 5	100	768	768	1'829'988

Table 3.2.: Configuration of all trained models

A list of all training runs with the specified learning rate and the best validation error and associated test error can be found in the appendix under chapter A.1.

4. Results

4.1. Training

As expected, the larger models achieve a lower validation error. However, no model can achieve a validation error below 0.15. The test error is only slightly higher for each configuration. A list of all training runs with the specified learning rate and the best validation error and associated test error can be found in the appendix under chapter A.1.

The relationship between model size and performance (measured with the test error) is shown in figure 4.1.

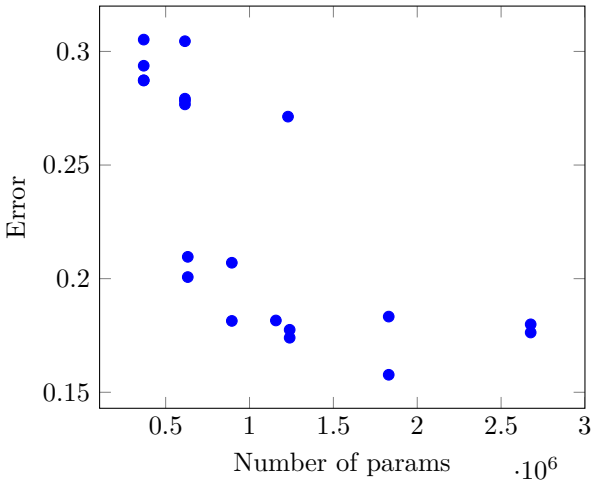
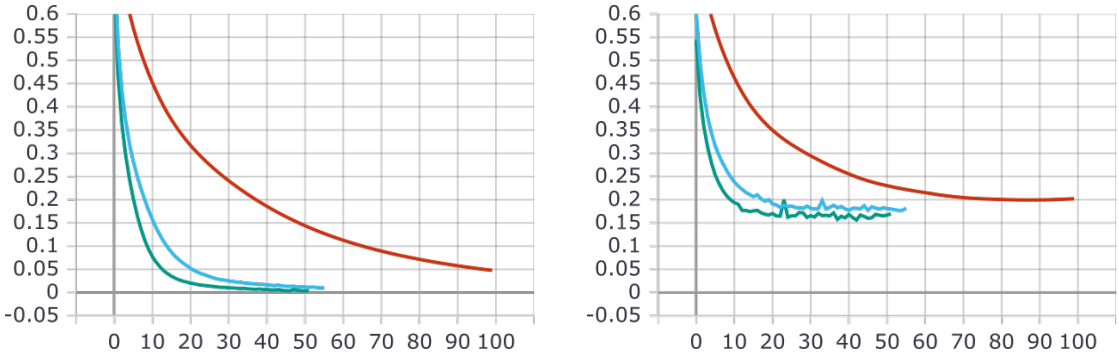


Figure 4.1.: Test error of all trained models

As can be seen in figure 4.2, all configurations tend to overfit strongly. For the larger models, the train error can be reduced almost to 0, while the validation error never falls below 0.15. The best performing models shown in figure 4.2 are listed in table 4.1.



(a) Train error

(b) Validation error

Figure 4.2.: Train and Validation Error of the best performing models

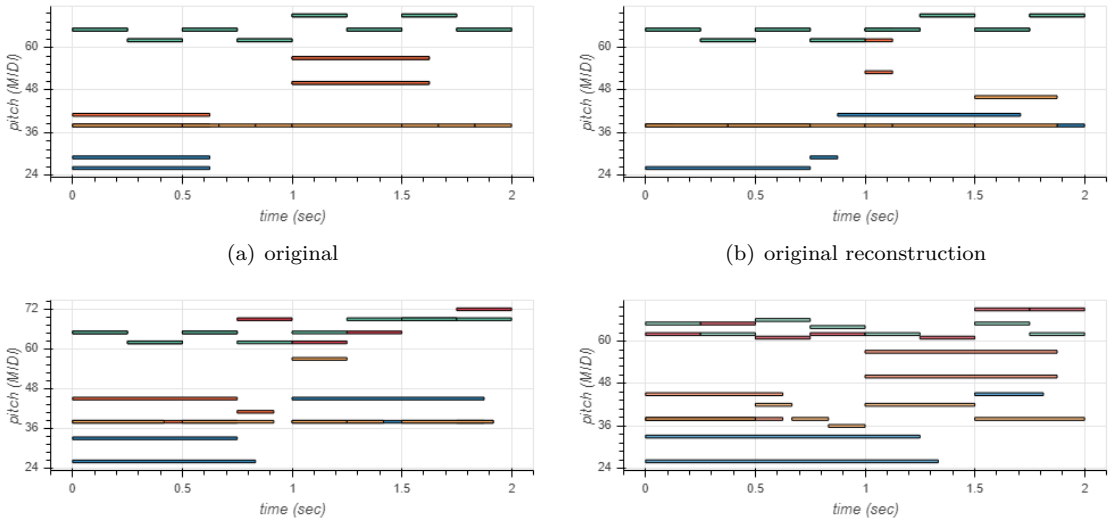
Color	Config Name	learning rate (LR)	# Params	Val Error	Test Error
Red	60c-512	0.00003	631868	0.1994	0.2007
Blue	60c-512-512	0.0001	894524	0.1771	0.1814
Green	100c-768-768	0.0001	1829988	0.1562	0.1577

