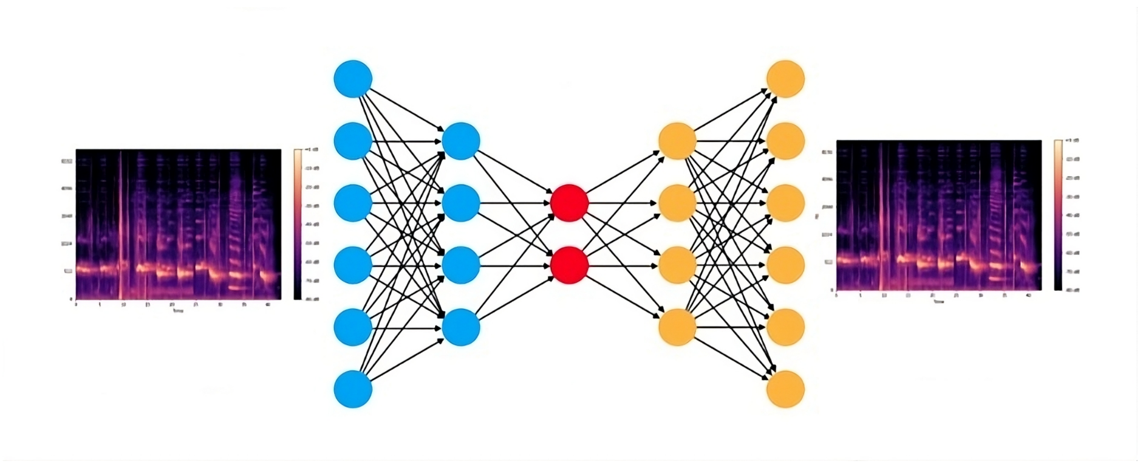
Kantonsschule Im Lee

Maturitätsarbeit HS 2024/25



Generation of Audio Effects using Deep Learning Algorithms

Author: Robert Paun

Class: 4a

Supervising teacher: Cyril Wendl

Winterthur, January 6th 2025

Image from [26]

Contents

Chapter 1

Introduction

1.1 Background and Motivation

With the current rapid advancement of **Artificial Intelligence** (AI) through tools like ChatGPT, Claude, DALL-E, and many others, the appeal of using **AI** to solve simple everyday problems, for greater convenience, has become increasingly compelling. In my matura thesis, I will specifically target **Deep Learning**, a subfield of **Artificial Intelligence**, in the field of audio synthesis, as it connects two of my passions, namely music-production and computer science and holds significant potential for driving innovation in the music production industry.
In contrast to the extensive research and application of **AI** language models (e.g. ChatGPT, Claude) the exploration of **generative AI** in music is a relatively under-researched field.

1.2 Goals and Objective

The overall goal of this thesis, as the title suggests, is the generation of audio effects using deep learning algorithms implemented in Python using a **Variational Autoencoder** (VAE), but more to that in Chapter 2. An additional aim is to possibly contribute to the yet under-researched area of **generative AI** in music, while establishing a bridge between my passions for music production and computer science. Furthermore, a significant focus lies on enhancing my personal understanding, skills and knowledge in **Deep Learning** and **AI** development.
The objectives can be divided into three subcategories: *Research Objectives, Technical Objectives and Evaluation Objectives*:
- **Research Objectives:**
Understanding the current limitations of traditional audio synthesis, in contrast to methods using Artificial Intelligence, such as the Deep Learning algorithm used for this thesis and studying the role of a **Variational Autoencoder** (VAE) in generative audio synthesis, as well as other possible approaches.
- **Technical Objectives:**
Implementing a Deep Learning model as a **VAE** for the generation of audio and fine-tuning the model for practical use in music production.
- **Evaluation Objectives:**
Comparing generated sounds with traditional methods and assessing the usability and quality from a music producer's perspective.

Chapter 2

Methodology

2.1 Introduction to the Methodology

The following chapter will outline the basics of the methodological approach of developing a **Convolutional Neural Network** (or CNN for short) capable of generating audio samples, such as simple percussion sounds like, for example, a snare. This includes exploring possible techniques for generating audio, a comprehensive explanation of the chosen method, and the reasoning behind key decisions. The main objective of this project is to design a deep learning model, more precisely a so-called **Variational Autoencoder**, capable of generating novel but yet, at the same time, familiar audio effects, based on a custom dataset provided as input. Training the model on specific categories of sound should enable the model to create new audio outputs that fall into the same category as that of the input data with every execution while managing to preserve sufficient diversity in each of the generated sounds at the same time.
Audio data is inherently complex, as it is produced by the vibration of objects that cause an oscillation of air molecules in the surrounding area. These oscillations can be described by the *frequency* and *amplitude*. For this thesis, these structures needed to be broken down into a more simplified form that the neural network can learn from. This is where techniques such as **Fourier Transform** (FT), precisely **Short-Time Fourier Transform** (STFT), come in handy.

2.2 Overview of Possible Approaches

When it comes to generating audio using deep learning, there are several possibilities available, each with their own advantages and disadvantages. The three main approaches considered for this project are the following: Generative Adversarial Networks (GANs), Recurrent Neural Networks (RNNs) and Autoencoders (AEs).

Generative Adversarial Networks are composed of two networks: a *generator*, which, as the name suggests, creates new data and a **discriminator** evaluating the data based on how realistic it is. The adversarial process then trains both networks, which generates highly realistic outputs. However, GANs are known for being difficult to train, due to *mode collapses*, that refers to a lack of diversity in generative samples [14], in which only a small range of outputs will be produced, and *instability*, referring to the sensitivity of a neural network or **AI** in general to minimal changes in input or initial conditions, leading to the model getting *unreliable* and *unpredictable* [20]. With the goal in mind of generating new and especially unique data rather than highly realistic data, GANs were not selected.

