Mikael Metsälä

Possibilities of convolutions in AI-reconstructed music

School of Technology and Innovations
Master's thesis in Computing Sciences

Vaasa 2024

UNIVERSITY OF VAASA
School of Technology and Innovations
Author: Mikael Metsälä
Title of the Thesis: Possibilities of convolutions in AI-reconstructed music
Degree: Master of Science in Technology
Programme: Automation and Computer Science
Supervisor: Teemu Mäenpää
Year: 2024 **Page count:** 71

ABSTRACT:

This thesis investigates the application of convolutional layers within an autoencoder to reconstruct one-dimensional audio data in systems with limited computational resources. The primary objective of this study is to explore whether convolutional layers could improve autoencoder performance by retaining key audio characteristics during the reconstruction process. While deep generative models have shown promise for audio synthesis, research has predominantly focused on large-scale implementations, leaving open questions about the adaptability of these approaches to smaller systems. This study hypothesized that convolutional layers would enable improved reconstructions compared to fully connected (FC) layers within a limited VRAM environment.

To test this hypothesis, a controlled experimental approach was employed, which involved a detailed comparison of the performance of both fully connected and convolutional architectures. Each model was trained from scratch on one-dimensional audio sequences until reaching convergence. This approach allowed for a clear and precise evaluation of the relative effectiveness of each model type. To ensure a comprehensive assessment, several key metrics were selected, including mean squared error as one of the primary metrics, alongside observations of convergence rate and memory efficiency to evaluate model performance.

The findings indicate that the convolutional autoencoder achieved superior reconstruction quality, as evidenced by its lower mean squared error and faster epoch-wise progression to accuracy, despite taking slightly longer per epoch than the FC model. These results highlight convolutional architectures' potential to facilitate high-quality audio reconstruction on smaller systems, making advanced AI-driven audio analysis more accessible. The convolutional model's ability to represent low-frequency components more effectively and with less added noise than the FC model supports the hypothesis, although challenges, such as limitations in replicating high-frequency components, were noted in both models. Overall, these results suggest that convolutional autoencoders could offer a promising approach for efficiently reconstructing audio data on constrained hardware.

The study contributes valuable insights to music analysis and AI audio research, particularly in the context of scalable model design for low-resource environments. It acknowledges limitations, such as subjective sound quality assessment and hardware constraints, and recommends future work. Further research might focus on enhancing frequency representation within convolutional networks and improving audio separation capabilities. By advancing methods that operate effectively on smaller systems, this study encourages further exploration of accessible AI applications in music analysis and digital audio processing.

KEYWORDS: audio processing, artificial intelligence, machine learning, neural networks

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen akateeminen yksikkö
Tekijä: Mikael Metsälä
Tutkielman nimi: Possibilities of convolutions in AI-reconstructed music
Tutkinto: Diplomi-insinööri
Oppiaine: Automaatio ja tietotekniikka
Työn ohjaaja: Teemu Mäenpää
Valmistumisvuosi: 2024 **Sivumäärä:** 71

TIIVISTELMÄ:

Tässä tutkielmassa tarkastellaan rajatuilla laskentaresurseilla toimivan konvoluutiokerroksia hyödyntävän autoenkoodaajan soveltamista audiosignaalin rekonstruointiin. Tavoitteena on selvittää, voivatko konvoluutiokerrokset parantaa autoenkoodaajan oppimiskykyä ja auttaa sitä säilyttämään musiikille ominaisia piirteitä rekonstruointiprosessin aikana. Aiemmissa tutkimuksissa on todistettu syvien generatiivisten mallien kyky audiosynteesissä, kun käytössä on ollut valtavasti laskentatehoa ja muistia, mikä jättää kysymyksen avoimeksi pienemmän laskentatehon omaavien järjestelmien osalta. Hypoteesina tässä tutkimuksessa on, että konvoluutiokerrokset voivat tarjota parempaa rekonstruktiota kuin täysin kytketyt kerrokset rajallisesti keskusmuistia sisältävissä järjestelmissä.

Hypoteesin testaamiseksi toteutettiin vertailukoe, jossa verrattiin täysin kytketyistä kerroksista koostuvan neuroverkon ja konvoluutiopohjaisten verkon suorituskykyä. Molemmat mallit koulutettiin audiodatan avulla, kunnes ne saavuttivat konvergenssin. Näin saatiin selkeä ja tarkka vertailu arkkitehtuurien tehokkuudesta. Mallien suorituskykyä arvioitiin ensisijaisesti keskineliövirheen avulla, ja lisäksi tarkasteltiin konvergenssinopeutta ja käytetyn muistin määrää.

Tutkimuksen tulokset osoittavat, että konvoluutiokerroksia sisältävä autoenkoodaaja rekonstruoi audiosignaalia paremmin, mikä käy ilmi sen matalammasta keskineliövirheestä sekä sen tuottamasta pienemmästä kohinan määrästä. Näiden tulosten perusteella konvoluutioarkkitehtuuri osoittaa potentiaalia korkealaatuisen audionsignaalin rekonstruointiin laskentateholtaan rajatuissa järjestelmissä, mikä parantaa tällaisten tekoälyyn perustuvien järjestelmien saavutettavuutta. Molemmissa malleissa havaittiin haasteita korkeiden taajuuksien rekonstruoinnissa.

Johtopäätöksenä voidaan todeta, että konvoluutiokerrokset parantavat autoenkoodaajan kykyä rekonstruoida audiosignaalia, erityisesti matalilla taajuuksilla ja vähentämällä kohinaa, mikä mahdollistaa mallin käyttämisen myös laskentateholtaan rajatuissa järjestelmissä. Tämä osoittaa konvoluutioon pohjautuvien arkkitehtuurien potentiaalin laadukkaaseen audiodatan rekonstruointiin ja mahdollistaa tekoälyn soveltamisen musiikkianalyysissä ja äänenkäsittelyssä laajemmalle yleisölle. Tulevissa tutkimuksissa voitaisiin keskittyä konvoluutiomallien kykyyn erotella eri taajuuskomponentteja entistä tarkemmin sekä parantaa niiden suorituskykyä korkeiden taajuuksien käsittelyssä.

AVAINSANAT: audio processing, artificial intelligence, machine learning, neural networks

Contents

1 Introduction 9

2 Basic concepts and technologies 12

2.1 Categorisation 12

2.1.1 Parameter-based 12

2.1.2 Non-parameter-based 13

2.2 Deep Neural Networks 13

2.2.1 Model 13

2.2.2 Dataset 15

2.2.3 Objective Function 17

2.2.4 Optimisation Procedure 19

2.3 Neural Network Architectures 20

2.3.1 Autoencoders 20

2.3.2 Variational Autoencoders 21

2.3.3 Generative Adversarial Networks 23

2.3.4 Transformers 25

3 Related Work 29

3.1 WaveNet 29

3.1.1 Causal and Dilated Convolutions 30

3.1.2 Non-linear Quantisation 31

3.2 Jukebox 32

3.2.1 Multi-scale VQ-VAE 32

3.2.2 Scalable Transformers and Upsampling 34

3.3 Summary 35

4 Methodology 38

5 Experiment Design and Model Evolution 40

5.1 Dataset and System 41

5.2 Model Evolution 42

5.2.1 Preprocessing 42

5.2.2 Model development 46
5.2.3 Testing setup 49
5.3 Results 50
5.4 Analysis 59
5.4.1 Common features 59
5.4.2 Separating features 61
6 Conclusion 66
References 68

Pictures

Picture 1. Screenshot of Task Manager during convolutional model training. 64

Picture 2. Screenshot of Task Manager during base model training. 65

Figures

Figure 1. A simple feedforward neural network by Goodfellow et al. (2016, p. 170) 14

Figure 3. Variational Autoencoder depicted as a figure. Blue boxes represent the encoder, white is the latent space, and green boxes represent the decoder. $\boldsymbol{x}$ is input, and $\boldsymbol{x}$ represents output. 22

Figure 2. Generative Adversarial Network architecture depicted by Bengesi et al. (2023). 24

Figure 4. Scaled Dot-Product Attention architecture (Vaswani et al. 2017). 26

Figure 5. Transformer architecture (Vaswani et al. 2017). The figure consists of an encoder on the left and a decoder on the right. 28

Figure 6. Different types of convolutions. Blue dots depict the input layer, grey marks the hidden layers, and yellow is the output layer. 30

Figure 7. Effects of μ-law companding on a waveform. 45

Figure 8. The encoder part of the Convolutional Autoencoder is in blue, and the first latent vector is in white. 48

Figure 9. The decoder part of the Convolutional Autoencoder is in green, and the second latent vector is in white. 49

Figure 10. Evolution of the validation loss during the test with global minima. The blue curve represents the base model's performance, and the red curve represents the convolution model's performance. 52

Figure 11. Waveform of the validation audio. 55

Figure 12. The base model's reconstruction data is depicted as a waveform after Epoch 0. 55

Figure 13. The base model's reconstruction data is depicted as a waveform after Epoch 56. 55

Figure 14. The convolution model’s reconstruction data is depicted as a waveform after Epoch 0. 56
Figure 15. The convolution model’s reconstruction data is depicted as a waveform after Epoch 75. 56
Figure 16. This represents the frequency distribution and corresponding magnitudes of the validation data. 57
Figure 17. Frequency distribution and magnitude of the base model's reconstruction data after Epoch 0. 57
Figure 18. Frequency distribution and magnitude of the base model's reconstruction data after Epoch 56. 58
Figure 19. Frequency distribution and magnitude of the convolution model's reconstruction data after Epoch 0. 58
Figure 20. Frequency distribution and magnitude of the convolution model's reconstruction data after Epoch 75. 58

Tables

Table 1. Summary of changes in the base model's dependent variables. 53
Table 2. Summary of changes in the convolution model's dependent variables. 54

Abbreviations

ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
BERT	Bidirectional Encoder Representations from Transformer
CNN	Convolutional Neural Network
CV	Computer Vision
EDM	Electronic Dance Music
FT	Fully Connected
GAN	Generative Adversarial Network
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
KL	Kullback-Liebler
MAE	Mean Absolute Error Multilayer Perceptron
MLP	Multilayer Perceptron

MSE	Mean Squared Error
NLP	Natural Language Processing
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RFFT	Real Fast Fourier Transform
RMSE	Root-Mean-Square Error
RNN	Recurrent Neural Network stochastic gradient descent
SGD	Stochastic Gradient Descent
SNR	Signal-to-Noise Ratio
STFT	Short-Time Fourier Transform
VAE	Variational Autoencoder
ViT	Vision Transformer
VQ-VAE	Vector Quantized Variational Autoencoder
VRAM	Video Random Access Memory

1 Introduction

Music generated with the help of Artificial Intelligence is a topic that has puzzled researchers for decades. AI-generated music research took its first steps as early as the 1950s when Hiller Jr. and Isaacson (1957) released a model that generated sheet music based on the Markov chain model. Many of these early parameter-based generative models were not multi-layered neural networks (Zhu et al., 2023), which at the time suffered from a lack of efficient training methods (Briot et al., 2019, p. 41). In the 2006 paper A Fast Learning Algorithm for Deep Belief Nets, Hinton et al. introduced a solution to this, paving the way for the rise of deep neural networks, and in 2012, AlexNet, a deep neural network, won the ImageNet image recognition competition, resulting in a paradigm shift, making deep learning the state-of-the-art solution for prediction problems (Briot et al., 2019, pp. 3, 41–42).

Deep learning is a vague term as it does not share a scientifically agreed-upon definition (Briot et al., 2019, p. 3). As a part of artificial intelligence, it usually refers to machine learning done with deep neural networks consisting of multiple layers that hierarchically extract and abstract data (Briot et al., 2019, p. 3). Briot et al. (2019, p. 3) highlight three major milestones that have fuelled the surge of deep learning: an increase in the quantity of data available, enhanced availability of computational resources, and technological advances, notably the meaningful application of convolutions, which are particularly relevant to the context of this thesis. This is supported by Bengesi et al. (2023) as they identified that prior to 2010, interest in deep learning was hindered by the limited availability of computing resources and insufficient large datasets.

After the major roadblocks were overcome and deep learning research gained wind in its sails, deep learning took a new course in 2013 when Kingma and Welling released Variational Autoencoder (VAE), followed by Goodfellow et al. (2014) with their Generative Adversarial Network (GAN), building the foundation for Generative Artificial Intelligence. At the time of their introduction, these new types of neural networks aimed to capture the underlying probability distribution of the data (Goodfellow et al., 2016, pp.

693, 697). The benefit of learning the distribution is the possibility of sampling it and generating novel data instances that resemble the original data (Goodfellow et al., 2016, p. 707)

These networks mainly operated with relatively small-resolution images where the input for the network was a complete image like the MNIST database of handwritten numbers (Goodfellow et al. 2014). One image from MNIST has a 28x28 resolution (LeCun et al., 1998). However, a 2-minute song sampled with the usual 44,1 kHz has roughly 5 million input parameters compared to an image of the MNIST set, which has a little less than 800 input parameters. As Dhariwal et al. (2020) note, it is very computationally demanding. This has led to new solutions in which the input is split into segments that are fed to the network as separate instances.

This “segmented learning” can still work for non-generative tasks. However, randomly sampling the latent space for multiple audio segments and combining them is unlikely to create a coherent song. This problem has given birth to autoregressive networks that calculate the probability of each new sample as a joint probability over all previous samples (van den Oord et al., 2016). Such networks are designed to generate long and coherent audio and usually consist of two separate neural networks, which are autoencoding and autoregressive in nature (Dhariwal et al., 2020). The first layers of the autoencoder are often convolutional layers designed to maximise the receptive field of the network, making it easier to model longer temporal dependencies (van den Oord et al., 2016).

The main aim of this thesis is to explore the use of convolutions for one-dimensional sequential audio, focusing on the regenerative capabilities of autoencoders in music. The inspiration for this study arises from the challenges identified by Dieleman et al. (2018), particularly the lack of long-term structure in AI-generated music. Previous research has demonstrated the feasibility of capturing local structures like timbre, but modelling higher-level structures, such as verses and choruses, remains elusive (Dieleman et al.,

2018). However, generative AI systems, especially in music, typically require extensive computational resources to achieve coherent and high-quality output (Dhariwal et al., 2020). This study aims to explore approaches that can be implemented on smaller systems, offering feasible solutions for researchers without access to vast computational resources. This study seeks to answer the question: Can convolutional architecture enhance the regenerative performance of autoencoder on one-dimensional audio data? This study hypothesizes that applying convolutions will improve the autoencoder's ability to capture and preserve essential structural features in the reconstructed output, thereby enhancing its understanding of relationships between preceding and succeeding data points.

This study follows a controlled experimental design to explore the effects of different neural network architectures on AI-regenerated music. A controlled experiment allows for precise manipulation of independent variables—such as the type of neural network used—and careful observation of their effects on the dependent variables, including the quality and characteristics of the music. By creating a structured environment, this study aims to isolate specific factors and assess their impact on model performance. The methodology employed in this thesis provides a systematic approach to validate the hypothesis and contribute to the broader understanding of neural network architectures in regenerative music through empirical evidence.

This study is divided into six chapters. The second chapter consists of an overview of neural networks, aiming to give readers an understanding of the basic concepts and technologies surrounding the field. The third chapter examines two influential studies that inspired this research. The fourth chapter discusses the methodology behind this study. Chapter five depicts the model and describes the experiment. The last chapter concludes this process and discusses possible future directions for this line of study.

2 Basic concepts and technologies

Artificial intelligence is a vast field that is constantly expanding. Its subset is Generative Artificial intelligence, which has recently become more popular in the eyes of the public (Bengesi et al., 2023). It can be hard to get a grasp of the field as it is moving so fast, but using categories can help make it easier to understand. There are various ways to categorise the process of generating music using AI models (Zhu et al., 2023; Bengesi et al., 2023). In addition to categorisation, this chapter aims to explain the building blocks of a generic neural network and provides an overview of popular Generative Artificial Intelligence architectures, their functionalities, and how they operate.

2.1 Categorisation

In their 2023 survey, Zhu et al. introduced an approach that divides models into two categories: parameter-based and non-parameter-based. A characteristic of this approach is that the models are differentiated by the type of input they use (Zhu et al., 2023). The non-parameter-based category is still divided into two subcategories: prompt-based and visual-based models.

Generative models can also be divided by model architecture, as shown by Bengesi et al. (2023). Architectures that have gained popularity are Generative Adversarial Networks, Variational Autoencoders, and Transformers (Bengesi et al., 2023). These architectures are described in-depth later in this chapter.

2.1.1 Parameter-based

Parameter-based models represent the majority of the models listed by Zhu et al. (2023). These models range from Hiller Jr.'s and Isaacson's (1957) Markov chain models to Dhariwal et al.'s (2020) multi-scale Vector Quantized Variational Autoencoder types of deep neural networks. Common to these models is that they require specific input parameters such as tempo or key (Zhu et al., 2023).