Table 4.1.: Best performing models

For the following analysis the model 60c-512 is used, which reaches a test error of 0.2007 with 631'868 parameters. It is not much worse than the model with the peak value of 0.1577, but massively smaller.

4.2. Song Reconstruction

In order to qualitatively test the performance of the model 60c-512, individual frames for various songs were taken from the test set and converted from the model to a melody. The figures 4.3 and 4.4 show the reconstruction of one song each. Here the original melody, the reconstruction by MusicVAE, the prediction with a train set frame and the prediction with a test set frame are compared. The song "Bolero of Fire" has a train error of 0.0097 and a test error of 0.1737. The song "Market" has a train error of 0.0059 and a test error of 0.0801 and is therefore the best performing song (see chapter 4.3). Two further reconstructions can be found in the appendix under chapter B.1.



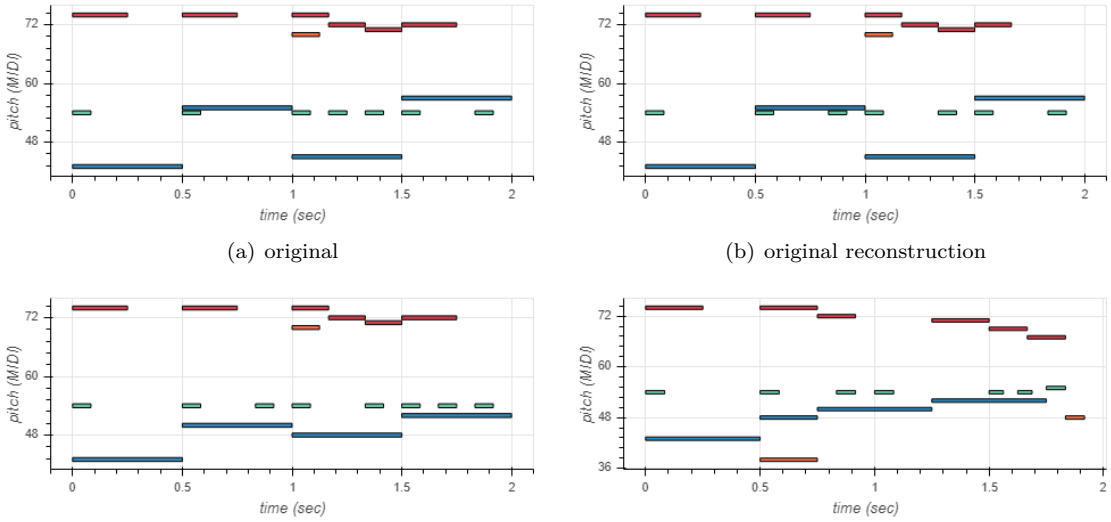
(a) original

(b) original reconstruction

(c) prediction reconstruction, train set

(d) prediction reconstruction, test set

Figure 4.3.: Reconstruction of the song "Bolero of Fire"