Recurrent Neural Networks are generally well-suited for tasks involving sequential data, like speech and natural language processing, as they effectively capture temporal dependencies, making them a strong candidate for audio synthesis. However, for this project, the complexity of RNNs is unnecessary, because the desired output should be individual audio effects rather than continuous sequences of audio. Furthermore, training RNNs also turns out to be challenging, due to *optimization difficulties*, which are inherent complexities within an **AI** model, that hinder the training process. The challenge lies within fine-tuning parameters for complex objective functions to be as efficient as possible.[1] .

The third option is **Autoencoders**, simpler models designed to learn a compressed, efficient representation of the data prior to reconstructing. To generate audio one can use vanilla autoencoders, although they lack the ability to introduce variations to the outputs. This limitation makes them less suitable for this project. A subset of the autoencoder, the so-called **Variational Autoencoder** (VAE), was selected instead.

Variational Autoencoders (VAEs) offer a significant advantage for this project due to the introduction of a probabilistic element to the latent space, diversifying the outputs, in contrast to simple reconstruction processes found in the mentioned approaches above. Moreover, the **VAE**s combination with a Convolutional Neural Network enables the model to handle spectrogram data, whose importance will be explained further in Section 2.4.

2.3 Data Preprocessing and Dataset Creation

Audio data is represented in the time domain as a waveform, where the amplitude changes over time. In order to extract meaningful data, the audio needs to be transformed into the frequency domain, as it is more intuitive and allows the analysis of its frequency components, using the **Fourier Transform** (FT). The **FT** breaks down complex sounds into a sum of *sine waves* oscillating at different frequencies, providing a clearer picture of all the frequency components within a sound. However, a major limitation of a standard **FT** is that it removes all temporal information, making it unsuitable for audio tasks where exactly that temporal information is crucial.

To address this, the **Short-Time Fourier Transform** (STFT) was used. The **STFT** divides the audio into short intervals and then performs multiple Fourier Transforms at different points in time, as shown in Figure 2.1 and Listing A.2. Using this method, both time and frequency information are retained, which allows capturing of the frequency changes over time. The result of this procedure is a **spectrogram**, a 2D representation of audio that maps time, frequency, and magnitude.

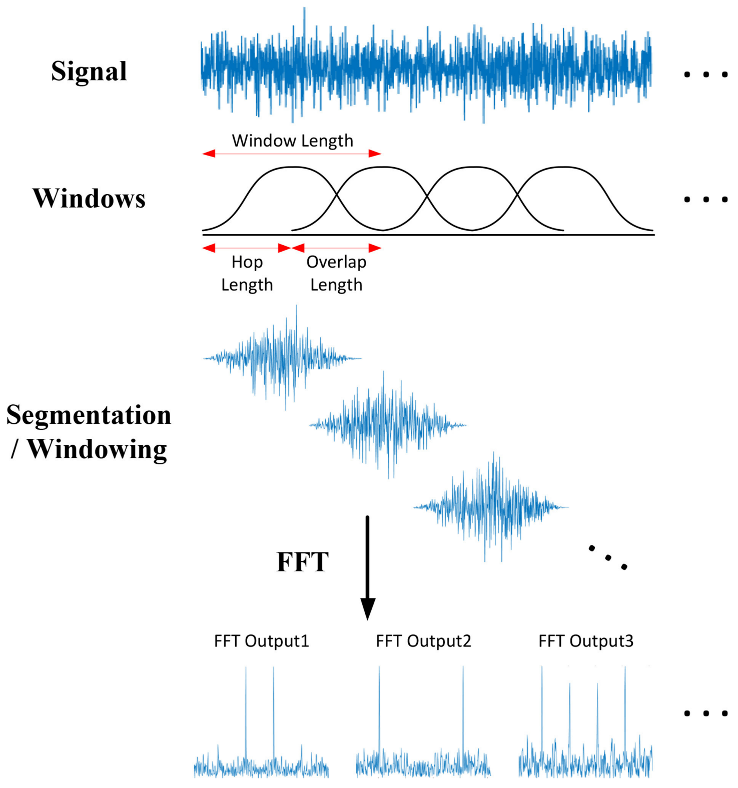


Figure 2.1: Short-Time Fourier Transform Steps [10]

Using the **Librosa** [24] library, in combination with **Keras** [22] and **Tensorflow** [25], widely used tools for audio and music analysis in Python, the initial raw audio data was converted into **Mel-spectrograms** (as in Figure 2.2), essentially spectrograms, in which all frequencies are *logarithmically scaled* to imitate the sensitivity to lower frequencies of humans ears [2].
The dataset (accessible at A) for this project was created by collecting audio files, in the basic .wav format, that can be classified into a specific category (e.g. snares, hi-hats), stemming from various soundkits of different music producers in the producing community. Before the effective conversion of the audio file into a **Mel-spectrogram**, the signal, currently stored as an *array*, a data structure consisting of a collection of elements of the same type [17], in some cases has to be *padded*. **Padding** signifies the process of augmenting the dimensions by, in this case **"zero-padding"**, appending zeros to either the left (left pad) or to the right (right pad) of the *array*, ensuring that all arrays are aligned prior to further processing [18]. These raw audio files were later transformed into **Mel-spectrogram** representations, produced by the preprocessing script as can be seen in A.2, providing a time-frequency visualization suitable for training the model.
Prior to the initiation of the training process a so-called **MinMaxNormaliser** is applied to the arrays, rescaling all data values between 0 and 1 to, as the name suggests, normalize the data values. These new *min and max values* are subsequently saved for the purpose of accurately reconstructing the input data later on [21].

