Faculty of Computer Science and Mathematics
Chair of Digital Image Processing

Fake Video Game Music Generation Using Generative Network

Master Thesis

Presented By:

Abhinandan Shetty Rathnakar

1. Examiner	2. Examiner
Prof. Dr. Tomas Sauer	Prof. Dr. Michael Granitzer

December 14, 2021

Contents

Abstract

Video games are those played on computers. The video game background music has become an integral part of successful video games. This music makes the player immersive in the game. The recent development in deep learning called Generative models has provided significant contributions in the field of generation of Image, Text, and Music. Variational Autoencoder and Generative Adversarial Networks are the most commonly used Generative models. The thesis explains the method of generating fake video game music using the Variational Autoencoder. For training the model, we use the retro video game music in the form of MIDI files. We manually download the training data from the Vgmuisc.com website. In this thesis, we consider video game music with a single instrument. Music constitutes various elements like notes, chords, velocity, and duration. For our thesis, we are only considering the notes, chords and keys for the music generation. The notes, chords, keys are extracted from the MIDI music file using the appropriate tools. We convert these notes,chords into a form suitable for processing in the neural network. The network is able to generate music that is different from the input music streams. Along with the generation of the music, we verify the influence of the varying hyperparameter values of the network on the music generated at the output. Due to the limitation in the data set and musical information provided to the model, we cannot wholly perceive the output music generated as video game music.

Acknowledgments

Firstly, I would like to express my gratitude to my supervisor Prof.Dr.Tomas Sauer, Chair of Digital Image Processing at the University of Passau, for providing me the opportunity to work on the state of the art topic in machine learning and guiding throughout my thesis work. The constant encouragement and support I received from the beginning of my thesis work to the end, as well as the timely feedback I received, aided me in completing my master's thesis effectively.

I would also like to acknowledge Prof.Dr.Michael Granitzer, for his time to have a discussion related to the topic and also for providing the constant support which helped me in a great deal.

I would like to thank my parents Mr.Rathnakar Shetty G and Mrs.Saraswathi Shetty R and my brother Mr.Arun Kumar Shetty R for showing their love and care for me along with standing by me and supporting me in all my endeavors.

My grandparents, friends and relatives thank you, everyone, for your motivation to carry out this work successfully. I would like to thank the Lord almighty for all the showers of blessings I received from him.

Finally, I would like to thank all the authors for providing the plethora of information on the internet, which was very helpful in carrying out this task

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Music is defined as the art of making compositions by combining sounds in time. Common elements of music include pitch, notes, rhythms, texture, and timbre [1]. The music generates a sense of emotions in the listener. Video games had been evolved rapidly with the advancement in technology since their beginning in 1950. Along with the game, the graphical display and music also have seen development. Video game music is those audio signals transmitted from a video game; it can be continuous music or sound effects [Col08]. The main difference between video game music and other entertainment music is that the former involves the gamer. Video game music influences the gamer such that the gamer responds to the situations of the game on the screen. To have a wonderful experience for the gamer, along with the video and storyline the background music also plays a vital role in making the player dwell into the game. In the early days, because of the limitation in technology, the programmer used to create music. With the advancement in technology, composers started to compose music [Fri13]. We can develop a tool to benefit the composers, which can generate artificial compositions that could inspire the music composers.

We find the use of computers in creative tasks such as the generation of images, lyrics, music. There has been extensive research in this area using different methods and technologies. Digitization in music has made listening to music very convenient and made the data available for analysis and other tasks. The advancement in Artificial Intelligence technology has led to the application of AI in music. The music industry is successful in applying AI for music recommendation systems. There has been intense research in AI music composition or generation with the development of deep learning technology. Due to the generality of deep learning models, it is advantageous in generating musical

[1]https://en.wikipedia.org/wiki/Music

content, which is unlike the music generation models like grammar-based, rule-based music generation systems. The Deep Learning model can be used for different categories of music, as they are unbiased and learn from random music corpus. We can extract the information from the largely available data set of old music to generate new versions of music. Deep learning is good at processing raw unstructured data, from which its hierarchy of layers will extract higher-level representations appropriate to the task.

The biggest motivation is to see how the existing deep learning model can come close in generating the game music compared with the original video game music and help the music composers with the above-discussed advancement in technology.

1.2 Challenges

This thesis focus on the generation of fake video game music with Variational Autoencoder (VAE). Previously there were works on the generation of music with VAE, in which they considered expressive elements in music such as bars, duration and other aspects . In this thesis, we focus on notes, chords along with their key. Handling large dimensional data posed a challenge in terms of the hardware limitation and the computation time; the computation time was increasing with the increase in the size of the training data set. Analyzing the influence of the different hyperparameter values of the neural network on the music generated at the output required extensive running of the model with several combinations of hyperparameter values; this was very challenging as we needed to examine the output for the range of values for each hyperparameter.

1.3 Objectives

1.3.1 Project Goals

The thesis goal is to design, develop and implement a generative deep neural network, namely Variational Autoencoder for fake video music generation, by training it on a corpus of MIDI files of retro video game music from the Vgmusic website. The generated music should not be identical to the input data. Analyze the effect of varying the hyperparameter value of the model on the music generated and to conduct the survey of the generated music from music enthusiasts to qualitatively evaluate the model's success.

1.3.2 Research Questions

When we analyze the requirements of this research thesis, several questions arise:

1. Can the Deep Generative Network like Variational Autoencoder generate the fake music by learning from the notes and chords of the training music data set?
2. Does the generated fake music sound close to the retro video game music?
3. How does the value of the model hyperparameters influence the output of the generated music?

1.4 Structure of the Thesis

This thesis organization is as follows:

Chapter 2. The chapter provides all the necessary background knowledge required to understand this thesis for the reader better. The topics covered here are the basic concepts of Music, Deep Learning, Generative Networks, Technology used. We have also discussed the previous works related to music generation using different methods and algorithms.

Chapters 3. In this chapter, we discuss the method implemented for the thesis, which includes the discussions related to the Architecture, Data set information, Data set extraction, Data pre-processing, tools implemented for music generation.

Chapter 4. This chapter includes experiments conducted on the hyperparameter values of the model and the results of these experiments. The result of the survey conducted on the generated music is discussed.

Chapter 5. This chapter discusses the answers to the research questions.

Chapter 6. In this chapter, we draw the conclusions with a summary of the main findings, along with the limitations of the model and the possible future work.

2 Background

In this chapter, we explain the background knowledge relevant to the thesis in detail, so the reader has a good understanding of all the concepts implemented in the thesis. Included are some basic music theory, the concept of deep learning, important terms related to deep learning as well as an explanation of the deep learning techniques used in the thesis. We also have a comprehensive review of the related work on the thesis.

2.1 Music

The state of the human mind can be affected by music; it can bring changes in the human's mood and also behavior. Music has been a tool for humans to self-regulate themselves. We do not delve deep into the understanding of musical theory but have a concise understanding of it. The concepts of musical theory explained here are based on readings from [Sch13]. Below listed are the essential notations and terms which are helpful for the current work.

2.1.1 Music Notations

- Notes and Chords
  Western music has 12 distinct notes. The twelve notes are A, A/B♭, B, C, C/D♭, D, D/E♭, E, F, F/G♭, G, G/A♭. Usually, middle C is the reference pitch. A,B,C,D,E,F,G are all the natural notes. The sharp note uses the symbol ♯ and the flat note uses the symbol ♭. The one-half step higher than the natural note is Sharp. The one-half step lower than the natural note is Flat. In figure 2.1 the white keys are the natural notes; the black keys are the sharp and flat notes. In the middle of E natural and F natural, there is no note. Notes are of different

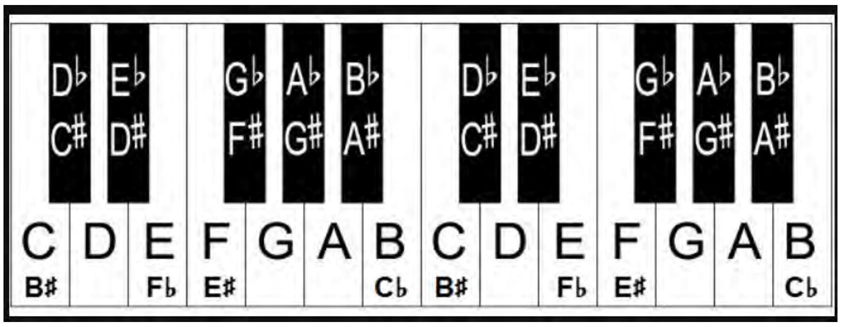


Figure 2.1: Representation of Notes on Keyboard [Key]

versions called an octave. For some notes, the sharps and flats are the same; this makes an Octave consisting of 12 notes. The Note played at twice or half the frequency of itself is called the Octave. The time at which the Note starts is called the onset, and the time at which it stops is called offset. A combination of notes played simultaneously forms the Chords.

- Pitch
  Pitch is nothing but variation in the Note's sound; that is how high or low the note sounds. The Note's frequency of the fundamental sound wave determines the pitch. The sound wave with a higher frequency has a shorter wavelength, but its pitch sounds higher.

- Duration
  Duration is measured in beats. The duration categories are fractions of a beat, mostly in powers of 2. In musical terminology, a whole note is a 4 beat note, and a half-note is a 2 beat note. The most common grouping of beats into regular units is by 4. Note lengths of two half notes or four-quarter notes last the same amount of time as one whole note. Although the whole note, two half notes, is the same in terms of duration, the way it is played is different. A whole note is played only at the beginning and allows it to sound until the end, but if there are two notes one after each other, we hit the first note lasting half of the measure and then hit the second note half of a measure after hitting the first note.

- Key Signature
  The group of notes arranged according to the ascending or descending order of the pitch is called the Scale. Ascending Scale is one in which the note is higher in pitch than the preceding one. In descending Scale, the note is lower

in pitch than the preceding one. In western music, there are two important scales, the Major and the Minor scale. All Major scales follow the same pattern: Whole-Whole-Half-Whole-Whole-Whole-Half. All Minor scales would follow the pattern Whole-Half-Whole-Whole-Half-Whole-Whole. The Base note, and the scale being Major or Minor, defines the Major and Minor keys. A Major key uses the Major scale, and the Minor key uses the Minor scale. These different scales provide different musical moods.

2.1.2 Music Representation

The representation of music can be in the form of [Mül15]

- Sheet music
- Symbolic Music

  MIDI

  Piano Roll
- Audio

We will consider the symbolic music representation, as our thesis makes use of MIDI files. Symbolic music is a digital representation of musical events in machine-readable format [Mül15]. The MIDI file format is explained in detail as the reader can have a good understanding of this format of music representation. We have not explained other music formats here.

MIDI

MIDI (Musical Instrument Digital Interface) is the widely used symbolic format for music representation, where the performance details are saved and reproduced by a computing system. MIDI is a digital protocol of industry standards used for communication between electronic musical instruments and computers. The design of MIDI involves recording musical performances using so-called high-level music features (i.e., features based on musical Note abstractions, such as musical Key, chord progressions, duration, pitch, velocity). Instead of using traditional low-level audio/sound features (i.e., features based on frequency data used to describe audio formats, such as spectral components of audio samples and frequency histograms, etc). Music composers have used it more commonly since its invention in the 1980s. The following explanation is as per the readings from [MID], [Ass+96], [Duv08].

MIDI devices fall into one of two categories they are:

- MIDI controlling devices: These devices translate the musical events into a MIDI format and transmit the MIDI messages.
- MIDI controllable devices: These devices generate sound according to the received messages.

Initially, the MIDI saw its use for keyboard players, and later it was used in various instruments. MIDI wind controllers, MIDI guitars, MIDI drums, and even MIDI accordions are available. MIDI controllers like electronics keyboards connect to a computer or other modules. In the olden method, for the interface between the MIDI controlling and the controllable device (computer), MIDI cables were used, which consisted of 5 pins MIDI ports; with the advancement in technology, the USB and Firewire connections have replaced it.

MIDI messages transmit musical information, this information consist of measurable characteristics such as pitch, timing, and velocity. Using this information, the synthesizer reconstructs the audio signal. An electronic keyboard is commonly referred to as a ”synthesizer” as it has built-in audio reconstruction capability. MIDI recordings are easier to edit, as they store simple instructions that are easy to view and change when compared with audio recordings, where extracting the actual notes is difficult. MIDI is convenient in analyzing patterns relating to notes, rhythms, chords, and instrumentation as it is easy to extract from MIDI compared to audio. It provides a method for interchanging time stamped data between different programs.

MIDI Message

MIDI implements unidirectional data transmission and it is asynchronous; the single transmitted bytes consists of 8 data bits, one start bit and one stop bit. The start bit is indicated by ’0’ and the stop bit is indicated by ’1.’ The middle 8 bits can be either a status byte or data byte; this makes 10 bits. The MSB bit of the byte determines if it

is a status or data byte. If the MSB is ’1’ then it is a status byte; else, if it is ’0’ it is a data byte. A status byte signifies the type of MIDI event taking place. The message consists of one status byte followed by one or more data bytes.
Over 16 channels MIDI messages can be transmitted, which includes information regarding performance. Conceptually MIDI messages are divided into either Channel message or System message. The status byte in the System message has no channel number, and it is not specific to the channel. In the System message, there is no transmission of information regarding the music events. System messages are of three types they are System Common message, System Real-Time message, System exclusive message.
Channel messages address the message to one of the sixteen channels; it uses the four bits of the status byte. Channel messages are of two types they are voice message and mode message; the voice message controls the voice of an instrument; the transmission of voice message takes place over the voice channel. The mode message affects the way the synthesizer responds to MIDI data. It is like a system message, and it differs by the way in which it affects the channel. The mode message affects the particular channel instead of the overall system. The channel voice messages are the information exchanged between the MIDI instruments; these include note on/off, program change, pitch-wheel change, after touch pressure, and controller changes. We will not go in-depth on these concepts here in this thesis; if the reader is interested in having more information, can refer to [Ass+96], [Duv08].
The MIDI events note on/off are the most commonly used channel voice message. The receiver receives the note on event upon pressing the key on a MIDI controller. The message consists of the status byte indicating the selected channel for transmission, followed by two data bytes consisting of information related to the note’s pitch and the velocity with which the key has pressed. The note off event is received by the receiver when the pressed key is released. A numeric value is associated with each note, it is transmitted along with the note on/off message.

