UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

Polyphonic music generation using neural networks

Author:

Federico Loyola

Supervisors:

Prof. Àngela Nebot

Prof. Enrique Romero

A thesis submitted for the degree of

Master in Artificial Intelligence

June 2021

Abstract

In this project, the application of generative models for polyphonic music generation is investigated. Polyphonic music generation falls into the field of algorithmic composition, which is a field that aims to develop models to automate, partially or completely, the composition of musical pieces. This process has many challenges both in terms of how to achieve the generation of musical pieces that are enjoyable and also how to perform a robust evaluation of the model to guide improvements.

An extensive survey of the development of the field and the state-of-the-art is carried out. From this, two distinct generative models were chosen to apply to the problem of polyphonic music generation. The models chosen were the Restricted Boltzmann Machine and the Generative Adversarial Network. In particular, for the GAN, two architectures were used, the Deep Convolutional GAN and the Wasserstein GAN with gradient penalty. To train these models, a dataset containing over 9000 samples of classical musical pieces was used. Using a piano-roll representation of the musical pieces, these were converted into binary 2D arrays in which the vertical dimensions related to the pitch while the horizontal dimension represented the time, and note events were represented by active units. The first 16 seconds of each piece was extracted and used for training the model after applying data cleansing and preprocessing. Using implementations of these models, samples of musical pieces were generated. Based on listening tests performed by participants, the Deep Convolutional GAN achieved the best scores, with its compositions being ranked on average 4.80 on a scale from 1-5 of how enjoyable the pieces were.

To perform a more objective evaluation, different musical features that describe rhythmic and melodic characteristics were extracted from the generated pieces and compared against the training dataset. These features included the implementation of the Krumhansl-Schmuckler algorithm for musical key detection and the average information rate used as an estimator of long-term musical structure. Within each set of the generated musical samples, the pairwise cross-validation using the Euclidean distance between each feature was performed. This was also performed between each set of generated samples and the features extracted from the training data, resulting in two sets of distances, the intra-set and inter-set distances. Using kernel density estimation, the probability density functions of these are obtained. Finally, the Kullback-Liebler divergence between the intra-set and inter-set distance of each feature for each generative model was calculated. The lower divergence indicates that the distributions are more similar. On average, the Restricted Boltzmann Machine obtained the lowest Kullback-Liebler divergences, indicating a superior capability in modelling the data's distribution in terms of the extracted features.

Contents

List of Figures

Acknowledgements

I would like to thank my co-supervisors Prof. Àngela Nebot and Prof. Enrique Romero for providing guidance and feedback throughout this project. I would also like to thank my friends and family for providing support and always being there to listen to music made by a computer when I needed them to.

Introduction

1.1 Introduction

Generative artificial intelligence models have been in existence and development for many decades. In contrast with discriminative models, which focus on modelling a decision boundary between different classes of data with the objective of classifying unseen examples, generative models aim to estimate the entire probability distribution of the training data with the final goal of replicating similar samples. While discriminative and generative models fundamentally carry out similar tasks in terms of estimating conditional probabilities, generative models take much longer to train due to the fact that they involve learning a substantially higher number of correlations within the data. Due to their increased complexity, and initially limited applications, these models have received comparatively less attention with respect to AI discriminative models. [1] However, in recent years, in line with the general trend of increased availability of computational power and data, along with rapid growth in computer vision, natural language processing and speech processing, the interest regarding generative models has skyrocketed and is currently amongst the most popular topics in AI research based on analysis of paper submissions. A recent innovation in particular, the Generative Adversarial Network, has enabled the use of AI to generate images, audio and text with a high extent of realism over a wide range of domains. In general, generative models find contemporary uses as powerful feature extraction tools, in unsupervised clustering and classification, pattern recognition followed by generation and recommendation systems. The focus of this project is to investigate a specific domain of content generation – the generation of music. [2]

Algorithmic composition is the field that is concerned, in the general sense, with the use of algorithms and particularly artificial intelligence methods, to automate the process of composing music. Composing music traditionally involves a series of activities, such as defining the main melody and rhythm, harmonization, arrangement, writing counter-point melodies and engraving the composition into a formal notation. All of these activities can be automated by computers to a certain extent, but the degree of automation of the composition can vary substantially. For small degrees of automation, the focus is on a set of frameworks and tools to provide support and assistance for a specific set of monotonous and repetitive tasks, whose automation is trivial. This specific subset of algorithmic composition is known as computer-aided algorithmic composition and is an active area of research and development due to the significant avenues for commercialization when integrated in software packages to aid musicians in the composition process. Many software packages for music production available today make use of computer aided algorithmic composition. Algorithmic composition in particular and as referenced in this project is concerned with higher degrees of automation, in which the system carries out the bulk of the creative process, and therefore requires more complex systems relying on artificial intelligence methods. At this level of autonomy, algorithmic composition is also known as music generation. [3]

Algorithmic composition falls under the loose category of computational creativity, which is the computational synthesis or analysis of works of ‘art’ in a partially or fully automated manner, in text, image or audio format. The motivation for undertaking such tasks can be quite varied. Some attempts at algorithmic composition and other computational creativity tasks are motivated mainly by artistic goals. In this case, the evaluation of the resulting generated content is mainly qualitative, and the output is deemed to be an acceptable or good result based on subjective criteria. Additionally, the end

goal may be of a commercial nature, in which case the quality of the produced content will again be evaluated in a subjective framework. In general, when dealing with computational creativity, the field is disadvantaged by the lack of a subject matter that can be defined in entirely formal terms. Within an academic context, attempts and research on algorithmic composition have taken place since the earliest days of the field of artificial intelligence itself. The idea of simulating a creative process such as composing music, was regarded as an appealing way to showcase the potential of AI emulating human intelligence. Since then, many attempts and methods have been developed for algorithmic composition, with significant improvements in terms of the range of tasks and complexity of the generated music successfully carried out. Despite the enduring issue of the evaluation of the generated music from an objective point of view, there is a focus on the objective evaluation of the models used for the task of algorithmic composition. [4] While still a relatively fragmented topic of research without strong continuity between projects, the problem of generating music with artificial intelligence remains interesting, with much progress to be made.

1.2 Objectives

The main aim of this project is hence to develop a generative model for the task of algorithmic composition, in line with the current state-of-the-art of the field. This involves identifying the specific tasks of the composition process which are currently interesting, as well as the level of complexity of the music generated that can be achieved through generative models. Due to the differences in the models available, consideration is required regarding the representation of the musical pieces and the format in which it will be processed and subsequently generated. The specific aims of the project are summarized as follows:

- Development of an appropriate process for the representation and encoding of the musical content, and application of the relevant domain knowledge
- Applying an artificial intelligence model to the generation of musical content, including optimization and modification of the model architecture or features to maximize the model's performance
- Development of a suitable methodology for assessing the model's performance and to evaluate the quality of the generated content
- Analysis and discussion of the model's performance and identification of potential further work

To achieve these objectives, a wide range of systems were considered, both taking into account models used in the earlier days of the field, to gain an understanding of how the problem of algorithmic compositions has evolved with time, and the more recent generative models that have gained popularity and are now widely documented. Throughout the project, two distinct models were used. The first model is a Restricted Boltzmann Machine (RBM). Due to limited results achieved with this model, a second model was also used, with a greater degree of success, the Generative Adversarial Network (GAN) previously mentioned. However, both models will be included in this project's description, and the analysis and results for both models are provided. For that reason, the theoretical background on both the models, not just the GANs, is provided.

1.3 Background

The concept of composing music via the use of algorithms predates computers. While undoubtedly a creative process, composition makes extensive use of music theory. It does not have an axiomatic foundation in modern mathematics (though music theorists may sometimes use mathematics), but there are a set of rules and general principles that guide compositions which largely depend on the musical style. The principles of music theory have been used to describe algorithmic processes to compose music. A well-known example is "Mozart's dice game", a system that uses dice to randomly guide a set of instructions which would result in a simple composition. "Schoenberg's twelve-tone technique" describes a series of steps to use to compose music. [3] Hence, in principle, it is sufficient to describe a set of instructions to obtain a musical composition, despite the fact that the resulting compositions will be very limited. One of the earliest examples of automated composition using computers was a program created using a computer called DATATRON, to generate 'Tin Pan Alley' melodies in the style of early 20th century American popular music. The melody "Push Button Bertha" was created using this system and aired in July 1956. [5] The computer made use of random number generation for initializing the set of available notes, and the program selected successive notes at random, checking for their suitability against rules extracted from examples of Tin Pan Alley melodies. This, at the time, resulted in a seemingly viable simulation of music composition. A similar method had been developed independently by Hiller and Isaacson, who described a series of compositional experiments using the ILLIAC computer. Some of the most successful experiments were performed and collected in the Illiac Suite for string quartet. [6] The system used two main procedures for composition, the first being similar to the DATATRON in which random selection of notes constrained by rules was perform, and a second approach which made use of Markov chains. Robert Baker, a collaborator of Hiller's, devised the composing computer known as MUSICOMP, which facilitated the development of composing programs, and was used in 1962 to create the "Computer Cantata", which made use of several statistical and stochastic methods. [7]Further uses of the MUSICOMP computer were achieved by Herbert Brun, who created a number of works such as the Sonoriferous Loops in 1964 by applying statistical methods to rhythm, note events and pitch, and then manually intervened to add more elements to the composition. [8] Simultaneously to these interlinked research efforts in algorithmic composition, other institutions had been independently developing algorithmic composition algorithms. One of the most well-known composers to employ computers has been Iannis Xenakis, who introduced statistical methods of composing, exploiting Gaussian distributions in 1956 and Poisson's distributions in 1957. His methods inspired much of the work on algorithmic composition carried out in the 1960s, such as Koenig's work which began in 1964 with a program called PROJECT1, that incorporated stochastic algorithms but also applied new techniques such as serial composition (from musical theory) that achieved some successful results. [5]

Many of these earlier attempts, in retrospect, are perhaps more adequately considered computer-aided algorithmic compositions, in which the computer automates only some aspects of the composition process, and there is some significant degree of human intervention in the resulting compositions. They are also mainly based on rule-based constraints and stochastic and statistical processes. As the availability and power of machines increased, the field of algorithmic composition slowly expanded. Aside from the main research group involved with ILLIAC and MUSICOMP, at Urbana university, there was generally little continuity in research, and attempts at algorithmic compositions often involved independently applying similar techniques in an ad-hoc manner, which led to a slow growth

in the field. [9] Nonetheless, many attempts have been made since the early days, and a wide range of techniques have been applied to the computer composition problem, achieving varying degrees of success. The general techniques applied, at different times, are overall in line with the general trends in the field of artificial intelligence throughout the second half of the 20th Century. In the following section, an overview of the methods applied and some characteristic examples are provided. Before explaining the different methods used, it is useful to describe some of the main concepts of algorithmic composition, particularly regarding the possible composition tasks, and briefly state some of the main musical theory concepts that are often referred to in works relating to algorithmic composition.

2 Literature Review

2.1 Algorithmic Composition tasks

Algorithmic composition is a broad area and covers many different tasks. These relate to the level and complexity at which the music is generated, and the proportion of human to computer input in the generation of the pieces. The various systems already developed to generate music tackle a broad range of tasks, and the specific task to carry out will inform the choice of generative model and data representation. In this section, the main tasks are outlined.

2.1.1 Composition from scratch

This is the main focus of algorithmic composition as it involves no human input in terms of the creative output, and the system composes all the different aspects and parts of the musical piece. The tasks can be divided in turn based on the complexity of the musical pieces used as training data:

- **Monophonic melody -** a monophonic melody is a series of notes with the same or different pitch, usually organized by specific pitch variation rules and rhythm relations. They are the simplest form of a melody in which a maximum of one note is played in each timestep, meaning that a generative model only needs to predict the probability of a single note to be played at each time step.
- **Polyphony -** polyphonic music is composed of two or more independent melodies, which exhibit complex patterns along both the temporal axis and harmonic intervals between simultaneous notes. This means that generation of such music is significantly more complex, but also more interesting in terms of the potential results obtained. At each timestep, the model needs to predict the probability of any combinations of notes played- The resulting compositions however, if successful, are complete musical pieces and do not necessarily require further human input. While in music theory there is a distinction between polyphonic and homophonic music, which is when a single melody is accompanied by one or more harmonic parts, in the context of a generative model this is considered to be the same task, so in this work homophony and polyphony are used interchangeably. Examples of the three types of melody are shown in musical notation in Figure 2.1, Figure 2.2, and Figure 2.3.
- **Multi-track music –** multi-track music is an extension of polyphonic music generation, and has the added complexity of generating different instruments simultaneously, each with their own temporal dynamics- This type of generation is the ideal type for modern music genres which make use of many different instruments and have a rhythm section which is integral to the genre.

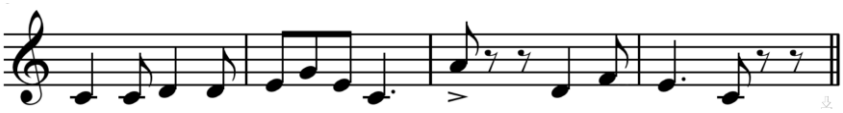

Figure 2.1 Monophonic melody

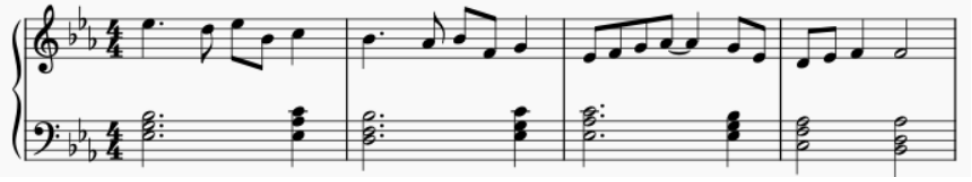

Figure 2.2 Homophonic melody

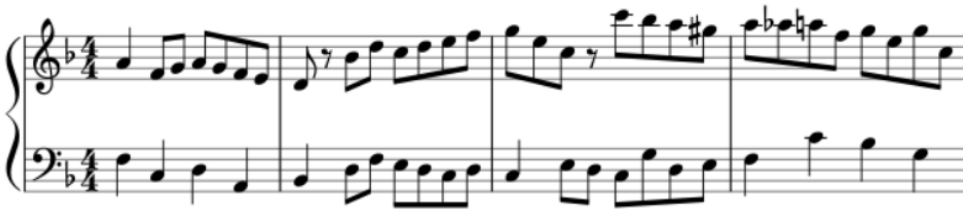

Figure 2.3 Polyphonic melody

2.1.2 Conditioning

Aside from composition from scratch, a popular method in the literature is to generate music by conditioning. These tasks involve conditioning models on additional information such as a predetermined melody or chord progression, which act as constraints on the generation process. Based on the constraints, the aim is to complete the composition.