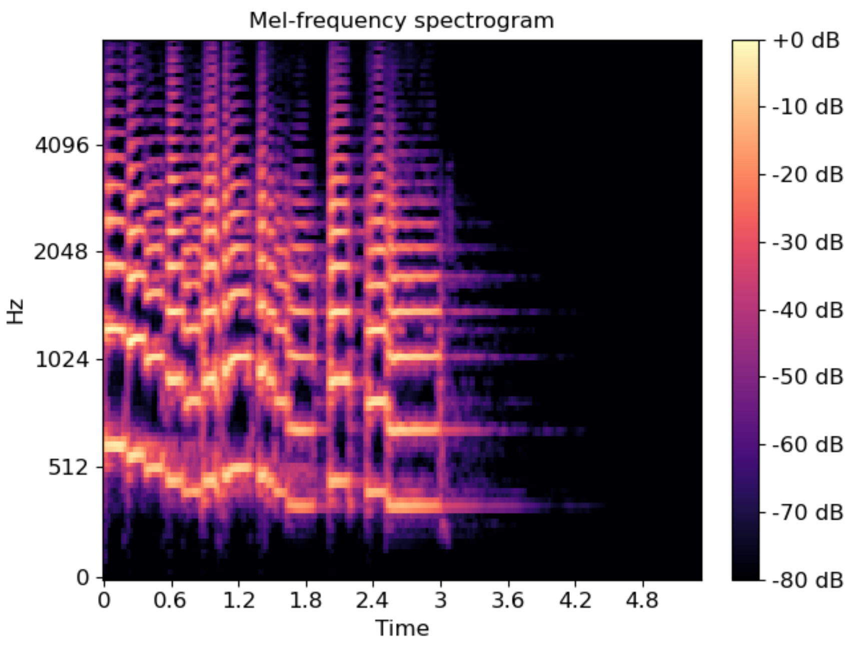


Figure 2.2: Mel-Spectrogram [23]

2.4 Model Architecture

At the heart of this architecture lies the principle of **deep learning** [7], where multiple layers of a neural network cooperate to process data. Each layer learns increasingly abstract and complex features, allowing the model to capture high-level patterns as well as more close-up details.

This architecture is based on the idea that each layer of the network is activated in response to specific properties of the input data. Lower layers might detect basic patterns (e.g. edges in an image or simple frequency changes in a spectrogram), while deeper layers might capture more complex patterns such as tonal or rhythmic features.

The architecture of the **Variational Autoencoder** (VAE) consists of three main components: the *encoder*, the *latent space* and the *decoder* [11]. The purpose of this composition is to learn the simplified representation of the input audio, generate variations of it, and then, lastly, decode these representations back into a usable audio format .wav.[6].

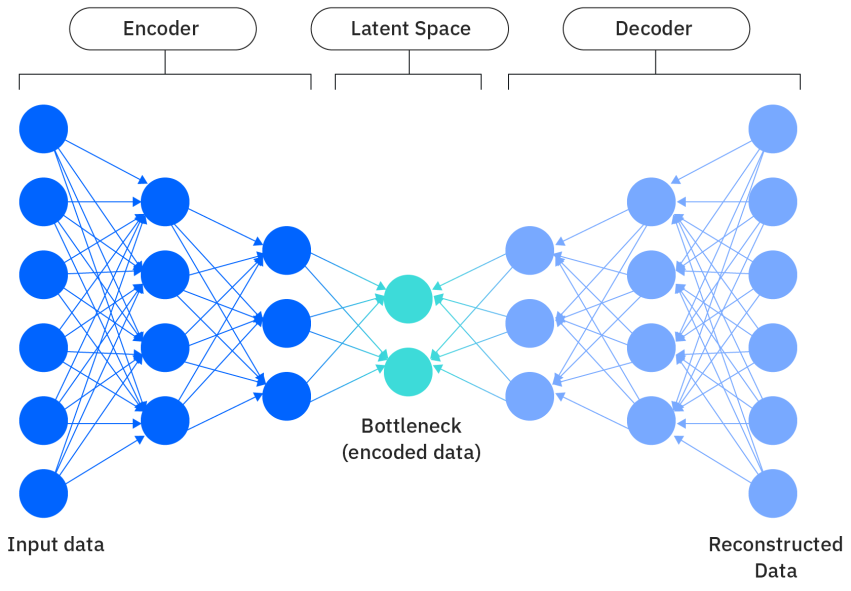


Figure 2.3: Structure of a VAE [8]

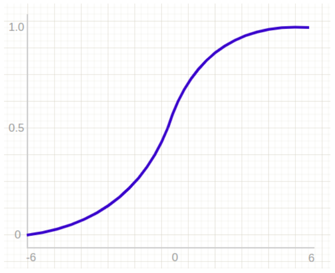
The **encoder** compresses the input data, consisting of already pre-processed *spectrograms* in this case, into a lower-dimensional latent space. This is done for simplification purposes so that the model captures only the necessary and most important key features of each sound for training.

The **latent space**, which can be regarded as the *bottleneck* of the model, serves as the core of this generative process in a **VAE**. Rather than compressing the data into a fixed point, the **VAE** models the latent space as a *probability distribution*, allowing for output variations to emerge from it. The encoder provides the *mean and variance vectors*, describing a so-called *multivariate normal distribution* (as shown in Figure 2.5). During generation, a random vector is sampled from this distribution by adding **stochasticity**, referring to the randomness or unpredictability involving probabilistic elements [19], to the process, allowing the model to generate several new outputs for the same inputs.

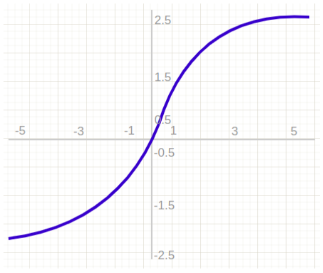
The **decoder** then takes the sampled latent vector and reconstructs the original spectrogram, reading and restoring the *min max values* from the preprocessing steps. The operation performed is essentially the *reverse* of what the encoder did, taking the low-dimensional latent representation back into a full-scale spectrogram. Once the decoded latent vector has been transformed into a spectrogram, the final output represents the time-frequency structure of the generated audio. This spectrogram will then be further transformed into an audio signal by applying the **inverse Short-Time Fourier Transform** (iSTFT), converting the spectrogram from the frequency domain back to the time domain.