(a) original

(b) original reconstruction

(c) prediction reconstruction, train set

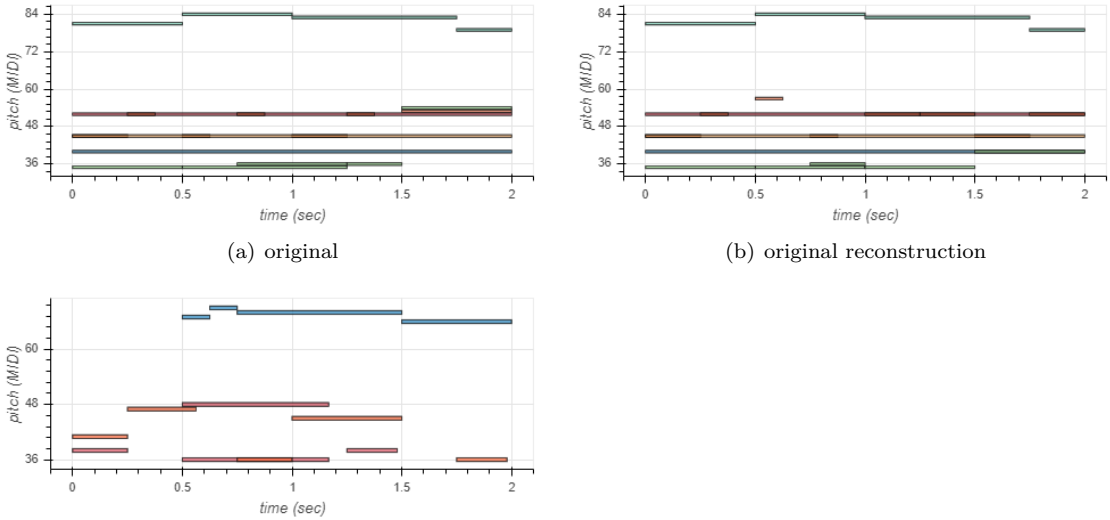
(d) prediction reconstruction, test set

Figure 4.4.: Reconstruction of the song “Market”

To determine the extent of the overfitting, a screenshot from another game was tested. This is “The Legend of Zelda: Majoras Mask”, the direct successor to “The Legend of Zelda: Ocarina of Time”. The frame was taken from level “Stone Tower Temple” (see figure 4.5). The figure 4.6 shows that the prediction has no similarity at all to the original.

Figure 4.5.: Test frame from “The Legend of Zelda: Majoras Mask”

Due to the overfitting the model fails miserably on this song as expected. The test error is 1.0508.



(a) original

(b) original reconstruction

(c) prediction reconstruction

Figure 4.6.: Reconstruction of the song “Stone Tower Temple”

4.3. Song Comparison

In the following all songs are compared with each other. The figure 4.7 shows the test error for each song. It can be seen that the songs are sometimes very different. The test error varies between 0.04 and 0.6. The high values can be explained by the fact that the frames of the corresponding songs are harder to distinguish from the others or generally have less distinctive features.