- **Melody generation given the chord progression** - a common way to condition the generative models is to provide an input sequence of chords onto which the melody is to be generated. A chord is a harmonic collection of arbitrary pitches composed of two or more notes played simultaneously, and a series of ordered chords is called a chord progression.
- **Melody harmonization** - this task involves the generation of harmonic accompaniment, meaning the generation of a harmonically appropriate chord progression based on an input melody, or the generation of an additional melody that harmonizes with the input.

The two main tasks are composition from scratch and conditioning, though there are have been many other attempts at algorithmic compositions that used different approaches. One particular branch, for example, focuses on music performance. Music performance refers to various characteristics of a piece, which are partly explicitly stated within musical scores, but in part arise through the musician's performance and serve to give a musical piece a more expressive sound and which overall makes the music sound more vivid and interesting. [10] These characteristics include slight variations in tempo, dynamics and articulation, and are considered to be an essential part of music performance. Music generation without taking into account these characteristics can result in a lower quality of composition. Encoding expressive characteristics into a model can be done via the choice and features of the representation of the training data, or it can be an algorithmic composition task in itself. Furthermore, a branch of algorithmic composition focuses specifically on interactive generation, in which the model improvises melodies or harmonies in real-time based on a human player. [11]

Overall, while each task has interesting characteristics, the focus of this work has been selected to be composition from scratch. This was chosen as it is the most complete and autonomous form of music generation and relies less on musical theory concepts compared with other tasks such as rendering performance expressive. Compared with conditioning, which relies on a human input for the melody, and thus the evaluation of the result will be with respect to the input melody, by performing a composition from scratch task, the results can be compared directly with the training data which will enable a more analytical assessment of the results.

2.2 Representation

In this section, an overview of the representation methods available for encoding musical content is provided. The encoding strategies are important to consider as they are closely related to the algorithmic composition task to carry out, and lead to different outcomes and characteristics of the generated music.

In most cases, for the majority of applications in algorithmic composition and especially tasks which involve generation of music from scratch, the training data is composed of musical tracks, with the output then being itself a musical track, melody or a component of a musical track. To input musical pieces into an AI system requires an appropriate representation of the piece. The two main types of representation are audio and symbolic representations. In a broad sense, this corresponds to a divide between a continuous (audio) and discrete (symbolic) representation, and each have their own advantages and disadvantages.

2.2.1 Audio

The most fundamental type of representation for musical content is the audio signal, starting with the raw waveform, encoded in computers as an audio bitstream. The waveform representation contains the two main dimensions of time and amplitude of the signal. As this format is not transformed, it contains the full resolution of the original sound, though entails a heavy computational load in terms of memory and processing. Figure 2.4 shows an example of a waveform.

Figure 2.4- Waveform representation of an audio file

Transformations may be applied to waveforms to compress the data and extract higher level information, though the loss of information means introducing some bias. A common transformation is the Fourier transform, which results in the waveform's spectrogram, resulting in a representation containing the time, frequency and intensity of the sound. An extension of this specifically used in music is the chromagram, in which the frequency dimension is discretized onto the musical scale.

2.2.2 Symbolic

Symbolic representation of music deals with fundamental concepts such as notes, rests and chords, and additional characteristics such as dynamics which serve to make the musical piece more expressive. A very brief overview of these concepts is provided below.

- Notes: symbols which denote a musical sound. They specify its value in terms of pitch, relative duration and dynamics. The dynamic notation used in music theory is a range of qualitative values that suggest the loudness at which the note should be played, such as pianissimo (ppp) or fortissimo (fff).
- Intervals: the relative transition between two notes, in terms of semitones (the smallest possible increase in pitch class).

- Chords: set of at least three notes played simultaneously, described by the pitch class of its root note and the type, such as major or minor which depends on the exact intervals between the notes.
- Time signature: describes the temporal dynamics of the musical piece. Composed by beats, the unit of pulsation, which are grouped into measures. The number of beats in a measure constitutes the time signature of the piece, and is often represented as a fraction such as ¾, meaning 3 beats per measure.

For the purpose of algorithmic composition, this information can be encoded in a variety of ways to be interpretable by a computer. The most popular format that makes use of symbolic information is the Musical Instrument Digital Interface technical standard, which describes a protocol for interoperability between electronic instruments, software and devices. A MIDI file carries messages that specify real-time note performance data as well as control data. The two main messages of concern for algorithmic composition are:

- Note on – to indicate that a note is played.
    - Channel number, to indicate the instrument or track, specified by an integer between 0-15
    - Note number, to indicate the note pitch, specified by an integer between 0-127
    - Velocity, which indicates how loud the note is played, also specified between 0-127
- Note off – to indicate that a note ends. The velocity in this case reflects how fast the note is released.

The note events are embedded in a data structure that contains a delta-time value which specifies the timing information, and is represented by relative time using the number of ticks from the beginning, with metadata to relate the number of ticks with absolute time.

A piano roll implementation, inspired from automated pianos, is a representation in two dimensions, in which time is represented on the x axis and discrete pitch classes are represented on the y axis. A third dimension may be added, the dynamics, by changing the colour intensity of the notes. An example is shown in Figure 2.5.

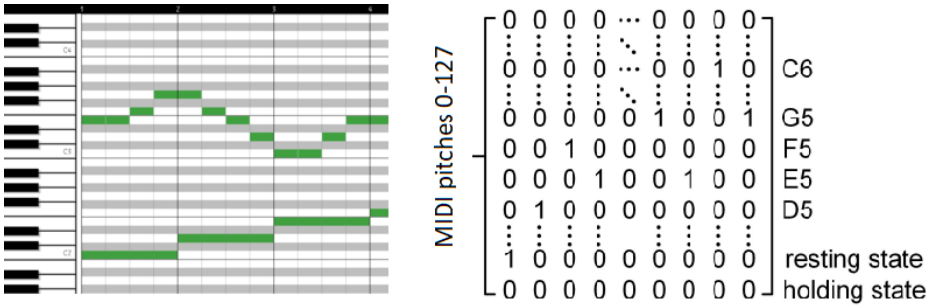

Figure 2.5- Piano-roll with corresponding matrix representation

This is commonly used as a basic visual representation in musical softwares, as it is more intuitive than traditional music notation. For algorithmic composition, this can be in the form of an image, or directly as a 2D matrix, either binary or from a range of 0-128 to denote the intensity of each note. This representation has a significant limitation compared to MIDI, which is the lack of an explicit note off instruction, meaning that additional steps are required to distinguish between a long note or two short consecutive ones. For example, the holding and resting states of each note can be explicitly encoded as a separate row in the matrix.

From an artificial intelligence perspective, the processing of these two types of representation is essentially the same. This means that the different generative models can theoretically be applied to both forms of representation. However, the choice of representation will have significant effects on the overall results of the model, and will influence the model choice, data preprocessing and post processing, and feature selection and extraction from the training data. The audio format requires techniques for processing the data which are based in signal processing. Raw audio signals contain the full initial resolution of the input musical piece, with all the information regarding the temporal and acoustic dynamics. Because of this, working with audio representation is significantly more computationally demanding than the symbolic representation. The audio waveform, as stated, can be transformed in order to reduce the computational requirements, and operations can be applied to the waveform to extract a wide range of acoustic features, giving information on timbre and sound quality that cannot be derived from the symbolic representation. Still, while these qualities ultimately make a musical piece more enjoyable, the symbolic representations encode all the necessary information for generating music, and at a lower computational demand. Therefore, since the aim of the project is to generate music on a more abstract level, the generation of timbre and acoustic characteristics is not as important, making the symbolic representation the most suitable choice. It is worth noting that the majority of currently existing systems use symbolic representations for the training data, though some of the most successful in terms of the quality of the generated music have used audio representation. [4]

2.3 Overview of Artificial Intelligence Methods for Music Generation

In this section, the various methods used in algorithmic composition will be discussed. This section provides a more historical perspective and does not consider the state of the art in the field, but rather the different approaches taken so far and how the paradigms for solving algorithmic composition tasks have shifted over time. The methods used to attempt autonomous music generation come from a wide range of artificial intelligence and computational fields such as complex systems. These are shown in Figure 2.6.

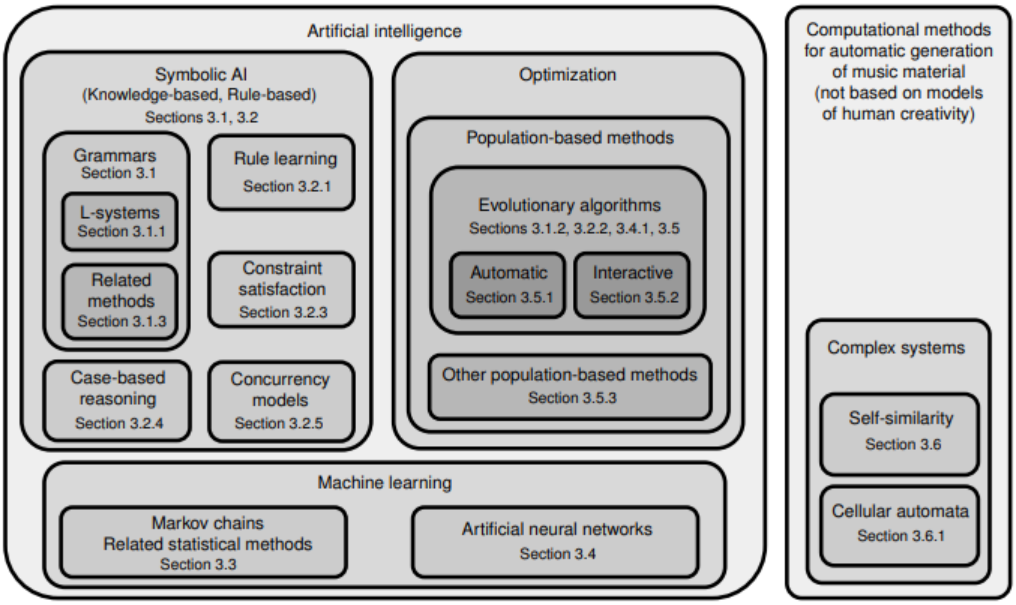

Figure 2.6- AI methods for algorithmic composition [3]

2.3.1 Grammars

A formal grammar is a set of rules used to expand high-level symbols into more detailed sequences of symbols representing elements of formal languages. These elements are obtained by applying rules repeatedly in steps known as derivation steps, and are suited to represent systems with hierarchical structure. Most styles of music have hierarchical structures and have been represented symbolically in a rich notation, so the use of grammars was a natural approach to algorithmic composition, especially in the earlier days when knowledge-based systems were a popular AI approach.

The main step in composing music through grammars is to define the set of rules for the grammar to drive the generative process. These can be directly hand-written from music theory, or can be learned from a corpus of compositions. It must then be determined in which case to apply the different rules, which is possible through several approaches such as using activation probabilities (stochastic grammars).

An early example of the use of a formal grammar to compose simple rhythms was proposed by Lidov and Gabura [12] while another example by Rader [13] made use of a grammar, defined by hand from simple music concepts, and made use of activation probabilities to enrich the rules of the grammar. Later, in the 1980s, there was a shift in the implementation of grammars for music composition, whereby the aim was to extract the grammar from existing compositions. Holtzmann [14] described a language to define music grammars from a corpus and automatically compose music from it. In this time, many methods were developed which focused on the analysis of musical corpora to extract grammars, such as Steedman's [15] grammar for analysis of jazz chord progressions. A general concern regarding grammars is that they fail to capture the internal coherency and subtleties required for music composition. Using hand-written rules derived from music theory is limiting, and even in the case of extracting rules from a corpus, the structure of grammars does not allow for particularly interesting compositions, and was suitable as an early demonstration of algorithmic composition. In fact, it is not currently a commonly used method, though there are some more recent examples of the use of grammars. Chemillier [16] adapted Steedman's grammar for analysing jazz music to develop

a system that could improvise on jazz music in real time, while Gillick [17]implemented a method of grammatical inference, extracting stochastic grammars from a corpus, to then generate jazz solos.

2.3.2 Knowledge-based systems

Knowledge-based systems is used as an umbrella term to include various rule-based systems which represent knowledge as a more or less structured set of symbols. Musical compositions have been traditionally structured using formalized rules, so similar to grammars, knowledge-based systems in general are a natural way to implement algorithmic composition. The earliest works in algorithmic composition, such as the Illiac Suite, are rule-based. Gill [18]presented the first application of classical AI heuristics to AC, they used a hierarchical search with backtracking to guide a set of compositional rules from music theory. An important issue with rule based systems is that by using certain rules for a particular style, the results are very limited, and the system cannot be adapted to more general tasks. To overcome this, rule learning is an important step, and was developed extensively throughout the 1990s. Schwanauer [19]implemented MUSE, a rule-based system for solving tasks in harmonization. The core ruleset was static, though a series of constraints was used to dynamically change the rule priorities. When a task was carried out successfully, the system could deduce new composite rules by extracting patterns of rule application. Widmer [20] and Spangler [21] implemented rule based systems for harmonization which could extract composition rules from a training corpus to perform the composition process, giving the systems flexibility over a wide range of genres. Morales [22] designed a rule learning system based in logic programming.

Different knowledge-based approaches have been used for algorithmic composition. One approach that became popular was constraint satisfaction, which made use of logic programming to solve algorithmic composition tasks posed as constraint satisfaction problems. Case-based reasoning is another formal framework which was used. Using different music pieces as cases, the system can store and analyze examples to extract rules and learn rule application patterns dynamically. It has been used especially in improvisation. Case-base reasoning has also been applied relatively recently by Lopez [10] to address the issue of rendering expressing performances, applied to the domain of jazz music.

2.3.2 Markov Chains

A Markov chain is a stochastic process which transits in discrete time step through a finite set of states, in which the successive state depends only on the current state. Usually, they are represented as either labeled directed graphs, or as probability matrices. In algorithmic compositions, the probability matrices can either be extracted from a corpus of training data or obtained by hand through music theory. When developing the Markov chain, the mapping between states and musical objects is an important design decision, usually done as each note being an individual state. It is also important to consider the order of the chain: an n^{th} order Markov chain considers not just the previous state, but the previous n states, which gives the resulting compositions a more global structure at the cost of a higher computational load.

In the early years, Markov chains were a very popular method. The very first attempts at algorithmic composition were based on Markov chains, such as the Illiac Suite and MUSICOMP. Around the same time, there are other non-scholarly examples of systems which made use of Markov chains, such as the GENIAC machine and the “Banal Tune-Maker”, both systems which would generate a simple monophonic melody after analysing a corpus of melodies. The main issue with the use of Markov chain is that using a low n for the order, the resulting compositions would sound random and not musical, lacking any structure, while higher n chains would end up copying entire segments from

the training corpus. As these limitations became apparent the research interest in Markov chains receded. However, they still remained popular for smaller subtasks within algorithmic composition and have been extensively used even in commercial settings. Many musical software suites make use of Markov chains to provide composers with suggestions on potential continuation of melodies. More recently, they have been used in improvisation systems, such as Grachten [23] and Manaris, [24] who combined Markov chains to generate a population of candidate melodies, selected using an evolutionary algorithm.