2.1.2 Non-parameter-based

A good example of prompt-based models is MusicLM, developed by Agostinelli et al. (2023). The model takes the text prompt as an input and uses sequence-to-sequence modelling to generate multiple-minute-long songs that adhere to the text prompt (Agostinelli et al., 2023). Applications of visual-based models like V-MusProd by Zhuo et al. (2022) include generating background music for videos by conditioning the model with images or video.

2.2 Deep Neural Networks

As Bengio et al. (2012) neatly put it, AI's goal is to ”understand the world around us”. The pursuit of this goal has led researchers to turn to deep learning, which, as previously established, involves the use of deep neural networks for machine learning (Briot et al., 2019, p. 3). Goodfellow et al. (2016, p. 151) describe the fundamental independent elements necessary for constructing such a machine learning algorithm as a model, a dataset, an objective function, and an optimization procedure. Next, this study expands on these four basic elements and what are their implications.

2.2.1 Model

In the context of machine learning, the model, often referred to as a neural network, is an artificial construction that mimics the neurons of the human brain (Nwadiugwu, 2021). Each neuron in a neural network has attributes called a weight and a bias; these two, together with the input and activation function, for example, sigmoid activation, determine the strength of the signal that is passed to the neuron in the next layer (Goodfellow et al., 2016, pp. 107, 65-66). This relationship is defined in Equation 1, and in Figure 1, each arrow represents a weight. In the model, the flow of the information or signal happens in two passes, forward pass and backward pass, also referred to as forward propagation and back-propagation (Goodfellow et al., 2016, p. 200).

$$\hat{y} = \sigma(wx + b) \quad (1)$$

A feedforward network, or Multilayer Perceptron (MLP), is a basic neural network architecture composed of layers of neurons where, in the forward pass, the signals flow unidirectionally from the input layer to the output layer (Goodfellow et al., 2016, p.164). Figure 1 illustrates a typical feedforward network, designed to approximate a function by mapping inputs to outputs, consists of a clearly defined structure with multiple layers: an input layer x, one hidden layer h, and an output layer y (Goodfellow et al., 2016, pp. 164-165). While feedforward networks are foundational, other architectures like Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) cater to specific data types and tasks, leveraging unique structural features to process spatial and sequential data, respectively (Goodfellow et al., 2016, pp. 326, 367).

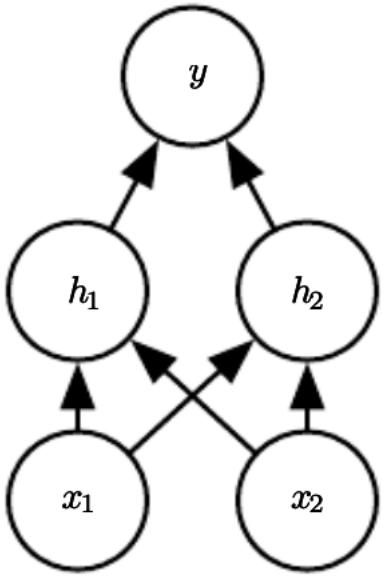


Figure 1. A simple feedforward neural network by Goodfellow et al. (2016, p. 170)

CNNs are a specialised type of feedforward network where one or more layers are replaced with convolutional layers (Goodfellow et al., 2016, p. 326). The convolutional operation in these layers involves sliding a filter or kernel w over the input data x to produce a feature map s (Goodfellow et al., 2016, p. 328). This process captures local

patterns by applying the same filter across various parts of the input, thus enabling CNNs to process spatial or multidimensional data like images efficiently (Goodfellow et al., pp. 330-333). To better understand the convolutional operation, consider a simplified one-dimensional example: a signal x is processed by a shorter filter w, which is a set of weights. During convolution, the filter w is slid along the signal x, with the element-wise multiplication of the filter and the segment of the signal filter covers being computed at each position.

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \quad (2)$$

Mathematically, the convolution operation is depicted by Goodfellow et al. (2016, p. 327) as Equation 2, where the asterisk (*) denotes convolution. In this equation, t represents the index of the element in the feature map that is being calculated. Importantly, the filter w is shorter than the signal x, which leads to an interesting property in the equation: an infinite sum calculated in a finite space (Goodfellow et al., 2016, p. 328). This is because outside the bounds of the filter, w is zero, resulting in any multiplication involving w outside its bounds also being zero. Another notable property involves $t - a$, which suggests that the filter is reversed, a process often called flipping (Goodfellow et al., 2016, p. 328). However, Goodfellow et al. (2016, p. 329) note that in machine learning implementations, this flipping is often not performed, and the operation should more accurately be called cross-correlation, though it is still commonly referred to as convolution.

2.2.2 Dataset

The dataset consists of the representation or features of the data and plays a crucial role in the model's performance (Bengio et al., 2012). This connection between representation and performance has inspired researchers to develop algorithms capable of representation learning, which has limited the requirement for feature engineering (Bengio et al., 2012). Feature engineering refers to a process where the dataset is manually configured into a form that is more acceptable for the model (Bengio et al., 2012). The reduced need for manual labour has speeded up the process of utilising artificial

intelligence (Bengio et al., 2012). In its essence, representation learning refers to a concept in which the neural network is presented with raw data and during training, it automatically learns the meaningful features of the data (Bengio et al., 2012). There are multiple different ways to make a neural network learn, as described in the book Deep Learning by Goodfellow et al. (2016, pp. 103-104).

Two of the most prominent ones are called supervised and unsupervised learning. Although there is no formal definition, supervised learning typically involves solving regression and classification problems, while unsupervised learning aims to understand the underlying probability distribution of the data (Goodfellow et al., 2016, p. 103). One way to understand this is that in supervised learning, the model is given a label y in addition to the data x, and it tries to do classification by learning the probability $p(y|x)$ (Goodfellow et al., 2016, p. 103). In the unsupervised learning process, the model tries to learn the underlying probability distribution $p(x)$ automatically (Goodfellow et al., 2016, p. 142). The model can then be used in, for example, anomaly detection, where deviations from the expected distribution can signal atypical events (Goodfellow et al., 2016, p. 100).

A third learning method derived from the above-mentioned approaches is self-supervised learning, which is usually associated with more complex deep neural networks (Ericsson et al., 2022). Ericsson et al. (2022) have divided self-supervised learning into a pretext task and a downstream task. In the pretext task, the model uses unsupervised learning to capture a meaningful data representation, for example, in a lower dimension (Ericsson et al., 2022). The downstream task then utilises this new domain for improved learning (Ericson et al., 2022). The related works chapter discusses OpenAI's music generation model, Jukebox, which can be thought to represent this learning method where Vector Quantized Variational Autoencoder (VQ-VAE) training is considered the pretext task and Scalable Transformer training matches the downstream task description.

2.2.3 Objective Function

An objective function is a mathematical function that guides a machine learning model to adjust its weight and bias parameters in an attempt to minimise or maximise the objective function (Goodfellow et al., 2016, p. 80). In supervised learning, the goal is often to minimise the difference between inputs and outputs, and this is measured with a cost function, also known as a loss function (Goodfellow et al., 2016, p. 80; Nielsen, 2015, p. 16).

In regression tasks, also known as quantitative tasks, the Mean Squared Error (MSE) is widely used to quantify the average of the squares of the errors, effectively measuring the variance between estimated and actual values (James et al., 2023, p. 28). In contrast, for classification tasks or qualitative tasks, Cross-Entropy Loss is frequently employed as it quantifies the divergence between the actual labels and the predicted labels and results in faster convergence compared to Mean Squared Error (James et al., 2023, p. 28; Nielsen, 2015, p. 63). Equation 3 depicts MSE and in the equation y_i is the actual value, and $\bar{y}_i$ is the predicted value.

$$MSE = \frac{1}{n} \sum_i^n (y_i - \bar{y}_i)^2 \tag{3}$$

As the cost function is used to direct learning, it can be thought that decreasing cost is a sign of learning (James et al., 2023, p. 28). However, this is not always the case, as machine learning models often suffer from overfitting, a phenomenon where the model learns the training data very well but fails to generalise effectively to new, unseen test data (Nielsen, 2015, p. 75; Goodfellow et al., 2016, pp. 109). More specifically, it can occur when the model parameter count is high and the amount of training data is low (Nielsen, 2015, p. 74). Luckily, it is not necessarily a sign that the model is inherently unable to learn, as prolonged training can be the cause of overfitting (Nielsen, 2015, p.75).

To prevent overfitting, regularisation techniques such as L2 regularization are introduced into the cost function (Nielsen, 2015, p. 79). Regularization tries to ensure the model does not overly adapt to the noise within the training data (Nielsen, 2015, p. 84). In L2 regularization, a regularization term is summed to the cost function, and in a machine learning setting, it is often squared L2 norm depicted in Equation 4, where λ is the regularization parameter, which balances how well the model fits the data and how diverse the weight domain gets (Nielsen, 2015, p. 79; Goodfellow et al., 2016, pp. 117, 227).

$$\lambda \|w\|_2^2 = \lambda \sum_w w^2 \tag{4}$$

Nielsen (2015, p. 86) states that there is no entirely convincing theoretical explanation that explains why regularization works. Regularization simplifies the network, and that is often offered as a general scientific principle as to why it works, but Nielsen (2015, p. 85) points out that simpler does not always equal better. Goodfellow et al. (2016, pp. 117-118) talk about the importance of domain knowledge when designing machine-learning solutions and how excessive regularization can hinder the model's ability to learn and lead to underfitting. Underfitting is the opposite of overfitting, and it occurs when the model is unable to learn the training data (Goodfellow et al., 2016, p. 109). The use of regularization boils down to a bias-variance trade-off described by James et al. (2023, pp. 242-243), where increased regularization decreases variance but increases bias. Based on the above, it can be derived that the goal of regularisation is to restrict the model's capacity to overfit while still having a low enough bias that the model does not underfit. Finding this balance is crucial when trying to achieve the best possible generalization.

More complex neural networks have different types of regularisation methods, one of which is Kullback-Liebler (KL) Divergence, which is used as a regularisation term in VAEs (Goodfellow et al., 2016, p. 693). It penalises deviations from expected probability distributions, ensuring desirable properties such as continuity for the posterior distribution

(Goodfellow et al., 2016, p. 72). These cost functions are foundational to the optimisation procedure, which is discussed in the subsequent section.

2.2.4 Optimisation Procedure

An optimisation procedure is generally a very difficult task, and it is also the fourth element of a machine learning algorithm described by Goodfellow et al. (2016, pp. 151, 279). It refers to the process of minimising or maximising the objective function $f(x)$ by optimizing x (Goodfellow et al., 2016, p. 80). One of the most prominent optimisation procedures is gradient-based optimisation (Goodfellow et al., 2016, p. 80). Previously, gradient-based optimization was described as "slow or unreliable", but since it has been accepted that it provides useful results in a reasonable time even though it does not always give the optimal solution (Goodfellow et al., 2016, p. 150). In other words, gradient descent converges to a local minimum or close to it but seldom finds the global minimum. Gradient descent is a process that utilises partial and directional derivatives to calculate the gradient $\Delta_x f(x)$ and the objective is to determine the direction that decreases $f(x)$ the most rapidly (Goodfellow et al., 2016, pp. 82-83). Gradient shows the direction of the steepest ascent (Goodfellow et al., 2016, p. 83). Equation 5 describes the optimisation of x by nudging it in the direction of the negative gradient (downhill). The coefficient ϵ is called the learning rate, and it determines the step length for the optimisation process (Goodfellow et al., 2016, p. 84). Usually, the learning rate is a small constant (Goodfellow et al., 2016, p. 84).

$$x' = x - \epsilon \Delta_x f(x) \tag{5}$$

Nowadays, the machine learning field is dominated by the stochastic gradient descent (SGD) algorithm (Goodfellow et al., 2016, p. 150). SGD is an extension of basic gradient descent, and its existence becomes obvious when larger and larger training sets are introduced to improve generalization (Goodfellow et al., 2016, p. 149). The problem with regular gradient descent is that when the amount of data x grows so does the computational cost (Goodfellow et al., 2016, p. 149). This is represented with $O(x)$ which means

that the cost is linear (Goodfellow et al., 2016, p. 149). SGD solves this problem by drawing a uniform representation from the data called a minibatch and doing the gradient calculation on that limited number of samples (Goodfellow et al., 2016, p. 149). This leads to the computational cost becoming independent of the amount of data, and it is denoted with $O(1)$, meaning that the computational cost is constant. (Goodfellow et al., 2016, p. 149).

2.3 Neural Network Architectures

Neural network architectures encompass a wide range of models that are based on the idea of deep neural networks, yet models based on these architectures differ in their objectives and applications. While some neural network architectures, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), are designed to generate new data based on learned probability distributions, other architectures, such as standard Autoencoders, focus on data reconstruction (Bengesi et al., 2023; Goodfellow et al., 2016, p. 499). This chapter will discuss key neural network architectures, including Autoencoders, Variational Autoencoders, Generative Adversarial Networks, and Transformers, highlighting their purposes and underlying mechanisms.

2.3.1 Autoencoders

Autoencoders are a neural network architecture focused on unsupervised learning tasks, including dimensionality reduction and feature extraction (Goodfellow et al., 2016, p. 499). The foundation for autoencoders was laid by Rumelhart, Hinton, and Williams in 1986 with the introduction of backpropagation, enabling neural networks to learn internal representations (Rumelhart et al., 1986). The concept of autoencoders as a specific neural network structure was later advanced by researchers such as LeCun (1987), Bourlard and Kamp (1988), and Hinton and Zemel (1994) (Goodfellow et al., 2016, p. 499). Typically, an autoencoder comprises three main components: the encoder, a bottleneck or latent vector, and the decoder (Goodfellow et al., pp. 499-500). This structure is similar to the one illustrated in Figure 3. As the data flows through the network (from left to

right in Figure 3), the encoder reduces the size of the input data to fit through the bottleneck of the latent space, and the decoder tries to reconstruct the original input from this representation (Goodfellow et al., 2016, p. 499).

Training an autoencoder involves using backpropagation and gradient descent to minimise the reconstruction error between the input and the output (Goodfellow et al., 2016, p. 499). This process allows the neural network to capture essential features of the data while discarding irrelevant information (Rumelhart et al., 1986; Goodfellow et al., 2016, p. 499). Goodfellow et al. (2016, p. 499) highlight that learning results can be improved by feeding the network incomplete input data and calculating the error with reconstructed data and complete input data.

Autoencoders are widely used for various purposes, including noise reduction, where they learn to reconstruct clean data from noisy inputs (Vincent et al., 2008; Goodfellow et al., 2016, p. 499). They are also applied in anomaly detection, where deviations between the input data and its reconstruction indicate unusual patterns, making them valuable for identifying outliers in data (Chalapathy & Chawla, 2019). Moreover, autoencoders serve in dimensionality reduction, compressing high-dimensional data into a more manageable form and aiding in tasks such as data visualisation and feature extraction (Hinton & Salakhutdinov, 2006). These applications highlight the versatility of autoencoders across various unsupervised learning tasks.

2.3.2 Variational Autoencoders

The concept of Variational Autoencoding was introduced in the paper Auto-Encoding Variational Bayes by Kingma and Welling in 2014. It was developed as a general solution for problems with intractable posterior distributions $p(z|x)$ in which parameters or latent variables are continuous. The proposed solution utilises the stochastic gradient variational Bayes estimator to optimise the approximate posterior distribution $q(z|x)$.

As described in van den Oord et al.'s 2017 paper Neural Discrete Representation Learning, a Variational Autoencoder consists of practically three parts: an encoder, a latent space, and a decoder (illustrated by Figure 3), where the latent space is often continuous (van den Oord et al., 2017). These components map probability distributions that are called a posterior distribution $q(z|x)$, a prior distribution $p(z)$, and $p(x|z)$ probability distribution (van den Oord et al., 2017). Kingma and Welling (2013) refer to $q(z|x)$ as a probabilistic encoder and $p(x|z)$ as a probabilistic decoder. Initially, a prior distribution $p(z)$ is defined, representing the latent space's expected shape or form, and it directs the posterior distribution in a specific direction; usually, the prior is standard Gaussian (van den Oord et al., 2017).

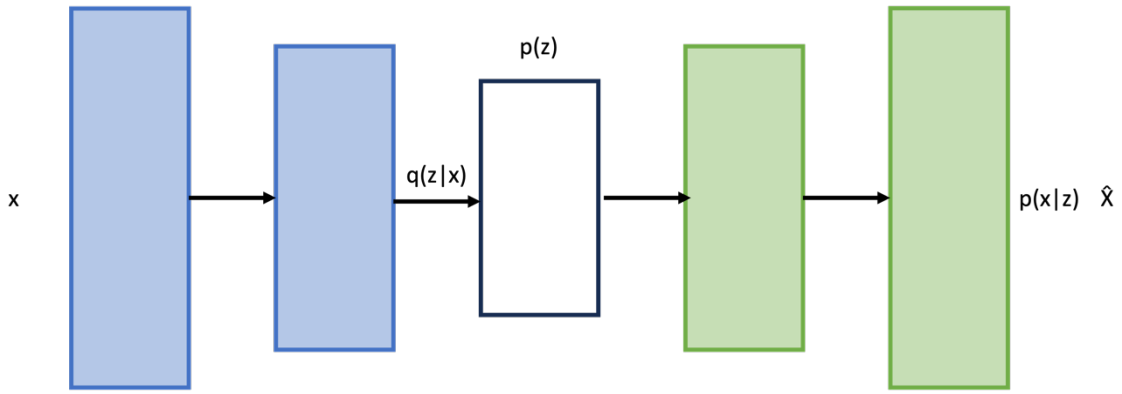


Figure 2. Variational Autoencoder depicted as a figure. Blue boxes represent the encoder, white is the latent space, and green boxes represent the decoder. $\boldsymbol{x}$ is input, and $\overline{\boldsymbol{x}}$ represents output.