MIDI File Format

The standard MIDI files provide the standardized file format specification to store the MIDI file. It provides a standardized way to save, transfer and play the music. It organizes the MIDI messages in multiple parallel tracks and time-stamps to play the events in sequence. Multiple MIDI streams are present in the MIDI files, along with the timing information of each. It also includes information regarding the structure of the

Number(hex)	**Representation(hex)**
00000000	00
00000040	40
0000007F	7F
00000080	81 00
00002000	C0 00
00003FFF	FF 7F
00004000	81 80 00
00100000	C0 80 00
001FFFFF	FF FF F7
00200000	81 80 80 00
08000000	C0 80 80 00
0FFFFFFF	FF FF FF 7F

Table 2.1: MIDI File Variable Length Quantities [Duv08]

track, tempo, name of the track, and the lyrics stored as metadata. The files are of the extensions ".mid," any general MIDI player will be able to play this file. The standard MIDI file uses an 8-bit binary data stream that is converted to HEX ASCII for general viewing. MIDI files use a convention called variable-length quantity to represent the numbers in the file, as one of the aims of the MIDI file is compactness. There is no set length for the parameters in the MIDI file. For the representation of the numbers, it uses 7 bits per byte; the MSB is used to indicate length. Excluding the last byte, all bytes have their 7^{th} bit '1,' and the last byte has the 7^{th} bit '0.' OFFFFFFF is the largest possible number that a variable representation can fit into 32 bit. The table 2.1 shows examples of variable-length quantities numbers representation

MIDI files constitute chunks. The chunk consists of 4-character type and a 32-bit length, which is the number of bytes in the chunk, this length describes the bytes of data to follow, The following section has a series of bytes of MIDI events that form the data section. They are two types of chunks, a header chunk and track chunks. Header chunks contain less information regarding the entire MIDI file, and track chunks contain a sequential stream of MIDI data. The MIDI files begin with the header chunk followed by one or multiple track chunks.

- Header Chunks
  A MIDI file begins with the header chunk having the identifier "MThd." It specifies the information related to the number of tracks in a file, track chunks, delta times.

- Track Chunks
  The track chunk holds the data related to the actual song with the identifier "MTrk." It also stores the MIDI and non-MIDI events with their delta values.

Depending on the number of track chunks, the MIDI file format is in forms 0,1, and 2. The '0' file format has one track chunk, the file format '1' has two or more track chunks intended to play simultaneously, and the file format '2' has two or more track chunks intended to play independently. Standard MIDI files contain events at the top level. The individual events contain two components, a MIDI time and MIDI message they follow each other. The time value called the delta time specifies the waiting time before playing the following message in the MIDI stream. For the first event, the delta time would be zero due to the absence of a preceding event [OO17]. Here we have tried to give some fundamental insight into the MIDI files.

2.2 Artificial Intelligence

Artificial intelligence is a well-known subfield of computer science concerned with machines that automate tasks involving "intelligent" behavior [RN02]. Artificial Intelligence tends to mimic human intelligence by allowing computers to perform tasks and functions that humans would normally perform. Learning, logic inference, and assessing our emotions are among several examples of human actions and behaviors that humans indulge in regularly.

Machine Learning

Machine Learning is a branch of Artificial Intelligence (AI) in which machines or computers learn on their own by analyzing data samples, developed as a new approach to artificial intelligence. In Machine Learning, rather than intending to hard-code everything, the programmer will devise algorithms that allow the computer to derive its own logical rules from given data. There are two main categories of Machine learning techniques Supervised learning and Unsupervised learning.

- Supervised Learning

A machine trained on a set of inputs along with its corresponding output for each input is called Supervised learning. The machine learns a pattern by analyzing all of these training data samples. Upon providing the previously unseen data as input, it can predict the label/output [Broa].

Classification and regression are the two categories in supervised learning problems [Broa].

- Classification: This supervised learning method classifies the output variable into one among many categories. For example, to classify the color of the input data into either 'green' or 'yellow.'
- Regression: This supervised learning method predicts the continuous value. It determines the relation between the dependent variable, called the 'outcome' or 'response,' with an independent variable, called 'features.' The output variable is the real value, for example, house price prediction, given a set of input features such as size, area.

Problems like recommendation systems use a classification method of supervised learning, whereas time series analysis uses the regression method of supervised learning.

- Unsupervised Learning

A machine trained on input data samples without its corresponding output is called Unsupervised learning, and the machine learns the underlying structure to make predictions. Real-world applications use this more due to the difficulty in acquiring labeled data. This approach develops the learning algorithm and makes predictions for the unseen data by recognizing the pattern from the input data [Broa].

Clustering and Association are the two methods of Unsupervised learning problems [Broa].

- Clustering: It involves the task of dividing the data points into multiple groups or clusters such that the data point of a similar type is in one group. There are two types of clustering, namely hard and soft clustering. In Hard clustering, the association of individual data points is limited to only one group. In soft clustering, the association of individual data points is a probability likelihood with each of the pre-defined groups.

- Association: It is the rule-based technique in which it finds the relation between the variable and the feature, which means it maps the dependency of one item with the other item.

Now we shall discuss the neural networks. A brief history of neural networks, along with the functionality and different forms of neural networks, followed by definition and description of the model used in this thesis.

2.3 Neural Network

In the mid-20th century, there was research on artificial neural networks, being inspired by the biological mechanisms of the human brain in processing the signals. Inspired by the discoveries in neuroscience, McCulloch and Pitts in 1943 proposed the first artificial neuron [MP43]. Based on mathematical algorithms, Frank Rosen-Blatt built a neural network [Ros58]. Similar to the neurons of the human brain, artificial neurons can also learn upon adjusting the parameters of the neurons.

2.3.1 Perceptron

A Perceptron is the basic block of a neural network. It allows neurons to learn and process individual elements in the training set. Let X be the input vector to the neuron that is (x_1, x_2, ..., x_n) . Multiply every component of the vector by their respective weight factor given by w_i, and these weights determine if a neuron fires or not. The product of the input x_i along with their respective weight factor w_i gets added up to form the weighted sum. The constant value term called 'bias' gets included in this weighted sum to offset the result. Thus obtained sum is then made to pass through the activation function; the resulting value thus obtained is the output. If the sum exceeds a certain threshold of the active function, it outputs a signal; else, it does not return an output [1]. Figure 2.2 shows the representation of a single perceptron. Many of these neurons together form the neural network [Neu]. Several layers of perceptron cascade to

[1]https://en.wikipedia.org/wiki/Perceptron

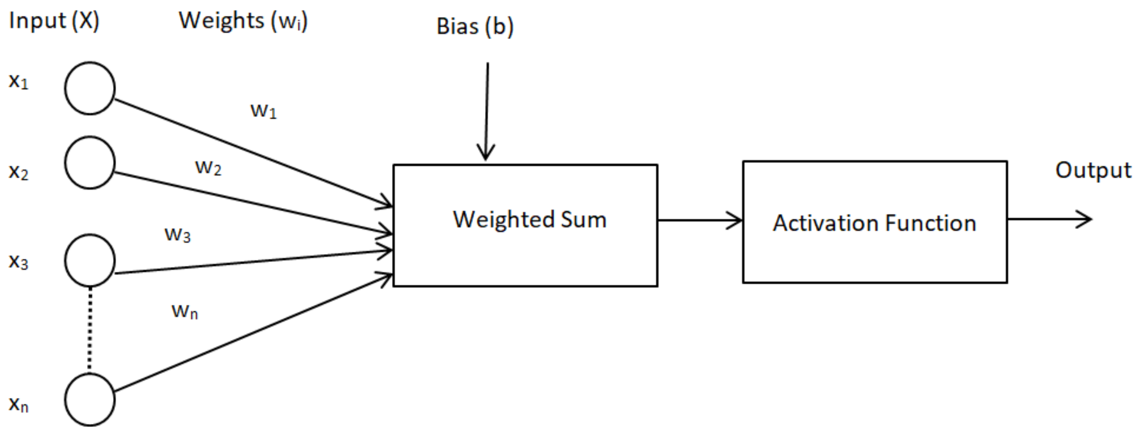


Figure 2.2: Representation of Perceptron

form the Artificial Neural Network(ANN). There are many different ANN architectures, such as Feedforward Neural Networks (FNN), Recurrent neural networks (RNN), and convolutional neural networks (CNN). We will focus on Feedforward neural network as our thesis implement this.

2.3.2 Feedforward Neural Network

A group of neurons entwined in the form of a Directed Acyclic Graph (DAG) in this network. FNN's combine multiple neurons to form a directed graph without cycles [Zel94]. The information flows from the input layer to the output layer. Every neuron in each layer connects to every other neuron in the next layer without any feedback to approximate a function given input x. This type of neural network forms a fully connected network. The layers in this network are called fully connected layers. It consists of one input layer, an intermediate layer called the hidden layer, and the output layer. The number of the hidden layer is arbitrary, which depends on the designer, and the output layer for computing the output. It represents the foundation of most deep learning applications.

2.3.3 Deep Learning

It is a subfield of Machine learning, concerned with the algorithms inspired by the structure of a human brain and modeled to work like it. Like humans use the brain to analyze and classify different types of information by learning from a large set of examples. The Deep Learning model consists of multiple layers of neurons to learn from the large data sets and perform some tasks [GBC16]. The early 1980s witnessed the initial research in the area of Deep Learning. However, this area has experienced a significant increase in research in recent years, owing to the introduction of new and faster CPUs and GPUs. Deep learning uses a given Machine learning algorithm in cascade to carry out the task, that is, an algorithm applied to input data and obtain the output; afterward, apply it again to that output, and so on in a recursive manner. Deep Learning approaches use many stacked processing layers to automatically learn representations and internal structure of raw input data [LBH+15]. On the input, these layers usually compute simple differentiable functions based on their parameters, which are known as weights. For example, in image classification, the output of the first few layers typically consists of edges at particular orientations and locations in the image, while later layers can encode more abstract concepts such as patterns or even objects by utilizing the information of the early layers. These are the type of artificial neural networks with multiple hidden layers when compared with usual ANNs. Thus it can be used to carry out more complex tasks with improved results.

The two models, discriminative and generative, implement deep learning techniques in their output predictions [Jeb12]. Let us consider a random variable x which is multidimensional, and these correspond to the features observed in a dataset. Let y be a discrete random variable where each state corresponds to a class of the objects in the datasets.
Discriminative models calculate the distribution $p(y \mid x)$. In other words, a discriminative model concentrates on a particular task, with less emphasis on the distribution p(x) of the dataset. Discriminative models perform the task by modeling the conditional probability, without making any assumptions about the data point and unable to generate new data instances.
Generative approaches aim to model the equation provided by the Bayes Rules. In contrast to the discriminative model, it will model the data points joint probability along with it using the maximum likelihood. In practice, discriminative models are more efficient when the goal is to accomplish a specific task. On the other hand, reliable

generative models are more powerful when many tasks are to be solved by the same model or when missing values are present in datasets.

2.3.4 Deep Generative Model

We have now had a basic knowledge of the deep neural network. As our thesis implements Generative Deep Neural Network, let us understand more about this. Generative modeling is one of the unsupervised learning methods in which the model learns from the training data sets underlying distribution to generate new data. This model finds its application in different domains, such as in the generation of Image, Text, and Audio. For example, it constructs new variations of cars by learning from the enormous examples of a training data set consisting of a car and understanding the minute details of the training data to generate a new car. The generated car can vary in color, height, shape, and various other attributes. We need to achieve a distribution that is nearly close to the original distribution, from where we can obtain a new sample. Depending on the generated samples from the model, we can conclude whether the model has successfully understood and learned from the underlying data pattern without supervision. Explicit density functions and Implicit density functions are the two classifications of generative models as shown in figure 2.3. We have given a brief overview of these models below, the following explanation is from the readings of [YU20].

- Explicit Density
  This generative model defines an explicit density function P_{model}, it assumes the prior data distribution, which allows us to vary the training data parameters and evaluate the model. On the training data, It attempts to increase the function's likelihood. The main disadvantage of this is, some density functions are intractable. Depending on the intractability, Tractable density and approximate density are the two types of explicit density models [Goo16].

  1. Tractable Density

     a) Fully Visible Belief Nets [FVBN]
        In 1996 Frey introduced this model [FHD+96], the m-dimensional probability distribution of vector x is decomposed into a probability distribution of

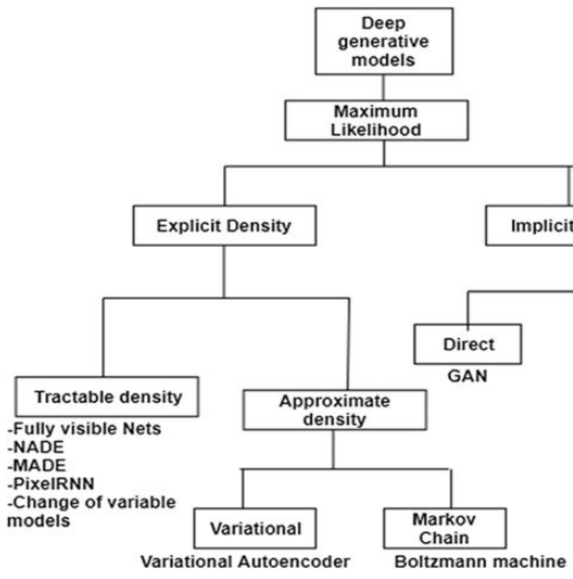


Figure 2.3: Taxonomy of Generative Model [Goo16]

1-dimensional product using the chain rule.

$$P_{model}(x) = P_{model}\,(x_1) \prod_{i=2}^{n} P_{model}\ \ (x_1 \mid x_i\ , x_{i-1}) \tag{2.1}$$

Multiply the x_1 probability distribution with the distribution over the following element until the distribution over the vector's final element, that is x_2 to x_n. This model has been improved and forms the basis for implementing the model Pixel Cnn [VKK16]. These model generates high quality data. The downside of the model is that the sample generation is prolonged; that is, it generates one sample at a time, and mainly the generation is not controlled by a latent code.

b) Nonlinear independent components analysis
This is based on transformations between two different spaces , for example simple Gaussian distribution is transformed into another space using the non-linear function [HP99].

$y = g(x) \Rightarrow P_x(x) = P_y(g(x)) \mid det\left(\frac{\Delta g(x)}{\Delta x}\right) \mid$

The simple distribution over the latent space along with transformation g(x) can be made density tractable by properly designing g. The disadvantage of this model is that it imposes a restriction on the choice of g. The model should be invertible, which means the dimension of the latent variable should be equal to the dimension of the input data.

2. Approximate Density
This explicit density functions are defined to overcome the limitations of the tractable density functions. These are the one which is intractable but then use the tractable approximations to increase the likelihood. Variational approximation (variational method) and Stochastic approximation (Markov chain) are the two types of Approximate Density [Goo16]

   a) Variational Approximation
   It is a deterministic technique to approximate inference for the statistical model parameters. The Variational Autoencoder is the popular deep generative model in variational approximation. The expression for the density function is

   $\log p(x) \geq \log p(x) - D_{KL}(q(z) \| p(z \mid x)) = E_{z \sim q} \log_p(x, z) + H(q)$ [YU20]

   Because we need to marginalize out the random variable z, the density function is intractable. The z variable provides the compressed data for the input data. Because of the intractability problem, we must apply a variational approximation technique that incorporates a distribution q over the latent variable z to the extent that this distribution q is closer to the true posterior over the latent variable z. The distribution q allows us to construct a lower limit that describes the true density and allows us to make a bound that gets smaller.