2.3.3 Artificial Neural Networks

In this section, a survey of artificial neural networks applied to algorithmic composition is provided. This type of method is covered in greater detail compared to the others as it is currently the most promising and successful approach, coinciding with the general advent of deep learning and the predominance of neural networks in generative AI. Hence, it is useful to cover in more depth. For music generation, ANNs are used as a machine learning methods, making use of a set of input examples to train the network, to generate similar patterns. This means that the majority of the musical structure must be learned from a training corpus, resulting in pieces which imitate the style of the corpus. The architecture of choice has historically been recurrent neural networks, in which nodes in the network receive inputs from deeper layers in order to take temporal structure into consideration.

While ANNs have been used for the analysis of musical composition between the 1970s and 1980s, the first example of a network used for generation is the implementation by Todd [25] who used a three-layered recurrent ANN to produce a temporal sequence of outputs encoding a monophonic melody. The network used was inspired from recurrent networks applied in speech applications such as word-to-speech and phoneme recognition. The domain is similar to the music generation domain, as the data input and output has a temporal element with heavy dependencies between each element. In this work, they encode each pitch as an absolute value, though another approach is proposed by Duff [26] who developed a recurrent neural network that used relative pitches for composing music based on a corpus of Bach. The idea of encoding relative values for pitches is interesting as it gives the results translation-invariance.

Artificial neural networks can be applied in many different ways, and following these examples, new approaches emerged. Mozer [27]developed a recurrent neural network devised to capture both local and global patterns in the set of training examples, with an output mapping of pitches that aimed to capture a formal psychological notion of similarity between different pitches. Lewis [28]proposed a feedforward neural network that was trained with a set of melodies that had been assigned a score, ranging from random to actual musical compositions. Once the network had been trained, the network was fed a random pattern, with the network modifying the pattern to maximise the score. Aside from composition of monophonic melodies, earlier algorithmic composition attempts also focused on harmonization tasks. Shibata [29]implemented a feedforward ANN whose output units corresponded to chords, with the aim of generating appropriate chord progressions for a given melody. While feedforward and recurrent ANNs were the main model used, some other models were implemented, such as Bellgard & Tsang [30] who implemented a system for four-part chorale harmonization using an effective Boltzmann machine. Another composition problem extensively addressed was improvisation, particularly jazz as improvisation is integral to jazz music. This task is quite similar to harmonization, though the chord progressions are used as inputs instead of outputs, and it is the melody that is generated by the network. Nishijimi and Watanabe [31], and Toiviainen [32]used ANNs to generate jazz improvisations based on a set of training examples. Franklin [33] used recurrent ANNs for the same task, though with an additional component that made use of

reinforcement learning to improve the quality of the compositions. This example illustrates the possibility of devising hybrid systems that make use of different paradigms of artificial intelligence to achieve a final composition.

2.3.4 Hybrid Systems

For the task of algorithmic composition, it is possible to improve results by developing hybrid models that make use of more than one artificial intelligence method for different stages of the process. With artificial neural networks, different methods have been combined to improve the results, particularly rule-based systems. This is due to the fact that within music composition, there are some basic rules which can be used as constraints, which can help to either set constraints for the generation of musical pieces, or in post-processing to modify the result. HARMONET [34] was designed for chorale harmonization and was composed of a three-layered architecture: a feedforward ANN to extract harmonization information, a rule-based constraint satisfaction algorithm to generate the chords, and a second ANN to add an additional voice to the melody. NETGEN [35] made use of a rule-based multiagent system that would elaborate music generation from a feedforward neural network.

Finally, evolutionary and population-based methods are worth mentioning as these have been extensively applied in the domain of music generation. These involve a changing set of candidate solution which undergo evaluation, selection and reproduction over many cycles, with each candidate being evaluated using a heuristic fitness function. Evolutionary algorithms have been combined with every other method mentioned above. With grammars, they may be used to evolve and refine the grammatical rules, and an entire class of methods, known as grammatical evolution, can be applied specifically to grammars. For rule-based system, a common approach was to use the ruleset to craft a fitness function, and from that obtain the generated musical piece. There are also numerous examples of evolutionary algorithms combined with ANNs, in which typically the network is trained to act as the fitness function of an evolutionary algorithm. [3]

2.4 State-of-the-Art in Algorithmic Composition

2.4.1 Monophonic melody generation

For state-of-the-art melody generation, recurrent neural networks are still amongst the most popular methods. Sturm [36] used a symbolic text representation, representing musical symbols and notes as tokens (such as ABC corresponding to the pitch). They train an LSTM network using around 23,000 music transcriptions to generate new transcriptions. The model, trained on Celtic folk music and known as folk-RNN, was built with three hidden layers having 512 LSTM blocks each, a significant increase in size from smaller networks previously applied to algorithmic composition. The generated pieces are evaluated against the training set using some statistical measures. Google Brain's Magenta project has produced some notable results in music generation. They introduce the Melody RNN models [37]. The first model, Lookback-RNN, uses LSTM to create a monophonic melody with long-term structure. The second, Attention-RNN, uses a technique known as attention to maintain even longer-term structure [38]. The attention method assigns a measure of importance to previous time steps to determine their effect on the current time-step, and has led to improved results in music generation. Anticipation-RNN aims to expand the interactivity of music generation, which typically relies on a seed vector of initial notes to generate the remainder of the composition. The network architecture accepts an entire composition piece, in which the notes are "constraints" and guide the compositional process.

Variational autoencoders have also been successfully applied to music generation. Roberts [39] proposed the MusicVAE model, which employs a hierarchical decoder to send the latent variables generated by the encoder into the underlying decoder to generate each subsequence. This model imposed long-term structure on the generated pieces, though they were monotonous and lacking in expressiveness. Other architectures which had notable results include VRASH [40] and GLSR-VAE. [41]

For monophonic generation, traditional GANS are limited due to their difficulty in generating discrete tokens. Hence, the focus of GANs in monophonic melody generation was to be able to overcome this shortcoming. SeqGAN [42] makes use of reinforcement learning to improve sequence generation, and can be used in a wide range of tasks. SentiGAN [43] was later developed which addresses the issue of mode collapse, a problem found with generators in GANs, by using a penalty target instead of a reward-based loss function.

2.4.2 Polyphonic melody generation

The significant increase in complexity in polyphonic music generation warrants the use of more complex architectures. The first major attempt at polyphonic music generation was a model that combines RNNs and Restricted Boltzmann Machines, known as RNN-RBM. Proposed by Boulanger-Lewandowski et al. [44], they investigated the problem of modeling symbolic sequences, applied to polyphonic music, using a general piano-roll representation. The architecture combines a series of RBMs to learn the complicated pitch distribution at each time step, with a recurrent neural network. This work has generated strong results and has inspired several methods that make use of Restricted Boltzmann Machines. Qi et al. [45] have developed a model that uses recurrent temporal RBMs, whose structure is similar to the RNN-RBM, and make use of LSTM for memorizing historical information. Lattner developed an implementation of the Convolution RBM to generate polyphonic music with high level structure and has achieved notable results. [46]

For recurrent neural networks, a major issue was the fact that they do not have transposition-invariance, meaning that the relationship between notes is absolute and not relative. Johnson [47] proposed two methods for overcoming this problem. The first makes use of neural autoregressive distribution estimation (NADE) which consists of a neural network inspired by RBMs, used in unsupervised distribution and density estimation. This is used to develop the Tied Parallel LSTM-NADE, which is of a similar architecture to RNN-RBM but makes use of tied parallel networks to ensure translation invariance. The second is the Bi-axial LSTM (BALSTM), which makes use of LSTM both in the time dimension and the pitch dimension. The translational invariance is achieved though the networks do not accurately model global structure of the musical pieces. An interesting model that was developed combines recurrent neural networks with generative adversarial networks for training to model the whole joint probability of a musical sequence.

Several examples of variational autoencoders for polyphonic music have been successful, in which the model architecture is generally based on VAEs such as MusicVAE with modifications to account for polyphony, such as the MahlerNet [48]. An interesting example is the PianoTree VAE which utilizes a tree structure representation of music syntax, encoding hierarchies of music concepts to learn polyphonic music distributions [49]. Recently, transformers have become a popular architecture for generation from sequential data. These models make use of the previously described attention mechanism to weigh the influence of different components of the input data, without relying on recurrence or convolutions. The first successful application was by Huang. [50] MuseNet is based on a transformer model to generate up to 4-minute musical compositions in a variety of styles. Zhang

proposed a novel transformer architecture that makes use of adversarial learning with attention to generate long sequences with high musicality.

2.4.3 Multi-track polyphonic generation

Multi-track music generation adds another dimension of complexity, due to the fact that the network needs to consider different instruments each with their own temporal dynamics, but interdependent with the other tracks.

In terms of recurrent neural networks, Chu [51] proposed a hierarchical RNN to generate pop music. Here, the network layers each produce a key aspect of the song, with the bottom layers generating melody, and the higher layers generating rhythm and chords, though made use of prior rules on how pop music is composed to condition the generator. Zhou [52] trained an RNN on a Beatles dataset to generate multi-track music in the style of the Beatles. Their model, BandNet, was able to generate music clips of variable lengths, but expert knowledge was integrated in the learning process, reducing the system's overall flexibility and autonomy. Transformers have shown more promising results recently to generate music. Donahue [53] proposed a transformer model for multi-instrument music generation, based on transfer learning to improve the model's ability to generalize. Ens developed the Multitrack Music Machine (MMM), which is based on transformers. This system takes advantage of the attention mechanism which can appropriately deal with sequences not ordered in time, to concatenate different tracks into a single sequence. [54]

Amongst the most well-known models for multi-track music generation is MuseGAN, proposed by Dong et al. [55] They use a piano roll representation of different musical tracks, implementing a generator and discriminator as deep convolutional neural networks and using different strategies to create a relation between the different tracks. This model was highly successful in creating tracks with a global structure and different tracks which work well with each other, though it was noted by the authors that the model still falls behind the level of human compositions. Nonetheless, this model has spawned numerous follow-up studies. A further implementation of MuseGAN was proposed by Dong that made use of binary neurons to improve a problem in music pieces generated by MuseGAN, which was the presence of highly fragmented notes. Guan [56]implemented a Dual Multi-branches GAN architecture, which makes use of the attention mechanism to extract temporal and spatial features more accurately. This method was found to improve upon the results of MuseGAN, and had a reduced training time. Another concern with MuseGAN and many other methods is the fact that the resulting compositions may imitate the training set to a high degree, which means a lack of novelty in the compositions. Chen developed a model, Musicality-Novelty GAN [57], which focused on the novelty of the generated samples. While novelty in a musical composition is hard to define, they used a basic measure of distance to be maximized between the training and generated samples.

2.4.4 Other

Many other attempts at algorithmic composition have been made using the methods described above. Aside from the tasks of symbolic melody generation, a wide range of tasks have been attempted. One prominent aspect of algorithmic composition is conditioning, particularly the task of composing a melody given an input sequence of chords. For this task, a successful system has been MidiNet which makes use of a CNN-based generative adversarial network to generate melodies. [58] Their architecture makes use of a unit named the conditioner, a convolutional neural network that allows for the chord sequence to be encoded appropriately. JazzGAN [59] combines GANS with RNNs to generate jazz melodies conditioned over a backing jazz chord progression. The result was strong when compared with other systems and is worth noting the additional complexity of the jazz domain

which arises from frequent key changes, off-beat rhythms and complex chord progressions. Recurrent neural networks have also been successfully applied, with ChordAL being used to first generate a chord sequence, which is then used as the input to create a relevant melody, and the two results combined. [60] This method resulted in very coherent harmonical structures, while the temporal structure was found to be lacking. Additional notable results that have made use of generating melodies using current chords as inputs were achieved using ImprovRNN, developed by Magenta and based on the previously mentioned MelodyRNN. [61]

Another group of models applied to algorithmic composition worth mentioning are those used in audio generation. These models make use of audio representation of music to generate songs and are based in methods that have been already applied in other sectors such as speech synthesis. WaveNet has been a recent breakthrough in audio synthesis, using a PixelCNN-based generative model developed by Google DeepMind. [62] It makes use of convolutions to create original waveforms of speech and music, and is able to generate highly realistic music clips. The recurring issue regards long-term consistency. WaveNet has sparked many follow-up studies, such as Engelìs [63] autoencoder model which could be conditioned on global attributes to ensure the long-term consistency. SynthNet, based on WaveNet, has been developed with the more specific aim to extract the timbre of musical instruments and generate novel sounds for the instruments. [64] While not strictly a composition task, their methodology can be extended to a composition task. RNNs have also been applied in the audio representation domain. SampleRNN was used for audio generation from waveforms, using an architecture composed of different modules to deal with the problem of the very high temporal resolution of waveforms. [65] The general problem with the audio representation was found to be that the models end up being very complex, with large number of parameters, a very vast number of training samples required, and very long training times. For this task, GANs have also been applied and achieved strong results- Models such as WaveGAN and SpecGAN [66] used an image generation architecture to generate music waveforms in time-domain and frequency-domain. The use of GANs has reduced to a significant extent the computational costs associated with the audio representation, with the results being comparable with other methods. GANSynth, a GAN to generate audio waveforms, was compared with systems such as WaveNet and was found to be superior. [67]

3 Theory

3.1 Restricted Boltzmann Machines

Boltzmann machines are bidirectionally connected networks of stochastic processing units which can be interpreted as neural network models. They are probabilistic graphical models which can be used to learn aspects of an unknown probability distribution based on samples from this distribution, and in general have a complicated and time-consuming learning process. By imposing certain restrictions on the network topology, you obtain a restricted Boltzmann machine. This is a parameterized generative model and is typically used in the context of unsupervised learning. Its original implementation was used as a nonlinear generalization of principal component analysis, though other applications have been implemented from generative models to recommender systems. [68]

Each node in a Boltzmann machine can be either hidden or visible, depending on whether they represent components of an observation or input. The visible nodes are components of the input data or the observation, while the hidden nodes capture dependencies between the visible nodes that cannot be modeled via direct pairwise interaction between the visible nodes, or in other words act as feature detectors. For example, in an image classification task, each visible unit could represent an individual pixel. For each configuration of the model's hidden and visible states, an energy function is defined from which the probability the model assigns to a particular visible vector can be derived. A diagram of a fully-connected Boltzmann Machine is shown in Figure 3.1:

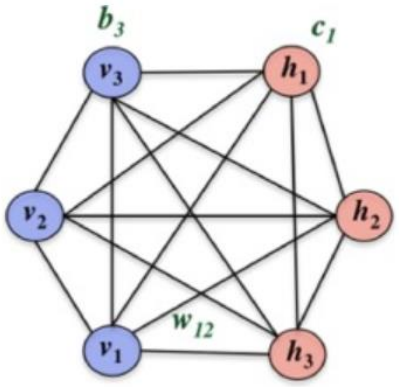

Figure 3.1 - Fully connected Boltzmann machine

For a Boltzmann Machine consisting of *m* visible nodes $v = (v_1, v_2 \ldots v_m)$ and *n* visible nodes $h = (h_1, h_2 \ldots h_n)$ with each $v_i \in \{0,1\}$ and $h_i \in \{0,1\}$, the energy function is given by:

$$E(v, h; \theta) = -\frac{1}{2} v^T \widehat{W} v \; -\frac{1}{2} h^T \overline{W} h \; - v^T W h - b^T v - c^T h \qquad \textit{Eq (3.1)}$$

where $\theta = \{W, \widehat{W}, \overline{W}, b, c\}$ are the model parameters, with $W, \widehat{W}, \overline{W}$ representing the visible-to-hidden, visible-to-visible and hidden-to-hidden weights, while *b* and *c* are the bias terms of the hidden and visible units. The energy function is a function of the configuration of latent variables, and the configuration of inputs provided in an example. The energy function can be used to calculate the probability assigned by the model to a visible vector *v*:

$$p\,(v;\theta) = \frac{1}{Z(\theta)} \sum_h e^{-E(v,h;\theta)} \qquad \textit{Eq (3.2)}$$

where $Z(\theta) = \sum_v \sum_h e^{-E(v,h;\theta)}$ is the partition function. The conditional probabilities of the visible and hidden nodes are given by:

$$p(v_i = 1|h, v_{-i}) = \sigma\left(\sum_{j=1}^{n} W_{ij}\, h_j + \sum_{k=1\setminus i}^{m} \widehat{W}_{ik}\, v_k + b_i\right), \qquad \textit{Eq (3.3)}$$

$$p\left(h_j = 1\middle|v, h_{-j}\right) = \sigma\left(\sum_{i=1}^{m} W_{ij}\, v_j + \sum_{k=1\setminus j}^{n} \overline{W}_{jk}\, h_k + c_j\right), \qquad \textit{Eq (3.4)}$$

where $\sigma(x) = 1\,/(1 + e^{-x})$ is the sigmoid function.

To train the Boltzmann machine, the aim is to adjust the weights and biases in an iterative process such that the marginal probability distribution $p\,(v;\theta)$ fits the training data as well as possible. This is theoretically done by maximizing the likelihood of the training data given the model parameters, but while it is impracticable to do so analytically, gradient ascent on the log-likelihood can be carried out instead. Computing the gradients takes a time that is exponential in the number of visible and hidden units, which means in practice, the expectation values must be estimated by drawing samples from the probability distributions using Markov Chain Monte Carlo methods. [69]

Restricted Boltzmann Machines (RBMs) are a type of Boltzmann machine that make the learning process more efficient by imposing limitations on the connectivity of the units. Instead of having visible-to-visible or hidden-to-hidden connections, the RBM is composed of a hidden layer and a visible layer and is bipartite, meaning there are no interconnections between the nodes in each layer. A diagram of the RBM is shown in Figure 3.2:

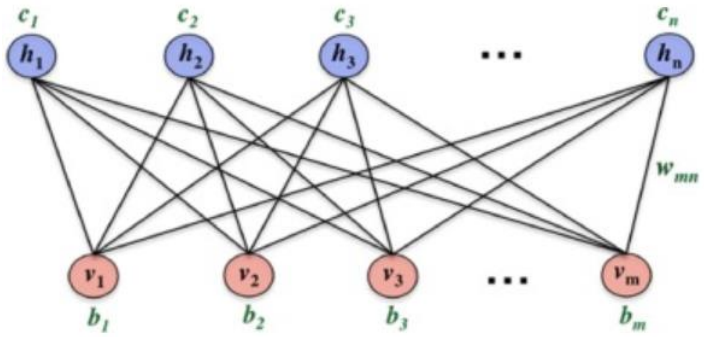

Figure 3.2 - Restricted Boltzmann Machine

Compared with the Boltzmann Machine, this limited connectivity results in a simplified energy function:

$$E(v,h;\theta) = -v^T W h - b^T v - c^T h = -\sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij} v_i h_j - \sum_{i=1}^{m} b_i v_i - \sum_{j=1}^{n} c_i h_i \qquad \textit{Eq (3.5)}$$

Because the graph is bipartite, the hidden units are conditionally independent of one another given the visible layer, and vice versa. Thus, the conditional probabilities for the hidden and visible units reduce to:

$$p(h_j = 1|v) = \sigma\left(\sum_{i=1}^{m} W_{ij}\, v_i + c_j\right), \qquad \textit{Eq (3.6)}$$

$$p(v_i = 1|h) = \sigma\left(\sum_{j=1}^{n} W_{ij}\, h_j + b_i\right). \qquad \textit{Eq (3.7)}$$

This conditional independence of the hidden and visible units means that Gibbs sampling may be performed more efficiently. Gibbs sampling is a Markov chain Monte Carlo algorithm, in which a Markov chain of samples is generated, and the samples within the Markov chain are correlated with neighbors in the chain. Since the nodes in each layer are independent from each other, the nodes of the same type can all be sampled simultaneously. Thus, each iteration of Gibbs sampling consists of two steps:

1) Sample a new hidden vector $h^{(t)}$ based on the conditional probability $p(h|v^{(t-1)})$
2) Sample a new visible vector $v^{(t)}$ based on the conditional probability $p(v|h^{(t)})$

This is also referred to as block Gibbs sampling. The RBM can be interpreted as a standard feed-forward neural network with one layer of non-linear processing units, in which each individual conditional probability variable is the firing rate of a neuron with a sigmoid activation function. From this perspective, the RBM is viewed as a deterministic function that maps an observation to the expected value of the hidden neurons given the observation. [70]

For an RBM, the computation of the log-likelihood gradients remains intractable because the complexity is still exponential in the size of the smallest layer. This could theoretically be approximated using Gibbs sampling, but in practice it would require running the Markov chain long enough to ensure convergence to stationary, which has still computational costs which are too large to yield an efficient learning algorithm. Therefore, training the RBM requires additional approximations. As stated, estimating the log-likelihood gradient using Gibbs sampling requires many iterations. However, it was shown that the estimates obtained after running the chain for a few steps only can be sufficient for model training. This technique is known as contrastive divergence, which has become the standard way to train RBMs. The idea behind contrastive divergence is to run a Gibbs chain for only k-steps (where usually k = 1). The chain is initialized with a training example $v^{(0)}$ of the training set and yields the sample $v^{(k)}$ after k steps. Each step t consists of sampling $h^{(t)}$ from $p(h|v^{(t)})$ and $v^{(t+1)}$ from $p(v|h^{(t)})$ subsequently. [71]

3.2 Generative Adversarial Networks

The generative adversarial network model was proposed in 2014 by Goodfellow et al. [72] At the time, deep learning had already achieved significant results in discovering probability distributions of complex data such as natural images and audio waveforms. The models developed were mainly of the discriminative kind, where the high-dimensional representations were mapped to class labels for prediction. Generative models on the other hand, had not been developed as extensively due to the difficulty of approximating many intractable probabilistic computations arising in maximum likelihood estimation and similar strategies, as well as the difficulty of making use of the benefits of piecewise linear units for generative applications. The model proposed aimed to sidestep these difficulties.

Prior to generative adversarial networks, most of the efforts on deep generative models focused on models which provided a parametric specification of a probability distribution function, which is then trained by maximizing the log likelihood – the Boltzmann machine is an example of such a model. As explained previously, the likelihood functions for these models are intractable and must be estimated through approximations. While these approximations have been shown to be successful in certain use cases, this has motivated the development of models known as "generative machines" which do not explicitly represent the likelihood but are still able to generate samples from the desired distribution. Generative stochastic networks are an example of a generative machine – they are trained with exact backpropagation rather than relying on approximations such as contrastive divergence. Generative adversarial networks extend the generative machine model by eliminating the Markov chains used in generative stochastic networks.

3.2.1 Original GAN

The original adversarial framework developed makes use of two multilayer perceptrons. A diagram is shown in Figure 3.3. The two networks have the following properties:

- Generator network: aims to learn a probability distribution p_g over data x, by defining a prior input noise variable p_z and then represent a mapping to data space as $G(z; \theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g.
- Discriminator network: a multilayer perceptron $D(x; \theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than the generator's distribution p_g.

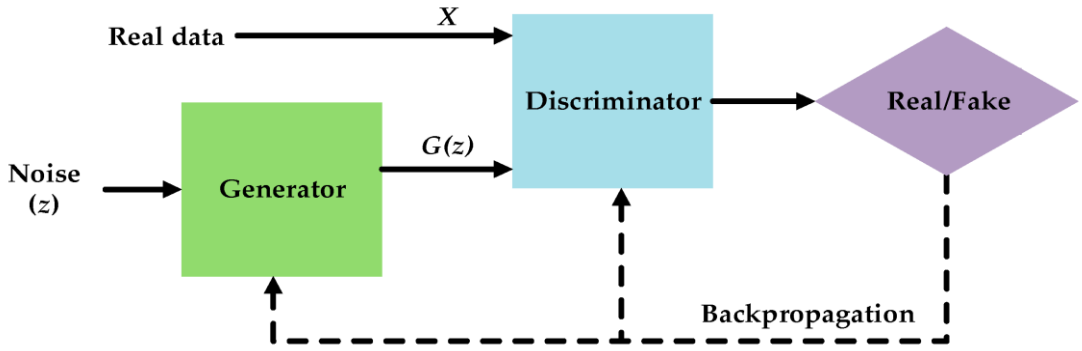

Figure 3.3 - Schematic diagram of generative adversarial network

The discriminator is trained to maximize the probability of assigning the correct label to both the real and generated samples, while the generator is trained to minimize the term $\log\left(1 - D\left(G(z)\right)\right)$, or minimizing the likelihood that the discriminator will correctly label its generated sample as such. This results in a minimax learning process with value function $V(D, G)$:

$$min_G max_D\, V(D, G) = E_{x\sim p(x)}[\log D(x)] + E_{z\sim p(z)}[\log(1 - D(G(z)))], \qquad \textit{Eq (3.8)}$$

where E is the expectation. The minmax learning process is implemented using an iterative approach – optimizing the discriminator to completion in the inner loop of the training is computationally prohibitive and results in overfitting on finite datasets. The approach used is therefore to optimize the discriminator for a number of steps and then the generator for one step, meaning that the discriminator is maintained near its optimal solution. It is worth noting that in practice, the learning process may not provide sufficient gradient for the generator to learn well. This is because early in the learning, the discriminator can reject samples with high confidence, which may lead to a situation where $\log\left(1 - D\left(G(z)\right)\right)$ saturates.

The discriminator and generator's training must be therefore synchronized carefully in order to arrive at a point where the generator is producing highly realistic samples. This makes GANs difficult to train and optimize. If the generator is trained too much without updating the discriminator, there is the risk of "mode collapse", where the generator learns a particular configuration for a sample which the discriminator believes to be from the training data and will therefore subsequently collapse many random noise vectors to the same value, and thus does not model the actual probability distribution of the data. Additionally, even in successful cases, there is no explicit representation of the data's distribution. Despite these drawbacks, the GAN does not require Markov chains and can make use of backpropagation and does not require inference during learning. These characteristics entail significant computational advantages. Also, unlike other generative models, the GAN can make use of a wide range of differentiable functions in theory, making the model more flexible.

3.2.2 Deep Convolutional GAN

Radford et al. 2016 [73]proposed an architecture for GANs, known as Deep Convolutional GANs, which aimed to address some of the limitations of traditional GANs in image generation. Previous attempts at scaling GANs using CNNs had been unsuccessful. By performing extensive model exploration, the authors identified a set of architecture features that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models. The success of their model is partly due to three improvements that had been recently developed for convolutional neural networks that they made use of.

Convolutional neural networks typically made use of pooling layers, in which, after each convolutional operation, a kernel was applied to the convolved features to reduce its dimensions for further convolution or to finally flatten the features to feed into a fully connected layer. Below, the scheme of a traditional convolutional neural network is shown in Figure 3.4.

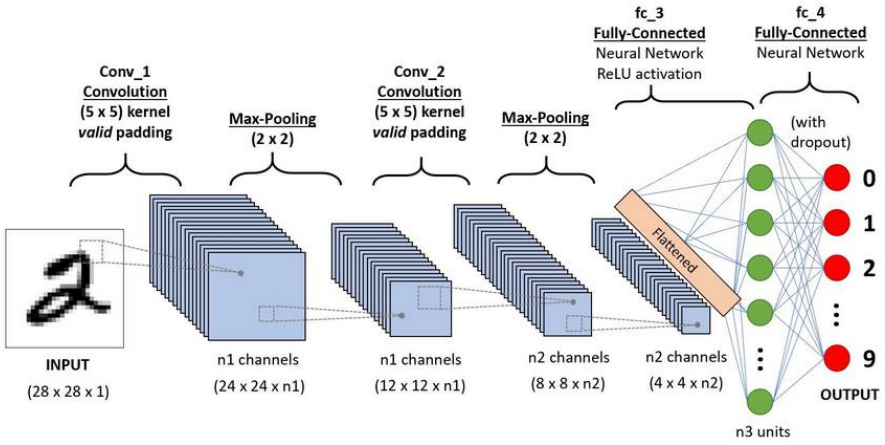

Figure 3.4 - Convolutional neural network architecture

The pooling layer is used for dimensionality reduction, to decrease the computational power required, while also being useful to extract translation invariant features. The first feature of the DCGAN is the all-convolutional net, which replaces the deterministic spatial pooling functions with strided convolutions. This means that the network learns an appropriate spatial down-sampling instead of fixing this. In the DCGAN, both the generator and discriminator are "all convolutional" networks.

The fully connected layer typically used in convolutional neural networks was a computationally inexpensive way to perform feature classification and was structured as a multilayer perceptron. However, it has been shown that other techniques, such as global average pooling, eliminate the need for the fully connected layer and improve the performance in terms of model stability. However, it is also more resource-intensive, so a middle-ground implemented for DCGAN is to connect the highest convolutional features to the input and output respectively of the generator and discriminator. For the generator, the random noise vector input can be regarded to be fully-connected, though with the result being reshaped into a 4-dimensional tensor. A diagram of the generator architecture is shown in Figure 3.5. For the discriminator, the final convolution layer is flattened, and fed into a single sigmoid output.