During the Variable Autoencoder's training, the model learns to refine the mappings of the encoder and decoder by optimising the parameters that define these conditional probability distributions $q(z|x)$, the probability of z given x, and $p(x|z)$, likelihood of observing x given z (van den Oord et al., 2017). To enable effective training through stochastic gradient descent, the latent variable z is transformed into a deterministic function $g_\phi(\epsilon, x)$, parameterised by ϕ, and an independent noise variable ϵ , sampled from a standard distribution (Kingma & Welling, 2013). This method, called the reparameterization trick by Kingma and Welling (2013), allows for gradient flow during backpropagation. This can also be visualised with the following code snippet of the

reparameterization function, where mu and log_var are the ϕ parameters and eps is the noise variable ϵ.

```
def reparametrize(self, mu, log_var):
     std = np.exp(0.5 * log_var)
     eps = np.rand_like(std)
     return mu + eps * std
```

VAEs have been applied in numerous fields, such as image processing, medical applications, and language modelling (Wei et al., 2020). In the image processing category, it is considered to be state-of-the-art in image classification, image compression, and image resolution (Bengesi et al., 2023). More recent advancements have been made in the 3D imaging domain, as VAE enables the efficient compression of high-dimensional spaces (Molnár & Tamás, 2024).

In 2017, van den Oord et al. introduced a variant of VAEs called a Vector Quantized Variational Autoencoder (VQ-VAE). Unlike traditional Variational Autoencoders, which have a continuous latent space, VQ-VAEs have a discrete latent space. This type of autoencoder uses a codebook in the quantization process, which makes the latent space discrete (van den Oord et al., 2017). VQ-VAEs are discussed more in-depth in the related works chapter.

2.3.3 Generative Adversarial Networks

As previously mentioned, the Generative Adversarial Network was first developed by Goodfellow et al. (2014). The architecture of GAN comprises two neural networks that compete with each other. The first network is called a generator, denoted by G, which aims to generate convincing samples that can deceive the second network. The second network is known as the discriminative network depicted by D, which aims to differentiate between generated samples and real data (Goodfellow et al., 2014). GAN architecture is visualised in Figure 2.

Training the model's two separate networks happens simultaneously (Goodfellow et al., 2014). Convergence is considered to be reached when the discriminator network classification probability approaches 0.5, and the generator network's probability distribution resembles the distribution of the real data (Goodfellow et al., 2014). The advantages listed by Goodfellow et al. (2014) are mainly computation improvements compared to previous models. A unique aspect of the model that sets it apart from other generative architectures discussed in this study is that the generator network is not directly exposed to embeddings of real data, which, according to Goodfellow et al. (2014), may lead to some statistical advantages.

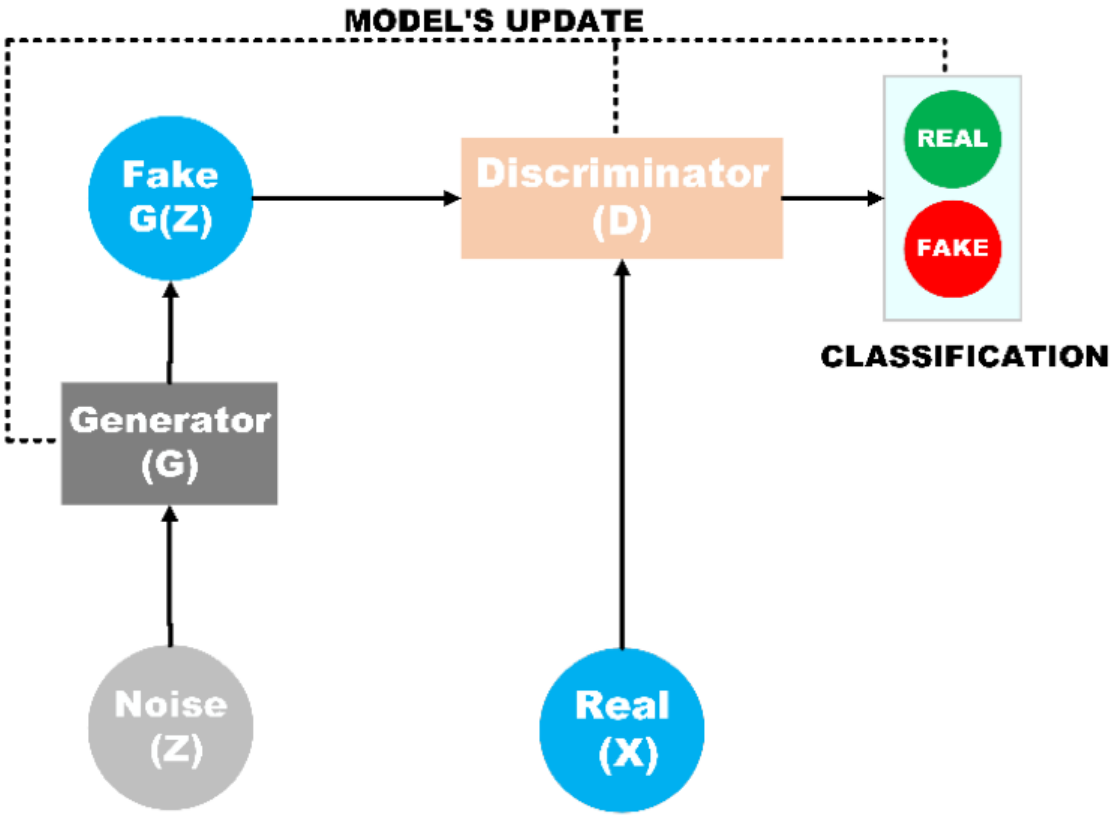


Figure 3. Generative Adversarial Network architecture depicted by Bengesi et al. (2023).

Generative Adversarial Networks (GANs) have been extensively studied since, as evidenced by Jabbar et al.'s 2020 survey on Generative Adversarial Networks: Variants, Applications, and Training. Applications include image generation in the form of hand-

written font, image blending, texture synthesis, and 3D image synthesis (Jabbar et al., 2020). Other notable implementations mentioned by Jabbar et al. (2020) are music generation, video synthesis, and applications in the medical field.

2.3.4 Transformers

Transformers were initially developed by Vaswani et al. (2017) in a study titled Attention Is All You Need. The transformer networks consist of encoder and decoder networks, each forming a stack of N identical layers. The fundamental unit of these layers is known as the Attention Head, which is responsible for updating each embedding based on its relationship with surrounding embeddings (Vaswani et al., 2017). In the paper, Vaswani et al. (2017) describe stacking these attention heads to create Multi-Headed Attention. An overview of the Transformer model architecture is shown in Figure 5.

Even though the Transformer model's architecture is similar to the previously mentioned Variational Autoencoder architecture, which consists of encoder and decoder networks, there are some interesting differences worth examining a little further. In a VAE, the encoder network is only required during the training phase, and the inference happens by sampling latent space and using the decoder to decode the sampled embeddings (Kingma & Welling, 2013). However, in a Transformer network, as visible in Figure 5, the encoder is connected to the decoder so that input embeddings serve as context throughout the autoregressive decoding process (Vaswani et al., 2017).

The most interesting and maybe the most ground-breaking output from Vaswani et al.'s (2017) study was the attention mechanism, which they refer to as Scaled Dot-Product Attention. In the study, the form of attention is more broadly called self-attention, which means that the attention head computes the representation of each element in a sequence by considering how it relates to every other element in the same sequence (Vaswani et al., 2014). In contrast to Recurrent Neural Networks (RNN), which rely on sequential processing, this attention method allows each position to attend independently

to all positions simultaneously, making the model more computationally efficient (Vaswani et al., 2017).

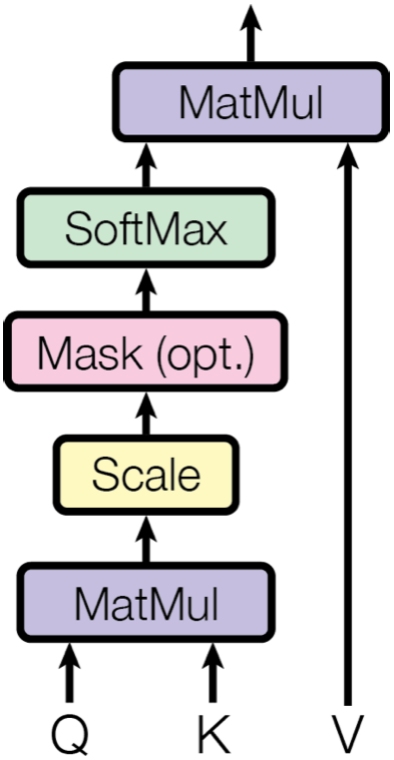


Figure 4. Scaled Dot-Product Attention architecture (Vaswani et al. 2017).

Figure 4 visualises the architecture of Scaled Dot-Product Attention. To better understand the process depicted in Figure 4, it is helpful to go through it step-by-step. Figure 5 shows how input is transformed into input embeddings and enhanced with positional encoding. For each of the embeddings, the process depicted in Figure 4 is performed. The variable Q represents a query matrix that is a product of all the embeddings E_T multiplied by W_Q, and similarly, K represents a key matrix and equals E_T multiplied by W_K and then matrix multiplication is calculated between Q and K (Vaswani et al., 2017).

According to Vaswani et al. (2017), the next step of scaling the result with $\sqrt{d_k}$ is what sets this method apart from regular Dot-product attention. In the study, Vaswani et al. (2017) talk about the theoretical complexity of additive attention and dot-product attention being similar and justify the use of dot-product due to it being faster because of

optimised matrix multiplication calculations. However, with sufficiently large d_k the additive attention outperformed the dot-product attention, introducing the need for scaling to prevent the vanishing gradient problem in the softmax layer (Vaswani et al., 2017).

The next step is a masking operation that is only done on the decoder side, as shown in Figure 5. During the autoregressive process, this prevents the decoder from attending to future data that otherwise would influence the present prediction (Vaswani et al., 2017). This is achieved by replacing the result of the matrix multiplication between Q and K with negative infinity for all the connections onwards from E_t, where t is the index of the embedding currently being processed (Vaswani et al., 2017). This adjustment ensures that during the softmax calculation, which produces the weights in the form of a probability distribution, only the desired weights are assigned a coefficient of zero, effectively disregarding their influence (Vaswani et al., 2017). The full context is maintained for the encoder side, where the masking operation is skipped (Vaswani et al., 2017).

The final step in the process illustrated in Figure 4 involves the matrix multiplication of the weight matrix with the value matrix V, which is the product of E_T and W_V. This operation yields ΔE_T, representing the direction in which the original embedding E_T should be adjusted. In the Multi-Head Attention model, there are h attention heads, each generating a ΔE_T. These matrices are concatenated to determine the unified direction for adjusting the original embeddings, enhancing the model's capacity to integrate various contextual insights (Vaswani et al., 2017).

The most famous transformer adaptation is probably Generative Pre-trained Transformer (GPT), which was released in 2018 by OpenAI (Bengesi et al., 2023). It is a large language model aiming to generate human-like text, and multiple versions have been released since (Bengesi et al., 2023). Other notable adaptations of transformer architecture are Bidirectional Encoder Representations from Transformer (BERT) and Vision Transformer (ViT), which have produced state-of-the-art results in Natural Language Processing (NLP) and in Computer Vision (CV) tasks, respectively (Chitty-Venkata et al., 2023).

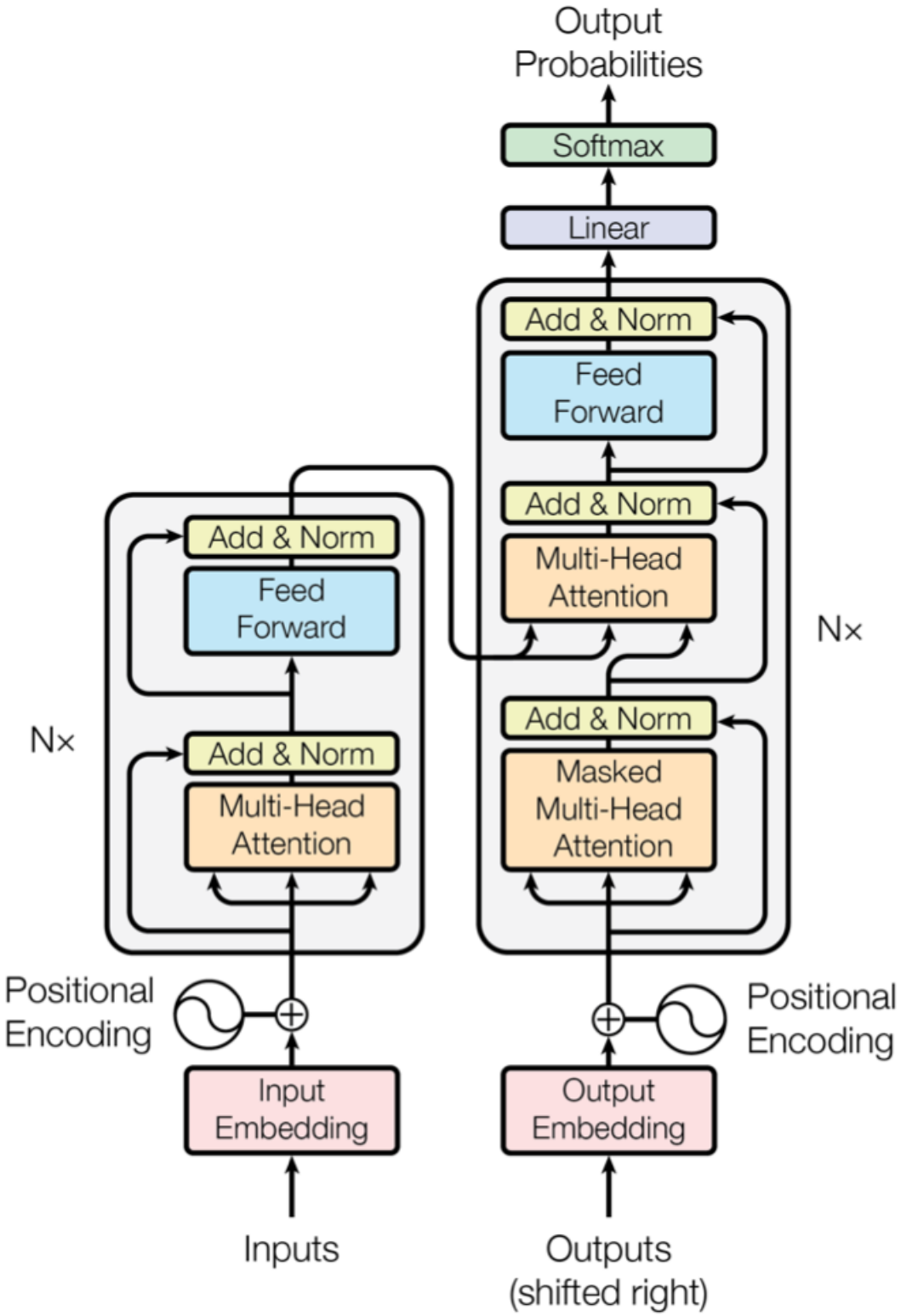


Figure 5. Transformer architecture (Vaswani et al. 2017). The figure consists of an encoder on the left and a decoder on the right.

3 Related Work

With the relatively recent increase in computational power and some breakthroughs in the adjacent field of image generation, AI-generated music has started to take bigger and bigger leaps forward. Some of the most notable papers in the context of this thesis will be discussed in detail below. Common to all these models, in the spirit of this thesis, is the use of raw audio data as input and output for the neural network, along with the incorporation of convolutions. To gain a deeper understanding of AI-generated music, it is essential to explore the key research problems that have shaped the field.