   This method defines a lower bound

   $\mathcal{L}(x; \theta) \leq \log P_{model}(x; \theta)$

   The data generated by this method tend to have lower quality. There is no adequate way to measure sample quality quantitatively, this is a personal judgment rather than an empirical truth.

   b) Markov Chain Approximation
   Another major family of models is the Boltzmann machine (Hinton and Sejnowski 1986) [HS+86]. These models also make use of an intractable

explicit function. In this case, energy function and the probability of a particular state proportional to e raised to the power of that function defines the Boltzmann machine. In order to convert this value to an actual probability, it is necessary to renormalize by dividing the sum over all the different states, and that sum becomes intractable. Using Monte Carlo approximation methods, we are able to approximate this density function [GBC16]. There are challenges with using Monte Carlo approximations, one of them being the functions failing to mix between different modes. Also, this methods perform poorly in high dimensional spaces because the Markov chains break down for very large images, because of this, it is rare to see Boltzmann machines implemented for image modeling tasks, such as we have seen with GAN's and other generative models [YU20].

- Implicit Density
  These are the models trained without explicitly defining the density function. By interacting indirectly with P_{model}, sampling from it offers a way to train the model. These are on the right side of our taxonomy, as depicted in figure 2.3. The primary example from this family is the generative stochastic network [TAY14]. The scaling of high dimensional space is not successful in Markov chains, and this results in increased computational costs for implementing the generative model; the design of GAN's overcame this issue. GAN's were the only member at the time of their introduction of this family; additional models have added to this later on.

For our thesis, we have chosen the Variational Autoencoder. In order to understand how this model generates an output, we need to first understand the basic working of Autoencoder, which forms the base for the VAE.

2.3.5 AutoEncoder

Autoencoders are neural networks that provide the output the same as the input data by compressing the input data X $\in R^n$ to lower dimension called latent dimension representation h $\in R^m$ and then reconstructing the output from this low dimension representation [Jor18].

It is an unsupervised learning model without the corresponding output labels to the input data. Autoencoders consist of Encoder, Code space or Latent space, Decoder.

- Encoder: compresses the high dimension input data X of n-dimension into smaller m-dimension latent space.
- Code Space or Latent Space: This layer is called the bottleneck layer because of its lower dimensionality than the input layer. This layer decides on the crucial aspects of input data that are significant to reconstruct the input data and discards the information that does not impact the output..
- Decoder: The compressed data from the latent space of m-dimension is re-mapped to the input data of n-dimension in this layer. The reconstructed output will be lossy; that is, the output will be close to the input. Figure 2.4 shows the structure of an AE. The encoder and decoder have an equal number of neurons. The figure of AE includes FNN as the encoder and decoder network. The small circles inside the rectangular box of the encoder and decoder indicate the number of neurons present in the input and output layers; the number of neurons decreases from the input layer to the latent layer.
  The number of neurons in the Latent space defines the dimension of the code space or the latent space. The encoder and the decoder network are identical. The process of moving the data from the input layer to the latent layer is done by the encoder, moving from the latent layer to the output layer is done by the decoder. Unlike other models that predict certain information about the input, the AE's will instead reproduce the output closer to the input.

Let h=g_θ(x) represents the encoder function, and x'=f_ϕ(h) represents the decoder function, where the θ, ϕ are the parameters of the encoder and decoder. The Autoencoder aims at minimizing the difference between the input and the output.

The 7 different types of auto-encoders are [2]:

1. Denoising Autoencoders
2. Sparse Autoencoders
3. Deep Autoencoders

[2]https://iq.opengenus.org/types-of-autoencoder

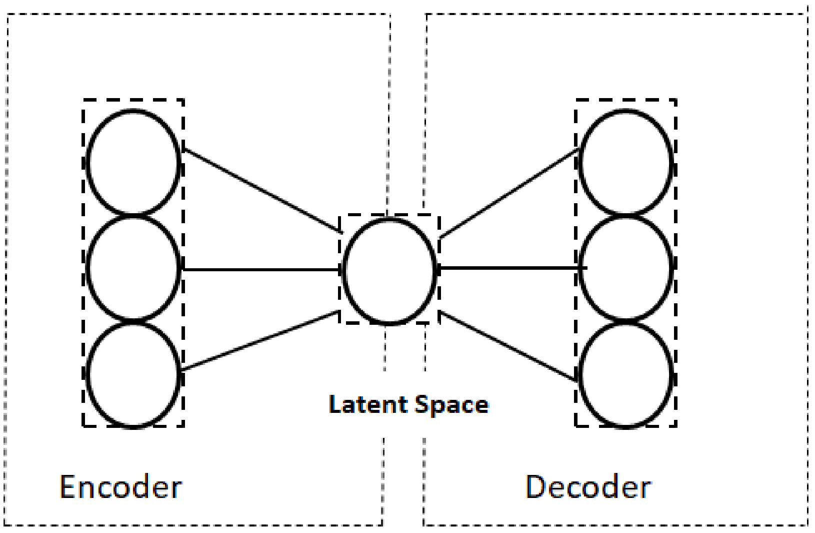


Figure 2.4: Autoencoder

4. Contractive Autoencoders
5. Undercomplete Autoencoders
6. Convolutional Autoencoders
7. Variational Autoencoders

2.3.6 Variational Autoencoder

The limitation of the AE is, there is no real generation of new content; this is because it is very difficult to guarantee that the organization of the latent space happens in a smart way that is compatible with the generative process. The generative model learns the joint probability distribution of the underlying data; by sampling this distribution, it generates new data. In order to use the Autoencoder for generative purposes, we consider the VAE. Knigma and Welling first proposed VAE [KW19], it is an autoencoder that avoids overfitting by regularized (technique to reduce the errors by appropriately fitting the function on the given training set) training and has the latent space that enables the generative process. Instead of building the encoder that learns to output a fixed value for each latent state attribute, the model aims to find the probability distribution of each latent state attribute which is capable of explaining most of the data samples.

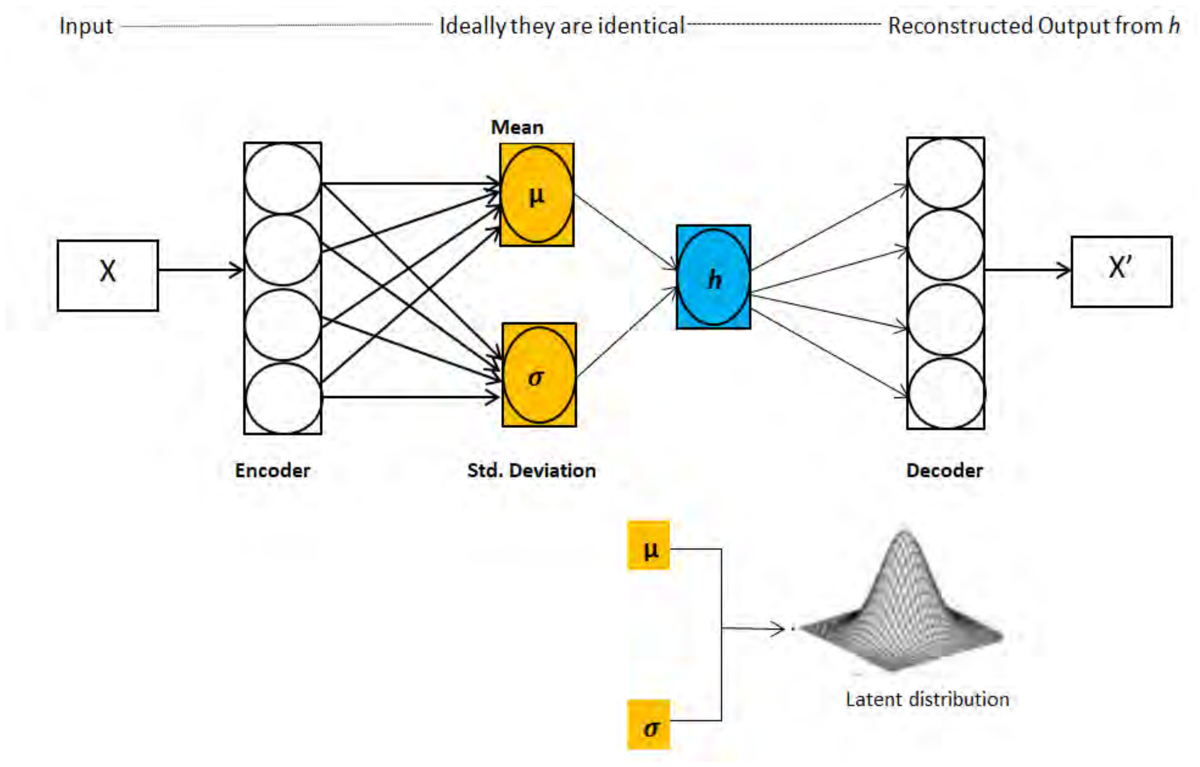


Figure 2.5: Variational Autoencoder

The following explains the Variational Autoencoder functionality based on the readings from [Doe16]. The Variational Autoencoder consists of Encoder, Decoder, and latent space similar to Autoencoder shown in figure 2.5. The Variational Autoencoder differs from the Autoencoder in the representation of the latent space. Instead of the point representation of the input data in the latent space, the latent space consists of a continuous normal distribution from which data sampling takes place, as shown in figure 2.6. Given the input X at the encoder, it encodes the input data into the latent vector h at the encoder's output, which forms the latent representation or latent space. Unlike in Autoencoder, the latent space is not obtained by directly mapping the input, instead sampled from the normal distribution with the mean(μ) and variance (σ) obtained at the output of the encoder. Thus sampled values from the distributions form the latent vector or hidden vector h. These mean and variance are the parameters of the Gaussian distribution. The latent space is also called the sampling layer. The encoder efficiently compresses the data into lower dimensions and outputs vectors of mean and variance. This lower dimensional space is stochastic. The number of the mean (μ) and variance (σ) at the output of the encoder depends on the latent dimension. The decoder has the input h, and these latent values pass through the decoder to reconstruct the input back.

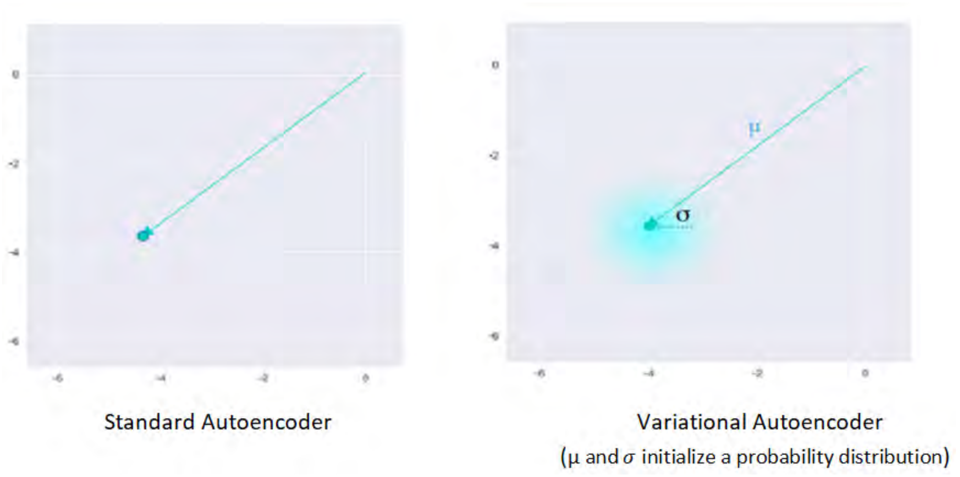


Figure 2.6: Difference in Latent Space representation between Autoencoder and Variational Autoencoder [Var]

The encoder and decoder consist of one or more hidden layers, where each of the hidden layer at the encoder extracts important features from the input data, thereby reducing the dimension of the input data in the latent space, whereas the hidden layer at the decoder upsamples the latent data to obtain back the input data. Thereby, the decoder increases the latent space dimension into the input dimension. Thus the encoder maps the input X into mean, variance, and the decoder maps the h into mean, variance at the decoder to produce the output similar to the input. The encoder maps the input X probabilistically to the latent space; in turn, the decoder maps the latent space h probabilistically back to the input space.

We can better understand the model functionality by considering probabilistic concepts. Variational Autoencoders are Probabilistic Generative Models(PGM). From a graphical model perspective, with the graph theory, we can see the dependency of the random variable. Based on the stochastic process, the unseen continuous variable h generates the observed variable X. We obtain this unseen h by the prior distribution $P(h)$ (profile of multivariate Gaussian) and to generate X, we use the conditional distribution. Let us consider a data set X. As the data dimension is reduced from a larger dimension to a smaller dimension through the encoder, there will be a data loss. Also, sampling from this bottleneck makes our reconstruction inaccurate to the input at the encoder. We can maximize the reconstruction likelihood of each data point by optimizing the parameter Φ. Φ is the parameter of the decoder network.

$$\Phi' = \arg\max_{\Phi} P_{\Phi}\left(X\right) \tag{2.2}$$

Which makes the likelihood of each data point X as high as possible. This, in turn, means that the region around the input having more samples has a high probability mass. By applying the logarithmic rules to the above equation, it gets modified to

$$\Phi' = \arg\max_{\Phi} \log P_{\Phi}\left(X\right)(2.3)$$

This logarithmic equation makes the calculation easy as it changes from product to sum. Using the Bayes theorem, the posterior distribution of the hidden space is given by $P_{\Phi}(h \mid X)$.
Using Bayes theorem we can write

$$P_{\Phi}(h \mid X) = \frac{P_{\Phi}(X \mid h)P_{\Phi}(h)}{P_{\Phi}(X)} \tag{2.4}$$

The joint distribution can be defined using the total probability law as the product of simpler distributions given by $P_{\Phi}(X, h) = P_{\Phi}(X \mid h)P_{\Phi}(h)$. The term $P_{\Phi}(X)$ is called the evidence which is calculated by marginalization of vector h of latent space, which is given as follows

$$P_{\phi}\left(X\right) = \int P_{\Phi}\left(X \mid h\right)P_{\Phi}(h)dh \tag{2.5}$$

The computation of the above term for each data sample becomes very expensive and intractable. To make further calculations, we need to calculate the posterior distribution $P_{\Phi}(h \mid X)$. For this, VAE's implement the Variational approximate inference [Ran+16] in which it uses proxy distribution $Q_{\varphi}(h \mid X)$. It is the estimation of the true intractable posterior distribution because $Q_{\varphi}(h \mid X)$ is only an estimation of the true posterior. $Q_{\varphi}(h \mid X)$ is an encoder distribution function, where φ are the parameters of the encoder. There is a need to keep track of the difference of these two probabilities as this approximation technique will lead to a large gap between the true and approximated distribution. Identifying the appropriate parameters φ results in an additional problem of optimization, for this, there is a need to minimize the distance between the true and the approximate distributions. For this measurement, we introduce the Kullback-Lieber Divergence (KL), it measures the similarity between the two distributions.