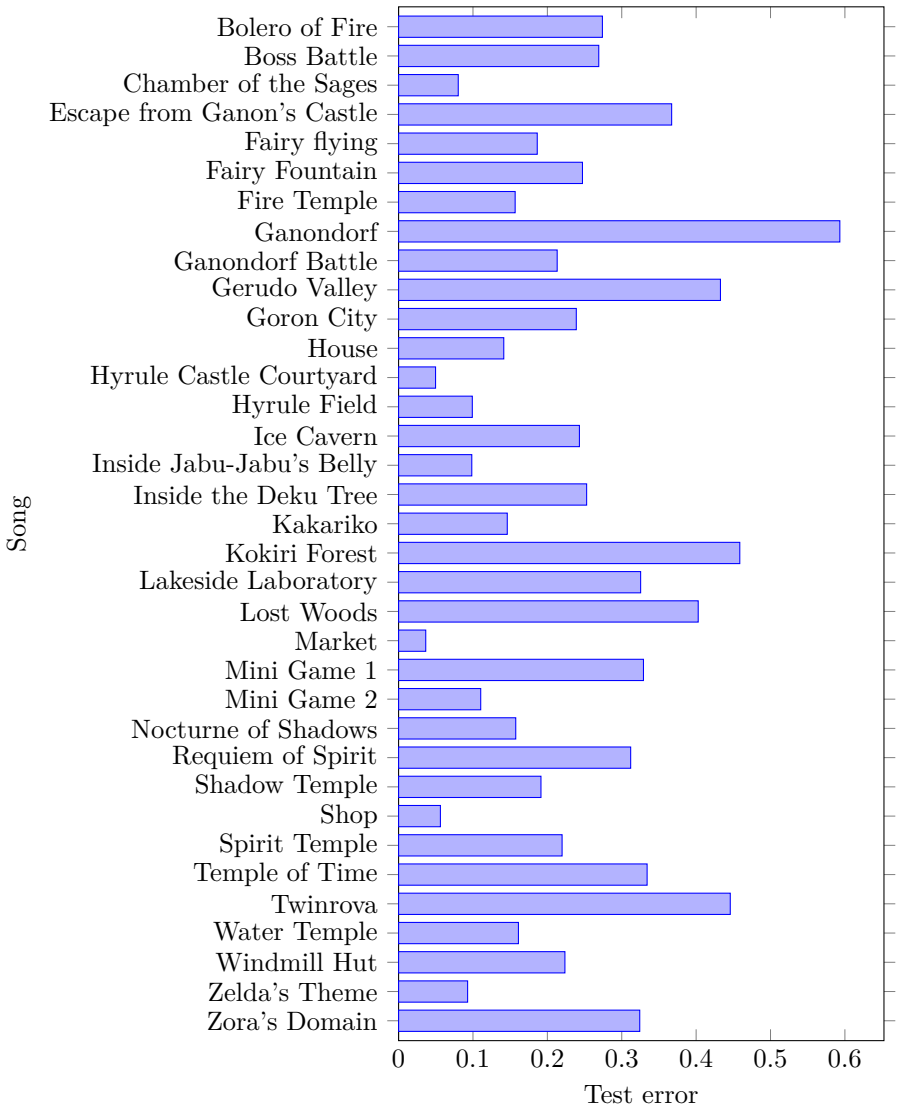


Figure 4.7.: Test error of songs

5. Discussion

5.1. Evaluation

The results from chapter 4.7 show that the model is basically able to assign the appropriate background music to a model. However, the performance of the whole model is limited by the decoder of the MusicVAE, which itself is not able to reconstruct all songs without loss.

Due to the strong overfitting (see chapter 4.1) the performance in the validation and test set differs from the training set. Predicting melodies outside of the trained data set is not possible in the current state.

Of all models tested, those with 1-2 hidden layers perform best. enlarging the layers brings only minimal improvement, while the number of parameters increases significantly.

5.2. Problems

A melody cannot always be clearly assigned to an image. Different could songs correspond to very similar screenshots. This way the songs for levels with bright colors and clear unique characteristics can be predicted much more accurately.

Another problem is currently the strong overfitting. The reason for this is a too small data set and a correspondingly too large model.

Due to the limitations of the MusicVAE, the output of the model is currently still restricted to only one bar. Theoretically, the MusicVAE can be extended to several bars, but it must first be trained from scratch with a large data set.

5.3. Future Work

In order to increase the performance of the presented model, it will be possible in the future to start at different positions.

- optimization of the network architecture through regularization and adaptation of the topology
- extending the data set to multiple video games (see chapter 3.2.1)
- extending the data set through data augmentation
- improve the performance of the VAE by training it with songs from the data set

As a further aspect, it could also be investigated in the future to what extent graphical overlays on screenshots have an impact on the performance.

Bibliography

[1] Google AI. Magenta: Musicvae.
`https://magenta.tensorflow.org/music-vae`, 2019. [online; accessed on 23.10.2019].

[2] MIDI Manufacturers Association. Standard midi files (smf) specification.
`https://www.midi.org/specifications-old/item/standard-midi-files-smf`, 2019. [online; accessed on 15.01.2020].

[3] Gwenaelle Cunha Sergio, Rammohan Mallipeddi, Jun-Su Kang, and Minho Lee. Generating music from an image. In *HAI*, 2015.

[4] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. 2018.

[5] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. 2017.

[6] Jay A Hennig, Akash Umakantha, and Ryan C. Williamson. A classifying variational autoencoder with application to polyphonic music generation. *ArXiv*, abs/1711.07050, 2017.

[7] KHInsider. Video game midi files.
`https://www.khinsider.com/midi`, 2019. [online; accessed on 15.08.2019].

[8] Sang-gil Lee, Uiwon Hwang, Seonwoo Min, and Sungroh Yoon. A seqgan for polyphonic music generation. *CoRR*, abs/1710.11418, 2017.

[9] Mike Newman. Video game music archive.
`https://www.vgmusic.com`, 2019. [online; accessed on 15.08.2019].

[10] George Papadopoulos and Geraint Wiggins. Ai methods for algorithmic composition: A survey, a critical view and future prospects. pages 110–117, 1999.

[11] Adam Roberts, Jesse Engel, and Douglas Eck. Hierarchical variational autoencoders for music. 2017.