Goel et al. (2022) identify three major challenges encountered by researchers when designing architectures to model waveforms. The first challenge is maintaining global coherence in the modelled waveform, which requires the neural network to effectively capture long-range dependencies (Goel et al., 2022). This is supported by Dieleman et al. (2018) as they state that attaining globally coherent music has proven difficult. The second challenge discussed by Goel et al. (2022) is computational efficiency. High-fidelity audio involves orders of magnitude more input parameters than a simple image, as explained in the introduction chapter. For instance, the models in Dhariwal et al.'s (2020) Jukebox were trained for two to four weeks using up to 512 GPUs, a feat achievable only by the most well-funded research initiatives even by today's standards. The third challenge is sample efficiency, which is closely related to computational efficiency. According to Goel et al. (2022), sample efficiency refers to the model's ability to converge with fewer training samples, thereby enhancing overall computational efficiency. By improving sample efficiency, the model can achieve effective performance with less data, reducing the computational resources and time required for training.

3.1 WaveNet

In 2016, Google DeepMind research laboratory released WaveNet, a deep neural network model capable of generating music. As described in the 2016 paper by van den Oord et al., the model is autoregressive and probabilistic, meaning that each newly

generated audio sample is conditioned by its priors. More concretely, the model calculates a conditional probability distribution from which each new output is sampled. As previously mentioned, the probability distribution is conditioned by all previous samples. All these probability distributions are joined to form the joint probability of waveform x, as observed in Equation 1.

$$p(x) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1}) \tag{1}$$

To achieve this functionality, van den Oord et al. (2016) depict a stack of causal convolution layers which model the conditional probability distribution.

3.1.1 Causal and Dilated Convolutions

One-dimensional causal convolutions, as described by van den Oord et al. (2016), are different from standard 1D convolutions. In causal convolutions, padding of size K-1, where K is the kernel size, is applied asymmetrically to the past or left side of the sequential input data. This practice prevents the model from accessing future information when predicting present data, as shown in Figure 6.

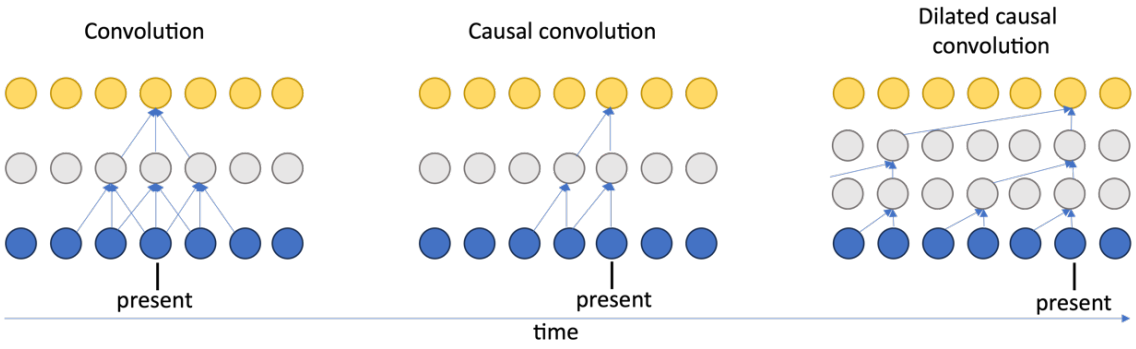


Figure 6. Different types of convolutions. Blue dots depict the input layer, grey marks the hidden layers, and yellow is the output layer.

WaveNet not only utilises causal convolutions but takes it one step further by implementing dilated causal convolutions, where the dilation doubles between layers (van

den Oord et al., 2016). This technique expands the model's receptive field, enabling it to capture longer temporal dependencies in audio sequences, which is vital for generating coherent and realistic sound over extended periods. Additionally, it is more computationally efficient and maintains the same input and output size, which is not the case with other convolution techniques like strided convolution that can be used for capturing longer temporal structures.

3.1.2 Non-linear Quantisation

Another notable technique in terms of music generation that van den Oord et al. (2016) discuss is performing a non-linear quantisation for the input signal. Their proposed method involves transforming the input signal using the μ-law companding technique visible in Equation 2, where $-1 < x_t < 1$ and $\mu = 255$, and quantising the data into 256 possible values. As a result, the data has a higher resolution on lower amplitudes due to its logarithmic nature (ITU-T, 1988).

$$f(x_t) = sign(x_t)\frac{\ln(1+\mu|x_t|)}{\ln(1+\mu)} \tag{2}$$

To tie it together, van den Oord et al. (2016) utilise a softmax distribution to predict the likelihood of each of the 256 quantised values. Based on this, the most probable value can be selected. Alternatively, to increase variety, the distribution can be sampled. The benefit of quantisation is clear: It simplifies the representation of audio samples, reducing the data complexity from 16-bit to 8-bit per sample, thereby decreasing the softmax distribution's output space and computational cost.

In contrast to previous studies, van den Oord et al. (2016) decided to use a gated activation function instead of a rectified linear activation function as it produced better results when modelling audio signals. Their gated activation unit consists of two adjacent convolutional layers, filter and gate, with respective tanh and sigmoid activation functions.

In addition, the units contain residual and skip connections, making the model convergence faster and allowing the use of deeper networks.

3.2 Jukebox

The next big leap in AI-generated music came in 2020 when OpenAI researchers Dhariwal et al. released a Jukebox model. The research listed its achievements as generating multiple-minute songs that stay coherent and contain singing. The actual model consists of three layered VQ-VAEs, Vector Quantized Variational Autoencoders, which learn to encode the music into embeddings. For inference, Dhariwal et al. (2020) used autoregressive Scalable Transformers. When generating music with the model, it is possible to prime it with text or audio data.

3.2.1 Multi-scale VQ-VAE

The Jukebox model comprises three VQ-VAEs operating on different temporal scales. Each VQ-VAE is trained separately to prevent the model from relying solely on the one that learns the highest temporal resolution (Dhariwal et al., 2020). Each VQ-VAE utilises WaveNet-like 1D convolutions mirrored for the encoder and decoder, intending to increase the model's receptive field (Dhariwal et al., 2020).

In Jukebox, Dhariwal et al. (2020) describe the quantisation process of a one-dimensional VQ-VAE where an input signal $x = \{x_1, \dots, x_T\}$ is learnt to encode with $z = \{z_1, \dots, z_S\}$ indices, where T is the input length, S is the count of indices, and T/S is the hop length, indicating the level of dimension reduction. During the encoding process, x is transformed into latent vectors $h = \{h_1, \dots, h_S\}$, and each latent vector is mapped to the nearest codebook vector $e_{z_s} \in C = \{e_1, \dots, e_K\}$,where K is codebook size. This results in the discretisation of the latent space.

For the training, Dhariwal et al. (2020) use a loss sum that consists of three separate loss functions. The first one is trivial reconstruction loss calculated between x and decoded codebook vector sequence as described in Equation 3.

$$\mathcal{L}_{reconstruction} = \frac{1}{T}\sum_t \|x_t - \overline{x}_t\|_2^2 \tag{3}$$

The other two loss functions are called codebook loss and commit loss. Both utilise the stop gradient (sg) function, which essentially means that in backpropagation, the sg vector is locked in place as the gradient is set to zero. The codebook loss penalises the model when the codebook vector e_{z_s} is far from the encoded vector h_s and in commit loss, it is the other way around, and the model is penalised if the encoded vector h_s is far from the codebook vector e_{z_s}. The respective loss functions are detailed in Equations 4 and 5 (Dhariwal et al., 2020)

$$\mathcal{L}_{codebook} = \frac{1}{S}\sum_s \left\| sg[h_s] - e_{z_s} \right\|_2^2 \tag{4}$$

$$\mathcal{L}_{commit} = \frac{1}{S}\sum_s \left\| h_s - sg[e_{z_s}] \right\|_2^2 \tag{5}$$

In combination, these loss functions produce the total loss function for the model detailed in Equation 6. Dhariwal et al. (2020) state that $\mathcal{L}_{commit}$ is added so that the model would try to constrain the values of h_s closer to possible e_{z_s} values and is said to have a stabilizing effect. The weight of commit loss can be controlled with the β value.

$$\mathcal{L} = \mathcal{L}_{reconstrcution} + \mathcal{L}_{codebook} + \beta\mathcal{L}_{commit} \tag{6}$$

In the paper, Dhariwal et al. (2020) discuss a problem in which the model only learns to reconstruct low frequencies when using sample-level reconstruction loss. The proposed solution uses the Short-Time Fourier Transform (STFT), which helps the model to learn

mid-to-high frequencies. The resulting loss is called spectral loss and is defined in Equation 7.

$$\mathcal{L}_{spectral} = \|STFT(x_t) - STFT(\overline{x}_t)\|_2 \quad (7)$$

VQ-VAEs have some known problems, one of which is codebook collapse. This means that most of h_s get mapped to only a few of e_{z_s}. To mitigate this problem, Dhariwal et al. (2020) introduce random restarts, which means that if the average usage of an embedding is too low the e_{z_s} vector gets randomly replaced by one of the h_s vectors.

3.2.2 Scalable Transformers and Upsampling

In addition to the VQ-VAE neural networks, Jukebox consists of autoregressive transformers for the actual music inference. In the paper, these transformers are divided into a top-level prior and upsamplers called middle and bottom. The prior $p(z)$ is a joint conditional probability distribution, with each component similar to Equation 1. The complete distribution is defined in Equation 8 (Dhariwal et al., 2020).

$$p(z) = p(z_{top})p(z_{middle}|z_{top})p(z_{bottom}|z_{middle}, z_{top}) \quad (8)$$

The approach chosen by Dhariwal et al. (2020) simplifies the prediction task, as it occurs in a discrete space and can be categorised as a classification problem instead of a regression problem. Basically, the transformers predict codebook indices autoregressively.

In the model, inference with transformers happens top-down, and each prediction layer consists of the same number of discrete codes that map to shorter and shorter segments in the raw audio domain. Upsamplers are conditioned only with the previous layer's discrete codes that match the raw audio length of the current layer. These codes go through a conditioning layer that uses dilated convolutions, similar to WaveNet, leading to an increase in the number of discrete codes after each layer, recreating the lost information and resulting in increased audio resolution (Dhariwal et al., 2020).

To decrease entropy, Dhariwal et al. (2020) encode artist and genre to embedding vectors. This can also be used to direct the model during the generation phase. They also provide the model with the total length of the audio signal and the start and end times of the current segment.

3.3 Summary

Both studies claimed state-of-the-art performance and results at the time of release. Even if this is true, it doesn't necessarily provide a complete understanding of the models' actual capabilities. Therefore, it is valuable to analyze both models in light of the challenges highlighted by Goel et al. (2022). One general concern that can be made from this line of study is the lack/difficulty of directing the generation in the wanted direction. Compared to other models that use prompt-based sequence-to-sequence generation like MusicLM (Agostinelli et al., 2023), it can be more difficult to direct the generation in the desired direction, especially into novel directions when existing audio does not exist, and hence it cannot be used as a prior. Also, it is important to note that although WaveNet possesses music generation capabilities and those are talked about in the study, the main focus of the study was on text-to-speech generation.

The first challenge brought up by Goel et al. (2022) is the requirement for global coherent generation. Jukebox’s approach clearly enables it to obtain some longer-range structure and maintain it throughout the generated sample, but at the cost of the audio's fidelity. This observation is aligned with previous research, as Dieleman et al. (2018) noted that using hierarchal autoregressive inference led to improved long-range structure and decreased signal quality, indicating a trade-off. Similarly, WaveNet's application to music generation demonstrated the importance of a large receptive field to produce samples that sounded musical (van den Oord et al., 2016). Despite this, the models struggled with long-range consistency, resulting in second-to-second variations in genre, instrumentation, and volume (van den Oord et al., 2016). However, the generated samples were often harmonic and aesthetically pleasing, particularly when conditional models were

used to control specific aspects of the output based on tags like genre or instruments (van den Oord et al., 2016).

The assessment of Jukebox compliance with the second point raised by Goel et al. (2022) reveals that computational efficiency is still a problem when training neural networks to model audio waveforms. In their paper, Dhariwal et al. (2020) mention four different training events: the two upsampler networks were trained with 128 GPUs for 2 weeks, top-level prior training took 4 weeks with 512 GPUs, and lastly, the lyrics conditioning training they performed with 128 GPUs for a total of 2 weeks. These numbers highlight significant monetary and time constraints for conducting research on this subject. The cost estimate for the GPUs alone amounts to millions of dollars, approximately 6.4 million USD, and does not include other necessary hardware expenses for building a system capable of training these networks.

The computational demands of using the original WaveNet model are significant due to its autoregressive nature, which requires generating audio one sample at a time (van den Oord et al., 2016). While the original WaveNet paper by van den Oord et al. (2016) does not discuss the hardware or training durations in detail, additional insights can be obtained from the 2017 paper "Parallel WaveNet: Fast High-Fidelity Speech Synthesis" by van den Oord et al. (2017). In this paper, the authors state that the re-engineered WaveNet's inference was 1,000 times faster than the original, with the ability to generate one second of audio in just 50 milliseconds (van den Oord et al., 2017). This implies that the original model took approximately 50 seconds to generate one second of audio. Based on the fact that generating one second of audio took 50 seconds and that further research was conducted to improve this, it can be deduced that even though the training of the original WaveNet was efficient, the total time it took to use it made it computationally inefficient, not overcoming the second challenge defined by Goel et al. (2022).

The last challenge mentioned by Goel et al. (2022) is sample efficiency, closely intertwined with computational efficiency as both affect the model’s performance. Sample

efficiency aims to achieve better performance through inductive biases (Goel et al., 2022). This means that the model's learning can be enhanced by making the right design and architectural choices (Hüllermeier et al., 2013, p. 1018). In the case of WaveNet, an example is its autoregressive inference, which assumes that each new sample depends on the previous ones (van den Oord et al., 2016). For humans, it is intuitive that in music, each note depends on the previous ones, but for machines, this connection can be challenging to learn without the aid of inductive bias. Similarly, Jukebox employs a hierarchical VQ-VAE architecture, which introduces inductive biases by modelling music at multiple levels of abstraction. This approach allows Jukebox to efficiently capture both the long-term structure and the fine details of music, enhancing its sample efficiency (Dhariwal et al., 2020).

4 Methodology

In this thesis, the chosen methodological approach is a controlled experiment. As the term implies, the objective is to establish a controlled environment where the experiment can be conducted (Järvinen, 2018; Walliman, 2010, p. 11). The fundamental purpose of experimental research is to establish causality by carefully controlling and manipulating variables; researchers can isolate specific factors and observe their direct impact on outcomes (Järvinen, 2018; Walliman, 2010, p. 103). This approach is used to test hypotheses, validate theories, and contribute to the body of knowledge in a systematic and replicable manner that allows making informed decisions based on empirical evidence.

In Järvinen's 2018 study On Research Methods, two critical factors for the new knowledge are identified: the necessity for rich and applicable knowledge and the need for reliable knowledge. These requirements often conflict. Specifically, when designing research, imposing numerous constraints to enhance reliability can strip the experimental context of factors that connect it to real-world conditions (Järvinen, 2018). Consequently, achieving a balance between these two factors is crucial to obtaining valuable insights from an experiment (Järvinen, 2018).

The basic terminology of controlled experiments includes dependent variables, independent variables, and intervening variables (Järvinen, 2018). The dependent variable(s), as defined by Järvinen (2018), represent the quantitative outcomes of the study, measuring aspects that indicate improvement or deterioration, a definition corroborated by Walliman (2010, p. 11). The independent variable(s) are the parameters controlled by the researcher, with the premise that manipulating these independent variables should result in a measurable change in the dependent variable(s) attributable to the independent variable(s) (Järvinen, 2018; Walliman, 2010, p. 11). The intervening variable(s) are those that are not under the control of the researcher but still affect the dependent variable and hence cannot be excluded from the research (Järvinen, 2018). The variables

that do not fall into these categories are commonly referred to as unknown variables, which are assumed not to have an impact on the research (Järvinen, 2018).

Experimental research can be enhanced with a control experiment in addition to the main experiment to increase the certainty that these unknown variables do not affect the result (Järvinen, 2018). This is done to rule out that arriving at the result truly was caused by the manipulation of independent variables and not by other unknown variables or other factors (Järvinen, 2018). Two additional criteria, listed by Järvinen (2018), that can be evaluated to gain more proof of the causality are association or relationship and temporal precedence. The association or relationship evaluation relies on proving the existence of covariance between independent and dependent variables, i.e., change in the independent variable shows reliable and observable change in the dependent variable (Järvinen, 2018). Proving the temporal precedence of events means that in order for two things to be causally linked, the change in the dependent variable must always happen after the change in the independent variable (Järvinen, 2018).

The practical implementation of this research involved conducting a controlled experiment to test whether a convolutional architecture could improve an autoencoder's ability to learn audio data representation. The process began with building a baseline model—a fully connected autoencoder with sufficient performance for comparison. Following this, a convolutional autoencoder was created to evaluate the effects of convolutional layers on the model's performance. Both models were trained from zero until convergence, and the results were analyzed to assess differences in performance, model size, and output quality.

5 Experiment Design and Model Evolution

This research is focused on testing a hypothesis through a controlled experiment. It involves comparing two models to determine if convolution can improve the model's ability to learn audio data representation. The first step was to create a prototype model with a decent baseline performance, against which the convolution model could be compared. The next step was to create a convolution model and, thirdly, analyse the differences in performance, size, and quality of the result.