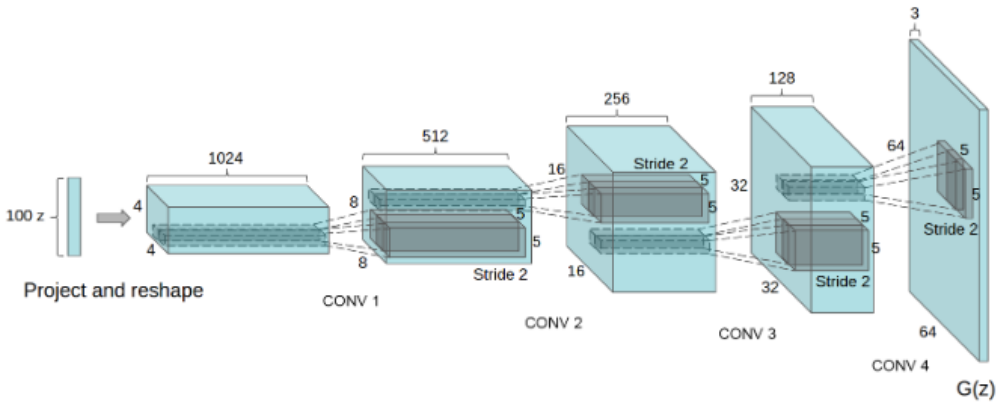

Figure 3.5 - Deep convolutional GAN generator architecture

An additional feature of Deep Convolutional GANs is the use of batch normalization. This type of normalization is performed for each training minibatches, ensuring that the input to each unit has zero mean and unit variance, which has the result of mitigating training problems that arise due to poor initialization of the weights. Batch normalization was found to be critical in reducing the problem of

mode collapse in generative adversarial networks. It is worth noting that the authors found that batch normalization should not be applied to the generator output or the discriminator input layers, as this caused model instability. Finally, regarding the activation functions for the networks, it was found that the rectified linear unit activation (ReLU) worked will for the generator, with the exception of the output layer where a Tanh function is used instead. For the discriminator, the leaky ReLU activation function worked well for all layers.

A summary of the architecture for stable Deep Convolutional GAN training is therefore:

- Replace pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator)
- Use batch normalization in both the generator and discriminator
- Remove the fully connected layers
- Use the ReLU activation function for all layers of the generator except the output, where Tanh is used
- Use LeakyReLU for all discriminator activations

3.2.3 Wasserstein GAN

Generative neural networks essentially minimize divergences between the probability distribution of the data and of the generator. However, it was noted that these divergences are not necessarily continuous with respect to the generator's parameters, which is what ultimately leads to training difficulties. Therefore, an alternative approach is to use the Earth-Mover distance, also called the Wasserstein distance $W(q,p)$, which is the minimum cost of transforming the distribution q into distribution p. Using a set of constraints and assumptions, $W(q,p)$ is continuous everywhere and therefore can be differentiated almost everywhere. This leads to the Wasserstein GAN (WGAN) which has a value function defined as follows:

$$\min\ W(P_r,P_g) = \ min_G max_{D_L \in D}\ \ E_{x\sim p_r}[D(x)] - E_{x\sim p_g}\big[D(\tilde{x})\big] \qquad \textit{Eq (3.9)}$$

where D_L is the set of functions that satisfy the continuity constraints (1-Lipshchitz functions) and P_g is the generator distribution implicitly defined by $\tilde{x} = G(z)$. Under an optimal discriminator, minimizing the value function with respect to the generator minimizes the Wasserstein distance. In this formulation of the GAN, the discriminator network is also known as the critic, as it does not simply classify the samples as belonging either to the training data or the generator's distribution, but rather assigns a score to the sample. The continuity constrained mentioned previously are enforced using a method known as weight clipping, where the weights of the critic network lie within a predetermined range. This form of constraints can lead to several issues during training. In practice, implementing a continuity constraint via weight clipping biases the critic/discriminator to much simpler functions. The interactions between the weight constraint and cost function during WGAN optimization can lead to vanishing or exploding gradients, which is when the derivatives calculated during the backpropagation step either grow or shrink exponentially, making the model incapable of learning. [74]

To overcome these issues, gradient penalty can be implemented. [75]Gradient penalty is an alternative way to ensure the continuity constraint instead of weight clipping. Since the continuity constraint requires a function that has gradients with a norm of at most 1 everywhere, gradient penalty aims to

constrain the gradient norm directly, and is applied to the gradient norm for random samples in the training data $\hat{x} \sim P_{\hat{x}}$. The new value function becomes:

$$\mathrm{L} = \; E_{x \sim p_r}[D(x)] - E_{x \sim p_g}\big[D(\tilde{x})\big] + \lambda \, E_{\hat{x} \sim p_{\hat{x}}}[(||\nabla_{\hat{x}} D(\hat{x})||_2 - 1)^2] \qquad \textit{Eq (3.10)}$$

where λ is the penalty coefficient. The first term corresponds to the original critic loss, while the second term is the additional gradient penalty. When using gradient penalty, batch normalization may not be used as this changes the form of the discriminator's problem from mapping a single input to a single output to mapping an entire batch of inputs to an entire patch of outputs, since the norm of the gradient is penalized with respect to each input independently. Therefore, batch normalization may be omitted, and layer normalization may be used instead.

3.3 Evaluation Metrics

A fundamental issue that arises in algorithmic composition, and more generally any generative model that aims to create something with an artistic aspect, is the difficulty in the evaluation of the generated content. The importance of the evaluation of the generated music will depend on the final objective of the algorithmic composition model. Often, in algorithmic composition, the evaluation is carried out by the designers themselves only, based on a subjective listening test. This could be an acceptable approach if the aim is a proof of concept to investigate the possibility of generating music using a particular model – in that case, the only condition for whether the experiment was successful is whether the model can generate something that sounds acceptable to the designer. If the aim is to develop and improve the performance of a model, or especially compare the performance of different models, then a more objective evaluation measure is essential.

In this project, some objective evaluation metrics are employed in order to better assess the performance of the different models and model hyperparameters. From the literature review of a broad range of different algorithmic composition systems, the main evaluation measure was a qualitative listening test. The test involves either the designer only or a range of participants whose knowledge of music and music theory may vary from no knowledge to expert. It is worth mentioning that generally, subjective evaluation is preferrable for evaluating generative modeling, especially when a large sample of test subjects is used, although the resources required to acquire a statistically significant number of participants, as well as challenges in terms of reliability, validity and reproducibility of the results, have limited the use of rigorous subjective evaluations. Regarding quantitative evaluation metrics, there is no unified evaluation criterion that is used in most cases. However, there are numerous criteria which can be executed to get a measure of the quality of the results and despite their limited musical relevance, they can be used over a large set of results to get an estimate of the general quality of the generated musical content.

The music metrics or descriptive statistics are derived from music concepts, and several have been used across different algorithmic composition projects. The general mode of use of these metrics involves obtaining the metrics for the generated content and comparing these with the metrics calculated for the training dataset and other similar datasets for comparison. The metrics each give an indication of an aspect of the musical piece that is or is not present, and while the metric does not guarantee that the quality is good, by employing a range of different metrics and comparing with the training data, it can be expected that these values should not differ significantly.

3.3.1 Signature Vector

The first set of characteristics are informed by domain knowledge but simple to interpret. Based on the 2-dimensional generated images, which represent the pitch and time dimensions, the main information to be extracted regards either the pitch information or the rhythmic information. More advanced information that can be derived from an audio representation, such as timbre, are not relevant to the symbolic representation chosen. The signature vector is composed of the following parameters:

- Number of notes – the number of distinct notes in the piece (normalized by length of piece).
- Occupation rate – the percentage of non-null values in the piano-roll representation. This metric is useful for removing examples from the training data which are too sparse.
- Polyphonic rate – the number of time steps where two or more notes were played simultaneously, divided by the non-empty time steps.
- Pitch range descriptors – maximum, minimum, mean and standard deviation of the nun-null pitches in the piece.
- Duration range – maximum, minimum, mean and standard deviation of all durations in the piece.
- Qualified note ratio – the ratio of qualified notes, meaning notes no shorter than three timesteps.
- Qualified rhythm frequency – the frequency of note durations within the valid beat ratios within the set: $\{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}\}$

The signature vectors for the training data and for the generated data may be compared to obtain an indication of the quality of the results. [11]

3.3.2 Information Rate

An aim in generating music is that the resulting pieces not only possess low-level or local structure in terms of harmonic intervals, but also must maintain a global coherency in order to sound realistic and pleasing. Because of this, a useful measure, obtained from information theory, is the Information Rate. This is the mutual information between present and past observations and is at a maximum when repetition and variation are balanced. For both random sequences and very repetitive sequences, the IR is low. It has been shown to provide a meaningful estimator for musical structure. [46]For a given sequence, the average IR is defined by:

$$IR(v_0^N) = \frac{1}{N} \sum_n^N H(v_n) - H(v_n | v_0^{n-1}), \qquad \textit{Eq (3.11)}$$

where $H(v)$ is the entropy of v, estimated based on the statistics of the sequence up to the event v_n. The entropy can be described as the average level of uncertainty inherent in a variable's possible outcomes. Given a discrete random variable X, with possible outcomes $x_1, \dots, x_n$, which occur with probability $P(x)$, the entropy of X is given by:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log P(x_i), \qquad \textit{Eq (3.12)}$$

The entropy for the conditional probability is approximated using a first-order Markov chain. This involves obtaining a transition matrix describing the probabilities of particular transitions from the

previous state to the current state, and assuming the Markov property, which states that the conditional probability distribution for a system at future steps depends only on the current state and not on the state of the system at previous steps.

3.3.3 Krumhansl-Schmuckler algorithm

Perceiving the tonality of a musical passage is a fundamental aspect of the experience of hearing music. In Western music, the individual tones making up a piece of music are organized around a central reference pitch, called the tonic or tonal centre, and music organized in this fashion is said to be in a particular musical key or tonality. The tonic note is considered the most representative note of the key, and the remaining pitches vary in terms of how representative they are of this tonality relative to the reference pitch. Additionally, the two major types of musical keys in Western music are the major and minor keys, which are constructed differently in terms of the pitch intervals between notes within the scale. Because these keys can be built on any of the 12 pitch classes (from C to B) there are a total of 24 keys or tonalities. Figure 3.6 shows the C major scale:

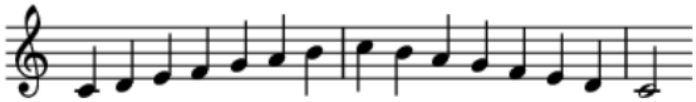

Figure 3.6 - C major scale

The Krumhansl-Schmuckler key-finding algorithm is one of the most well-known techniques for estimating the key of a musical piece and is derived from cognitive psychology experiments in which participants were asked to assess the relations between different pitch classes in different keys. [76] From that, a key profile was devised for each key - a vector of 12 values which represent the stability of the pitch relative to a given key (shown in Figure 3.7).

major profile

do	do#	re	re#	mi	fa	fa#	so	so#	la	la#	ti
6.35	2.23	3.48	2.33	4.38	4.09	2.52	5.19	2.39	3.66	2.29	2.88

minor profile

la	la#	ti	do	do#	re	re#	mi	fa	fa#	so	so#
6.33	2.68	3.52	5.38	2.60	3.53	2.54	4.75	3.98	2.69	3.34	3.17

Figure 3.7 - Major and minor key profiles for K-S algorithm (from [76])

To estimate the key of a musical piece, a vector of the total duration of each pitch class within the musical piece must be extracted, and from that, the Pearson correlation must be calculated for each of the key profiles. For the major key, this involves finding the Pearson correlations between the major profile and the vector of pitch durations, transposing the profile by one position for each subsequent note. The Pearson correlation is given by:

$$r = \frac{\Sigma(x-\bar{x})(y-\bar{y})}{(\Sigma(x-\bar{x})^2\Sigma(y-\bar{y})^2)^{1/2}} \qquad \textit{Eq (3.13)}$$

where x is the input vector values, y is the key profile values, and $\bar{x}, \bar{y}$ are the average values. The Pearson correlation is calculated with each key profile, and the key yielding the highest value is chosen as the preferred key. Despite being the most well-known method for key-finding, there are still significant problems regarding its utility. The results obtained vary substantially based on the musical pieces – tests on Bach preludes have yielded an accuracy of 91.7%, while on Chopin preludes, the accuracy was only of 70.8%. Many pieces may contain key changes and not be in a single key, and other similar factors complicate the assessment of a piece's key. Nonetheless, it provides a good estimate for the key, and in general, the correct key is given a relatively high correlation.

For this project, the Krumhansl-Schmukler algorithm was implemented. Applying this algorithm to the generated pieces will give a strong indication of the musicality of the piece. For a random, unmusical generation, no key will yield a significantly high correlation. On the other hand, if a key with a high correlation is obtained, it will indicate that the generated piece follows a coherent tonal structure.

3.3.4 Evaluation

The final evaluation of the generated dataset follows the approach of using relatively simple features based on domain knowledge, encompassing both pitch-based and rhythm-based features. After extracting these features for both the training dataset and the generated dataset, an absolute and relative measurement is obtained. The absolute measurement is the average and variance of each of the features, and gives insight about the training set properties and generative system's characteristics. The relative measurement allows for the comparison of the distributions of the two sets. The following section outlines how to obtain the relative measurement.

To enable the comparison of different sets of data, such as the training data and the generated samples, a relative measure is required to generalize the results. This is done by obtaining the intra-set distances of each feature (between samples from the same dataset) and inter-set distances (between the two datasets). As mentioned previously, a signature feature vector is extracted for each sample within a set of data. Using the intra-set distance as an example, for each feature, the Euclidean distance between the feature of that sample and all other samples is calculated, and the process repeated for each sample. The result of this process is a histogram of distances for each feature. The histogram is smoothed using kernel density estimation, a process which estimates the probability density function. The probability density function using kernel density estimation is:

$$f(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \qquad \textit{Eq (3.14)}$$

Where K is the kernel, and h>0 $h > 0$ is the smoothing parameter known as the bandwidth. The kernel used in this project is a Gaussian kernel given by:

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}, \qquad \textit{Eq (3.15)}$$

and the bandwidth used is given by Scott's rule of thumb bandwidth estimation method:

$$h \approx 1.06 \sigma n^{-\frac{1}{5}}, \qquad \textit{Eq (3.16)}$$

where σ is the standard deviation of the data.

The distance between probability density functions of the target dataset's inter-set distances and intra-set distances can be computed using the Kullback-Liebler Divergence (KLD). The Kullback-Liebler Divergence is a measure of how one probability distribution differs from a reference probability distribution and is also known as relative entropy, as it is essentially a measure of the entropy of a probability density function compared with a reference density. It is given by:

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right), \qquad \textit{Eq (3.17)}$$

Where P, Q are two probability distributions defined on the same probability space X. By measuring the Kullback-Liebler divergence between the training and generated datasets, an estimate of how well the generative model was able to model the input data distribution can be obtained. In addition, both the intra-set and inter-set distances themselves can be used to determine useful information on the generated samples. By determining the mean and standard deviation in the intra-set distance probability density functions, it can be determined whether the generated samples have lost variety and can be used to identify mode collapse in GANs, which is when the model generated very similar samples. [77]

4 Methodology

All the implementations for this project have been carried out in Python 3.7 and are available online following this link [78].