Because of intractability in (2.3) we cannot optimize the likelihood directly, for that instead, we have to define and optimize a lower bound of the likelihood. Assuming that using the proxy distribution $Q_{\varphi}(h \mid X)$ we will be able to estimate h. By taking the logarithm of the expectation in (2.3) we get

$\log P_{\Phi}(X) = E_{h \sim Q_{\varphi}(h|X)}\left[\log P_{\Phi}(X)\right]$

$\log P_\Phi(X) = E_{h\sim Q_\varphi(h|X)} \left[\log \frac{P\Phi(X|h)P_\Phi(h)}{P_\Phi(h|X)}\right]$

$\log P_\Phi(X) = E_{h\sim Q_\varphi(h|X)} \left[\log \frac{P_\Phi(X|h)P_\Phi(h)}{P_\Phi(h|X)} \frac{Q_\varphi(h|X)}{Q_\varphi(h|X)}\right]$

$\log P_\Phi(X) = E_{h\sim Q_\varphi(h|X)} \left[\log P_\Phi(X \mid h)\right] - E_{h\sim Q_\varphi(h|X)} \left[\log \frac{Q_\phi(h|X)}{P_\Phi(h)}\right] + E_{h\sim Q_\varphi(h|X)} \left[\log \frac{Q_\varphi(h|X)}{P_\Phi(h|X)}\right]$

Applying the Kulback Lieber divergence in the above equation after some mathematical calculations, we obtain the following result

$$\log P_\Phi(X) = \underbrace{E_{h\sim Q_\varphi(h|X)} \left[\log P_\Phi(X \mid h)\right]}_{Reconstruction\ term} - \underbrace{D_{KL}\left(Q_\varphi(h \mid X) \| P_\Phi(h)\right)}_{Closed\ form} + \underbrace{D_{KL}\left(Q_\varphi(h \mid X) \| P_\Phi(h \mid X)\right)}_{Intractable, but \geq 0} \tag{2.6}$$

The first term is related to reconstruction likelihood, and it measures how well the approximation $P(X \mid h)$ produces data samples from the given latent state. The second term is a KL divergence between General and normal Gaussian; this makes sure that the learned approximate posterior distribution is similar to the true prior distribution(this term must be very small). The third term by the property of Jensen's inequality it will be always non-negative $D_{KL}(P\|Q) \geq 0$. The first and the second term together form the Evidence lower bound(ELBO) [Yan17]; it is the tractable lower bound, which means by maximizing this we can increase the probability of the data observation. As the third term is always positive, the ELBO is indeed the lower bound of log-likelihood. As we see the intractable term in (2.6) the ELBO is a lower bound but not the pure optimization. This is the loss function on which we can apply back propagation and we work on the negative ELBO as loss function is minimized, hence .

$$E_{h\sim Q_\varphi(h|X)} \left[\log P_\Phi(X \mid h)\right] - D_{KL}\left(Q_\varphi(h \mid X) \| P_\Phi(h)\right) = -ELBO \tag{2.7}$$

The left-hand side of the equation (2.7) is the main objective function of the VAE which we want to maximize. The final objective function for the VAE is given by.

$$L(\Phi, \varphi; X) = -ELBO = E_{h\sim Q_\varphi(h|X)} \left[\log P_\Phi(X \mid h)\right] - D_{KL}\left(Q_\varphi(h \mid X) \| P_\Phi(h)\right) \tag{2.8}$$

In order to minimize the ELBO, we need to tune the parameters Φ and φ, for that we apply backpropagation on the encoder and decoder networks. Sampling from the output of the encoder network forms the hidden vector, which is non-deterministic and makes the node stochastic. In order to apply the backpropagation, the node should be deterministic; thereby we cannot apply random sampling in the bottleneck layer. In order to overcome this problem, we introduce a technique called the reparametrization

trick.
The Reparameterization trick was the key insight given by Kingma and Welling in order to change the parameters straight forward. This solution reparameterizes the h so the stochasticity becomes independent of the parameters of the distribution. We can express $h \sim Q_\varphi(h \mid X)$ as a two step process. Sample noise from a simple distribution $\epsilon \sim P(\epsilon)$ and according to this technique it is possible to transform $h \sim Q_\varphi(h \mid X)$ into a deterministic function by considering $g_\varphi(\epsilon, X)$ as a function of another random variable, this shapes the random noise into an arbitrary distribution. Due to this reparameterization, we externalize the randomness in h, and the differentiation can be applied straightforwardly with respect to Φ and φ. The encoder and decoder network parameters are now independent of stochastic sampling. Instead of sampling h from μ and σ of the normal distribution, we sample h from

$$\mathbf{h} = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1) \tag{2.9}$$

The reparameterization trick is showed in figure 2.7.
The μ and σ are the deterministic outputs generated by the encoder and the $\odot$ operator determines the element-wise product. Since the approximated posterior is a multivariate Gaussian distribution, the sample noise must also follow a Gaussian distribution to get a similar distribution. We can achieve this by sampling ϵ from the standard normal distribution. Even after the reparameterization technique, there is the presence of a stochastic node. As we added the ϵ, $\epsilon \sim \mathcal{N}(0, 1)$, the stochastic sampling does not happen in h. The backpropagation applied to the modified model looks as shown in figure 2.8.
To generate new samples during testing, we consider only the generative part which is the decoder. For generation, we simply input the values h into the decoder. The encoder is separated, including the multiplication and addition operations that change h.

2.4 Important Deep Learning Terms

This section describes some commonly used deep learning terms related to the networks and architectures and is helpful for the readers as they encounter these terms further in the readings.

- Backpropagation :

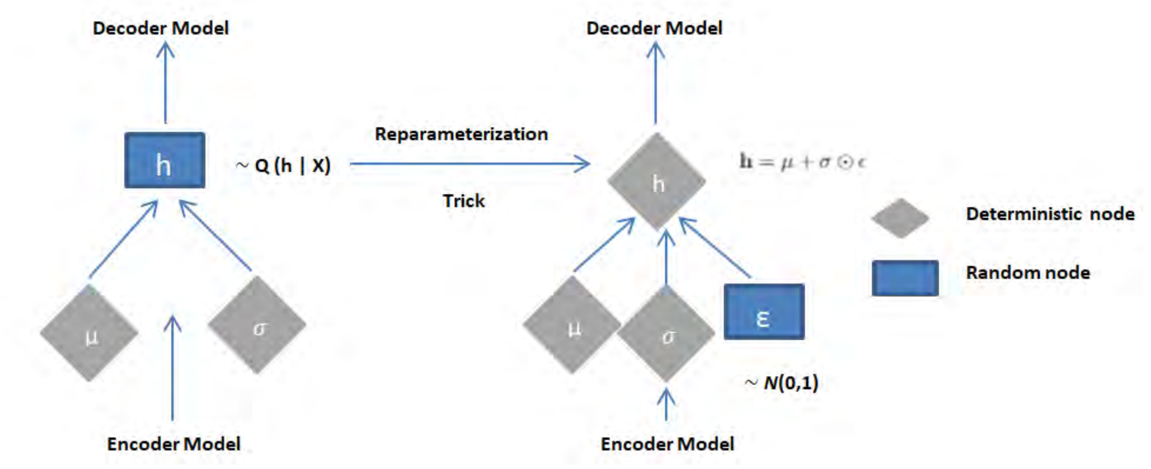


Figure 2.7: Reparametrization trick [Rep]

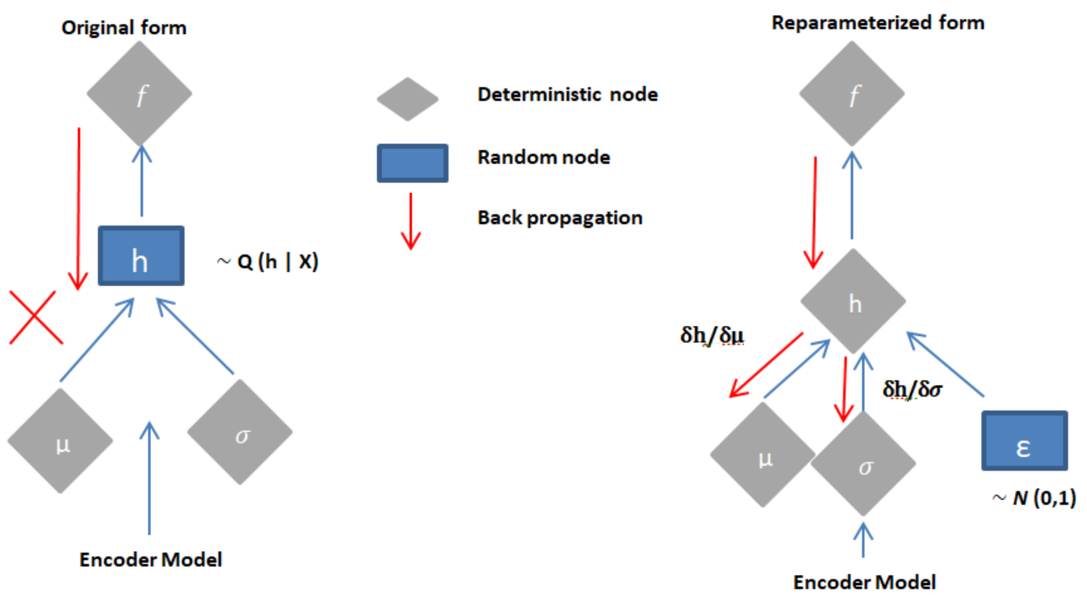


Figure 2.8: Backpropagation on Reparameterized model [Rep]

The parameters of the neural networks are updated using this method, in which it calculates the gradient. It calculates the derivative of the loss function with respect to the parameters. Implements the chain rule of derivations, as it does derivations of output passing back to the input, moving through several network layers. This method can be applied only on the deterministic nodes. The result of the gradient calculated give information on how to tune the parameters of the model to get the best result out of the network [Hec92].

- Epoch
  The process of passing the entire data set once through the neural network and calculating the gradient of the parameter using backpropagation is called an epoch. In one epoch, the network gets to see all the training data samples only once [3].

- Batch Size
  The number of training samples given for the network in a single iteration, means the number of samples moving through the network forward and backpropagating. The training dataset gets divided into several batches depending on the batch size, and processing of all batches of data makes one epoch [Ter].

- Learning Rate
  This hyperparameter controls the parameter(weights) updates of the network. Updating the network weights with respect to the gradient loss obtained by the backpropagation is monitored by this hyperparameter. The lower the value of this hyperparameter, the slower the movement of the objective function towards the slope; this results in slower learning processing for the network to reach the global minimum. Higher learning rate values speed up the descent process, but there is a chance for the objective function to miss the global minimum. Choosing the value of this hyperparameter is done on a trial and error basis [Brob].

- Activation function
  The activation layer brings in non-linearity into the network. Without an activation mechanism, a neural network is basically a linear regression model. With any number of linear functions in the hidden layer of the neural network, all layers will behave the same way, as the combination of two linear functions results in a linear function. Neurons cannot learn with just a linear function. The activation function transforms the input in a non-linear way, allowing it to learn and perform

[3]https://deepai.org/machine-learning-glossary-and-terms/epoch

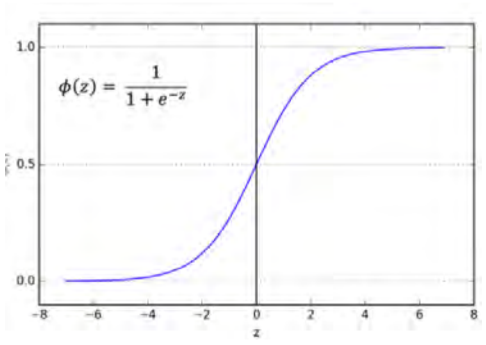


Figure 2.9: Sigmoid
[SS17]

more complex tasks. The different types of activation functions are the Heaviside step function, Sigmoid, ReLU, Leaky ReLu [SS17].

- Sigmoid
  The Sigmoid Function, also known as the logistic function, is a common non-linear activation function. It works by squeezing the output of the neurons into a range of 0 to 1. The activation function has the following mathematical form:

  $$\sigma(z) = \frac{1}{1+e^{-z}}$$

  'This function is of the form 'S.' The steep middle part of the curve causes this shape. Since the curve is very steep, minor changes in the input result in significant changes in the output. It takes the real number and reduces it to a number between 0 and 1. It is useful in the model where the task is to find the output as a probability. The Y values appear to respond less to changes in X as it gets close to either end of the sigmoid function. At the edges of the graph, the bigger the input values, the smaller the difference in the output. In other words, the gradient at these regions becomes small [SS17].

- ReLu
  The Rectified Linear Unit (ReLU) is a computationally efficient activation that does not saturate in real values in the positive region [SS17] . The output for this function is the input as long as the input is positive; if the value is negative, the result is 0. Since the feature ignores all negative values, the networks are relatively quick to train. With ReLU, negative values will cause neurons not to fire (zero output). Since approximately half of the neurons

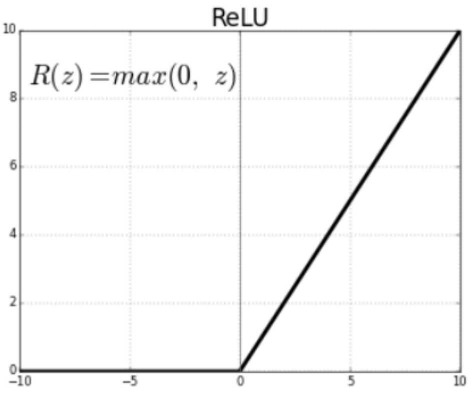


Figure 2.10: ReLu
[SS17]

will be inactive after initialization, ReLU networks will be faster to train from scratch. It is less costly computationally than the tanh and sigmoid and requires fewer mathematical operations.

– tanh
It is similar to the sigmoid activation function, only difference is that on the lower range values it approches -1 unlike 0 in sigmoid activation [SS17].

- Cost Function
The estimation of the network about its performance is provided by the cost function. The cost function is also termed as Loss function. It estimates the correctness of the model in evaluating the relationship between Input(X) and Output(Y). The cost function calculates the distance or difference between the predicted and the actual value. That is the distance between the expected output Y for the given X and the actual response Y' in a typical neural network environment. This indicates the network, whether or not its predictions are correct, and by how much. The network is performing poorly if the cost function returns a large amount, indicating that the distance is large. The aim of the network is to close the gap between its predictions and the actual response that is in turn to minimize the loss function. It is also referred to as an error; this can be estimated by iteratively running the model and estimating the ground truth against the predicted value. Therefore, the objective of an ML model is to find parameters, weights, or a structure that minimizes the cost function. It is the most common method for adjusting the weights of the nodes, with the random weights assigned

initially. Commonly used Loss Functions are Categorical Cross-Entropy, Binary Cross-Entropy, Mean Squared Error, Mean Absolute Error [Asl]

2.5 music 21

It is a toolkit based on python language developed to aid the learning of music on a computer. This toolkit supports studying the music of large data sets. The music 21 library enables the music enthusiast to know the overall profile of the music. In the year 2008, the professors of MIT Michael Cuthbert, Christopher Ariza, Benjamin Hogue, and Josiah Wolf Oberholtze introduced this [Mus].

2.6 Percentage Similarity

To measure the similarity between the two lists, we employ this method. It calculates the number of distinct and similar values in both the lists and calculates the quotient multiplied by 100 to get the percentage similarity between the two lists [4]. We use this method to measure the similarity of the Generated chords sequence with the input sequences. This value gives the confidence that whether the output notes/chords sequence generated copies any input notes/chords sequence or not.