The results are newly generated audio effects from the original data but with unique variations.

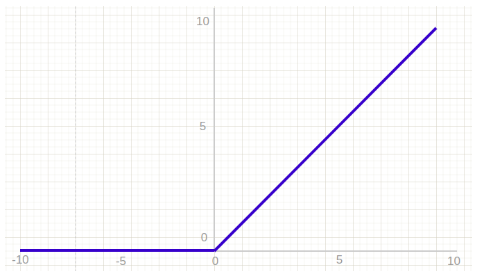
Detailed Breakdown of the Layers:
- **Input Layer (Spectrograms)**: The model takes a **Mel-spectrogram** as input. A **Mel-spectrogram** is a 2D matrix of frequency versus time, containing the essence of the audio, in a format suitable for further convolutional processing (as said in 2.3).
- **Convolutional Block (Encoder)**: Multiple convolutional layers progressively extract features from the spectrogram, starting with low-level features and progressing to more abstract features. Each convolutional layer is followed by a **non-linear activation function**, which in this case is a rectified linear unit or short **ReLU**, a widely used activation function in deep learning models operating by outputting positive inputs directly and otherwise returning "0" [4] and pooling layers, which are integral components of **CNN**s used to reduce the spatial dimension of the input and therefore decrease the number of parameters in the network, to *downsample* the data and focus on important features [3]. An activation function in a neural network determines whether a neuron (modeled by a layer) should be activated based on its input. This enables the network to handle more complex and intricate tasks. Without such *activation functions* all *neural networks* would be limited to linear functions. The most common activation functions include:
Sigmoid (2.4a)**:** Outputs values between "0" and "1", which is usually applied in probabilistic interpretations, **Tanh** (2.4b)**:** Outputs values between "-1" and "1" centering all the input data and **ReLU** (2.4c) [16].



(a) Sigmoid



(b) Tanh



(c) ReLU

Figure 2.4: Comparison of activation functions: Sigmoid, Tanh, and ReLU [16]

- **Latent Space**: The encoder outputs two vectors representing the *mean* and *variance* of the latent distribution. This is where **probabilistic modeling** occurs, introducing randomness through *stochasticity* that allows the generation of new data points.
- **Transposed Convolutional Block (Decoder)**: The decoder takes a sampled latent vector from this latent distribution and forwards it through a series of *transposed convolutional layers* in order to *upsample* it back into a full-size spectrogram. These layers invert the *downsampling* in the encoder and reconstruct the temporal and spectral features of the input audio.
- **Output Layer (Reconstructed Spectrogram)**: The final layer of the decoder produces a spectrogram much similar to the original input. However, because of the added variation in the latent space, the output is not an exact replica, but rather a new audio effect structurally similar to the input data.

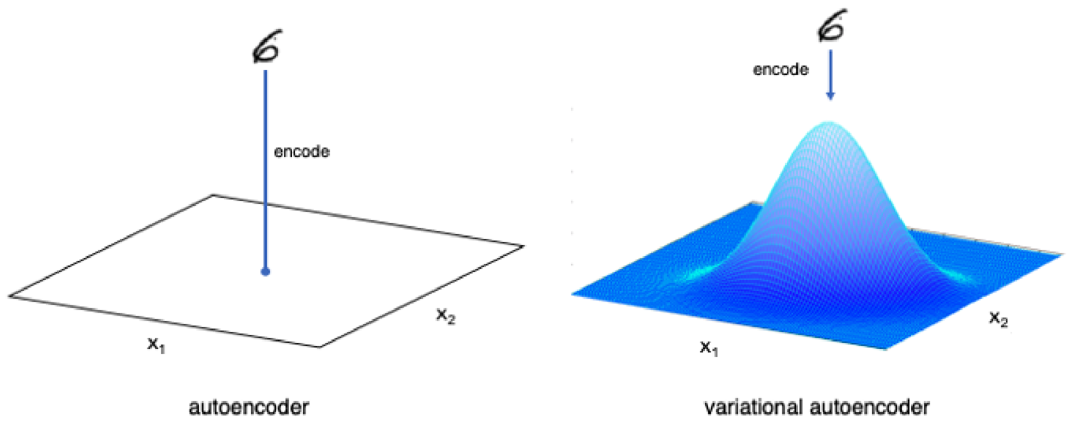


Figure 2.5: The Multivariate Normal Distribution of VAE [6]

2.5 Hyperparameter Selection and Model Training

The performance of a deep learning model is highly dependent on the careful tuning of its *hyperparameters*, which control the learning process and model complexity. This section will talk about the key hyperparameters selected for the **Variational Autoencoder** (VAE) and **Convolutional Neural Network** (CNN) architectures, and how they impact the model's performance during training. But what exactly are *hyperparameters*, what are their functions, and how do they differ from normal *parameters*?

Hyperparameters are so called because they operate at a higher level than standard model parameters, as they are external configurations that then drive the learning process. The key difference is that *parameters* are learned during training, whereas *hyperparameters* are predefined and control that very learning process [5].

- **Latent Space Size**: The size of the *latent space* will define how much information is kept when the encoder compresses the input data; a latent space too small might result in losing useful information and therefore lead to poor reconstructions and less variability of the generated outputs. On the other hand, a latent space too large may introduce irrelevant complexity and even result in overfitting, which is a term used when a model learns its training data *too* precisely, leading to poor generalization [9]. For this project, a size of *128 dimensions* for the latent space was chosen, as it provides a good balance between capturing the richness of the audio data while keeping the model computationally efficient [11].

- **Learning Rate**: The *learning rate* is a *hyperparameter* that controls how fast the model updates its weights during training. If the gradient for a specific weight is 5, the weight will be adjusted by $0.0001 \cdot 5$ with each step. A learning rate too high may make the model converge too fast to a suboptimal solution, while a low learning rate can result in slow convergence. This project settles on using a learning rate of *0.0001* since this allows for steady progress in minimization of the loss function, without introducing *instability* into the training process. The **Adam optimizer** was used to train because it self-adjusts the learning rate dynamically and suits complex *deep learning* models best [7], [12].