The initial idea was to study the effects of multidimensional convolution on AI-generated music. The prototyping phase turned out to be the most time-consuming part of the thesis work. Due to time constraints, the research scope had to be adjusted as the prototyping phase continued. The experiment also faced constraints imposed by the system used for testing, which led to the decision to omit the generative functionality from the neural network due to these limitations. As previously described, the reduced scope focuses on the performance differences between a fully connected Autoencoder network and a 1D convolution Autoencoder network.

The initial plan was to acquire a raw audio dataset and do minimal preprocessing before feeding that raw audio data into a generative neural network. After the initial model was ready, the goal was to play with the input data’s dimensionality and convolutions to see if that would improve the consistency of the structure in the long term. In the planning phase, a Variational Autoencoder was selected as the generative neural network type. The initial model was built, but during the training, the validation metrics showed that the model could not learn a meaningful representation of the data. This sparked the prototyping process that followed a continuous feedback loop with three key phases. First, the model underwent training and testing. Next, the results were analysed. Finally, based on these insights, the model and/or preprocessing techniques were refined and updated, leading back to the next iteration of training and testing. Numerous preprocessing methods were tried during the prototyping process, such as μ-law companding and quantisation, as explained by van den Oord et al. in 2016. Throughout the process,

the original model structure was adjusted to account for changes in the dimensionality of the input data. Initially, the 1D sequential audio data was converted into a 2D image-like format. Subsequently, it was further transformed into a 3D video-like format during the following prototyping iterations. As previously mentioned, due to the inability to construct a model capable of learning meaningful data representations within the constraints, the decision was made to use an Autoencoder neural network instead of a Variational Autoencoder network.

5.1 Dataset and System

This research was performed on Faraldo's (2017) Beatport EDM Key Dataset, which consists of 1486 songs in the Electronic Dance Music (EDM) genre. The data was divided into training, validation, and test sets, so the training set consisted of 1300 songs, the validation set had 130 songs, and the test set included 20 songs. These songs were split into segments. Different segment lengths were tested, but a 3-second segment length was selected. The audio data has a frequency of 44100 Hz, meaning there are 44.1 thousand data points per second. To make this manageable, data quality was down-sampled by a factor of ten, resulting in 4410 data points per second. Combined with the selected 3-second segment length, this results in a 13230 input length in the neural network.

As previously referenced, the importance of computational resources cannot be overemphasised. The models' prototyping and actual experiments were performed on a PC. The PC had an Intel i7-4770K processor and 24 GB of DDR3 RAM. Neural network training was performed on a dedicated GPU, Nvidia GeForce GTX 1080, which has 8 GB of dedicated VRAM and an additional 12 GB of shared GPU memory, for a total of 20 GB of memory. The bottleneck in the process turned out to be the GPU memory. The main limitation of GPU memory is its impact on the size of the neural network it can handle. In this context, size is closely related to complexity, as increasing the number of layers or the size of each layer in the neural network increases its memory usage. When prototyping, I found that a model with three fully connected layers containing 9800, 6500, and 3300 neurons resulted in a size of over 10 GB when loaded into the GPU memory.

5.2 Model Evolution

This research resulted in the development of two neural networks, each designed to capture specific representations of audio data. Additionally, it outlines the preprocessing steps necessary for the success of these neural networks. As discussed earlier, the model development process followed a prototyping approach, which created a feedback loop. This iterative process led to several unsuccessful combinations of preprocessing techniques and generative neural network models.

As the project timeline became more constrained, a strategic decision was made to narrow the scope by excluding the generative neural network. Following this decision, many previously developed preprocessing methods were reevaluated using a standard Autoencoder instead of a Variational Autoencoder. However, the performance outcomes were unsatisfactory, suggesting that the primary limiting factor in this study may have been the system's computational capacity in relation to network size.

5.2.1 Preprocessing

The preprocessing in this study can be divided into two categories: non-transformative and transformative methods. Non-transformative methods modify the data's dimensions without altering its internal structure. Examples of such preprocessing steps include data segmentation and sampling, which were applied consistently throughout the prototyping process. In the sampling step, the original 44.1 kHz signal was downsampled to 4.41 kHz, reducing the number of data points by a factor of 10. Similarly, in the data segmentation step, the downsampled 2-minute-long songs were divided into more manageable 3-second chunks. These methods were introduced primarily to reduce the computational load on the system, as processing large datasets in their original form would have been resource-intensive. Other preprocessing methods that fall into the first category and align with the study's initial plan were modifications to the input data dimensions.

In contrast, transformative methods involve altering the data's internal relationships or converting it from one coordinate system to another. Initially, the plan did not include any data-transforming preprocessing, as the goal was to work with the raw data as much as possible. However, when it became evident that the neural networks could not effectively learn from the raw audio data, transformative preprocessing steps were added to the workflow. These transformations aimed to modify the data to better align with the neural network's learning capabilities. Various transformative preprocessing techniques were tested individually and in combination as part of the iterative prototyping process.

Early attempts at transformative preprocessing involved data normalisation, which is beneficial in some instances, as pointed out by Singh and Singh (2019). Min-max normalisation was selected, with a range of [-1, 1], to align with the sigmoid activation function's output range used by the model at that time. Despite this alignment, no noticeable improvement was observed in the behaviour of the neural networks.

```
def mu_law_companded(x, mu=255):
    # Ensure the input is in the range [-1, 1]
    x = np.clip(x, -1, 1)

    # Apply μ-law companding
    x_mu = np.sign(x) * (np.log1p(mu * np.abs(x)) /
np.log1p(mu))

    return x_mu
```

Following preprocessing attempts, utilised μ-law companding, as described by van den Oord et al. (2016). The code snippet above demonstrates how the transformation was applied, where the input data was first clipped to the range of [-1, 1] before using the μ-law companding transformation. Clipping was necessary for this process, as indicated in Equation 2. When examining Waveform 1 in Figure 7, it is clear that most amplitudes remained within the range of [-1, 1], making clipping an effective solution. Listening tests comparing Waveform 1 and Waveform 3 revealed no significant auditory differences, even though clipping resulted in minimal data loss. This indicates that clipping did not significantly affect the quality of the processed audio.

```
def mu_law_decoding(y, mu=255):
    # Apply inverse μ-law decoding
    x = np.sign(y) * (1.0 / mu) * (np.power(1 + mu, np.abs(y))
- 1)

    return x
```

The above code snippet depicts how μ-law companding transformation was reversed, and from Figure 7, the effects of μ-law companding on a waveform can be observed. The first plot shows Waveform 1, the validation waveform, without any modifications. The second plot displays Waveform 1 after applying μ-law companding using the function visible in the code snippet above. The third plot exhibits Waveform 2 after reversing the μ-law companding using the function in the code snippet below. However, feeding the neural network with μ-law companded data did not lead to any performance improvements.

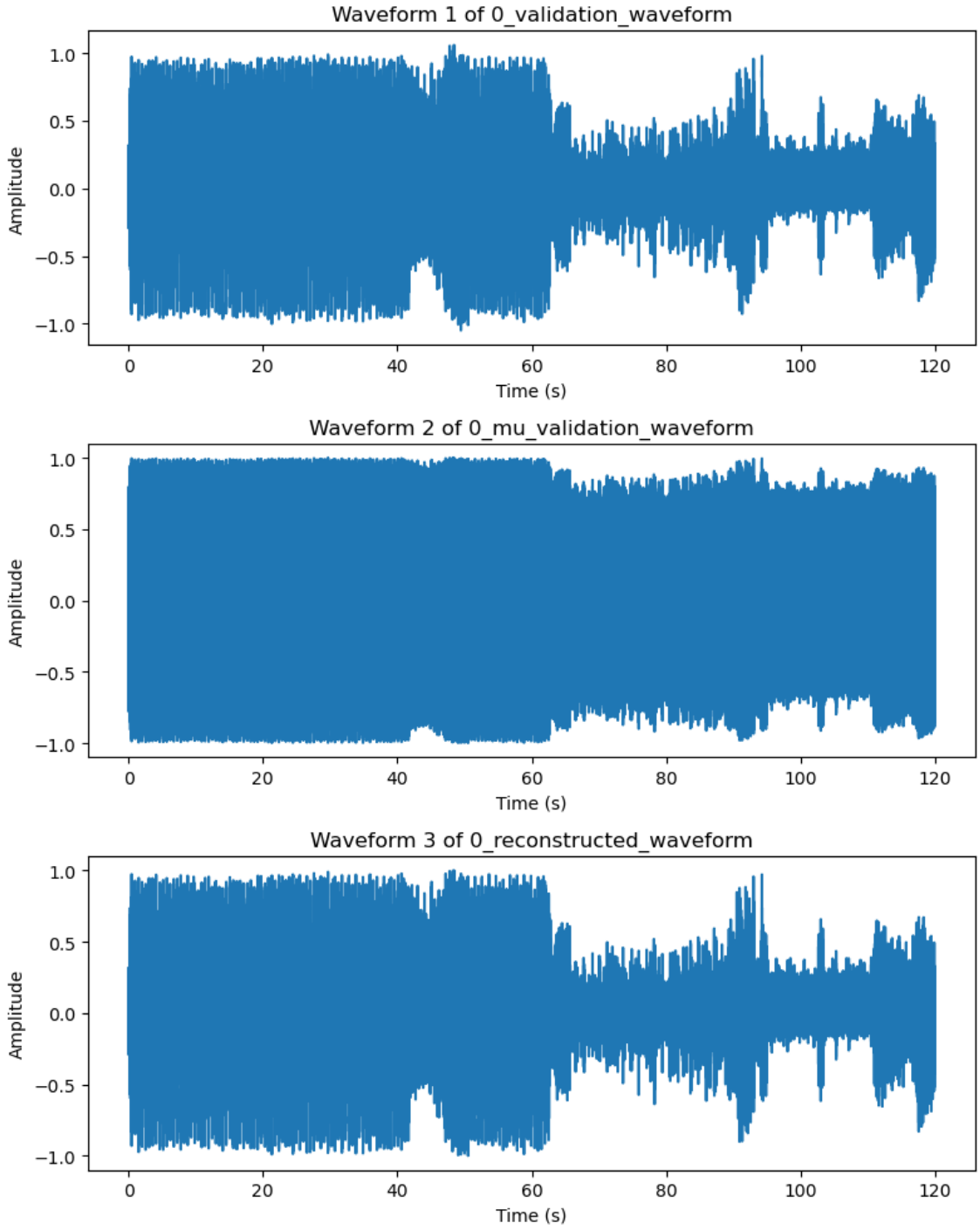


Figure 7. Effects of μ-law companding on a waveform.

Van den Oord et al. (2016) utilised μ-law companding along with quantisation, prompting the consideration of incorporating this technique in the preprocessing phase. This method involves transforming the problem from a regression to a classification problem.

However, as this was not the chosen preprocessing method for this study, even though providing a detailed explanation would be interesting, it would also divert focus from the current study. Therefore, it is only provided here for context.

The preprocessing method that ultimately proved effective involved a combination of steps. First, non-transformative techniques such as downsampling and segmentation were applied. Each 3-second segment was then transformed using a Real Fast Fourier Transform (RFFT), converting the data into the frequency and magnitude domain. After the RFFT transformation, the length of each 3-second segment was halved, with the resulting array containing values in the form of real ± imaginary coefficients. In the final step, the array size was doubled back to its original length by splitting each value into separate real and imaginary coefficient components. The final array alternated between real and imaginary coefficients, structured as [real_value, img_coeff, real_value, img_coeff, …], making the data suitable for input into the neural network. It's interesting to note that the length of the segment, when combined with the RFFT, impacts the neural networks' capacity to learn. When the process of segmenting and RFFT was reversed so that the complete 2-minute song was taken through the RFFT and only then segmented into “3-second” segments, it led to the network's inability to capture a meaningful representation of the data.

5.2.2 Model development

This study produced two distinct Autoencoder neural network architectures: the base model and the convolutional model. The base model was created as a reference point for comparing the performance of the convolutional model.

The base model consists of 5 fully connected layers, similar to the model shown in Figure 4. It comprises an encoder side (depicted in blue in Figure 4) and a decoder side (depicted in green in Figure 4). In the middle, there is a bottleneck layer that determines the minimum dimensionality through which the data is passed. This layer can significantly impact the model's ability to learn a representation of the data, as having too few

nodes can make it impossible to capture all the important attributes of the data. In a network layer, there are input and output nodes, which are similar to the two bottom-most layers shown in Figure 1. The connections are as depicted in Figure 1, where each node is connected to all consecutive nodes; hence, the name is fully connected. The preprocessing creates data blocks the size of 13230 attributes; this determines the first layer input size. The two layers in the encoder reduce the attributes first to 9800 and then 6500. The bottleneck layer, also known as latent space/vector, further reduces the dimensionality to 3300 attributes. The decoder mirrors the encoder and upscales the data back to 13230 in 2 layers. This model uses a Rectified Linear Unit activation function (ReLU) where each layer's output is pushed through the activation function except for the decoder's final layer.

The convolutional model, as described by Goodfellow et al. (2016, p. 326), is a type of neural network where one or more layers are replaced with convolutional layers. In this research, the model consists of 8 layers, with six being convolutional and two being linear (fully connected). The model design is shown in Figure 8 and Figure 9. Figure 8 illustrates the encoder side of the Autoencoder design, with each block representing the interface between layers. The numbers above the blocks represent the channels in each interface. For example, the first layer takes data in one channel, but after the first convolution, the data is channelled into 16 separate channels. This increase in the number of channels is represented by the increased thickness of the blocks in the figures. Additionally, each convolution reduces the input length for the next layer, which is visualised by the decrease in the "area" of each block.

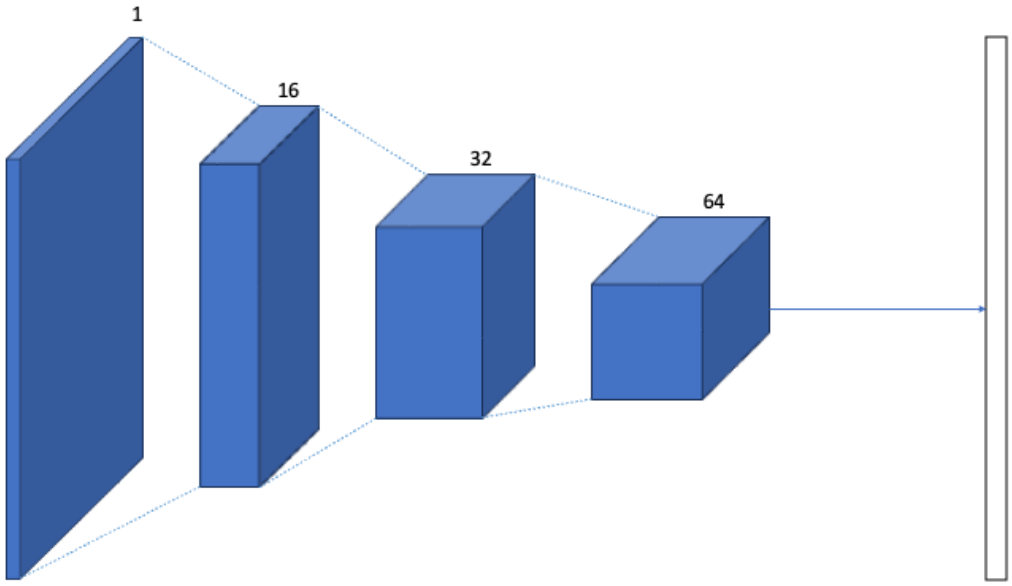


Figure 8. The encoder part of the Convolutional Autoencoder is in blue, and the first latent vector is in white.