4.1 Datasets

In terms of representation of the audio data, there are both symbolic and direct audio datasets. While in this project, the symbolic representation is used, audio datasets could be theoretically used and converted into a symbolic representation. One of the largest audio format music datasets used in music generation systems is the Free Music Archive (FMA) which consists of over 100,000 unique tracks and a broad range of over 100 musical genres. However, the audio format is particularly memory-intensive, and the entire dataset is of 917GiB, making the acquiring and converting of this data into a symbolic representation to be cumbersome and time-consuming. The only dataset found which was publicly available, already in symbolic MIDI format and of a comparable scale is the Lakh MIDI dataset, which consists of around 175,000 unique MIDI files. Like the FMA dataset, these files cover a broad range of musical genres, though the entire dataset is much smaller in terms of memory size. It is the most commonly used dataset in recently developed algorithmic composition systems. [11]

However, the Lakh dataset mainly consists of 20th century musical genres such as pop and rock and contains multi-track MIDI files with up to 10 instruments and a rhythmic drum section. For this project, the aim is to generate polyphonic music within the limit of a single instrument track. Furthermore, the choice of dataset in this project was influenced by the assumption that by limiting the training data to a single musical genre, the generative model would be able to generate more realistic music. This is based on the idea that it is more beneficial to learn patterns in the data such as scales, tonality, and rhythm which are consistent throughout the whole dataset, as opposed to having examples from completely different genres such as jazz and pop which have different characteristics.

For that reason, the final dataset chosen for this project is the GiantMIDI-Piano (GMP) dataset. It is a dataset in MIDI format of classical music only, in which each piece is a single-track polyphonic piano piece. It is significantly larger than any other classical music dataset available, and consists of 2786 composers, 10,854 unique pieces and a total of 1,237 hours. For this project, the dataset size was considered to be an acceptable size.

4.2 Data Preprocessing

While the GiantMIDI-Piano (GMP) dataset contains MIDI files, these need to be converted to a representation useable with the generative models. For both Restricted Boltzmann Machines and Generative Adversarial Networks, a piano-roll representation is suitable. A piano-roll representation is a 2D binary array where the vertical axis represents the pitch class of the various notes and the horizontal axis represents time. To obtain this information, the MIDI files were parsed using the "python-midi" library which extracts the MIDI messages [79]. The two main events within the MIDI files from which to extract the relevant information are "Note On" events and "Note Off" events, which contain the start or end time of the notes as well as their pitch class, in a range from 0-127. The start and end times are given in terms of relative ticks, and additional information in the MIDI files

is required to create the piano-roll. These are the resolution, tempo and time signature of the track. Figure 4.1 shows what the numerator and denominator in the time signature represent.

$$resolution = ticks/beat \qquad \textit{Eq (4.1)}$$

$$tempo = beats/minute \qquad \textit{Eq (4.2)}$$

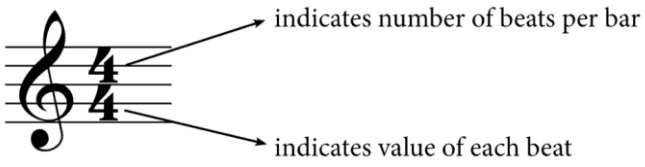

Figure 4.1 - Illustration of fractional time signature notation

Using the resolution and the tempo, it is possible to convert the relative ticks in the MIDI format to absolute time in seconds. Once calculated, a temporal resolution for the piano-roll must be chosen. Regarding the piano roll as an image, each pixel will represent a set amount of time, which must be chosen a priori based on the desired resolution. In the GMP dataset, all tracks are in a time signature of $\frac{4}{4}$, also known as common time, the most basic time signature. This means that each bar or measure within the piece is composed of four beats, each beat being a quarter note. The tempo of all the tracks in the dataset was of 120bpm, or 2 beats per second. As each second is 2 beats, or 2 quarter notes, this means that each second is a half-note. Figure 4.2 shows the structure of different note durations.

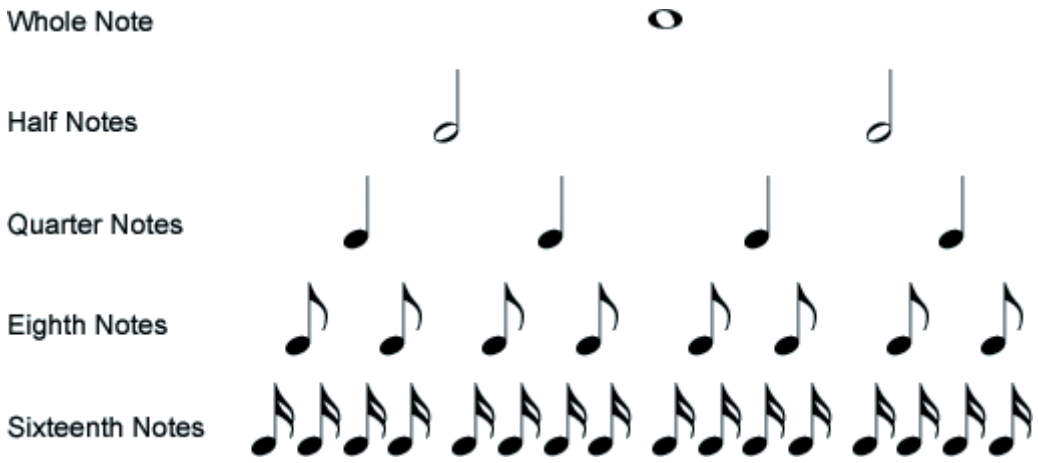

Figure 4.2 - Subdivisions of a whole note

As each second is a half note, the temporal dimension chosen for this project was to represent each second as 16 pixels. This means that 8 pixels represents a quarter note, and the minimum resolution of the piano-representation is a 32nd note, which using a tempo of 120bpm lasts a sixteenth of a second. Finally, the other parameter to decide before transforming the data to a piano-roll representation was the number of pixels which represent a single pitch class in the vertical dimension. For simplicity, this value was chosen to be 1, so a single pixel represents a distinct pitch class, giving a piano-roll of height 128 in total. An example of a MIDI piece converted to piano roll representation is shown in Figure 4.3:

Figure 4.3 Frederic Chopin, Waltz in A Minor B.150 in piano-roll/image representation

In terms of preprocessing techniques for applying generative models to music datasets, the main step to carry out is to transpose all examples into the same key or using samples from one key only. [53] This step is especially important if the model is not translation-invariant. For generative adversarial networks, which in this project make use of convolutional operations, there is some degree of translational invariance. For the restricted Boltzmann machine, transposition would lead to a more robust learning of the data's probability distribution. For some experiments, a subset of the data, belonging only to the keys of A minor and C major (as these keys are very similar) is used. Before using the dataset, additional preprocessing techniques are required.

Firstly, the duration of each piece is reduced from the original data. Figure 4.3 shows the first 20 seconds of Frederic Chopin's Waltz in A Minor. Considering a resolution of 16 pixels per second, this gives a total of 320 pixels in the horizontal dimension. The majority of pieces within the dataset are over 3 minutes long, which would result in images of significant dimensions which go beyond the capabilities of current generative models. Given that within 20 seconds of the piece, there are already complete melodic structures and chord progressions, and the evaluation metrics can still be applied, it is sufficient to make use of the first 20 seconds only.

Secondly, to reduce the size of the input data images and data sparsity, unused pitch classes may be removed. The range of MIDI pitches, as stated previously, is 0-127. However, this number is significantly larger than the actual pitches most commonly used and goes beyond the pitch range of a standard piano (which has 88 notes). Furthermore, the authors of the dataset report the histogram of note frequencies for each pitch class (Figure 4.4), [80] and some of the upper and lower pitch classes are not used in any example and can therefore be removed. Removing the upper and lower pitches gives a range from C1 to G7, or 80 pitches in total. This gives an input image dimension of 80x320.

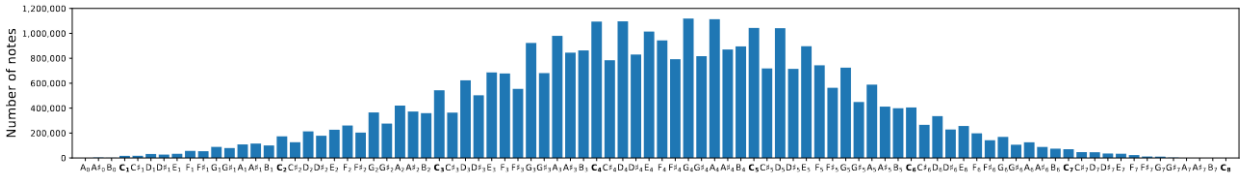

Figure 4.4 - Frequency of pitches in the GiantMIDI-piano dataset

While the dataset is too large to inspect each example manually, as the aim is to generate polyphonic music, it is important to verify whether the examples in the training data have an acceptable polyphonic rate. In addition, some of the examples may have pauses which take up a significant portion of the song piece and will therefore end up being very sparse. To verify this, both the longest pause in each piece and the total number of occupied timesteps are determined.

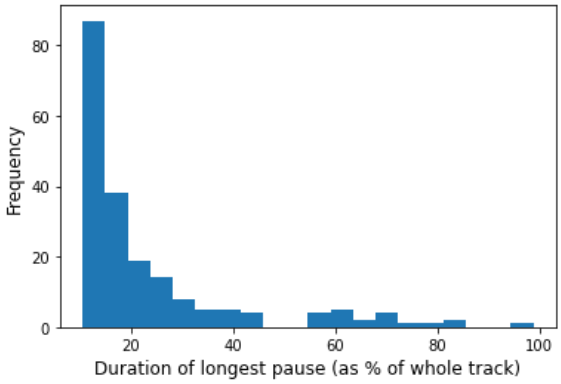

Figure 4.5- Histogram of maximum pause durations for selected samples

Figure 4.5 shows a histogram of pause durations, as a percentage of the whole track length, for the 1000 pieces that had the longest pause. The mean value for the longest duration for the dataset was of 1.64%, though there are a few outlier cases as shown in the histogram. Already in the 1000 pieces with the longest pause, the majority of the examples have a pause which is under 10% of the entire track. For that reason, an upper bound of 10% for the longest pause was chosen as a threshold to remove the outliers. Regarding examples in the dataset which are not polyphonic, Figure 4.6 shows the total number of occupied timesteps for each example, sorted in increasing order.

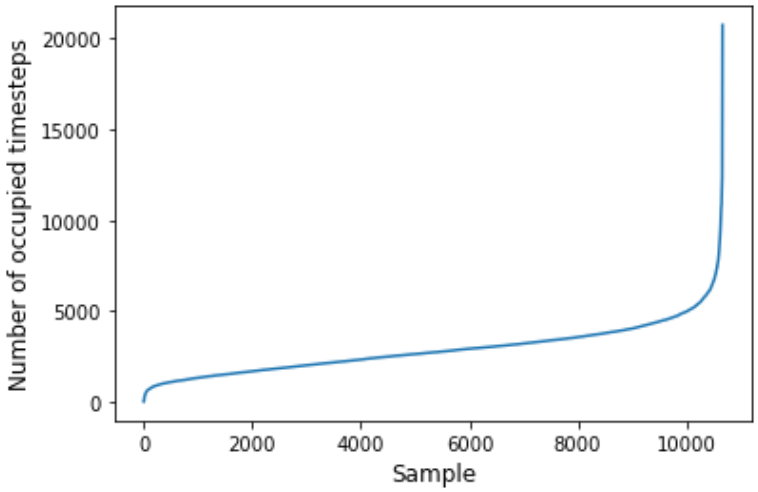

Figure 4.6 - Number of occupied timesteps in each training example

The figure shows that the increase is mostly linear with the exception of a select few samples that have a significantly larger number of occupied timesteps. In order to keep the distribution as uniform as possible, the upper bound chosen was of 6000, which roughly coincides with the beginning of the exponential increase in number of occupied timesteps shown. For the lower bound, a simplifying assumption made was that as the aim is to ensure polyphony and reduce data sparsity, a minimum of at least 600 occupied timesteps is required. For images of dimension 320, having 600 occupied timesteps equates to an average of just under 2 notes per timestep, indicating that the track is polyphonic.

Finally, while convolutional neural networks such as the Deep Convolutional GAN do not require a square image as an input specifically, and different aspect ratios are possible, for the sake of simplicity in utilizing square filters in the convolutional operations, the images have been reshaped to a square. The result is that instead of an 80x320 image used as an input, the new dimensions are of 160x160.

This reshaping is performed by dividing the original image into two 80x160 segments and stacking them vertically. An example of a reshaped image is shown in Figure 4.7:

Figure 4.7 - Waltz in A minor, B.150 (reshaped)

An alternative method used and tested for certain experiments is to encode the different sections of the piece in three channels as opposed to as vertically stacked segments. In this case, the image is divided in three equal segments and each segment assigned to a single channel. The aim of this is to attempt to enforce some repetition of patterns corresponding where the images overlap. Since entire chords and patterns repeat, if the image is segmented by bars, then melodies or chords which occur at regular intervals should overlap. This could make it easier for the networks to learn repeating patterns. Figure 2.1Figure 4.8 shows the encoding of Waltz in A minor in RGB format.

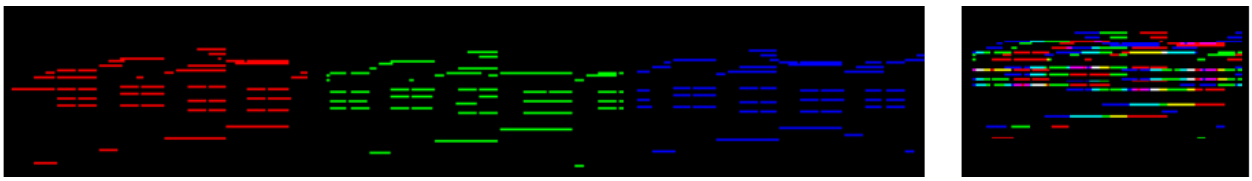

Figure 4.8 - First 16 bars of Waltz in A minor divided in three sections (left) and superimposed in RGB channels (right)

4.3 Evaluation Metrics – Implementation

This section outlines the implementation details of the evaluation metrics. As stated previously, there are three main evaluation metrics. These consist of a signature vector of simple features that act as indicators of rhythmic and melodic quality, and in addition, the average information rate and estimated musical key.