- **Batch Size**: "The *batch size* refers to the number of samples that get processed before the model's parameters are updated" [13]. Larger batch sizes offer more stable gradient estimates but require more memory, whereas smaller batches can result in noisier updates, due to less data per set, that would generalize better. The batch size chosen was *32*, representing a good trade-off between leveraging memory

efficiently and achieving stable updates.
- **Number of Epochs**: The model was trained for *150 epochs*, where one epoch is a single pass over the entire training dataset. Early stopping halted the training process when the loss value stopped improving for several epochs, ensuring that the model did not *overfit* the training data while still learning effectively.
- **Reconstruction Loss Weight**: The *reconstruction loss weight* α balances the trade-off between the reconstruction loss and the KL divergence (as can be seen in Formula 2.1). Selecting an appropriate value for α can significantly impact the model's performance and therefore is a delicate task. A α *too large* may result in the model behaving similarly to a regular **Autoencoder**, losing the ability of introducing variation to the output through the probabilistic features of a **VAE**, and a loss value too large. On the other hand, a α *too small* leads to an overly emphasis on the **KL divergence** (talked about in 2.6), leading to poorly reconstructed outputs. Therefore, α can be treated as a *hyperparameter*.

Loss Function of the Variational Autoencoder

$$\text{Loss} = \alpha \cdot \text{Reconstruction Error (MSE)} + \text{KL Divergence} \tag{2.1}$$

```python
import os

import numpy as np
import tensorflow as tf
from autoencoder import VAE


LEARNING_RATE = 0.0001
BATCH_SIZE = 32
EPOCHS = 150
```

Listing 2.1: The train.py script used for training the VAE with the set hyperparameters.

2.6 Evaluation and Assessment of the Results

Once trained, the performance of the model was evaluated objectively and subjectively, using a set of *quantitative* and *qualitative metrics.* This section addresses the methodology of model effectiveness testing in creating novel and high-quality audio effects.

Quantitative Evaluation

- **Reconstruction Loss**:

The main metric used for model evaluation was the **reconstruction loss**, which essentially measures how well the **VAE** can reconstruct an input spectrogram. A lower reconstruction loss means that the model has learned the essence of the most informative features of the audio data in a latent space with great efficiency. This metric was calculated as the **mean squared error** (MSE) between the original input and the reconstructed output.

- **Kullback-Leibler Divergence**:

Another important component of the loss function was the **KL divergence**, which made sure that the *distribution of the latent space* remained close to a standard normal distribution. The minimization of **KL divergence** also made sure that the model avoids *overfitting* and enforces a structured latent space, which is necessary for generating diverse audio outputs.

Qualitative Evaluation

- **Subjective Listening Tests**:

While quantitative metrics, like *reconstruction loss*, offer an objective measure through the use of mathematical expressions of the performance of the model, such measures do not present the actual quality of generated audio. Hence, subjective listening tests are conducted, as part of assessing the *perceptual quality* in the generated audio effects of the model. The evaluation of sound effects can be approached subjectively on the basis of various criteria: potential of usability and clarity of the sounds, although the judgment of the sounds as being of good or bad quality is highly subjective and differs from person to person.

Chapter 3

Results

3.1 Presentation and Interpretation of Key Results

The results of this project are comprised of several aspects: the newly generated sound effects (A.2) alongside the corresponding spectrograms and a **VAE model**, consisting of multiple Python scripts (A.1). This model has undergone training with the dataset and the set *hyperparameters*, discussed in Section 2.5, and is capable of generating new sounds.
Following a training period of 150 epochs (as outlined in 3.1 and 3.2) the loss value ranges between 40 to 42, which can be considered relatively high in the context of this thesis, which aims to generate novel and distinctive sound effects, all while maintaining a similarity to the source sound category.

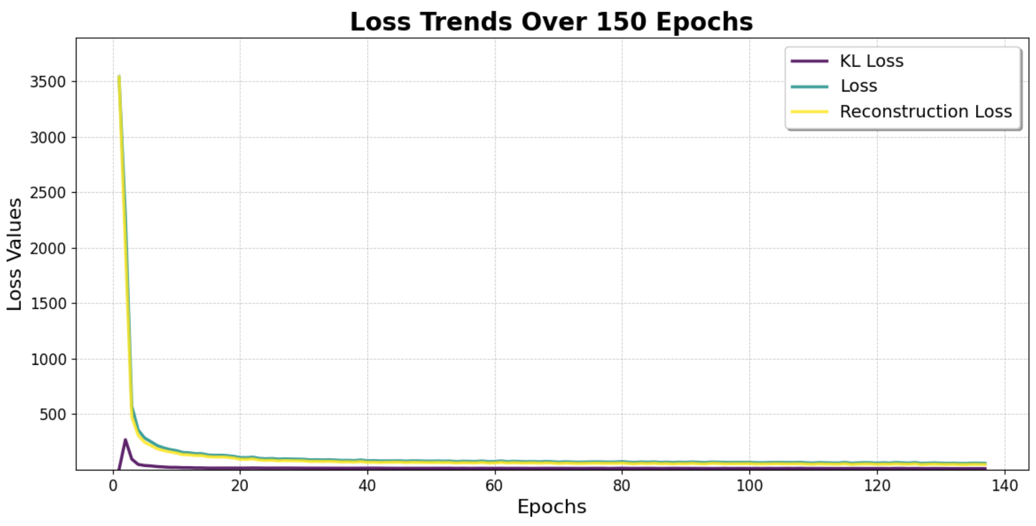


Figure 3.1: VAE model loss trend over 150 epochs [15]