The code snippet below illustrates how the input length decreases as it passes through the encoder side. In the code, 40 represents the batch size, while 16, 32, and 64 represent the channel counts, and the last number represents the input length. An interesting observation from the code is that doubling the channel size halves the input length. This is because of the kernel attribute selection for the convolutions. Each convolution layer uses the same kernel attributes, which are size: 4, stride: 2, and padding: 1. The "flatten" operation is essential for transforming the 2D data back to a 1D form, allowing it to be fed to the linear bottleneck layer.

```
Encoder input: torch.Size([40, 1, 13230])
Conv1: torch.Size([40, 16, 6615])
Conv2: torch.Size([40, 32, 3307])
Conv3: torch.Size([40, 64, 1653])
Flatten: torch.Size([40, 105792])
Bottleneck: torch.Size([40, 3300])
```

In Figure 9, we can see that the decoder side of the autoencoder mirrors the encoder side. One key difference is that the decoder's convolutional layers use transpose

convolution operations to reverse the effects of the convolution operations. The input length is doubled when the channels are halved in the decoder. However, this alone is insufficient to reach the original data dimensionality, as shown in the code snippet below. To address this mismatch, the decoder side uses a fourth kernel attribute called output padding, which increases the dimensions of the output by one. Similar to the base model, all the layers in the autoencoder go through the ReLU activation function except for the decoder's last layer.

```
Decoder input: torch.Size([40, 3300])
Decoder fc: torch.Size([40, 105792])
Reshape: torch.Size([40, 64, 1653])
Conv3: torch.Size([40, 32, 3307])
Conv2: torch.Size([40, 16, 6615])
Conv1: torch.Size([40, 1, 13230])
```

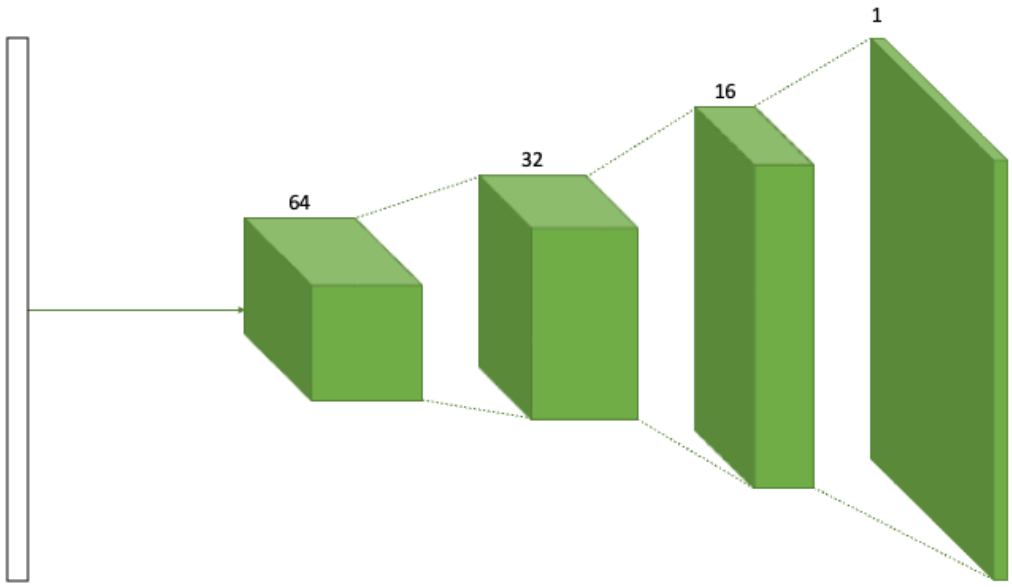


Figure 9. The decoder part of the Convolutional Autoencoder is in green, and the second latent vector is in white.

5.2.3 Testing setup

As previously established, this test evaluates the difference between linear and convolutional neural network architectures. Both networks' training starts from zero and is set

to last until convergence or until a max epoch limit of 500 is reached. In this study, convergence is considered to be achieved when there are 50 epochs with no improvement in validation loss. The testing setup consists of multiple variables, which in the controller experiment world can fall under the category of either unknown variables or intervening variables. To limit the number of variables that fall into either of the categories, all other variables are kept constant between the test runs. These variables include system settings, dataset composition, and training setup. Chapter 5.1 discusses system settings and the dataset in detail, so this section elaborates on the testing setup.

Two parts that critically affect the model's ability to learn are the objective function and optimization procedure, as explained in Chapter 2. In this study, the Mean Squared Error (MSE) function, presented in Equation 3, is employed as the objective function in alignment with established literature, given the regression nature of the problem (James et al., 2023, p. 28). Building upon the optimization concepts discussed in Chapter 2, this study utilizes the Adam optimizer for training the neural network model. Adam, short for Adaptive Moment Estimation, is an extension of stochastic gradient descent that computes adaptive learning rates for each parameter (Kingma & Ba, 2015). The choice of Adam is motivated by its efficiency and effectiveness in handling sparse gradients and noisy data, which are common in real-world datasets. The optimizer is configured with a learning rate of $1x10^{-4}$ and a weight decay of $1x10^{-5}$. A lower learning rate ensures that the model converges smoothly without overshooting the minimum. The weight decay term is a regularization parameter, penalizing large weights to prevent overfitting and improve the model's generalization capabilities (Goodfellow et al., 2016, p. 229).

5.3 Results

Following the principles of a controlled experiment, this study established one independent variable and seven dependent variables. The independent variable in this study can be broadly described as the neural networks' architecture. In controlled experiments, dependent variables are those expected to be affected by changes in independent variables. They can hence be used to measure results if it is also accepted that a change in

the dependent variable leads to a qualitative improvement in output data. The dependent variables can be divided into four error calculations, two graphical visualizations and one subjective listening review.

The selected error metrics were Mean Squared Error, Mean Absolute Error, Root Mean Squared Error, and Signal-to-Noise Ratio (SNR). As mentioned earlier, Mean Squared Error (MSE) was chosen as the validation loss function, given its effectiveness in regression tasks. In addition, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) were computed to give further insights into model performance. MAE assesses the average size of the errors in predictions, providing a clear view of how far predictions deviate from actual values in the original scale, making it easier to interpret. RMSE, which is derived by taking the square root of MSE, is more responsive to larger errors, giving insight into possible outliers in predictions. Signal-to-Noise Ratio was also employed as a metric to measure the clarity of the reconstructed audio compared to the original. SNR assesses the level of the desired signal relative to the background noise, with higher values indicating better reconstructions. Compared to other error metrics, SNR provides a complementary view of model performance and is particularly suitable for audio processing tasks. The validation process utilized two types of graphs: comparison graphs illustrating both the original and reconstructed waveforms and magnitude spectrum graphs comparing the frequency content of the validation and reconstructed data. Ultimately, the most sensible way to evaluate the music's results is through listening. The listening and the analysis were both conducted by the author.

$$MAE = \frac{\sum_i^n |\bar{y}_i - y_i|}{n} \tag{9}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_i^n (y_i - \bar{y}_i)^2} \tag{10}$$

This study includes two training runs, one for each model, allowing them to be trained from scratch to assess their performance. An analysis of the results is conducted after

both training runs are complete. The values of the dependent variable were automatically recorded at the beginning of the training and during training under two specific conditions: every 25th epoch and when the best validation loss improved. Figure 10 illustrates the validation error progression for both models. Figure 10 gives an excellent general view of the performance of both models. It can be seen that the convolution model was worse in the beginning, but in a matter of a few epochs, it was able to surpass the base model. Also interesting is that the base model reached its minima in epoch 56 and slowly started increasing after that. After that, the convolution model's validation loss continued to decline up to 75 epochs and remained very close to the global minima. It is important to highlight that the graph's resolution is highest during the initial epochs and diminishes toward the end of the runs, based on the chosen reporting criteria. The loss is recorded only for every 25th epoch, provided it does not improve the validation loss.

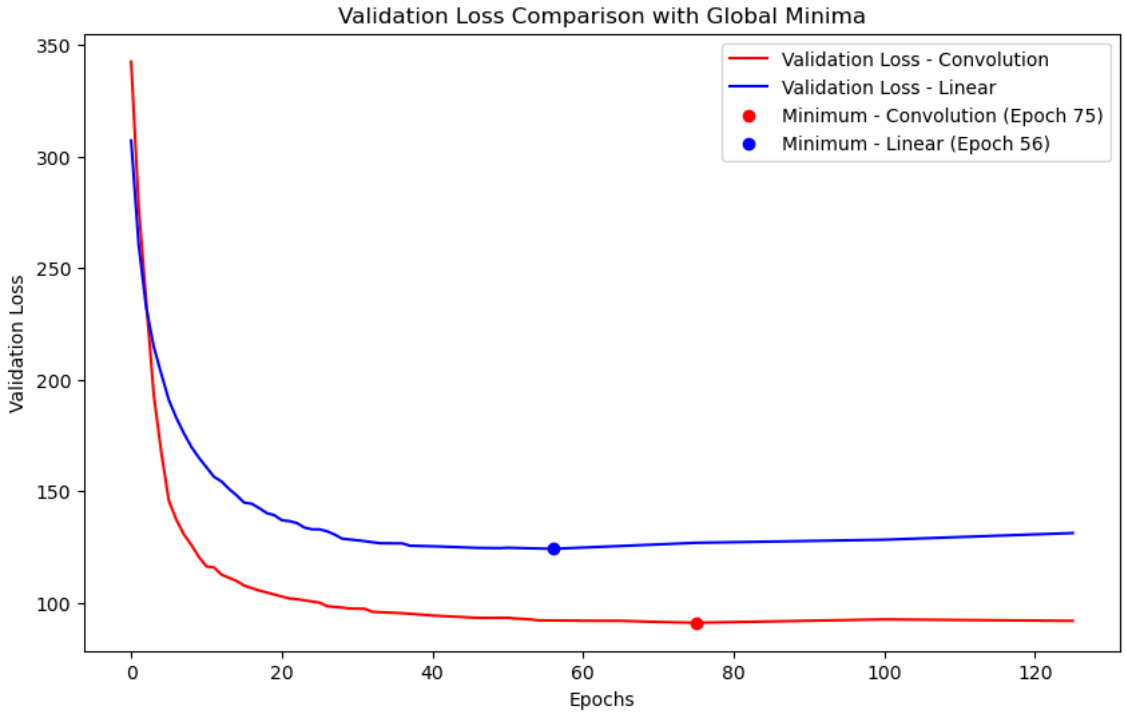


Figure 10. Evolution of the validation loss during the test with global minima. The blue curve represents the base model's performance, and the red curve represents the convolution model's performance.

Tables 1 and 2 give a deeper insight into the development of dependent variables. It should be noted that epoch numbering starts at zero, and at that point, one training round has already been completed. In the tables, the first three rows reflect the progress of models after one, five, and ten epochs. The fourth row presents each model's best result, while the remaining rows display all recorded outcomes following those best results. It is worth highlighting some notable findings from the results. Initially, the convolution model had a higher validation loss compared to the base model. However, after just five epochs, the convolution model had halved its validation loss, whereas the base model only achieved a 30% reduction. Remarkably, it took only ten epochs for the convolution model to surpass the base model's lowest validation loss. By the 10th epoch, the base model reached results similar to what the convolution model had achieved in just half that number of epochs.

Table 1. Summary of changes in the base model's dependent variables.

Base model	MSE	MAE	RMSE	SNR
Epoch 0	307.26	0.08	16.75	3.18 dB
Epoch 4	202.67	0.06	13.44	5.61 dB
Epoch 9	165.03	0.06	12.26	6.57 dB
Epoch 56	124.20	0.05	11.09	8.42 dB
Epoch 75	126.90	0.05	11.25	8.37 dB
Epoch 100	128.29	0.05	11.37	8.42 dB
Epoch 125	131.28	0.05	11.54	8.35 dB

Table 2. Summary of changes in the convolution model's dependent variables.

Base model	MSE	MAE	RMSE	SNR
Epoch 0	342.58	0.11	17.58	1.64 dB
Epoch 4	167.56	0.06	12.39	6.32 dB
Epoch 9	120.48	0.05	10.62	8.55 dB
Epoch 75	91.05	0.04	9.69	10.23 dB
Epoch 100	92.57	0.04	9.77	10.16 dB
Epoch 125	91.91	0.04	9.74	10.27 dB

The following Figures 11 to 20 offer an overview of the training processes for both models. These figures are divided into two groups: Figures 11 to 15 showcase amplitude waveform data, while Figures 16 to 20 illustrate the magnitude across frequency bins. Each figure represents an interesting stage in model progression, providing visual benchmarks of how closely each model approximates the target waveform and frequency distribution over successive epochs.

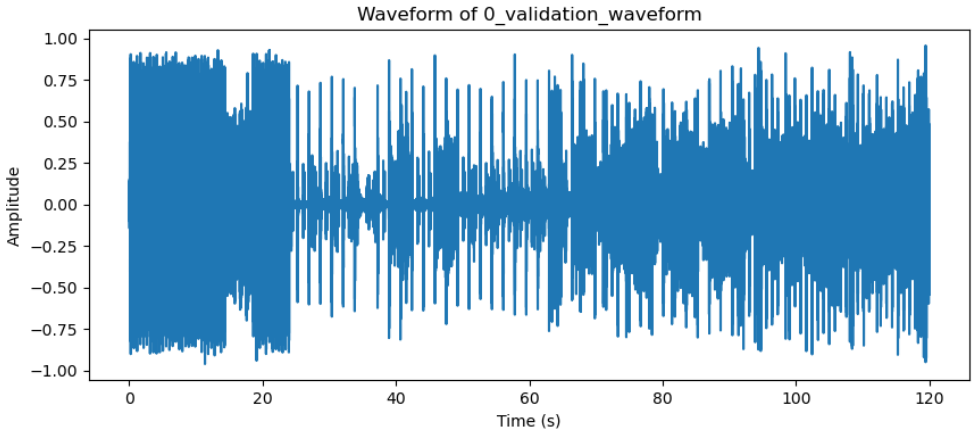


Figure 11. Waveform of the validation audio.

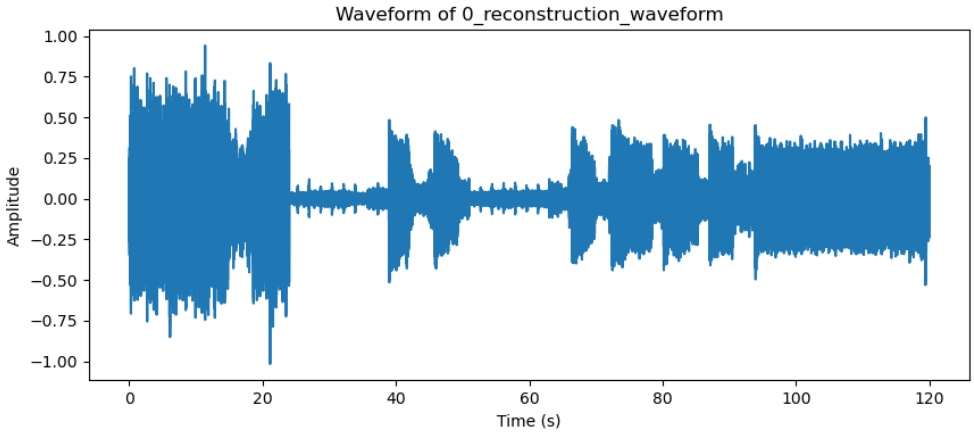


Figure 12. The base model's reconstruction data is depicted as a waveform after Epoch 0.

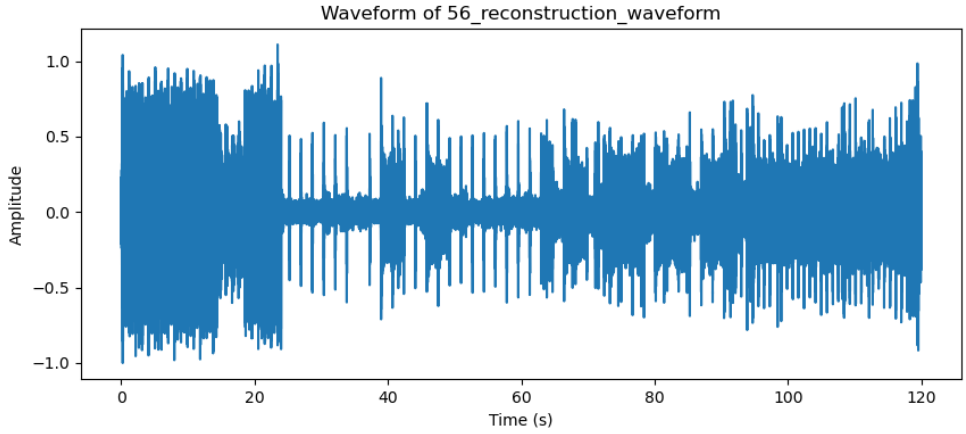


Figure 13. The base model's reconstruction data is depicted as a waveform after Epoch 56.

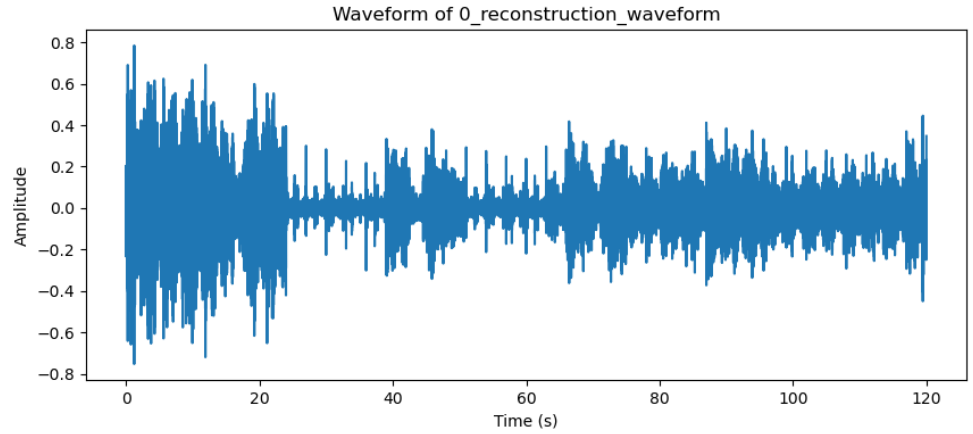


Figure 14. The convolution model's reconstruction data is depicted as a waveform after Epoch 0.