[12] Ian Simon, Adam Roberts, Colin Raffel, Jesse Engel, and Curtis Hawthorne. Multitrack musicvae: Interactively exploring musical styles.
`https://magenta.tensorflow.org/multitrack`, 2019. [online; accessed on 05.06.2019].

[13] Ian Simon, Adam Roberts, Colin Raffel, Jesse Engel, Curtis Hawthorne, and Douglas Eck. Learning a latent space of multitrack measures. *ArXiv*, abs/1806.00195, 2018.

[14] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[15] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. *CoRR*, abs/1609.05473, 2016.

[16] ZorZelda. Youtube channel.
`https://www.youtube.com/user/zorzxx`, 2019. [online; accessed on 20.11.2019].

List of Figures

1.1. Logo of “The Legend of Zelda: Ocarina of Time” 2

2.1. System diagram of the proposed MuseGAN model [4][p.37] 7

3.1. Schematic view of the MusicVAE 9
3.2. Schematic view of the entire network 10
3.3. Schematic view of the dataset creation 11
3.4. Reconstruction of the song “Bolero of Fire” 11
3.5. Distribution of latent variables 12
3.6. Rectangular frame cropping 12
3.7. Schematic view of the base network 13
3.8. Model Type 1 13
3.9. Model Type 2 14
3.10. Model Type 3 14
3.11. Model Type 4 14
3.12. Model Type 5 15
3.13. Definition of the models in python using Keras 16

4.1. Test error of all trained models 17
4.2. Train and Validation Error of the best performing models 17
4.3. Reconstruction of the song “Bolero of Fire” 18
4.4. Reconstruction of the song “Market” 19
4.5. Test frame from “The Legend of Zelda: Majoras Mask” 19
4.6. Reconstruction of the song “Stone Tower Temple” 20
4.7. Test error of songs 21

A.1. Sample frame of each song 28

B.1. Reconstruction of the song “Temple of Time” 29
B.2. Reconstruction of the song “Windmill Hut” 29

List of Tables

3.1. Architecture of all base models 15
3.2. Configuration of all trained models 16

4.1. Best performing models 18

A.1. All training configurations 27

Part II.

Appendix

A. Training

A.1. Training Configurations

Config Name	LR	# Params	Val Error	Test Error
60c	0.0001	369212	0.294	0.2937
60c	0.001	369212	0.3034	0.3052
60c	0.0003	369212	0.2864	0.2872
60c	0.0002	369212	0.2849	0.2873
100c	0.0002	615012	0.276	0.2783
100c	0.001	615012	0.3028	0.3045
100c	0.0001	615012	0.2733	0.2767
100c	0.0003	615012	0.2759	0.2792
200c	0.0001	1229512	0.2701	0.2713
60c-512	0.0001	631868	0.2081	0.2096
60c-512	0.00003	631868	0.1994	0.2007
60c-512-512	0.0001	894524	0.1771	0.1814
60c-512-512	0.00003	894524	0.1984	0.207
1024	0.0001	14681600	0.1549	0.1547
1024	0.00003	14681600	0.167	0.1673
200c-1024	0.0001	2676424	0.1721	0.1763
200c-1024	0.00003	2676424	0.1755	0.1799
2048	0.0001	29362688	0.1496	0.1497
100c-768	0.0001	1239396	0.1765	0.174
100c-768	0.00003	1239396	0.1739	0.1775
100c-768-768	0.0001	1829988	0.1562	0.1577
100c-768-768	0.00003	1829988	0.1727	0.1833
60c-512-512-512	0.0001	1157180	0.1736	0.1816