2.7 Technology

- Google Colab
  It is a special version of the jupyter notebook that runs on cloud. It offers free CPU, GPU, TPU resources well suited for running machine learning models. It has pre-installed libraries to carry out machine learning tasks.
- Tensorflow and Keras
  These are the popular frameworks in machine learning, which consists of packages and libraries to make the machine learning task easier. The Tensor flow consists of both high level and low level API's to carry out various machine learning tasks,

[4]https://www.geeksforgeeks.org/python-percentage-similarity-of-lists/

whereas Keras is a high level API consists of libraries related to neural networks, and it runs on top of Tensorflow [5].

- Python
  Python is a programming language that is most commonly used in machine learning tasks because its very simple and also has good community support. It has many open source packages, which helps to carry out the tasks.

2.8 Related Work

This section provides an overview of the neural network implementation in the generation of music. Algorithmic music composition was a popular computer science field. Improvement in deep learning and neural network brought about its implementation in the music domain. There has been work done on music generation by implementing the feedforward neural networks and Recurrent Neural Networks. Most of the music generation works, implement the use of Recurrent Neural network, as it has the ability to remember the previous state. In 1994 Michael Mozer introduced the Auto predictive recurrent neural network, which used Bach and European folks as training data. This network was able to generate a new piece of music by predicting the next pitch [Moz94].

Hadjres and Briot implemented a system called MiniBach to generate music based on melodies from the input. It employs a Feedforward Neural Network [BHP17], this is the reference for this thesis in implementing a feedforward network. The Deepbach was developed by improvising the Minibatch, which implemented the LSTM networks and two feedforward networks[HPN17].

After the introduction of Variational Autoencoder in 2014 by Kingma and Welling, it has seen increasing use in generative problems. Majorly it was used in the task of image generation; it also has few works in the music generation. The base reference for this thesis is from the work of Fabius [FV14], in this work, the author implements the Variational Autoencoder with the Recurrent Neural network for the generation of video game like music. They used 8 MIDI files of the retro video Game music from Vgmusic.com. The data preprocessing involved dividing the song into shorter lengths and mapping this into the data points for further processing. It shows how the music can

[5]https://analyticsindiamag.com/tensorflow-vs-keras-which-one-should-you-choose/

be generated using the latent samples but does not provide an in-depth understanding of latent code implementation. The generated music was identical to the input music.

Music generation with variational recurrent autoencoder supported by history [YT20], they created a system in which the data related to pitch, octave, and duration was encoded in the form of one-hot encoding and used the natural language processing technique for encoding the data. The author fedback the output of the decoder back as the input to the decoder as a reference of the previously generated notes. The network used four LSTM layers for encoder and decoder, and it could predict the next note depending on the previous notes.

The polyphonic music generation using VAE was implemented by Hennig [HUW17] even this implemented the LSTM network. This introduced the modification in the VAE architecture called classifying VAE. This consists of the trained classifier, along with the generative model to summarize the class of each data point. The author used this model for the algorithmic music generation in which the model learns to generate music in different keys.

Variational Autoencoder with the recurrent network called ”Music Vae” was used by Roberts [Rob+18] to reproduce the musical sequence of longer duration. The encoder implements a single layer Bi-directional LSTM and the 2-level hierarchical Rnn within the decoder. They used the method in which they represented melody loops as a sequence of 32 categorical variables taking one of 130 discrete states,128 out of 130 represents the note-on pitches, and the last two represent the rest and hold states. They could successfully generate 2 bar melodies.

3 Methods

The main intention of this research is to generate an exciting video game like music using the retro video game music MIDI file and also analyze the effects of the hyperparameters of the neural network on the generated music. We carry out the following workflow.

1. Extracting the training data.
2. Data preprocessing.
3. Training the Variational Autoencoder with the preprocessed data.
4. Generate new music with random input.
5. Analyze the effect of music generated by varying the values of the hyperparameters.
6. Evaluate the generated music by carrying out a survey with music enthusiasts.

3.1 Training Data

Obtain the dataset for training from the Vgmusic.com website [1], and thus acquired datasets are in MIDI format. It has roughly 31,810 MIDI files of different video game music. However, for our thesis, we are only interested in retro video game music, in which the music is composed with a single instrument. The category of music on the Vgmusic website we include in this thesis are Pokemon(Nintendo Gameboy), Piano only, and NES(Nintendo). The Vgmusic has organized the game music files into different folders per each console, which helps us extract the music at ease as per our requirements. We manually download the MIDI files from the Vgmusic website and store them on the computer's local disk. When using google colab, we need to store the music files on google drive.

[1]https://www.vgmusic.com/

3.2 Architecture

This section discusses the pre-processing of data, the Variational Autoencoder architecture implemented, and other components used for generating the music. Here we first explain the pre-processing of the input data, then explain the architecture with its components and the description of the encoder and decoder.

3.2.1 Data PreProcessing

The stored .mid file extracted from the Vgmuisc website is then loaded and parsed using the tools from music21 library modules. Our thesis involves only a single instrument Polyphonic music (multiple notes played together). In this, we consider that the multiple notes played simultaneously form the single chord. After parsing, we extract the notes, chords, and keys from the training music data streams using the music21 library modules. We include only the music stream consisting of the C major key because they are the most popular keys used in the music generation. Also, as we implement one-hot encoding, the matrix dimension gets very large; due to the hardware limitation, the system runs out of memory when we try to utilize the streams of all keys, and the processing becomes hard.

Once the information regarding the notes and chords are obtained, we need to transform it into the form of a matrix that is compatible with the neural network for further processing, because the network cannot process the input data in the form of notes and chords. For transforming it into the matrix, we implement the following steps.

1. We identify the unique notes and chords present in the training music streams.
2. Assign the different integer values for every unique notes and chords using the dictionary method.
3. Assign these unique integer values to the notes and chords in the training music stream. Now the input data is represented by integer values rather than notes and chords.

All these sequences form a list. This is the training data set with which we will train our model.

We illustrate the above-mentioned data pre-processing steps with an example for the reader to understand. We have considered a random MIDI file to explain this. Let us say that our training data set, after extracting all notes and chords, consists of the following sequence in the form of lists:

Original notes_chords = $[['B{-}3.B{-}4', 'G2.E4.G4.C5', 'B{-}2','C3.E3.G3.C6', 'E3.G\#4', 'F\#2.F\#4.B{-}4', 'B2.G4', 'E3.G4', 'G2.G4', 'C3.G\#4', 'B2.B4', 'E3.G4', 'G2.G4', 'G2.B4', 'G3.C4.E4.G4.G5', 'C3.G\#4', 'C3.E3.E5', 'A2.F4.A4.C5', 'G2.G4','C3.C5', 'G3.C4.E4.G4.G5', 'E3.G4', 'G2.E4.G4.C5', 'B - 2.C\#3.B - 4','C3.G\#4', 'E3.G4', 'E3.G\#4', 'E2.E4.G4.C5', 'C3.E3.E5','E3.G4', 'E3.G4', 'E3.G4']]$

Next we will identify the number of unique notes and chords present in the training data set. The unique set of notes and chords present in the training data set is shown below

Unique notes_Chords=['A2.F4.A4.C5' , 'B-2' , 'B-2.C#3.B-4' ,'B-3.B-4' ,'B2.B4' ,'B2.G4' ,'C3.C5' , 'C3.E3.E5' , 'C3.E3.G3.C6' , 'C3.G#4' ,'E2.E4.G4.C5' , 'E3.G#4' 'E3.G4', 'F#2.F#4.B-4' , 'G2.B4' , 'G2.E4.G4.C5' , 'G2.G4' ,'G3.C4.E4.G4.G5']

The dictionary is created in which unique notes and chords are enumerated by which the notes /chords are now identified by unique integers

{0: 'A2.F4.A4.C5', 1: 'B-2', 2: 'B-2.C#3.B-4', 3: 'B-3.B-4', 4: 'B2.B4', 5: 'B2.G4', 6: 'C3.C5', 7: 'C3.E3.E5', 8: 'C3.E3.G3.C6', 9: 'C3.G#4', 10: 'E2.E4.G4.C5', 11: 'E3.G#4', 12: 'E3.G4', 13: 'F#2.F#4.B-4', 14: 'G2.B4', 15: 'G2.E4.G4.C5', 16: 'G2.G4', 17: 'G3.C4.E4.G4.G5'}

We then assign these unique integer values to the respective notes and chords to the training data set(Original notes chords). After assigning these integer values following sequence list is obtained.
Original notes_chords = [3, 15, 1, 8, 11, 13, 5, 12, 16, 9, 4, 12, 16, 14, 17, 9, 7, 0, 16, 6, 17, 12, 15, 2, 9, 12, 11, 10, 7, 12, 12, 12]

For the further processing of data, we divide the training data lists into multiple sub-lists, with the length of each sub-list equal to the value of Sequence length. For this example, let us consider the sequence length equal to 28, and it results in the five sub-lists as shown below, with the length of each sub-list equal to 28.
[[3, 15, 1, 8, 11, 13, 5, 12, 16, 9, 4, 12, 16, 14, 17, 9, 7, 0, 16, 6, 17, 12, 15, 2, 9, 12, 11, 10],
[15, 1, 8, 11, 13, 5, 12, 16, 9, 4, 12, 16, 14, 17, 9, 7, 0, 16, 6, 17, 12, 15, 2, 9, 12, 11, 10, 7],
[1, 8, 11, 13, 5, 12, 16, 9, 4, 12, 16, 14, 17, 9, 7, 0, 16, 6, 17, 12, 15, 2, 9, 12, 11, 10, 7, 12],

$[8, 11, 13, 5, 12, 16, 9, 4, 12, 16, 14, 17, 9, 7, 0, 16, 6, 17, 12, 15, 2, 9, 12, 11, 10, 7, 12, 12]$,
$[11, 13, 5, 12, 16, 9, 4, 12, 16, 14, 17, 9, 7, 0, 16, 6, 17, 12, 15, 2, 9, 12, 11, 10, 7, 12, 12, 12]]$

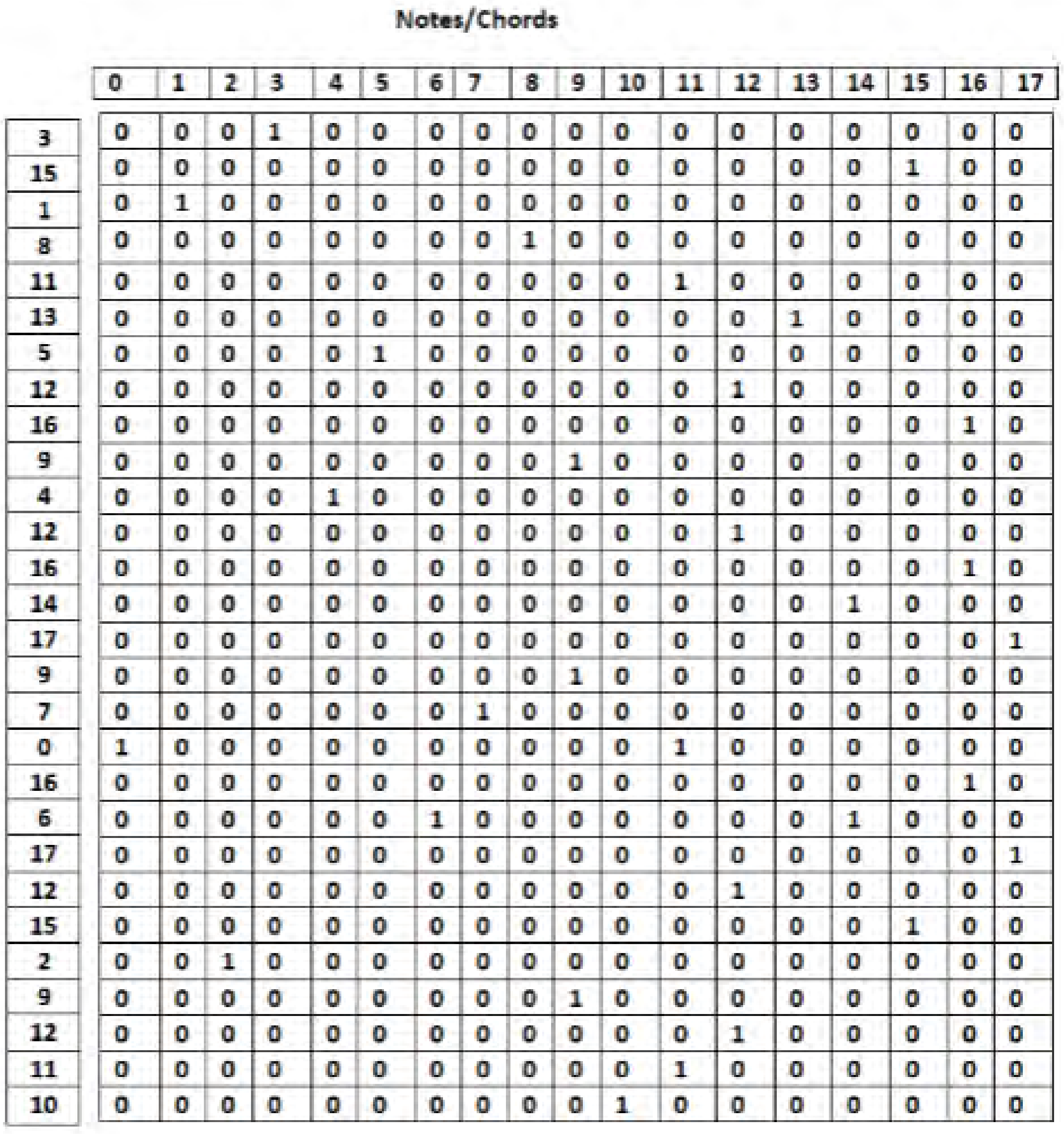
Now each of these sub-lists is converted into a one-hot encoded vector matrix. In which the column of the matrix represents the unique notes and chords, and each row represents the individual element in the sublist. The presence of the particular note/chord is represented by '1,' and the absence of note/chord is represented by '0' [BHP17]. In figure 3.1 shows the one-hot encoded matrix for one sequence sub-list. In a similar manner, we convert other sequence sub-list into one hot matrix. In our example, there will be five such one-hot encoded matrices. The One-hot encoded matrix for the first sequence has been shown in figure 3.1 for the reader to get a good understanding of this. From this, we can see the dimension of data involved; this increases with the increase in the number of input music streams. Next, we flattened this into the single dimension vector of length (no of unique chords x sequence length). For our above example, it will be (18 x 28).

3.2.2 Model Explanation

The architecture of the VAE model we implemented in this thesis is as shown in figure in 3.2. The details about the encoder and decoder sections are explained below.

Encoder

The architecture of the encoder consists of the input layer with the dimension equal to (no of unique notes and chords X sequence length). From the example mentioned in the previous section, it will be equal to (18x28). The hidden layer follows the input layer with its dimension lesser than the input layer. We have considered the value of 32, followed by the mean and log variance layer whose dimension is equal to the latent dimension, and we have considered the latent dimension of 2. All these layers are densely connected feedforward neural networks. We pass the flattened one-hot encoded vector through this densely connected neural network, and the output of this network produces the mean and the variance. It extracts the essential features from the input and compresses them into a lower dimension vector. As explained in section 2.3.3, it generates the normal distribution with these mean and variance values.