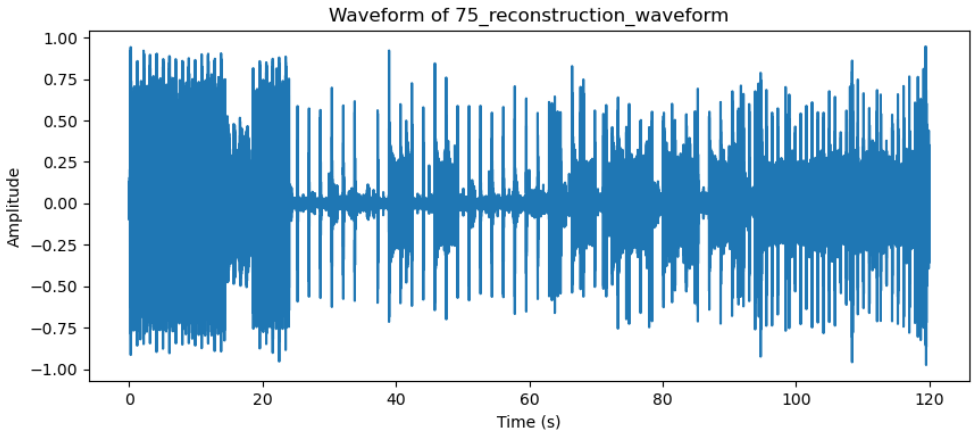


Figure 15. The convolution model's reconstruction data is depicted as a waveform after Epoch 75.

Figure 11 presents a waveform representation of the validation data, which serves as the target reference. Figures 12 and 13 illustrate the initial (Epoch 0) and optimal (Epoch 56) results for the base model. As the model progresses from Figure 12 to Figure 13, notable changes in the waveform appear particularly the development of distinct amplitude spikes that move closer to the target shape in Figure 11. Figures 14 and 15 similarly track the convolution model's progression, with Figure 14 displaying the initial output (Epoch 0) and Figure 15 presenting the optimal result (Epoch 75). Figure 14 exhibits more waveform volatility compared to Figure 12 but less than Figure 13. Interestingly, the convolutional model's mean squared error (MSE) at Epoch 0 was worse than that of the base

model, although visually, the waveform in Figure 14 resembles the target validation data in Figure 11 more closely than the base model's Figure 12. Figure 15 showcases the convolutional model's best result achieved after Epoch 75, and compared to earlier figures, it most closely resembles the target waveform in Figure 11. In comparison to the base model's best result shown in Figure 13, the individual amplitude spikes in Figure 15 are slightly stronger, making it more similar to Figure 11. However, Figure 15 is not perfect, lacking some of the mid-level amplitudes between weak and strong that are present in Figure 11.

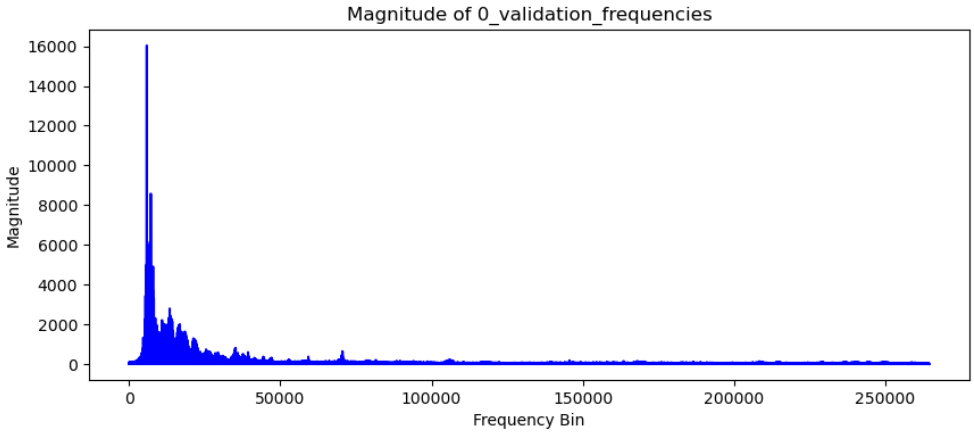


Figure 16. This represents the frequency distribution and corresponding magnitudes of the validation data.

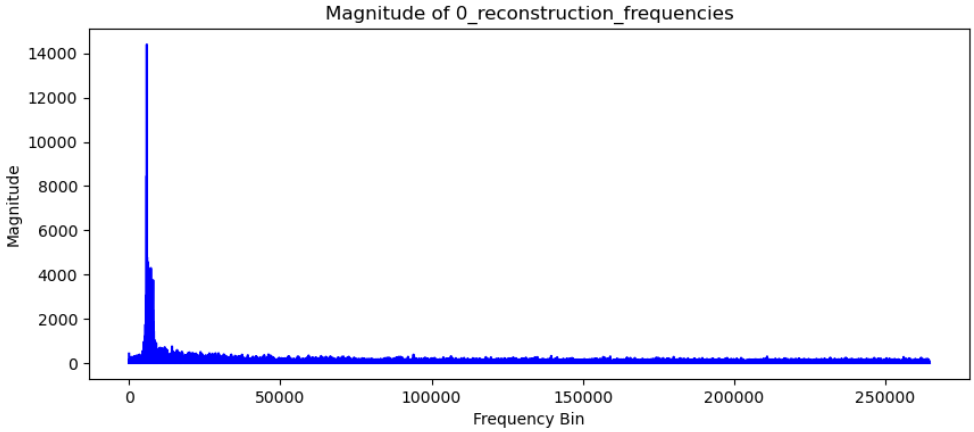


Figure 17. Frequency distribution and magnitude of the base model's reconstruction data after Epoch 0.

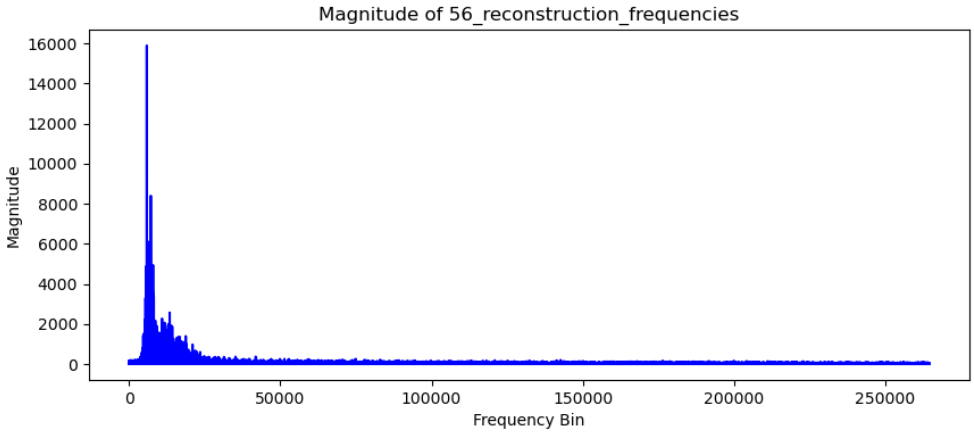


Figure 18. Frequency distribution and magnitude of the base model's reconstruction data after Epoch 56.

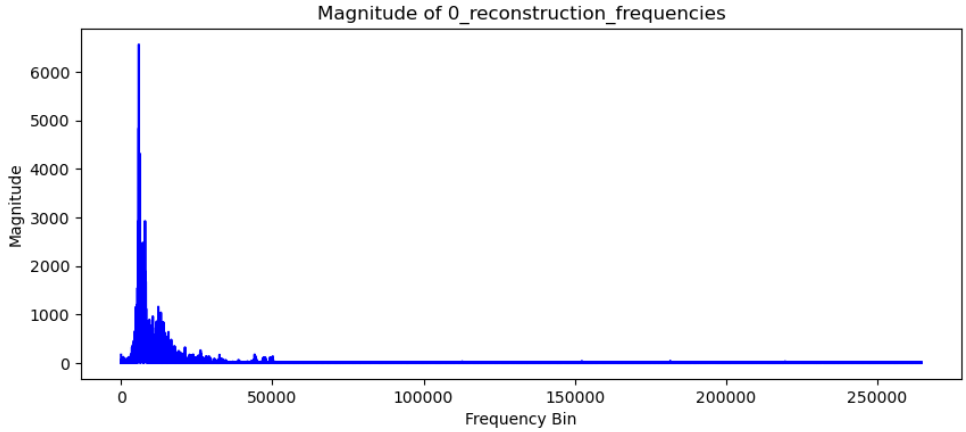


Figure 19. Frequency distribution and magnitude of the convolution model's reconstruction data after Epoch 0.

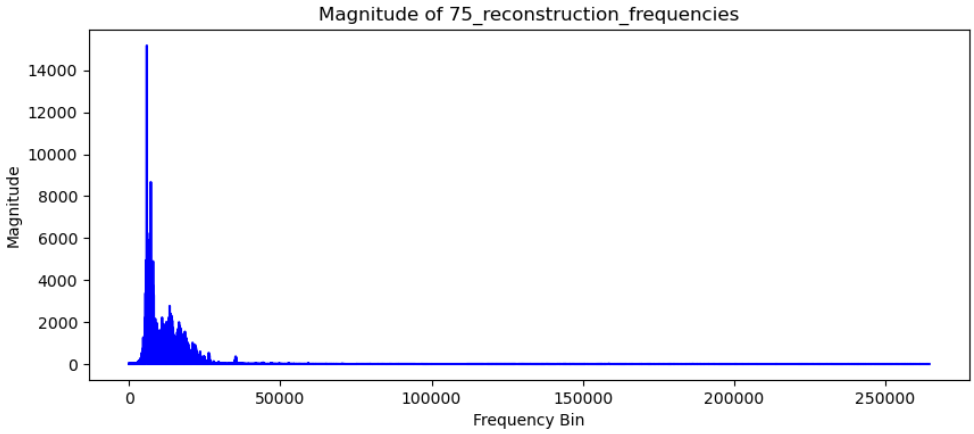


Figure 20. Frequency distribution and magnitude of the convolution model's reconstruction data after Epoch 75.

Figure 16 illustrates the frequency distribution and corresponding magnitudes of the validation data, essentially showing how prominent each frequency range is within the target data. Figure 17, representing the base model's initial reconstruction at Epoch 0, shows only a single major frequency spike; unlike the broader distribution seen in Figure 16, the mass in Figure 17 is concentrated in a narrow range. By Epoch 56, shown in Figure 18, the base model's frequency distribution widens, closely resembling Figure 16, but still missing smaller spikes at frequencies farther from the primary concentration. Figures 19 and 20 show the convolutional model's progression, starting with Epoch 0 in Figure 19, where the distribution already replicates much of Figure 16, with a wider base and even a secondary spike. However, this figure exhibits a cutoff beyond the 50kth frequency bin, with no data present past that point. Figure 20, showing the convolutional model's best result at Epoch 75, closely resembles Figure 16 with an even sharper cutoff than in Figure 19, beyond which no frequency data appears.

5.4 Analysis

This chapter focuses on analysing the models' results, considering the reasoning behind their performance and the factors that influenced their behaviour. The analysis can be divided into two sub-chapters: common features and separating features. Given the highly controlled environment and specific test conditions, one might expect that the models would share more commonalities than differences. Surprisingly, that is not the case.

5.4.1 Common features

Exploring audio data reconstruction of the models revealed a noteworthy commonality that deserves further examination. One notable commonality between both neural network models is the absence of sounds typically associated with higher frequencies—such as melodies or vocals—from the reconstructed audio. This observation raises the question of why these higher-frequency elements were not effectively retained by either

model, particularly focusing on the potential role of downsampling and its compliance with the Nyquist theorem.

A potential explanation for the lack of melodies could be the effects of downsampling. The Nyquist-Shannon sampling theorem states that the highest frequency that can be accurately captured by a system is half of the sampling rate (Shannon, 1949). For the audio used in this study, which was downsampled to 4410 Hz, the highest frequency that can be reconstructed without aliasing is 2205 Hz. However, this explanation alone seems insufficient, as most musical instruments produce fundamental frequencies well below this limit.

To contextualize the discussion, it's beneficial to consider the frequency ranges of various musical instruments. For instance, a piano spans from 27.5 Hz to 4186 Hz (A0 to C8), a violin from 196 Hz to approximately 3500 Hz, and a guitar from 82 Hz to up to 1k Hz. The human voice typically operates at around 120 Hz for males and approximately 200 Hz for females, though female singers can reach frequencies as high as 1500 Hz (Pulkki & Karjalainen, 2015, p. 82). These examples illustrate that most of the fundamental frequencies of these musical instruments and vocals fall within the capacity of the Nyquist limit applied in standard audio processing. Thus, the absence of higher frequency components like melodies or singing in both models' reconstructions cannot be solely attributed to limitations imposed by the Nyquist theorem. Further supporting this conclusion is a visual review of Figure 16. Upon examination, it is evident that the majority of the content lies below the 50,000th frequency bin. With equation 11, where dF represents the frequency resolution and T is the input signal length in seconds (120), it can be calculated that the frequency of the 50,000th frequency bin is roughly 417 Hz. What this basically means is that most of the data is well below the 2205 Hz Nyquist limit. Final support comes from listening to the validation audio samples, where the melody and vocals are clearly audible.

$$dF = \frac{1}{T} \tag{11}$$

5.4.2 Separating features

This chapter presents and compares the differences in the models' results, followed by an analysis of the underlying reasons for these variations. Key factors influencing these differences include the presence or absence of noise in the output, the rate of learning and each model's learning capacity. Additionally, a comparison of memory footprint highlights how resource usage varies across architectures. These aspects, among others, provide insight into the models' performance and efficiency.

So, the Nyquist-Shannon theorem could not explain the lack of audible melodies or vocals, which raises the question: what can? A bit of insight into this can be gained by observing and comparing Figures 18 and 20. Looking at the graphs, it becomes evident that the base model has a greater presence of higher frequencies. However, Tables 1 and 2 indicate that the base model, at its best, also has nearly twice the noise power of the convolutional model. This indicates a difference in the quality of the audio, and listening to the samples reveals that the base model exhibits significantly more noise compared to the convolutional model. From this, it can be concluded that the base model can create higher frequency content, but it is most likely just noise, while the convolutional model’s reconstructions completely lack the higher frequency content, bizarrely resulting in better audio quality due to the absence of the noise.

The rate of learning between the two models offers another clear distinction. The base model achieved its best result in epoch 56, whereas the convolutional model required just ten epochs to reach a comparable performance. One reason for this could be that the convolutional model uses of weight sharing, allowing it to use the same filter across the input to efficiently capture local patterns, such as repeated frequencies or temporal structures. Secondly, the convolutional model can generalize better because it can recognize important features regardless of their position in the data. In contrast, the base model must learn every feature independently without leveraging spatial relationships, which slows their learning process. Weight sharing and generalization

allow the convolutional model to capture essential features early, leading to faster convergence, while fully connected models require more epochs to fine-tune their extensive parameter set.