4.3.1 Krumhansl-Schmuckler Algorithm

The Krumhansl-Schmuckler algorithm compares the durations of each pitch class in a song with the major and minor profiles of each pitch class, which were determined empirically. The Pearson correlation is calculated with each key and the result is, for a single musical piece, the correlation with each of the 24 keys. As mentioned previously, the accuracy of the algorithm in terms of assigning the correct key can vary substantially, so there is no target accuracy when implementing the algorithm. However, it is worth assessing if the algorithm can assign the correct key, or a high correlation to the correct key even if this is not the highest. Using Chopin's Waltz in A minor as an example (as the key is known), the durations of each pitch class are extracted (Figure 4.9):

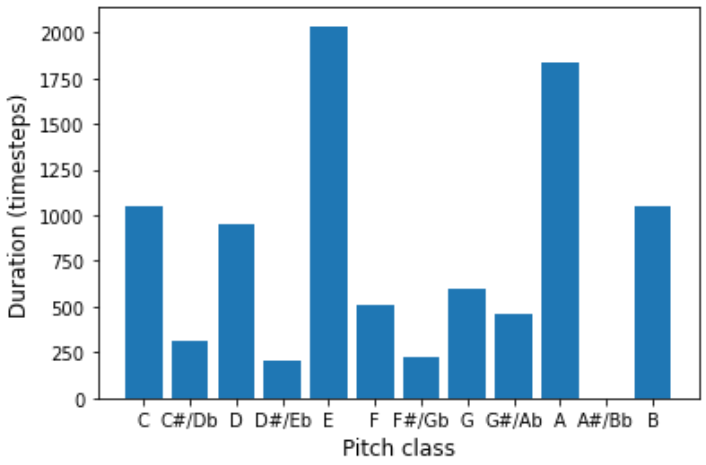

Figure 4.9 - Pitch class distribution for Waltz in A minor, B.150

The correlation between the pitch distribution and the major key profile is calculated once for each pitch class, keeping the values the same but shifting the pitch distribution vector to start on the relevant pitch class. The results for the keys with the highest correlations are shown in Table 4.1:

Table 4.1 - Pearson correlation of Waltz in A minor with different keys

Key	C major	D major	E major	E minor	A major	A minor
Pearson correlation	0.48	0.42	0.58	0.64	0.71	0.84

As shown, the algorithm identified the correct key, assigning the highest correlation to the key of A minor, while also assigning high scores to similar keys. In general, the algorithm has difficulty in differentiating between a major and minor key, and will usually assign similar values to both the major and the minor for a given pitch class. A test was carried out on a sample of the GMP dataset. Using 100 musical pieces which contained the piece's key in the title, of which half were in major key and half in minor key, the accuracy of the algorithm was assessed manually by counting the correct classifications. Results are shown in Table 4.2.

Table 4.2 - Correct assigning of musical keys

Key mode	**Correct key had highest correlation (%)**	**Correct key in top 3 of highest correlations (%)**
Major	64	26
Minor	68	26

While the sample was small, the accuracy was found to be pretty high, and only in around 10% of the cases, the correct key did not appear within the top 3 keys which had been assigned the highest correlation. It is worth noting that the incorrect assigning of the keys could be due in some cases to the title information being incorrect, or the actual song having been transposed in the MIDI file.

However, what is more important for this project is to obtain a high correlation with any key, as this gives an idea of whether the generated piece has musical structure. This is tested by comparing the highest correlation value of the sample set with both random and highly repetitive sequences.

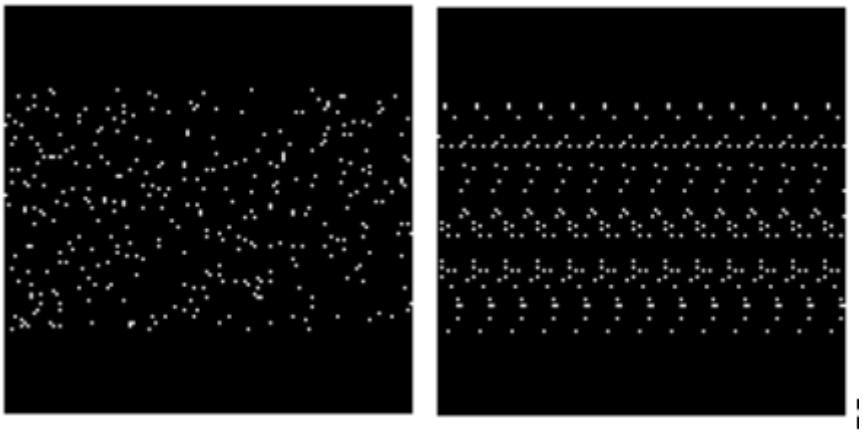

Figure 4.10 - Randomly generated (left) and repetitive (right) piano-rolls

Comparing the highest Pearson correlation of the test set of 100 tracks with 100 randomly generated images and 100 randomly generated repetitive sequences, it was found that the random and repetitive images obtained a significantly lower score on average. The plot of the Pearson correlation for each test sample compared with the Pearson correlation for a random and repetitive sample is shown in Figure 4.11

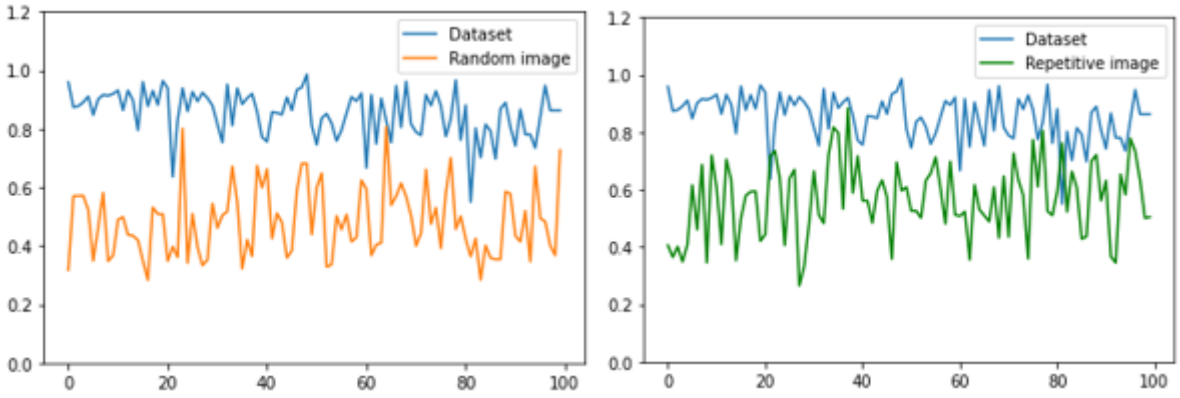

Figure 4.11 – Highest Pearson correlations with key profiles for each sample

As shown, the values for the Pearson correlation fluctuate for each sample. By looking at the average over a number of samples, the real music images obtained a value of 0.85, while the random and repetitive images obtained a value of 0.48 and o.56 respectively. This confirms that the key-finding method may be used as an indicator of musical structure.

4.3.2 Information Rate

The information rate in the context of musical structure is defined as the difference in entropy of the event at the current timestep, and the entropy of the event given the previous events. The IR is to be calculated at each timestep, from which the average for the whole musical piece can be obtained. The entropy of the event given the previous ones can be estimated using a first-order Markov chain. The objective is to obtain a relatively high information rate for the music images, and a low information rate for both random and very repetitive sequences.

As the images make use of 80 pitches, and at each timestep there is no strict limit on the number of pitches that can play simultaneously, the size of the transition matrix is substantial. Each combination of pitches, taking into account the octave it is in, is considered. For Chopin's Waltz in A Minor B.150,

the total number of unique combinations of notes was of 312, meaning the transition matrix is of size 312x312. The average information rate was found to be 4.87, while shows how the IR varied with time.

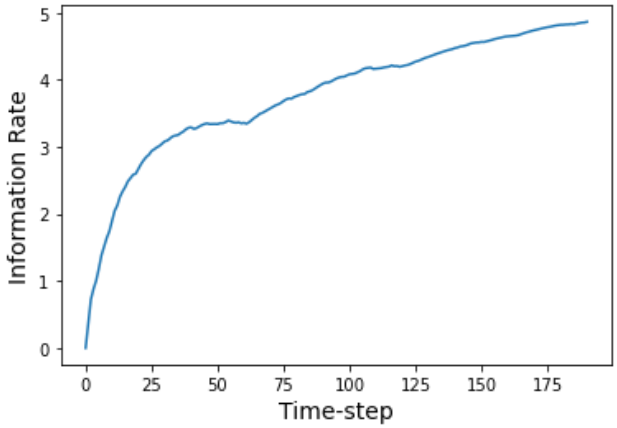

Figure 4.12 - Information rate with time

Figure 4.12 shows the change in the information rate with successive timesteps. As shown, the rate increases with time as the discrepancy between the current state and the previous state given the current increases. In terms of the current state, given by the uncertainty or entropy of the distribution of each pitch class, the uncertainty tends to increase. As new states are observed for the first time, the overall uncertainty tends to increase. Given the vast number of combinations of possible states, new states are observed constantly, increasing the uncertainty. For the uncertainty of the transitions, the inverse effect is obtained. Since their entropy depends on the current state given that of the previous, once a particular state is observed, and given the enormous number of theoretically possible transitions (312x312), it is highly unlikely that more than a few transitions from one state to the next will occur. However, given the repetitions in the musical piece, certain transitions will be very likely, thus reducing the uncertainty. The overall result is an increase in the information rate.

This can be tested against random and very repetitive sequences, to obtain an indication of whether the dataset or generated piece has a higher degree of structure. Using the same test sample as before, and the same random and repetitive image sets, the average information rate for each piece is calculated.

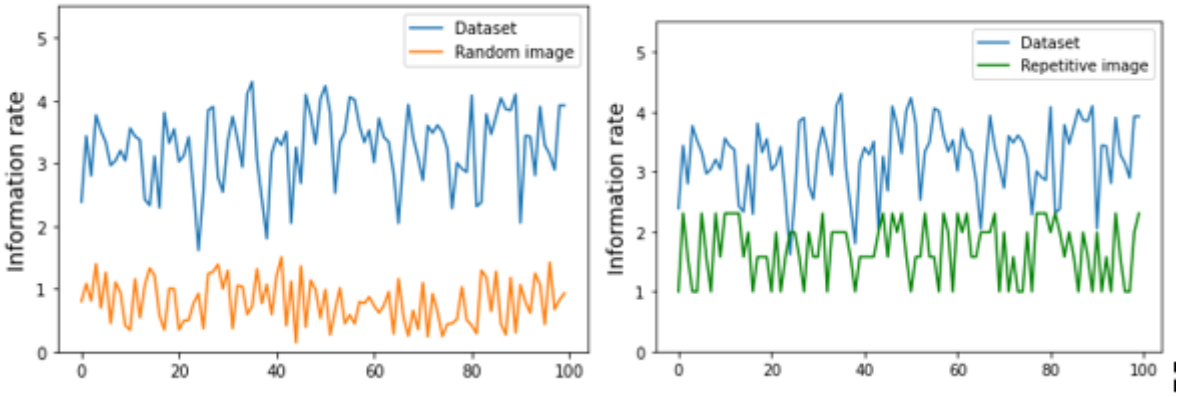

Figure 4.13- Average information rates for each sample

As shown in Figure 4.13, the information rates for the random images is significantly lower than for the images in the dataset. For the random images, the uncertainty for both the current state and the

transitions remains high throughout the entire piece, leading to a low information rate. Therefore, the information rate can be used as a measure to evaluate whether the generated images have more structure. The information rate will also be useful to ensure that the generated images are not too repetitive. In musical compositions, while there are repetitions in melodies and chords, the sequence of note events are not entirely predictable and repetitive. For the dataset of highly repetitive images (repeating every 2-10 timesteps), the average information rate was found to be much lower than for the test images from the dataset. This metric will be useful, as it will ensure that the generative models are not outputting very predictable patterns.

4.3.3 Kullback-Liebler Divergence of Feature Distribution

The final component of the evaluation involves extracting a set of features that describe rhythmic and melodic characteristics of the musical pieces for both the training and generated data, and comparing these with each other. The results of feature extraction on two subsets of the GMP dataset for selected features is shown in Table 4.3:

Table 4.3 - Extracted features for two subsets of the GMP dataset

	Polyphonic rate	Qualified rhythm frequency	Mean note duration	Pitch range
GMP subset 1	0.84	0.46	14,1	55
GMP subset 2	0.89	0.43	10.7	54

As shown, the values for each feature are similar for the two subsets. From these features, it is possible to make a series of observations regarding aspects of the compositions, both in a relative and an absolute way. For example, both subsets have a high degree of polyphony, with almost every timestep being occupied by at least two notes simultaneously. The mean note duration is of over 10 timesteps, indicating that the musical pieces are not fragmented. The values shown are the mean average values of that feature for the dataset. To enable a more robust comparison between the features, the Kullback-Liebler divergence between the probability density functions of each feature can be computed. This is done in three steps.

The first step involves computing exhaustively the pairwise distances between each sample, for each separate feature. Comparing two sets of samples, with n and m number of samples respectively, and k features for each sample, the result is k matrices of dimension n*m:

$$\begin{bmatrix} s_{11} & \cdots & s_{1m} \\ \vdots & \ddots & \vdots \\ s_{n1} & \cdots & s_{nm} \end{bmatrix}$$

where s_{ij} is the Euclidean distance between the value of the feature for samples i and j. As there are duplicate values in the matrix, the relevant triangular (depending on matrix dimensions) is chosen, and the result flattened to obtain a histogram of distances. The second step is to use kernel density estimation to convert the histograms into probability density functions. For the polyphonic rate, the probability density functions for the two datasets are shown in Figure 4.14.

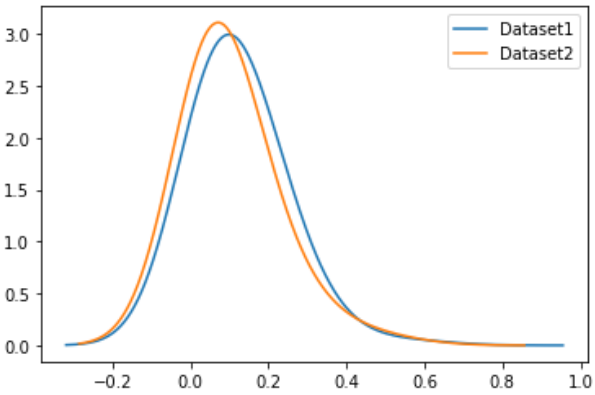

Figure 4.14 - PDFs of polyphonic rate distances for two sets of samples from GMP dataset

The PDF immediately gives a more accurate representation of the similarity between the two datasets with respect to the chosen feature. The final step involves computing the KL divergence between the two distributions. The computed value for the distributions shown above is of 0.015, which is expected as they are very similar distributions (a value of 0 meaning the distributions are identical). For the evaluation of the generative models, the feature vectors are extracted for all images in the training dataset, to be then compared with the generated samples.