Notes/Chords

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Figure 3.1: One hot Encoded Matrix

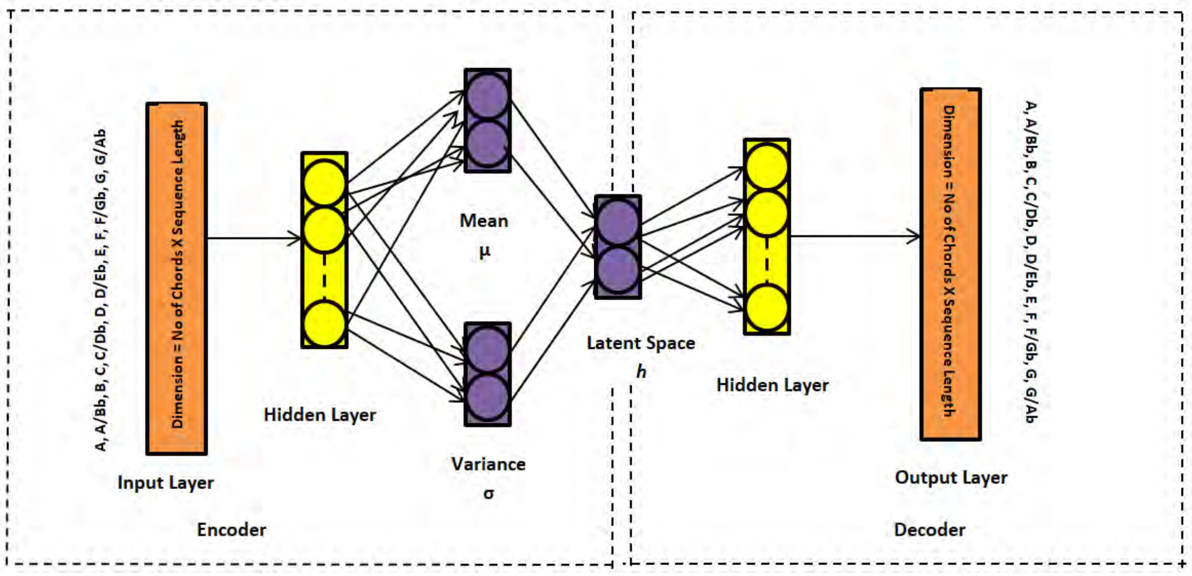

A, A/Bb, B, C, C/Db, D, D/Eb, E, F, F/Gb, G, G/Ab
Dimension = No of Chords X Sequence Length
Input Layer
Hidden Layer
Mean
μ
Variance
σ
Encoder
Latent Space
h
Hidden Layer
Dimension = No of Chords X Sequence Length
A, A/Bb, B, C, C/Db, D, D/Eb, E, F, F/Gb, G, G/Ab
Output Layer
Decoder

Figure 3.2: Architecture of Variational Autoencoder

Decoder

Sampling from the distribution obtained at the output of the encoder generates the latent vector h. The data in the latent vector h is sufficient to regenerate the input. We pass this through the densely connected decoder network, which produces the probabilistic output. The exact match of the input cannot be obtained at the output as the decoder receives the encoded information from the latent space. The reconstruction loss measures the difference between the original notes and chords and the reconstructed notes and chords. As we have used a one-hot encoded vector at the input of the network, we use binary cross-entropy to measure the reconstruction loss at the output. The sigmoid activation layer at the output gives the probability of the values between 0 and 1 for each of the input data [BHP17]. Our model is symmetric VAE, meaning that the dimension of the encoder and decoder hidden layers are identical. The activation function in the hidden layer of the encoder is ReLu and decoder is tanh. The output layer is of the same dimension as the input, and the output is the probability distribution of a single note/chord. For gradient calculation, we use Adagrad optimizer. We have used Keras to develop and compile our model. The derivative is calculated automatically via Tensorflow using the Adagrad optimizer. The Graphical representation of the model implemented is as shown in figure 3.3 and 3.4.

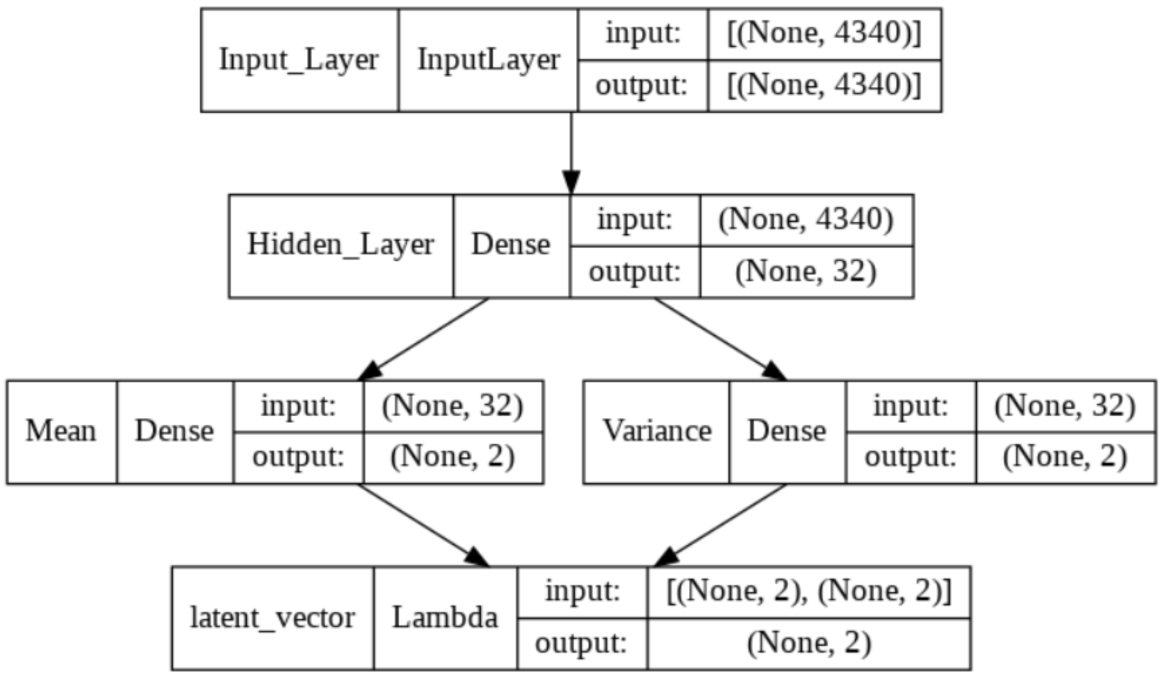


Figure 3.3: Encoder Network

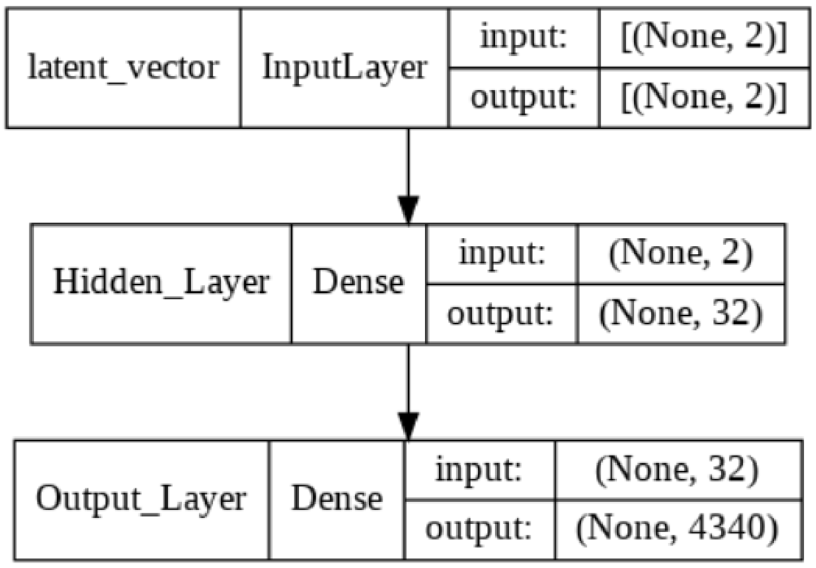


Figure 3.4: Decoder Network

Generating Music

After the training of the model as explained above, the model can generate the music. For the generation of the music, we consider only the decoder part. Provide the decoder with the random sample vector from the latent space, whose dimension is equal to the (1 X latent dimension); for our example considered in the section 3.2.1, we consider the following sample

[[0.16084939 0.00839284]]

The decoder undergoes the process as in training, and it generates the output. The output generated has the probability value vectors. We adjust the output dimension to that of the sequence length to obtain the notes and chords to generate the music. The output generated for our sample looks like the following.

```
[0.50036407  0.5013775  0.50090146:........... 0.5011115  0.50068814  0.499421]
[0.500394  0.50003284  0.49941427 ............. 0.4997231  0.4988158  0.49919748]
...
[0.50109214  0.49878985  0.5014525 .............. 0.500891  0.4995132   0.501209]
[0.49843007  0.5011684  0.49919248 ...............50014573  0.49926212  0.5003791]
```

The number of rows is equal to the number of unique notes and chords, and number of columns is equal to the sequence length, as per our example (18 X 28). We choose the maximum valued index from each column from the obtained vector output as per the readings from [BHP17]. Thus obtained integer values form the list with the size equal to that of the sequence length. We obtain the following.

```
[[15  2  3  5  4  9  10  6  2  17  8  4  0  6  3  9  1  7  12  9  3  9  1  6  9  3  16  17]]
```

Then we map back these integer values to their respective notes and chords using the dictionary implemented at the beginning. We obtain the following lists of notes and chords.

['G2.E4.G4.C5' , 'B-2.C#3.B-4' ,'B-3.B-4' , 'B2.G4' , 'B2.B4' , 'C3.G#4' , 'E2.E4.G4.C5' , 'C3.C5' , 'B-2.C#3.B-4' , 'G3.C4.E4.G4.G5' , 'C3.E3.G3.C6' , 'B2.B4' , 'A2.F4.A4.C5' , 'C3.C5' , 'B-3.B-4' , 'C3.G#4' , 'B-2' , 'C3.E3.E5' , 'E3.G4' , 'C3.G#4' , 'B-3.B-4' , 'C3.G#4' , 'B-2' , 'C3.C5' , 'C3.G#4' , 'B-3.B-4' , 'G2.G4' , 'G3.C4.E4.G4.G5']

Append these notes and chords using the music21 library to form the music stream and then convert them into a MIDI file. We then play the generated MIDI file on any music player to listen to the generated music.

4 Results and Evaluation

4.1 Experiment and Results

This section includes the experiments carried out in the thesis and also presents the results. We are interested in evaluating the performance of the model on varying the hyperparameter values. We are supposed to select the hyperparameter values before running the neural network. The selection of the hyperparameter value plays an important role in the performance of the neural network. There is no empirical evidence for selecting the appropriate hyperparameter values. Selecting the proper hyperparameter value is an expensive process. Choosing the hyperparameter within the range of values involves a heuristic search. As the search space increases, it becomes more expensive. The expensive here means the run time of the model. The model has to run from the beginning every time its hyperparameter values are changed. With the increase in input data size, the run time increases accordingly. Hence we have included only a single music stream from Vgmusic.com to analyze the impact of the hyperparameters on our model. It will be easy for us to monitor the changes in the output with changing hyperparameter values. We conduct a series of experiments to observe the behavior of the model. One hyperparameter value is changed, keeping the other hyperparameters constant. In each run, the model's output, the music generated, histogram of notes/chords of the generated music is analyzed. The selection of hyperparameter values is random, and it is selected based on trial and error. We will consider the song sequences with the following notes/chords for this experimentation, as shown in figure 4.1. Including fewer notes/chords will be easier to investigate the impact of each hyperparameter.

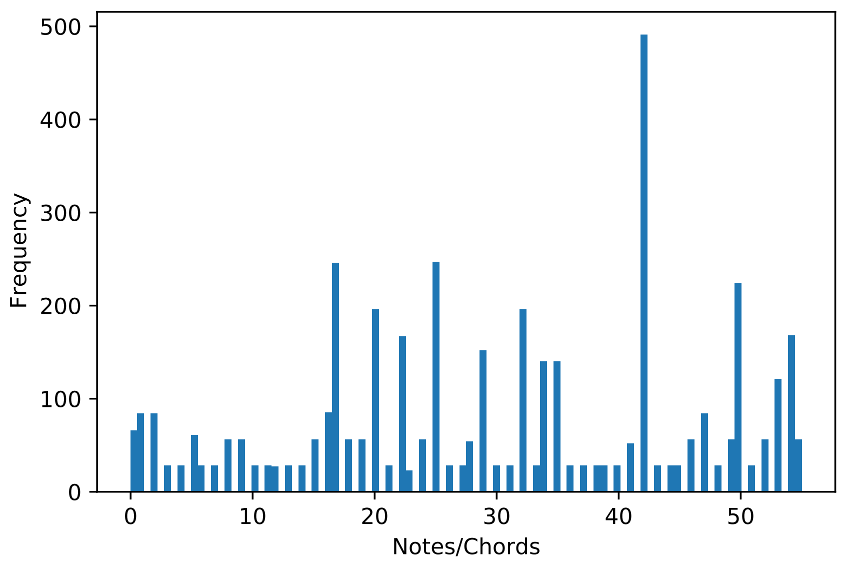


Figure 4.1: Frequency of Notes/Chords Present in the Training Data

4.1.1 Results

The important Hyperparameters considered are:

1. Learning Rate
2. Epochs
3. Latent Dimension
4. Number of Neurons in the hidden layer
5. Sequence Length

1. Learning Rate
   The model was trained on a sample music stream with the different learning rates of 0.1, 0.01, 0.001, 0.0001 and analyzed the influence of this hyperparameter on the model output. The generated notes/chord sequence is compared with the input notes/chords sequence in each run to see the percentage similarity of notes/chords present in both the sample music and the generated music. As we can see from figure 4.2, from the graph of validation loss, we can analyze that when training the model with a learning rate of 0.1 and 0.01, the loss after 10 epochs is low. The

highest loss after the 10 epochs is with a learning rate of 0.0001. The percentage similarity in the generated notes/chords was lowest when the learning rate was 0.1, and it was highest with the learning rate of 0.0001. The more the percentage similarity better the model is, as it avoids the repetition of notes/chords and instead it generates different notes/chords, which will be generating good music. For any learning rate, the generated sequence of notes/chords was not 100 percent similar to the sample music. The highest percentage similarity was not more than 50 percentage, for the learning rate of 0.0001. From the graph of validation loss, we can see that the model converges quickly for the higher learning rates, and the notes/chords generated for the higher learning rates concentrate on the most frequently occurred notes/chords at the input. The higher learning rate does not learn the data sequences in depth. We chose the batch size of 32, the latent dimension of 30, the number of neurons in hidden layers is 10, and the number of epochs of 10 for this experiment.