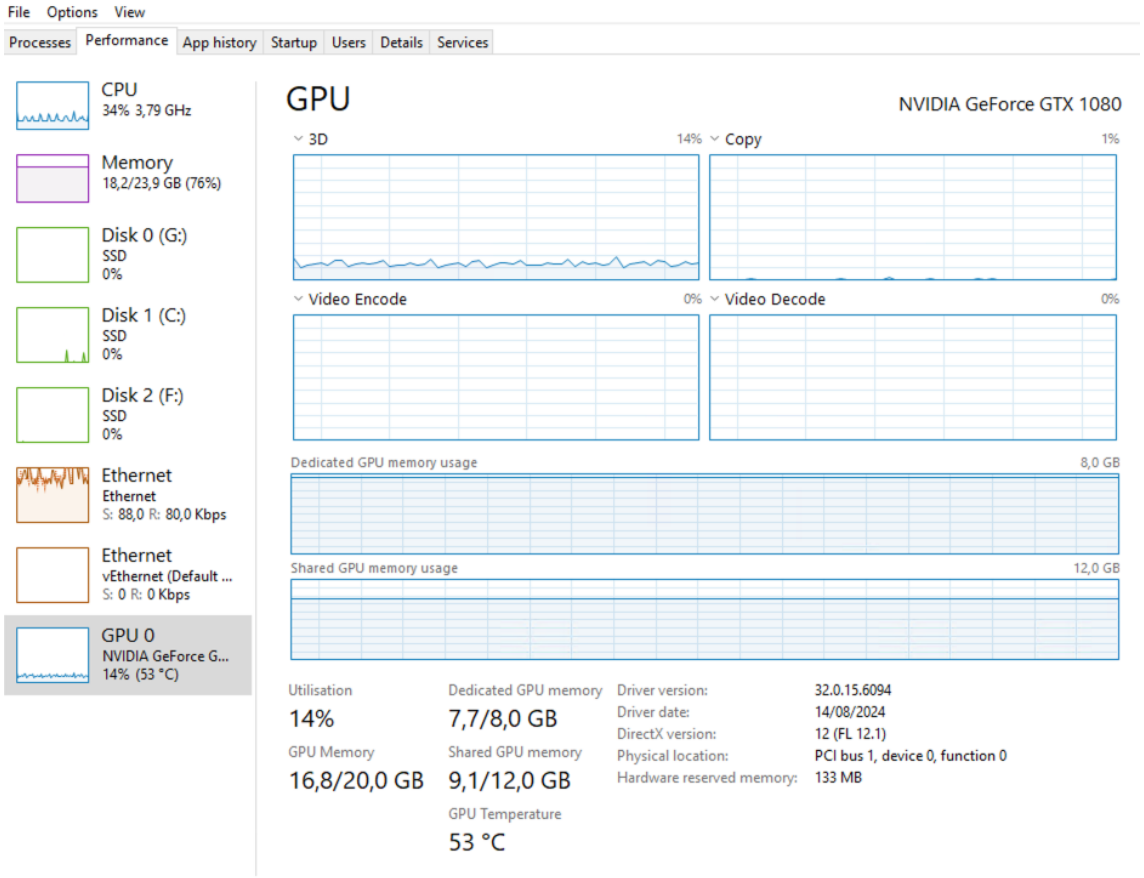
Another distinction is the learning capacity. Based on practically all the metrics, the convolutional model outperformed the base model. Most significant is the comparison done by listening to the samples where, while not perfect, the convolutional model clearly outperforms the base model. Reasons for this can be sought from the following code snippets. The following code snippets summarise the structure, parameter distribution, and parameter count for the base model and convolutional model. An interesting observation from them is that the base model's parameter count decreases when moving towards the centre of the neural network; in contrast, the convolutional model has the highest parameter count at the centre. Additionally, when looking at the latent space sizes, the convolutional model shows more than 10 times the number of parameters. This likely accounts for its superior learning capacity compared to the base models.

```
----------------------------------------------------------
Layer (type)          Output Shape      Param #
==========================================================
Linear-1               [-1, 1, 9800]     129,663,800
ReLU-2                 [-1, 1, 9800]               0
Linear-3               [-1, 1, 6500]      63,706,500
ReLU-4                 [-1, 1, 6500]               0
Linear-5               [-1, 1, 3300]      21,453,300
ReLU-6                 [-1, 1, 3300]               0
Linear-7               [-1, 1, 6500]      21,456,500
ReLU-8                 [-1, 1, 6500]               0
Linear-9               [-1, 1, 9800]      63,709,800
ReLU-10                [-1, 1, 9800]               0
Linear-11             [-1, 1, 13230]     129,667,230
==========================================================
Total params: 429,657,130
Trainable params: 429,657,130
Non-trainable params: 0
----------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 0.65
Params size (MB): 1639.01
Estimated Total Size (MB): 1639.71
```

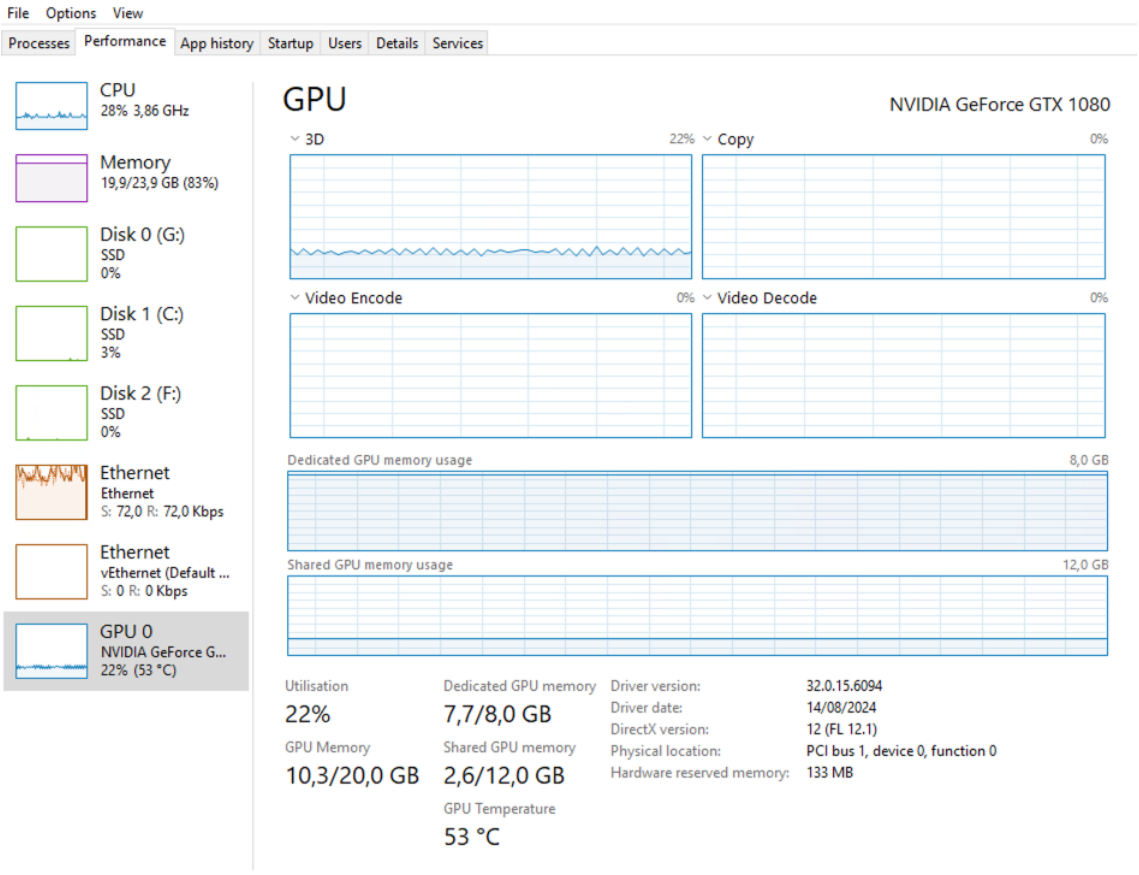
```
----------------------------------------------------------------


----------------------------------------------------------------
Layer (type)               Output Shape       Param #
================================================================
Conv1d-1              [-1, 16, 6615]              80
ReLU-2                [-1, 16, 6615]               0
Conv1d-3              [-1, 32, 3307]           2,080
ReLU-4                [-1, 32, 3307]               0
Conv1d-5              [-1, 64, 1653]           8,256
ReLU-6                [-1, 64, 1653]               0
Linear-7                  [-1, 3300]     349,116,900
ReLU-8                    [-1, 3300]               0
Linear-9                [-1, 105792]     349,219,392
ReLU-10                 [-1, 105792]               0
ConvTranspose1d-11    [-1, 32, 3307]           8,224
ReLU-12               [-1, 32, 3307]               0
ConvTranspose1d-13    [-1, 16, 6615]           2,064
ReLU-14               [-1, 16, 6615]               0
ConvTranspose1d-15    [-1, 1, 13230]              65
================================================================
Total params: 698,357,061
Trainable params: 698,357,061
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.05
Forward/backward pass size (MB): 9.84
Params size (MB): 2664.02
Estimated Total Size (MB): 2673.91
----------------------------------------------------------------
```

When examining the memory requirements of both models, a key insight arises. Contrary to expectations, the convolutional model used notably more memory (16.8GB) than the fully connected base model (10.3GB), yet it achieved better outcomes for its size under the limitations of this test system. Pictures 1 and 2 illustrate the resource metrics captured during the training of the convolutional and base models, respectively. This increased memory demand is likely attributed to the size of the fully connected layers within the convolutional network, especially the bottleneck layer, which greatly raised the parameter count. Attempts to increase the complexity of the base model by expanding its layers caused the model to run out of GPU memory during training, preventing meaningful improvements in performance. In contrast, the convolutional model allowed for more complex architectures while remaining within memory limits,

ultimately yielding superior results. This highlights the convolutional model's advantage in delivering better performance within a constrained computational environment, where scaling the base model was not feasible.



Picture 1. Screenshot of Task Manager during convolutional model training.



Picture 2. Screenshot of Task Manager during base model training.

Finally, the sound quality differences between the two models highlight some peculiarities in the convolutional model's performance. For instance, the reconstructed audio from the convolutional model occasionally contained low-frequency harmonics for vocals, which were particularly noticeable because the fundamental frequencies were completely missing. Additionally, the convolutional model introduced a distinct clicking sound every 3 seconds in the reconstructed audio. This periodic clicking likely stems from how the model processes and stitches together audio segments and may be linked to preprocessing, where the last two samples of each segment were dropped to achieve the correct input length. Despite these artefacts, the convolutional model consistently outperformed the base model in terms of overall audio quality, as evidenced by its lower MSE and reduced noise levels.

6 Conclusion

In this study, the exploration of convolutional and fully connected models for audio analysis provided several valuable insights, even though the focus shifted from generating music to understanding how different neural network architectures perform in a resource-constrained system. The convolutional model consistently outperformed the fully connected base model in terms of efficiency, accuracy, and learning speed, achieving superior audio quality with lower MSE loss. Despite the convolutional model's higher memory usage, its ability to scale more effectively highlighted the architectural advantages of convolutional layers in handling complex audio data. These findings contribute to a better understanding of how limited computational resources can still be leveraged to achieve meaningful results, especially for individuals or smaller teams without access to extensive computational power, an issue emphasized by Dhariwal et al. (2020) in large-scale generative models.

The study demonstrates that, even within the constraints of a 20GB VRAM system, careful model design can enable complex audio analysis tasks. This is particularly important in a field where larger players dominate with more resources. By focusing on architectural efficiencies, such as convolutional networks' capacity for better local pattern detection, the study highlights that advanced neural network prototyping is possible for individuals with smaller resources, making valuable contributions to democratizing AI research. These findings are consistent with Goodfellow et al. (2016, pp. 329-333), who noted that convolutional networks allow more compact representations due to sparse interaction and parameter sharing. Additionally, the study provides insights into the challenges faced when preprocessing audio, which can inform future research in the field.

Several limitations impacted the scope of this study. Firstly, the analysis of the generated audio was conducted by a single individual, introducing subjectivity into the evaluation of sound quality. The study also faced time constraints, which limited further experimentation and model refinement. Additionally, the system's memory constraints (20GB VRAM) restricted the size and complexity of the models, especially when working with

the fully connected architecture. Finally, the broad independent variable, model architecture, presents challenges in generalizing the results, and further studies would be needed to identify the impactful attributes.

Future research could explore several intriguing areas that emerged from this study. One area of interest is understanding why the autoencoder primarily learned low frequencies, as observed by Dhariwal et al. (2020), and finding ways to capture higher-frequency components. Another interesting direction is the development of neural networks capable of separating instruments in audio data, which could lead to practical applications like automatic vocal removal for karaoke or music sampling.

References

Agostinelli, A., Denk, T. I., Borsos, Z., Engel, J., Verzetti, M., Caillon, A., Huang, Q., Jansen, A., Roberts, A., Tagliasacchi, M., Sharifi, M., Zeghidour, N., & Frank, C. (2023). MusicLM: Generating Music from text. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2301.11325

Bengesi, S., El-Sayed, H., Sarker, M. K., Houkpati, Y., Irungu, J., & Oladunni, T. (2023). Advancements in Generative AI: A comprehensive review of GANs, GPT, autoencoders, diffusion model, and transformers. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2311.10242

Bengio, Y., Courville, A., & Vincent, P. (2012). Representation Learning: A Review and New Perspectives. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1206.5538

Briot, J., Hadjeres, G., & Pachet, F. (2019). Deep Learning Techniques for Music Generation -- a survey. In *HAL (Le Centre pour la Communication Scientifique Directe)*. https://hal.sorbonne-universite.fr/hal-01660772

Chalapathy, R., & Chawla, S. (2019). Deep Learning for Anomaly Detection: A survey. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1901.03407

Chitty-Venkata, K. T., Mittal, S., Emani, M., Vishwanath, V., & Somani, A. K. (2023). A survey of techniques for optimizing transformer inference. *Journal of Systems Architecture*, *144*, 102990. https://doi.org/10.1016/j.sysarc.2023.102990

Dhariwal, P., Jun, H., Payne, C. K., Kim, J. W., Radford, A., & Sutskever, I. (2020). Jukebox: a generative model for music. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2005.00341

Dieleman, S., Van Den Oord, A., & Simonyan, K. (2018). The challenge of realistic music generation: modelling raw audio at scale. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1806.10474

Ericsson, L., Gouk, H., Loy, C. C., & Hospedales, T. M. (2022). Self-Supervised Representation Learning: Introduction, advances, and challenges. *IEEE Signal Processing Magazine*, *39*(3), 42–62. https://doi.org/10.1109/msp.2021.3134634

Faraldo, Á. (2017). Beatport EDM Key Dataset [Dataset]. In *Zenodo (CERN European Organization for Nuclear Research)*. https://doi.org/10.5281/zenodo.1101082

Goel, K., Gu, A., Donahue, C., & Ré, C. (2022). It's Raw! Audio Generation with State-Space Models. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2202.09729

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. http://www.deeplearningbook.org

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1406.2661

Hiller Jr, L. A. and Isaacson, L. M. Musical composition with a high speed digital computer. In *Audio Engineering Society Convention 9*. Audio Engineering Society, 1957.

Hinton, G. E., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*(7), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507. https://doi.org/10.1126/science.1127647

Hüllermeier, E., Fober, T., & Mernberger, M. (2013). Inductive bias. In *Springer eBooks* (p. 1018). https://doi.org/10.1007/978-1-4419-9863-7_927

International Telecommunication Union. (1988). G.711: Pulse code modulation (PCM) of voice frequencies. ITU-T Recommendation. https://www.itu.int/rec/T-REC-G.711-198811-I/en

Jabbar, A., Li, X., & Omar, B. (2020). A survey on Generative Adversarial Networks: Variants, applications, and training. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2006.05132

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2023). *An Introduction to Statistical Learning: with Applications in Python*. https://www.statlearning.com

Kingma, D. P., & Ba, J. L. (2014). Adam: A method for stochastic optimization. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1412.6980

Kingma, D. P., & Welling, M. (2013). Auto-Encoding variational Bayes. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1312.6114

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

Molnár, S., & Tamás, L. (2024). Variational autoencoders for 3D data processing. *Artificial Intelligence Review*, *57*(2). https://doi.org/10.1007/s10462-023-10687-x

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination press. https://www.ise.ncsu.edu/fuzzy-neural/wp-content/uploads/sites/9/2022/08/neuralnetworksanddeeplearning.pdf

Nwadiugwu, M. C. (2021). Neural networks, artificial intelligence and the computational brain. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2101.08635

Pulkki, V., & Karjalainen, M. (2015). *Communication Acoustics: An Introduction to Speech, Audio and Psychoacoustics* (1st ed.). John Wiley & Sons, Incorporated.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. https://doi.org/10.1038/323533a0

Shannon, C. (1949). Communication in the presence of noise. *Proceedings of the IRE*, *37*(1), 10–21. https://doi.org/10.1109/jrproc.1949.232969

Singh, D., & Singh, B. (2019). Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, *97*, 105524. https://doi.org/10.1016/j.asoc.2019.105524

Van Den Oord, A., Li, Y., Babuschkin, I., Simonyan, K., Vinyals, O., Kavukcuoglu, K., Van Den Driessche, G., Lockhart, E., Cobo, L. C., Stimberg, F., Casagrande, N., Grewe, D., Noury, S., Dieleman, S., Elsen, E., Kalchbrenner, N., Zen, H., Graves, A., King, H., . . . Hassabis, D. (2017). Parallel WaveNet: Fast High-Fidelity Speech Synthesis. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1711.10433

Van Den Oord, A., Vinyals, O., & Kavukcuoglu, K. (2017). Neural Discrete representation learning. *Neural Information Processing Systems*, *30*, 6306–6315. http://papers.nips.cc/paper/7210-neural-discrete-representation-learning.pdf

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.1706.03762

Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P. (2008). *Extracting and composing robust features with denoising autoencoders* [ICML '08: Proceedings of the 25th international conference on Machine learning]. https://doi.org/10.1145/1390156.1390294

Walliman, N. (2010). Research Methods: The Basics. In *Routledge eBooks*. https://doi.org/10.4324/9780203836071

Wei, R., Garcia, C. a. S., El-Sayed, A., Peterson, V., & Mahmood, A. (2020). Variations in Variational Autoencoders - A Comparative evaluation. *IEEE Access*, *8*, 153651–153670. https://doi.org/10.1109/access.2020.3018151

Zhu, Y., Baca, J., Rekabdar, B., & Rawassizadeh, R. (2023). A survey of AI music generation Tools and models. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2308.12982

Zhuo, L., Wang, Z., Wang, B., Liao, Y., Peng, S., Bao, C., Liu, M., Li, X., & Liu, S. (2022). Video Background Music Generation: Dataset, Method and Evaluation. *arXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2211.11248