Table A.1.: All training configurations

A.2. Training Data Samples


(a) Bolero of Fire


(b) Boss Battle


(c) Chamber of Sages


(d) Escape from Ganon's Castle


(e) Fairy flying


(f) FairyFountain


(g) Fire Temple


(h) Ganondorf


(i) Ganondorf Battle


(j) GerudoValley


(k) Goron City


(l) House


(m) Hyrule Castle Courtyard


(n) Hyrule Field


(o) Ice Cavern


(p) Inside Jabu-Jabu's Belly


(q) Inside the Deku Tree


(r) Kakariko


(s) Kokiri Forest


(t) Lakeside Laboratory


(u) Lost Woods


(v) Market


(w) Mini Game 1


(x) Mini Game 2


(y) Nocturne of Shadows


(z) Requiem of Spirit


(aa) Shadow Temple


(ab) Shop


(ac) Spirit Temple


(ad) Temple of Time


(ae) Twinrova


(af) Water Temple


(ag) Windmill Hut

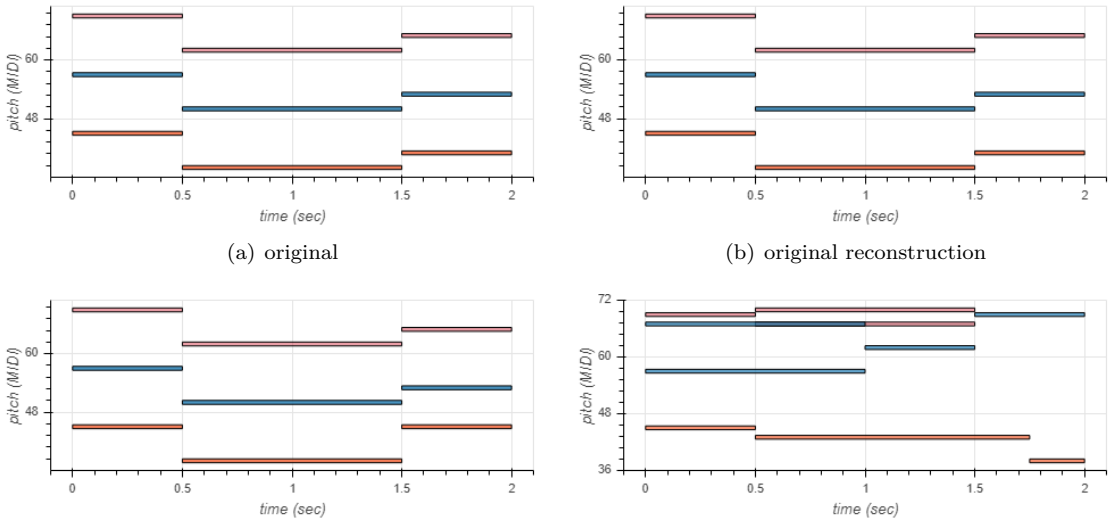

(ah) Zelda's Theme


(ai) Zora's Domain

Figure A.1.: Sample frame of each song

B. Result

B.1. Song Reconstruction



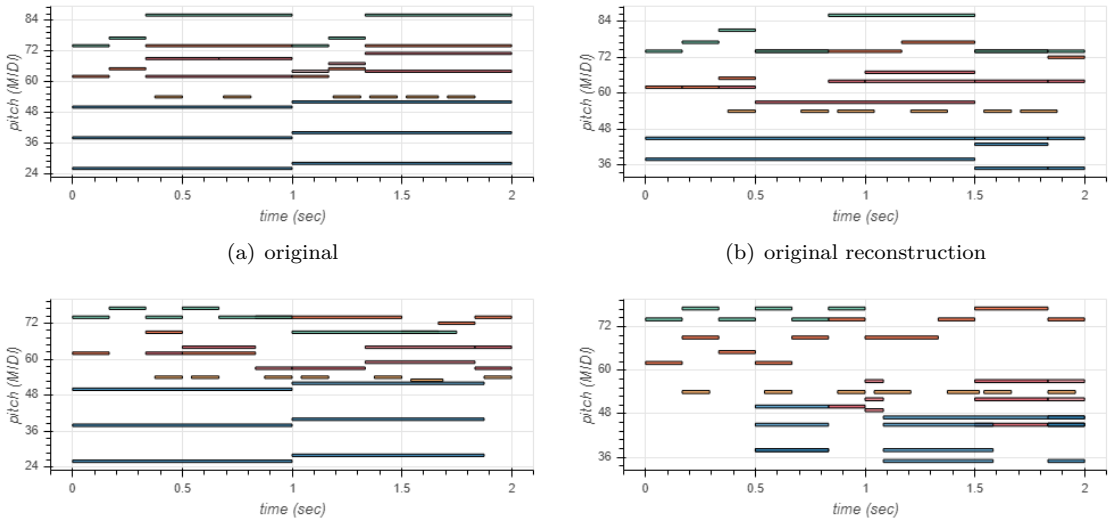
(a) original

(b) original reconstruction

(c) prediction reconstruction, train set

(d) prediction reconstruction, test set

Figure B.1.: Reconstruction of the song “Temple of Time”



(a) original

(b) original reconstruction

(c) prediction reconstruction, train set

(d) prediction reconstruction, test set

Figure B.2.: Reconstruction of the song “Windmill Hut”