2. Epochs Variation
   Experiments were conducted on various epochs to see the effect on the model output. We trained the model for epochs 10,50, and 100. For this experiment, we chose the learning rate of 0.001, batch size of 32, latent dimension of 30, and the number of neurons in hidden layers as 10. Figure 4.3 shows the generated sequence of notes/chords for each epoch. The generated notes/chords spread across all the possible ranges of input for lower values of epochs, whereas with the increase in epochs, the model generates the notes/chords, one with the highest frequency at the input.

3. Latent Dimension Variation
   The dimension of the latent space is varied, and its output is analyzed. We trained the model on the latent dimensions of 10, 128, and 512. The generated output sequence of chords for each is shown below in figure 4.4. From the output, we can see that with the increase in the value of the Latent Dimension, the generated chords sequence has the notes/chords with the one which has a high frequency at the input. For this experiment, we chose the learning rate of 0.001, batch size of 32, epochs of 50, and the number of neurons in hidden layers as 10.

4. Number of Neurons in the Hidden Layer
   The number of neurons in the hidden layer can influence the output of the neural network. The neural networks overestimate the complexity of the problem with

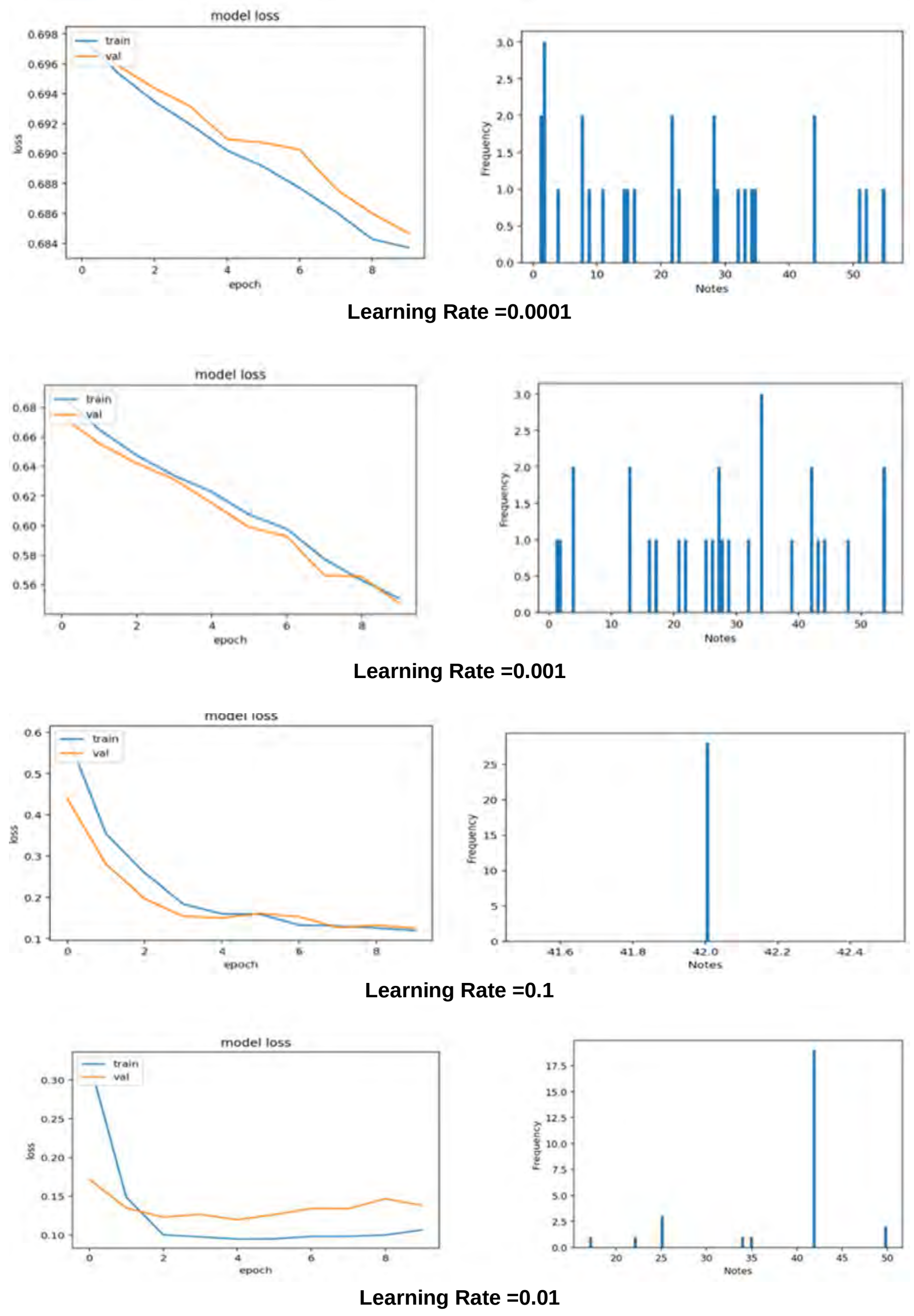
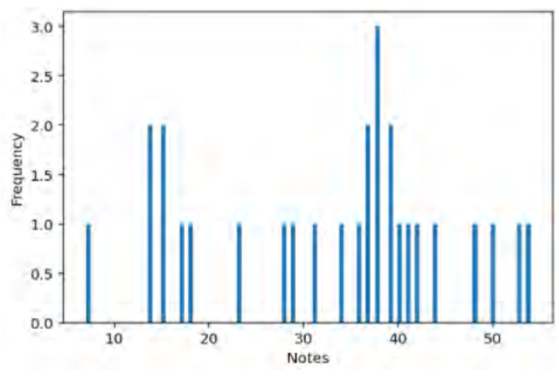
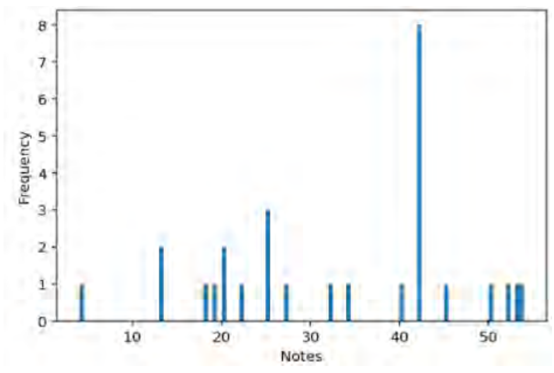


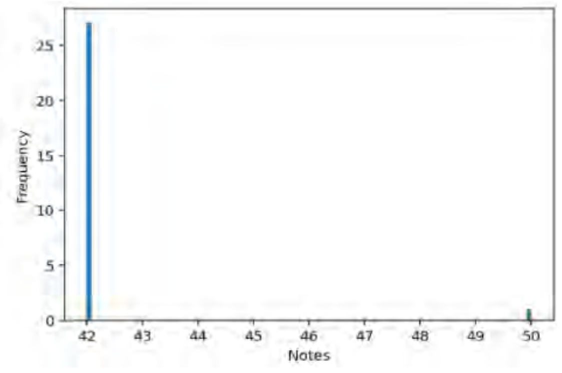
Figure 4.2: Graph of Loss and Histogram for different Learning Rates



EPOCHS 10

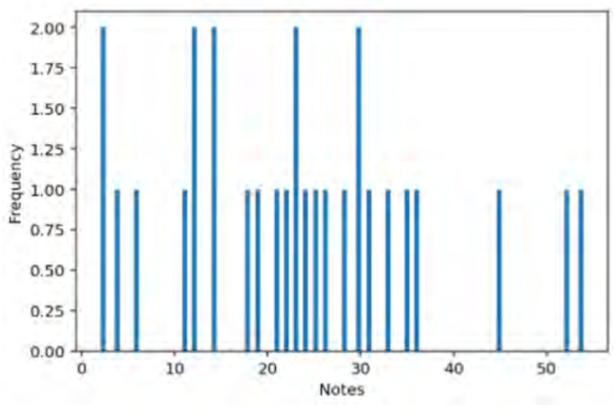


EPOCHS 50

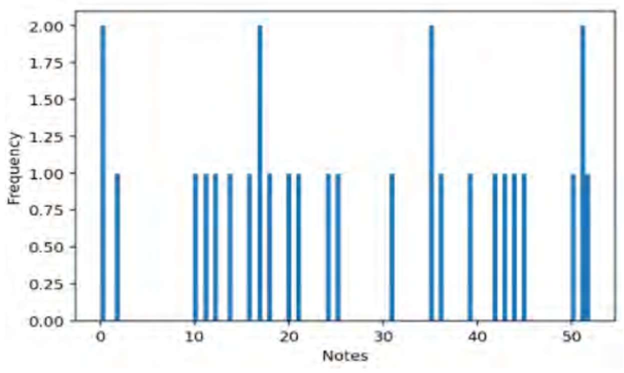


EPOCHS 100

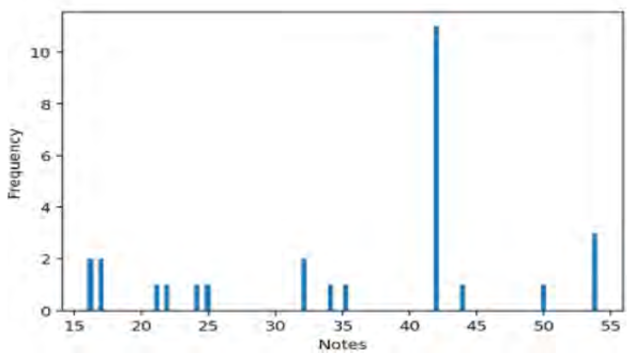
Figure 4.3: Graph of Histogram for different Epochs



Latent Dimension 10



Latent Dimension 128



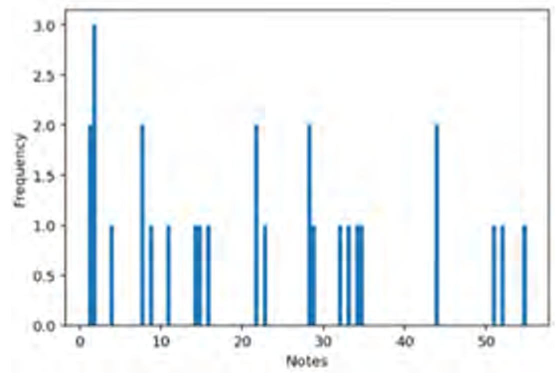
Latent Dimension 512

Figure 4.4: Graph of Histogram for different Latent Dimension

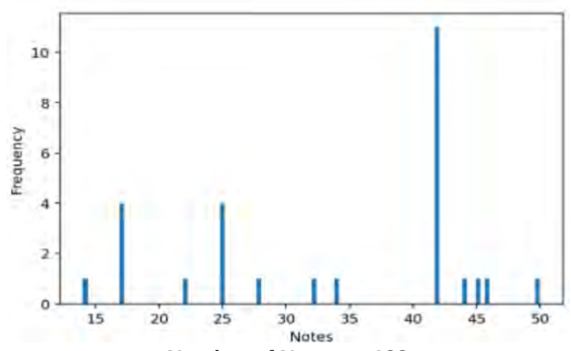
more hidden neurons, and this causes the problem of overfitting [KL08]. It reduces the generalization capability. We ran our model with the values of the hidden neurons of 10,128, and 512 and examined the notes/chords sequence generation. For this experiment, we chose the learning rate of 0.001, batch size of 32, latent dimension of 30, and epochs of 50. From figure shown in 4.5, we can observe that the generated chords spread across all possible values within the range for a lower number of neurons; with the increase in the number of neurons, it follows the most repeated notes/chords at the input.

5. Sequence Length
   We considered different values of sequence length to train the model. We considered the Sequence length of 28,64 and 128 for training the model. By the generated music from the model, we analyzed that varying sequence length could vary the duration of the generated music. The model trained with a shorter sequence length has generated music of shorter duration when compared with the model trained with the larger sequence length. We chose the batch size of 32, the latent dimension of 30, the number of neurons in hidden layers of 10, and the number of epochs of 50 for this experiment.

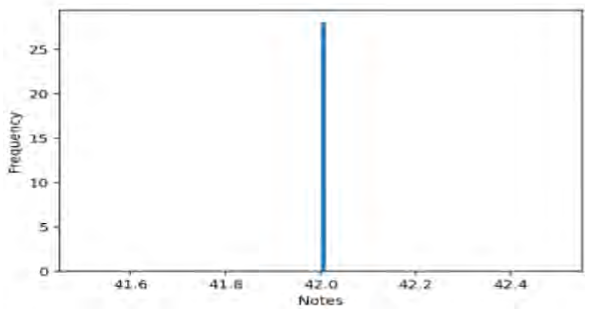
Upon analyzing the influence of the various hyperparameter value on the output of our model, we can choose the value according to the requirement of the generated music. For example, if we want to have a music of repetition, we can choose the larger value of hyperparameter. If we want to have a music of variation, we can choose the hyperparameter of low value. In all the above experiments, the percentage similarity of the generated chords sequence with the input chords sequence was not more than 50 percent in any case. By this, we can be confident that the output sequence generated was never the same as the input sequence. We cannot conclude from the above experiments that choosing the hyperparameter values mentioned above provides the same result on all data sets. It can vary depending on the number of training data samples. This experiment's main intention was to analyze whether the hyperparameter values impact the music generation.



Number of Neurons 10



Number of Neurons 128



Number of Neurons 512

Figure 4.5: Graph of Histogram for different Number of Neurons in Hidden layer

4.2 Evaluation

In this section, we present the result of a survey conducted on the generated music. We conducted this qualitative assessment as there cannot be a quantitative assessment of the generated music. This is because the music generated by the network does not follow the music theory concepts such as duration, pitch, and velocity. The thesis mainly concentrates on the generation of notes and chords, for which there is no quantitative assessment. The intention of this survey is to know how the music generated by the VAE model sounds to the participants and obtain their opinion about the generated music. To create a survey, we use Google forms to administer it online. We trained the model with different hyperparameter values and the different sets of MIDI files. Few of the MIDI file sets consist of music from one particular category(Piano, Pokemon, NES); the other set consists of MIDI files from all these three categories combined. Upon running all these sets of combinations on our model, there were several generated music streams. Among those, we have considered a few exciting music streams for this survey.

During the survey, the participants first listen to one original video game music from the training data set and one generated music to make sure that the participants get familiar with the type of music they will listen to in the survey provided. The length of music clips in the survey varies between 15-20 seconds. The participants were then allowed to listen to the ten music streams, which consisted of three original video game music from the training data and seven generated music obtained from the model with varying training data and the model's hyperparameter values. The participants were provided with three options to select from, and they were: Original Music, Generated Music, and Cannot Say. After listening to the given music clips, the participants selected one among the given option. There was also a provision for the participants to give the reason on which criteria they selected a particular music as the generated.