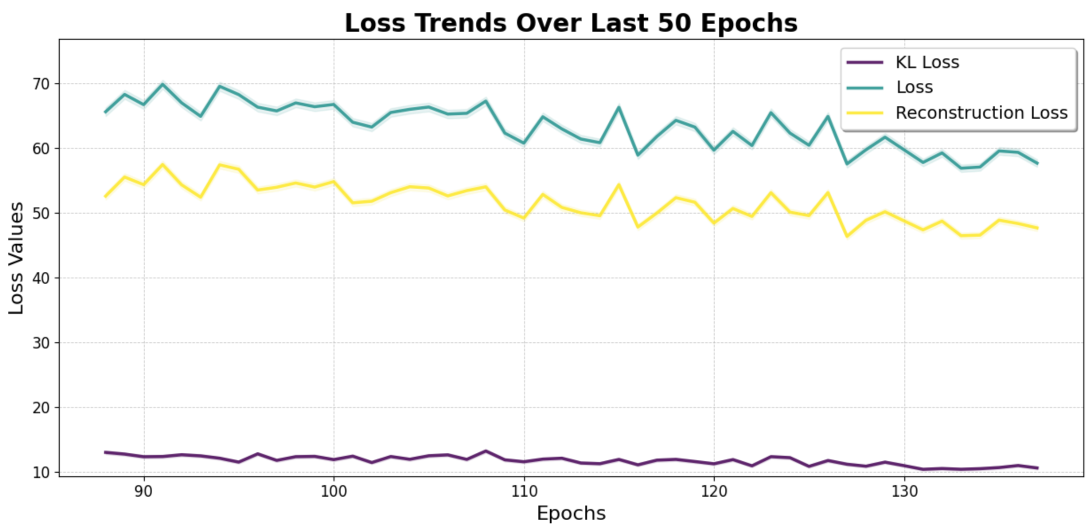


Figure 3.2: VAE model loss trend in last 50 epochs [15]

The limited number of sample sound effects generated by the **VAE**, through the *"generate.py"* and *"soundgenerator.py"* scripts, have been found to be merely poor reconstructions and variations of the input data set, as can be seen in 3.3, when comparing the original spectrograms of a random sound effect within the dataset and the generated output spectrogram. The underperformance of the generation can be attributed to numerous factors inherent in the training process and in the *hyperparameters* involved.

The **VAE** was only trained over *150 epochs* on a very small, hand-made dataset of 1006 audio effects, which is inadequate for such a complex task as sound generation. Training over a higher amount of epochs was not possible with such a dataset without risking to *overfit* the model operating with a small dataset. Apart from being too small for the algorithm, the dataset could potentially also be lacking diversity, leading to insufficient representations of sound characteristics. Another factor could be the *hyperparameter* configuration being disadvantageous, such as the learning rate being too small or too large, training the model either too slowly or leading to *instability*. Alternatively, the batch size may be set at an insufficient level, which, as previously mentioned in 2.5, may result in noisier updates. In conclusion, the objective evaluation provided by the overall loss function is confined to measuring numerical reconstruction. It is important to note that a *spectrogram* with an acceptable loss may still be perceived as distorted or unnatural by a listener.