Among these ten music clips, music stream numbers 2,3,5,6,8,9 and 10 were the generated music clips. The generated music clips 2,3 used the Piano only MIDI files as training data (50 MIDI files). The music clips 5,6 used the Pokemon MIDI files as training data (50 MIDI files). The music clips 8,9 used the NES MIDI files as training data (50 MIDI files). The music clip 10 had the MIDI files from Piano only, NES, and Pokemon combined (50 MIDI files). The Bar graph shown in figure 4.6 shows the result of our survey. The result shows that our attempt to generate a video game like music was not completely successful, where we had the belief that all the music generated by the model

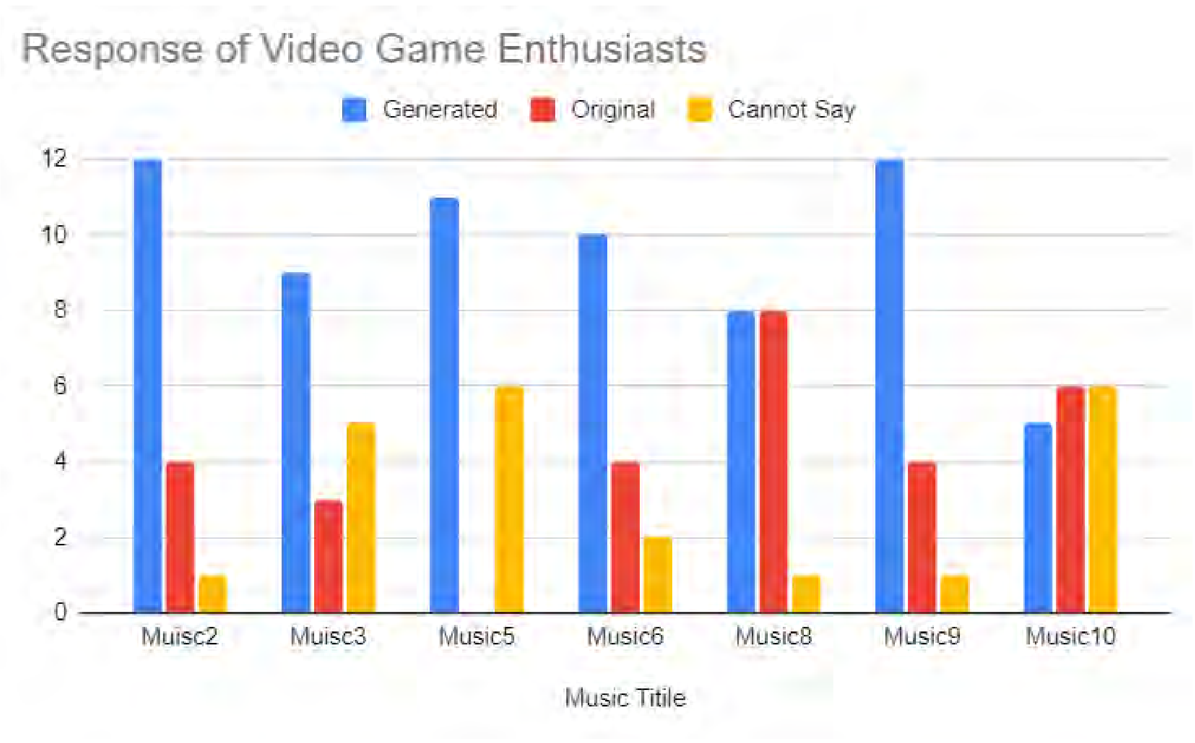


Figure 4.6: Survey Result

would sound like the original retro video game kind of music. Instead, our survey results suggest that to some extent, it has given satisfactory results, where the participants got deceived into believing that the music generated was the original music. Including more participants might give a better picture of the survey result. Among the reviews received from the participants, most felt that the not orderly placed sequences of the notes and chords as per music theory helped them guess the generated music. Few gave the opinion that the sample clip of the generated music helped them select the generated music clips; without that sample, it would have been difficult for them to select it.

5 Discussion

This section discusses how we could find the answer to the research questions described in chapter 1.

1. Can the Deep Generative Network like Variational Autoencoder generate the fake music by learning from the notes and chords of the training music data set?

The architecture we proposed was able to generate the fake music by only considering the notes and chords of the input training data. We converted the MIDI file into the format suitable for processing by the generative model VAE. We converted the model's output back to the MIDI stream, by which we generated the fake music.

2. Does the generated fake music sound close to the retro video game music?

By the survey we conducted with the music enthusiasts, we can say that even though the system was not completely successful in generating the fake retro video game kind of music, it has given the satisfactory result, in which people not much exposed to the video game music were deceived in believing it to be the original video game music , this is evident from the survey we conducted on the few of the generated music.

3.How does the value of the model hyperparameters influence the output of the generated music?

From the experiments we conducted using the music sample, we have seen that hyperparameters like learning rate, epochs, latent dimension, sequence length, and number of neurons in the hidden layer has an impact on the output generated, which we could analyze with the histogram of the generated notes/chords. The higher value of these hyperparameters produced the notes/chords sequence in which the major of the notes/chords are those which are the most frequent at the input training data. With the results from the experiments conducted we can say that, the hyper parameter values have influence in the generation of the music

6 Conclusion

6.1 Summary

We have implemented a Variational Autoencoder with the densely connected feedforward neural network that generates a fake video game music. We used software tools to extract the data related to notes,chords and keys from the MIDI file and encoded this extracted data into the neural network processing; We then converted the output data obtained from the model back to MIDI format; by this, we successfully developed a deep generative model for music generation. The proposed VAE architecture was able to generate music somewhat similar to retro video game music. The generated music was very much different from the input; this we measured from the percentage similarity. The generated sequence with the higher value of percentage similarity sounded better. The model could generate better-sounding music by considering other aspects of music. We could also verify that the hyperparameter values impacted the generated music.

6.2 Limitation

The proposed architecture has some limitations; here, we consider only the notes, chords, keys and discards other important elements of music such as velocity and duration. Even considering these elements in the generation process could enhance the music. Our proposed method does not keep track of the previous notes/chords, considering this adds a better listening experience; we can implement this by using Rnn or LSTM into the Variational Autoencoder network. We have included only the retro video game music of single instruments, and due to hardware limitations, we have considered only a few training data. Finding the perfect combination of hyperparameters was not completely successful in generating the music.

6.3 Future work

In this research thesis, we addressed the task of generating music with the single instrument retro video game MIDI music files. Future work can consider the music with multiple instruments and include the duration into the data processing, as it controls the playtime of the particular notes/chords. The inclusion of LSTM networks into this model might provide even better results as it learns the pattern of previous chords. Including more training data provides an increased number of different chords and notes, which might enhance the quality of music generated.

Bibliography

[Asl] Nazia Aslam. *Introduction of Different types of Loss Functions in Machine learning and Deep learning — by Nazia Aslam — Analytics Vidhya — Medium.* https://medium.com/analytics-vidhya/introduction-of-different-types-of-loss-functions-in-machine-learning-and-deep-learning-66ef7804668b. (Accessed on 12/01/2021) (cit. on p. 30).

[Ass+96] MIDI Manufacturers Association et al. "The complete MIDI 1.0 detailed specification". In: *Los Angeles, CA, The MIDI Manufacturers Association* (1996) (cit. on pp. 7, 8).

[BHP17] Jean-Pierre Briot, Gaëtan Hadjeres, and François-David Pachet. "Deep learning techniques for music generation–a survey". In: *arXiv preprint arXiv:1709.01620* (2017) (cit. on pp. 31, 36, 38, 40).

[Broa] Jason Brownl. *Supervised and Unsupervised Machine Learning Algorithms.* https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/. (Accessed on 12/06/2021) (cit. on p. 11).

[Brob] Jason Brownlee. *Understand the Impact of Learning Rate on Neural Network Performance.* https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/. (Accessed on 12/01/2021) (cit. on p. 27).

[Col08] Karen Collins. *Game sound: an introduction to the history, theory, and practice of video game music and sound design.* Mit Press, 2008 (cit. on p. 1).

[Doe16] Carl Doersch. "Tutorial on variational autoencoders". In: *arXiv preprint arXiv:1606.05908* (2016) (cit. on p. 21).

[Duv08] David Chris Duvall. *Real-time MIDI performance evaluation for beginning piano students.* Clemson University, 2008 (cit. on pp. 7–9).

[FV14] Otto Fabius and Joost R Van Amersfoort. “Variational recurrent auto-encoders”. In: *arXiv preprint arXiv:1412.6581* (2014) (cit. on p. 31).

[FHD+96] Brendan J Frey, Geoffrey E Hinton, Peter Dayan, et al. “Does the wake-sleep algorithm produce good density estimators?” In: *Advances in neural information processing systems*. Citeseer. 1996, pp. 661–670 (cit. on p. 15).

[Fri13] Melanie Fritsch. “History of video game music”. In: *Music and game*. Springer, 2013, pp. 11–40 (cit. on p. 1).

[Goo16] Ian Goodfellow. “Nips 2016 tutorial: Generative adversarial networks”. In: *arXiv preprint arXiv:1701.00160* (2016) (cit. on pp. 15–17).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 14, 18).

[HPN17] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. “Deepbach: a steerable model for bach chorales generation”. In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1362–1371 (cit. on p. 31).

[Hec92] Robert Hecht-Nielsen. “Theory of the backpropagation neural network”. In: *Neural networks for perception*. Elsevier, 1992, pp. 65–93 (cit. on p. 27).

[HUW17] Jay A Hennig, Akash Umakantha, and Ryan C Williamson. “A classifying variational autoencoder with application to polyphonic music generation”. In: *arXiv preprint arXiv:1711.07050* (2017) (cit. on p. 32).

[HS+86] Geoffrey E Hinton, Terrence J Sejnowski, et al. “Learning and relearning in Boltzmann machines”. In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1.282-317 (1986), p. 2 (cit. on p. 17).

[HP99] Aapo Hyvärinen and Petteri Pajunen. “Nonlinear independent component analysis: Existence and uniqueness results”. In: *Neural networks* 12.3 (1999), pp. 429–439 (cit. on p. 16).

[Jeb12] Tony Jebara. *Machine learning: discriminative and generative*. Vol. 755. Springer Science & Business Media, 2012 (cit. on p. 14).

[Jor18] Jeremy Jordan. “Introduction to autoencoders”. In: *Jeremy Jordan, Mar* (2018) (cit. on p. 18).

[KL08] Jinchuan Ke and Xinzhe Liu. "Empirical analysis of optimal hidden neurons in neural network modeling for stock prediction". In: *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application.* Vol. 2. IEEE. 2008, pp. 828–832 (cit. on p. 48).

[Key] Keyboard. *Piano Notes and Keys – How to Label Piano Keys.* `http://www.piano-keyboard-guide.com/piano-notes.html`. (Accessed on 12/07/2021) (cit. on p. 5).

[KW19] Diederik P Kingma and Max Welling. "An introduction to variational autoencoders". In: *arXiv preprint arXiv:1906.02691* (2019) (cit. on p. 20).

[LBH+15] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, et al. "Deep learning. nature 521 (7553), 436-444". In: *Google Scholar Google Scholar Cross Ref Cross Ref* (2015) (cit. on p. 14).

[MP43] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cit. on p. 12).

[MID] MIDI. *Standard MIDI file format, updated.* `http://www.music.mcgill.ca/~ich/classes/mumt306/StandardMIDIfileformat.html`. (Accessed on 11/30/2021) (cit. on p. 7).

[Moz94] Michael C Mozer. "Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing". In: *Connection Science* 6.2-3 (1994), pp. 247–280 (cit. on p. 31).

[Mül15] Meinard Müller. *Fundamentals of music processing: Audio, analysis, algorithms, applications.* Springer, 2015 (cit. on p. 6).

[Mus] Music21. *music21 Documentation — music21 Documentation.* `https://web.mit.edu/music21/doc/index.html`. (Accessed on 12/01/2021) (cit. on p. 30).

[Neu] Neuralnetwork. *What the Hell is Perceptron?. The Fundamentals of Neural Networks — by SAGAR SHARMA — Towards Data Science.* `https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53`. (Accessed on 12/10/2021) (cit. on p. 12).

[OO17] Hélio Magalhães de Oliveira and RC de Oliveira. "Understanding MIDI: A Painless Tutorial on Midi Format". In: *arXiv preprint arXiv:1705.05322* (2017) (cit. on p. 10).

[Ran+16] Rajesh Ranganath et al. “Operator variational inference”. In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 496–504 (cit. on p. 23).

[Rep] Reparameterization. *“Reparameterization” trick in Variational Autoencoders — by Sayak Paul — Towards Data Science.* `https://towardsdatascience.com/reparameterization-trick-126062cfd3c3`. (Accessed on 12/14/2021) (cit. on p. 26).

[Rob+18] Adam Roberts et al. “A hierarchical latent vector model for learning long-term structure in music”. In: *International conference on machine learning*. PMLR. 2018, pp. 4364–4373 (cit. on p. 32).

[Ros58] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), p. 386 (cit. on p. 12).

[RN02] Stuart Russell and Peter Norvig. “Artificial intelligence: a modern approach”. In: (2002) (cit. on p. 10).

[Sch13] Catherine Schmidt-Jones. “Understanding basic music theory”. In: (2013) (cit. on p. 4).

[SS17] Sagar Sharma and Simone Sharma. “Activation functions in neural networks”. In: *Towards Data Science* 6.12 (2017), pp. 310–316 (cit. on pp. 28, 29).

[Ter] DeepLearning Terms. *Epoch vs Batch Size vs Iterations — by SAGAR SHARMA — Towards Data Science.* `https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9`. (Accessed on 12/13/2021) (cit. on p. 27).

[TAY14] Eric Thibodeau-Laufer, Guillaume Alain, and Jason Yosinski. “Deep generative stochastic networks trainable by backprop”. In: *International Conference on Machine Learning (ICML).* 2014 (cit. on p. 18).

[VKK16] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. “Pixel recurrent neural networks”. In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1747–1756 (cit. on p. 16).

[Var] Variational. *Intuitively Understanding Variational Autoencoders — by Irhum Shafkat — Towards Data Science.* `https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf`. (Accessed on 12/01/2021) (cit. on p. 22).

[YT20] Ivan P Yamshchikov and Alexey Tikhonov. “Music generation with variational recurrent autoencoder supported by history”. In: *SN Applied Sciences* 2.12 (2020), pp. 1–7 (cit. on p. 32).

[Yan17] Xitong Yang. “Understanding the variational lower bound”. In: *variational lower bound, ELBO, hard attention* (2017), pp. 1–4 (cit. on p. 24).

[YU20] Chika Yinka-Banjo and Ogban-Asuquo Ugot. “A review of generative adversarial networks and its application in cybersecurity”. In: *Artificial Intelligence Review* 53.3 (2020), pp. 1721–1736 (cit. on pp. 15, 17, 18).

[Zel94] Andreas Zell. *Simulation neuronaler netze*. Vol. 1. 5.3. Addison-Wesley Bonn, 1994 (cit. on p. 13).

List of Abbrevations

VAE	Varitional Autoencoder
ANN	Artificial Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
FNN	Feedforward Neural Network
VRAE	Variational Recurrent Autoencoder
AE	Autoencoder

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie, dass ich die Masterarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, 14. December 2021

Abhinandan Shetty Rathnakar