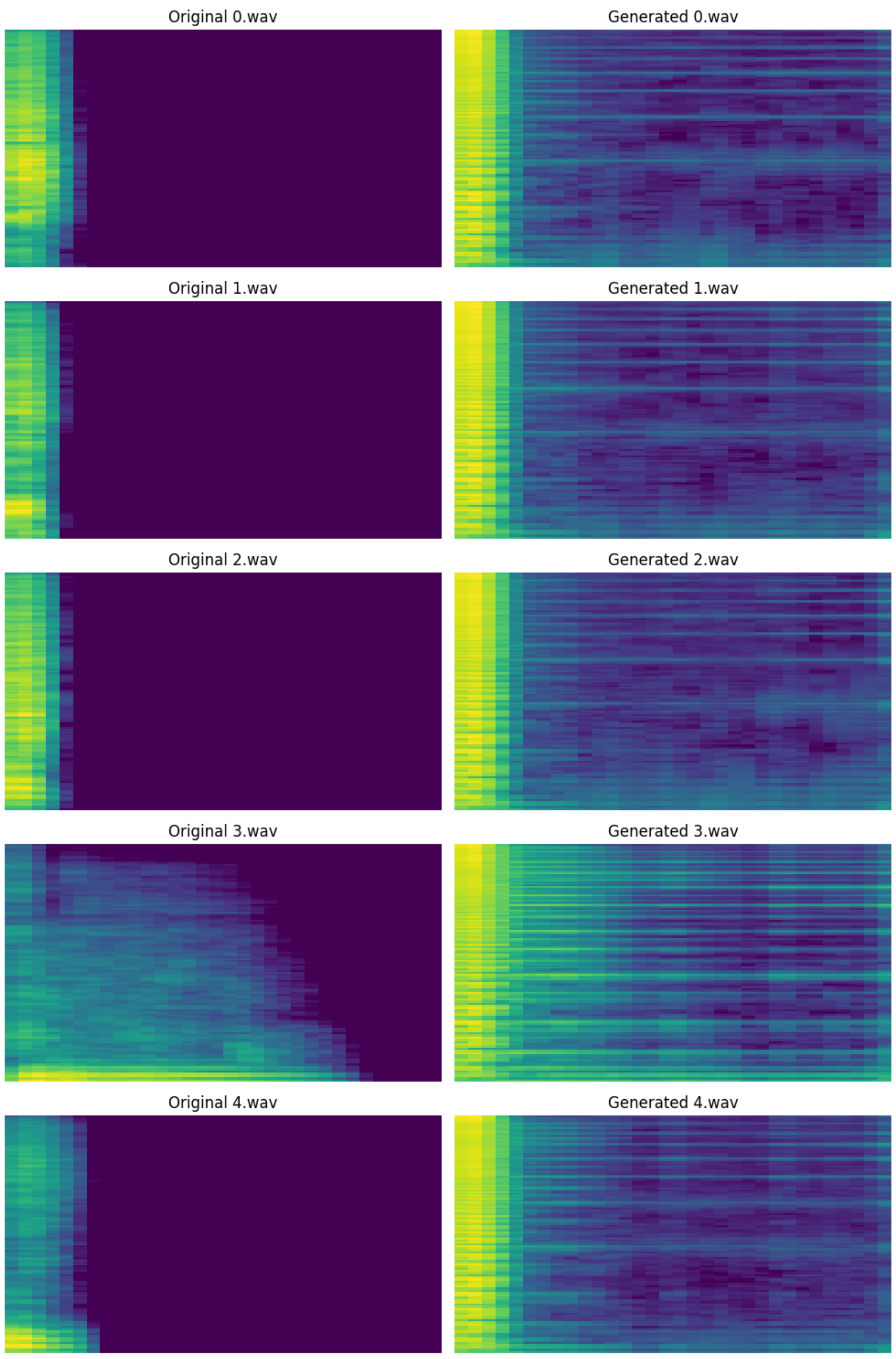


Figure 3.3: Comparison of Spectrograms visualized by "plotspectrograms.py" [15]

Chapter 4

Conclusion and Outlook

4.1 Summary of Contributions

This thesis aimed to explore the potential of Deep Learning system for audio effect generation, bridging music production and AI technologies. Overall the contributions can be split into the following categories:

Technical Contributions: The development of a **Variational Autoencoder** (VAE) for sound generation, implementation of several processing steps, such as the preprocessing pipelines, transforming raw audio into *spectrogram representations* compatible for model training and further processing and the integration of deep learning techniques in Python.

Research Contributions: The comparative analysis of generative models in audio generation, the identification of limitations and challenges in different generative models for this thesis and the contribution to the understanding of **AI** applications in audio.

Personal Development: Expansion of knowledge in **AI** and deep learning algorithms as well as enhanced skills in Python, machine learning frameworks and tools, such as *Keras*, *Librosa* and *Tensorflow* [22], [24], [25] and data preprocessing in audio.

4.2 Limitations and Challenges

During the development of the **VAE**, several limitations and challenges became apparent over time. One of those being the major update from *Keras 2* to *Keras 3* in 2023, which resulted in a complete restructuring of the backend with its functions. This in turn caused the code written with the assistance of a tutorial from 2021 [26] to become inoperable, necessitating the identification of alternative solutions and the restructuring of several functions within the code. Another challenge was (as mentioned in 2.5) to establish appropriate values for the *hyperparameters* to ensure model stability and a steady learning process. The overall loss function in the initial training runs exceeded *"250'000"*, due to an excessive *reconstruction loss* weight, which ultimately resulted in poor reconstructions.
Limitations include the limited training of the model of only *150 epochs* to prevent *overfitting*, which made it very challenging to achieve effective reconstruction. Another limitation, that played a major role in the early stages implementing the model and later heavily impacted the training process, was the lack of available audio datasets or even accessible audio effects in general, necessary for a generative deep learning model, such as a **VAE**, to be able to conduct reconstruction processes well. To pursue the potential training of a **VAE** on a larger scale, meaning having a larger dataset, minimizing the loss function even further and optimizing every *hyperparameter*, a lot of time and resources would be needed. Furthermore, the application of different reconstruction methods should be considered, as the **iSTFT** algorithm used for this project lacks the ability to realign the phases of the audio, resulting in robotic sounds.

4.3 Outlook

Looking into the future, numerous possibilities emerge for the continuation of research and development of this project. Assuming the availability of a substantially larger, high-quality dataset accompanied by simple sound effects, or a comprehensive optimisation of all the relevant *hyperparameters*, there is considerable potential for a more large scale and efficient training process. This, in turn, could result in the utilisation of the model for commercial applications, including film sound effect scoring, music production, and game audio development, among numerous others. The implementation of the model created in this project could also be applied in other domains requiring generative **AI**, such as image creation or manipulation, text and language processing, and many more. The potential for the emergence of a novel, more suitable form of generative deep learning system for this project is noteworthy, with the capacity to complement or enhance the methodologies employed in this thesis.

Appendix A

Appendices

A.1 Documentation

All the Python scripts and the dataset used for this project can be found under the following QR-Codes:

Figure A.1: GitHub repository "SoundGenDL" [15]

Figure A.2: Dataset used for thesis, as described in 2.3

A.2 Preprocessing Script Snippet

```
class Padder:
    #Padder  responsible to apply padding to array
    def __init__(self, mode="constant"):
        self.mode = mode

    def left_pad(self, array, num_missing_items): # [1, 2, 3]
       --> 2 --> [0, 0, 1, 2, 3] "zero-padding"
        padded_array = np.pad(array,
                              (num_missing_items, 0),
                              mode=self.mode)
        return padded_array

    def right_pad(self, array, num_missing_items): # [1, 2,
       3] --> 2 --> [1, 2, 3, 0, 0] "zero-padding"
        padded_array = np.pad(array,
                              (0, num_missing_items),
                              mode=self.mode)
        return padded_array



class LogSpectrogramExtractor:
    #LogSpectrogramExtractor extracts log spectrograms (in
       dB) from time-series signal
    def __init__(self, frame_size, hop_length):
        self.frame_size = frame_size
        self.hop_length = hop_length

    def extract(self, signal): # short-time fourier transform
        stft = librosa.stft(signal,
                            n_fft=self.frame_size,
                            hop_length=self.hop_length)[:-1]
                               # (1 + frame_size / 2,
                               num_frames) 1024 --> 513 (not
                               good - want even) --> 512
        spectrogram = np.abs(stft)
        log_spectrogram = librosa.amplitude_to_db(spectrogram)
        return log_spectrogram


class MinMaxNormaliser:
    #MinMaxNormaliser applies min max normalisation to array
    def __init__(self, min_val, max_val):
        self.min = min_val
        self.max = max_val

    def normalise(self, array):
        norm_array = (array - array.min()) / (array.max() -
           array.min()) # --> [0, 1]
        norm_array = norm_array * (self.max - self.min) +
           self.min
        return norm_array

    def denormalise(self, norm_array, original_min,
       original_max):
        array = (norm_array - self.min) / (self.max -
           self.min)
        array = array * (original_max - original_min) +
           original_min
        return array
```

Listing A.1: Snippet of preprocess.py script displaying Padding, Mel-Spectrogram creation and MinMaxNormalising

List of Figures

Glossary

AI: Artificial Intelligence. 1, 2, 4

CNN: Convolutional Neural Network. 2

FT: Fourier Transform. 2

STFT: Short-Time Fourier Transform. 2

iSTFT: inverse Short-Time Fourier Transform. 2, 4

GAN: Generative Adversarial Network. 2

RNN: Recurrent Neural Network. 2

AE: Autoencoder. 2

VAE: Variational Autoencoder. 1, 2, 3, 4

ReLu: Rectified linear unit, activation function. 2

MSE: Mean squared error. 2

KL: Kullback-Leibler Divergence. 2

Bibliography

[1] Alexander Bastounis, Anders C Hansen, and Verner Vlačić. *The mathematics of adversarial attacks in AI – Why deep learning is unstable despite the existence of stable neural networks.* 2021. URL: https://arxiv.org/abs/2109.06098.

[2] Diego de Benito et al. “Exploring convolutional, recurrent, and hybrid deep neural networks for speech and music detection in a large audio dataset”. In: *EURASIP Journal on Audio, Speech, and Music Processing* 2019 (June 2019).

[3] Jason Brownlee. *Pooling Layers for Convolutional Neural Networks.* Accessed: 08-10-2024. 2023. URL: https://www.machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/.

[4] Jason Brownlee. *Rectified Linear Activation Function for Deep Learning Neural Networks.* Accessed: 08-10-2024. 2023. URL: https://www.machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/.

[5] Jason Brownlee. *What is the Difference Between a Parameter and a Hyperparameter?* Accessed: 09-10-2024. 2019. URL: https://machinelearningmastery.com/difference-between-a-parameter-and-a-hyperparameter/.

[6] David Foster. *Generative deep learning.* " O’Reilly Media, Inc.", 2022.

[7] Ian Goodfellow. *Deep learning.* MIT press, 2016.

[8] IBM. *Variational Autoencoder (VAE): Explained.* Accessed: 12-08-2024. 2024. URL: https://www.ibm.com/think/topics/variational-autoencoder.

[9] IBM. *What is Overfitting?* Accessed: 03-11-2024. 2024. URL: https://www.ibm.com/think/topics/overfitting.

[10] Hohyub Jeon et al. "Area-efficient short-time fourier transform processor for time–frequency analysis of non-stationary signals". In: *Applied Sciences* 10.20 (2020), p. 7208.

[11] Yash Kavaiya. *Unlocking the Power of Discrete Latent Spaces: Autoencoders*. Accessed: 28-12-2024. 2023. URL: https://medium.com/@yash.kavaiya3/unlocking-the-power-of-discrete-latent-spaces-autoencoders-48cc462fac92.

[12] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv preprint arXiv:1312.6114* (2013). URL: https://arxiv.org/abs/1312.6114.

[13] Gunawan Kurniawan. *Understanding and Determining the Ideal Batch Size in Machine Learning*. Accessed: 11-11-2024. 2020. URL: https://medium.com/@gunkurnia/understanding-and-determining-the-ideal-batch-size-in-machine-learning-d3ef93acf79a.

[14] Ziqi Pan, Li Niu, and Liqing Zhang. "UniGAN: Reducing mode collapse in GANs using a uniform generator". In: *Advances in neural information processing systems* 35 (2022), pp. 37690–37703.

[15] Robert Paun. *SoundGenDL Github Repository*. 2024. URL: https://github.com/proddxterner/SoundGen_DL/.

[16] Roboflow. *What is an Activation Function? A Complete Guide*. Accessed: 08-10-2024. 2023. URL: https://blog.roboflow.com/activation-function-computer-vision/.

[17] Simplilearn. *Arrays in Data Structure Tutorial*. Accessed: 12-26-2024. 2024. URL: https://www.simplilearn.com/tutorials/data-structure-tutorial/arrays-in-data-structure.

[18] Simplilearn. *Arrays in Data Structure Tutorial*. Accessed: 12-26-2024. 2024. URL: https://www.simplilearn.com/tutorials/data-structure-tutorial/arrays-in-data-structure.

[19] *Stochastic*. Accessed: 17-10-2024. 2024. URL: https://en.wikipedia.org/wiki/Stochastic.

[20] Cecilia Summers and Michael J. Dinneen. *Nondeterminism and Instability in Neural Network Optimization*. 2021. URL: https://arxiv.org/abs/2103.04514.

[21] Codecademy Team. *Normalization*. Accessed: 11-11-2024. 2024. URL: https://www.codecademy.com/article/normalization.

[22] Keras Team. *Keras: Deep Learning for humans*. Accessed: 16-08-2024. 2015. URL: https://keras.io/about/.

[23] Librosa Team. *Librosa Documentation*. 2024-14-10. 2015. URL: https://librosa.org/doc/main/generated/librosa.feature.melspectrogram.html.

[24] Librosa Team. *Librosa: Python Library for Audio and Music Analysis*. Accessed: 16-08-2024. 2015. URL: https://librosa.org/doc/latest/index.html.

[25] TensorFlow Team. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Accessed: 16-08-2024. 2015. URL: https://www.tensorflow.org/.

[26] Valerio Velardo. *Sound Generation with Neural Networks*. Accessed: 21-09-2024. 2021. URL: https://www.youtube.com/watch?v=F1KA5Z3DOGU.