4.3.4 Qualitative Analysis – Listening Test

In addition to the quantitative evaluation criteria outlined above, as the final content generated is of an artistic nature, and therefore highly subjective in terms of quality, a qualitative analysis allows for a better understanding of whether the models implemented for music generation are successful. The qualitative measure comes in the form of a listening test. In terms of the project objectives, the main technical requirement outlined was to achieve a polyphonic music generation system which generates entire compositions from scratch, so the only formal constraint in those terms is the polyphony requirement. Nonetheless, the success of the project in achieving the musical compositions is in part based on the extent to which the generated musical samples are realistic and pleasing to listeners. Therefore, a listening test was devised for this project. In order to perform statistical analysis on the results of the listening tests, a large test subject base would be required, with additional information on the level of formal training they have in music as well as listening habits. Due to a lack of an adequately large sample size, the results of the listening test are discussed qualitatively and used mainly as a measure of how realistic and enjoyable the generated musical samples are.

In the literature for algorithmic composition systems, most make use of listening tests as the main evaluation criteria. [9] While these vary in terms of how rigorous the evaluation is carried out, there are many examples of tests. In this project, the tests carried out are based on a Turing test implemented for the LakhNES transformer [53] and a survey carried out by the authors of MuseGAN. [55]The Turing test is a test of a machine's ability to exhibit intelligent behaviour in a way that is indistinguishable from that of a human. In this context, it means testing whether a human listener could believe that the generated musical pieces had been composed by a human. Hence, the participants will be asked to listen to a set of samples created by the generative model, and assess whether they are artificial or composed by a human. In order to provide more context to the participant's answers, they are asked to rate samples not only from the model, but also actual compositions from the training dataset, as well as randomly generated images which are then converted to MIDI files. The random images are made to have characteristics roughly similar to the

training dataset in terms of how many notes there are in total, but the specific values for pitches and note durations are entirely random. Table 4.4 shows an example of the Turing test for one sample:

Table 4.4 - Example of Turing test Table for one generated sample

	Definitely not (1)	Probably not (2)	Not sure (3)	Probably (4)	Definitely (5)
Participant 1		X			
Participant 2			X		
Participant 3	X				

The participants are provided with a range of possible answers, which are then assigned a score from 1-5 in order to finally obtain an average for each sample. Aside from the Turing test, an additional brief survey is provided for each of the samples in which the participants are asked to rate certain characteristics of the musical pieces on a scale from 0-5, with 0 being the most negative score. Table 4.5 shows an example of the survey provided to a participant. It is worth noting that for this test, the comparison with the human and random musical pieces is not required.

Table 4.5 - Example of sample quality survey

	Sample 1	Sample 2	Sample 3
How pleasant are the harmonies?			
How pleasant is the rhythm?			
Is the musical structure clear and coherent?			
Overall rating			

4.4 Restricted Boltzmann Machine Implementation

The generative models were implemented in Python using NumPy and PyTorch. PyTorch is a library which makes use of tensor computing via graphics processing units and contains built-in functions for implementing deep neural networks. The training of all the generative models was performed using Google Colab, which provides a single 12GB NVIDIA Tesla K80 GPU. All the models used in this project are based on publicly available implementations, which are referenced within the project's GitHub repository. [78]

The first generative model implemented in this project was the Restricted Boltzmann Machine. To summarize the architecture of the RBM, the model consists of two layers of nodes, the visible and hidden layers. The visible units correspond to the input pixels of the image, while the hidden units correspond to feature detectors.

4.4.1 Model Architecture

The main RBM class defined contains the methods to train the model, and accepts the following hyperparameters:

- Number of hidden units
- Number of sampling steps in contrastive divergence
- Number of training epochs
- Mini batch size
- Learning rate, momentum & weight decay

The training dataset of images is flattened into a series of 1-D arrays (from dimension 160x160 to dimension 1x26500) . As the state of each pixel (timestep) can be either on or off, the visible units are binary. The class contains two main methods, *fit* and *train*. Once the model is fitted with the data, the visible units are initialized, as well as the weights and biases. The weights between visible and hidden units are initially selected from a random uniform distribution of zero mean and unit variance, while the biases are initially set to a value of 1. The momentum for the weights and biases is initially set to 0. During training, the model performs contrastive divergence for each mini batch and for each epoch. Contrastive divergence is carried out in three main steps:

1) Sample the hidden probabilities from the mini batch. The hidden activation energy is computed using the energy function, and the probabilities of the hidden units being activated is then computed via the sigmoid function. The resulting hidden unit probability tensor is compared with a random uniform tensor to stochastically convert the probabilities in binary states (on or off).
2) Once the hidden activations are determined, alternating Gibbs sampling is performed for a number of *k* steps. In this step, the visible probabilities are sampled from the hidden activations, from which new hidden probabilities are sampled, and the hidden activations stochastically assigned (with higher hidden probabilities having a higher likelihood on being assigned a value of 1).
3) In the final step, the parameters of the network are updated. The momentum is updated, after which the connection weights and biases are updated using the learning rate and batch size, with the weights also updated using L2 weight decay. Finally, the reconstruction error is computed as the difference between the input data's activated visible units and the final visible probabilities obtained.

4.4.2 Experimental Setup

As stated previously, there are numerous hyperparameters for the RBM network. In order to generate musical pieces using the network, and ultimately modelling the input data distribution accurately, the hyperparameters need to be fine-tuned. The selection of parameters requires practical experience, or is best determined experimentally through testing. Some general guidelines have been followed in this project. [81] Regarding the mini batch size, the suggested range is between 10-100. While not the most important parameter, if the mini batch size is too large could lead to weight updates which are too small. It is important to randomize the order of training examples to reduce the sampling error. For the learning rate, if a value which is too large is chosen, the reconstruction error will increase drastically and the weights explode. The learning rate can be modified using momentum and allows the learning rate to be larger without causing unstable oscillations. In this project, the value chosen for the momentum coefficient is fixed at a rule of thumb value of 0.5. It is a conservative momentum which is supposed to make learning more stable. The weight decay, which is a penalty function which penalizes large weights, is used to improve generalization to new data by reducing overfitting to the

training data, as well as dealing with hidden units which develop very large weights early in the training phase. The simplest type, and the one used for this project, is the L2 weight decay, which makes use of a coefficient typically in the range 0.01-0.00001. The value chosen in this project is fixed at 0.001. The choice of the number of features to model, or the number of hidden units, is crucial to learn a good model of the data's probability distribution. While in discriminative models, using more parameters than training cases can lead to significant overfitting, in the case of generative models, this is not necessarily the case. However, considering the large number of visible units in the network, including a similar or larger number of hidden units is very computationally intensive, which had to be taken into account for the choice of numbers of features. An acceptable range however is to use a number of features one order of magnitude less than the visible units, so in this case around 5000 hidden units. [81]

The experimental setup is therefore as follows. Different combinations of hyperparameters will be tested for the model taking into account the general guidelines and limitations mentioned. Due to the high number of total combinations, instead of sampling the final model for musical compositions for each combination, the reconstruction error will be used as an initial estimate of the performance, and the sampling and evaluation of the musical pieces will be performed for the most promising combinations only. The reconstruction error on the entire set should fall rapidly at the start and then slowly towards convergence. Since the reconstruction error is not used in the update of weights, it is not a robust measure of the progress of learning. This means that a low reconstruction error does not necessarily imply that the final distribution of the data was learned accurately. Nonetheless, it can be used as an initial measure before choosing the final hyperparameters. Once a few set of hyperparameter combinations are chosen, they can be used to then sample hidden probabilities and obtain a reconstruction, which is then reshaped back into an image and finally a musical piece. The final sampling of the RBM will be carried out following two strategies: in the first, a random uniform vector is used to sample the RBM, while in the second case, incomplete images from the training data, to be 'reconstructed', will be used. In this context, incomplete means that most of the pixels are turned off, leaving the "backbone" of the musical piece only, while removing the majority of melodies and chords.

4.5 Generative Adversarial Networks Implementation

The key feature of generative adversarial networks is the use of two models which interact in order to optimize a loss function, interrelated between the two models so that they reach an ideal equilibrium point. Since the original formulation of the GAN, that made use of multilayer perceptrons, many different types of models have been implemented and have achieved impressive results. Therefore, the choice of architecture is vast and requires consideration. The model architectures for both the DCGAN and WGAN-GP are implemented as outlined in the original papers.

4.5.1 Model Architecture - DCGAN

The stacking of layers follows the same process, though an additional layer was added to accommodate for the larger dimensions of the images used compared with the original implementations. Below, the architectures for the generator and discriminator are shown:

```
Generator:

    (0): ConvTranspose2d(100, 4096, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(4096)
    (2): ReLU
```

```
    (3): ConvTranspose2d(4096, 2048, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (4): BatchNorm2d(2048)
    (5): ReLU
    (6): ConvTranspose2d(2048, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): BatchNorm2d(1024)
    (8): ReLU
    (9): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (10): BatchNorm2d(512)
    (11): ReLU
    (12): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (13): BatchNorm2d(256)
    (14): ReLU
    (15): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (16): BatchNorm2d(128)
    (17): ReLU
    (18): ConvTranspose2d(128, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (19): Tanh()

Discriminator:

    (0): Conv2d(1, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU
    (2): Conv2d(32, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): BatchNorm2d(64)
    (4): LeakyReLU
    (5): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (6): BatchNorm2d(128)
    (7): LeakyReLU
    (8): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (9): BatchNorm2d(256)
    (10): LeakyReLU
    (11): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (12): BatchNorm2d(512)
    (13): LeakyReLU
    (14): Conv2d(512, 1024, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (15): BatchNorm2d(1024)
    (16): LeakyReLU
    (17): Conv2d(1024, 1, kernel_size=(4, 4), stride=(1, 1))
    (18): Sigmoid()
```

The *ConvTranspose2D* operation is a deconvolution and shows the number of input and output channels respectively, as well as the kernel size, stride and padding. The latter three determine the final dimensions of the image, while the number of channels at each step depends on the number of feature maps chosen, which is a hyperparameter. The generator takes an input noise vector of dimensions 100x1, and transforms it into an image with a single channel. The *Conv2D* operation has the same characteristics, except that it performs a convolution and transforms the input image to ultimately a single value. Aside from the layers, the weights are initialized as specified in the DCGAN paper, randomly drawn from a Normal distribution with mean 0 and standard deviation 0.02. Both models use the Adam optimizer, an extension to stochastic gradient descent which uses the first and second moments of the gradients to determine the moving average of the gradients. [82]The loss function used is the Binary Cross Entropy loss function. The training process takes place in two separate steps. The minibatches are constructed from just real examples, passed through the discriminator to calculate the loss and obtain the gradients in the backward pass. After this, a batch of fake examples is introduced and the corresponding loss calculated. The gradients are accumulated and a step of the optimizer is called, so that the gradients are updated taking into account both the real and fake batch. The second step involves training the generator. The labels assigned by the discriminator are used to obtain the loss in the generator, and once the gradients are calculated, a step of the optimizer is called.

4.5.2 Model Architecture - WGAN- GP

As mentioned previously, the main feature of the Wasserstein GAN is the use of the Wasserstein distance as the loss function to evaluate the distance between the training data and the generated

examples. The discriminator in this case is also known as a critic as it assigns a score on the real and generated images, as opposed to a probability. This has the advantage that the loss is now more informative with regards to the performance of the model. It has been shown in the original formulation of the Wasserstein GAN that there is a strong correlation between the loss and the sample quality, and as the loss decreases, the quality of the samples improves. This is useful in providing a direct and empirical way of assessing the performance, and does not rely on visual inspection or samples or other methods such as Inception classifiers. However, the authors note that the loss cannot be used as quantitative estimate, as the value itself offers little information in terms of comparisons between different critics. [74] This method requires weight clipping, which was shown to have issues in expressiveness of the network. Therefore, gradient penalty was introduced as an alternative, and was shown to work better in practice.

For the generator, the architecture is the same as in DCGAN. The discriminator architecture is shown below:

```
Discriminator:

    (0): Conv2d(1, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): LeakyReLU
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (3): LeakyReLU
    (4): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (5): LeakyReLU
    (6): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (7): LeakyReLU
    (8): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
```

The weight initialization is the same as in the DCGAN. The hyperparameters used are the same as in the original paper and are the following:

Table 4.6 - WGAN-GP default hyperparameters

Gradient penalty coefficient (lambda)	10
Critic iterations per generator iterations	5
Learning rate for optimizers	0.0001
First and second momentum terms	0, 0.9

The first term of momentum was set to zero as other values were found to lead to unstable training. In this formulation of the GAN, the critic or discriminator is trained for five iterations for every training iteration of the generator. It is worth noting that when using the gradient penalty, the layers in the discriminator cannot make use of batch normalization, as gradient penalty aims to penalize the norm of the gradients with respect to each input independently and not the entire batch.

4.5.3 Experimental Setup

Generative adversarial networks are notoriously difficult to train. [83] Combined with a significant computational demand, running many different hyperparameter combinations was found to be unfeasible using the available GPU. This also influenced the choice of image size and dataset size. Nonetheless, a series of experiments was devised in order to obtain indicative answers on the following aspects:

1) Effect of training dataset size on generated music
2) DCGAN hyperparameter tuning

3) Application of WGAN-GP to improve results
4) Final evaluation of generated music using best combinations of settings and hyperparameters

Initially, the model is implemented as suggested by the authors, with the only modification being an increase in the image size to 128x128 (from 64x64 in the paper) and therefore also the feature maps. This is to initially test the feasibility of the GAN for this type of image. A square representation is chosen since even though the result is a more limited range of notes and shorter duration compared with the images used for the RBM, the original DCGAN implementation makes use of a square image and to keep everything as similar as possible, the musical images were reshaped to a square. This setup is tested with three different sizes of the dataset – the aim of this is to find out whether a smaller dataset still yields acceptable results, as using a smaller dataset would mean that more tests can be performed. The hyperparameters used are shown in Table 4.7.

Table 4.7 - DCGAN default hyperparameters

Number of feature maps (generator, discriminator)	128,32
Minibatch size	64
Learning rate for optimizers	0.0002
First and second momentum terms	0.5, 0.999

Regarding the hyperparameter tuning, this process was limited by the availability of the GPU. Finding the equilibrium between the two networks is a more difficult process with more trial and error involved, compared with models which rely on optimizing an objective function. [84]Therefore, the approach used in this project is to start with the suggested parameters from the original implementations, and then apply modifications based on the results. Aside from the Adam optimizers and the minibatch size, some modifications on the architecture of the model could vary the results substantially. For example, the kernel sizes could be modified in order to change the results. Like the RBM, the full evaluation of the generated music will not be carried out for all combinations. Instead, the errors in the generator and discriminators will be used to understand the procedure of the training progress. As the errors do not necessarily correlate to sample quality, for each setting, an image will be generated at various steps and analyzed through visual inspection. This is just a preliminary test to make sure the images are not blank, or too noisy. If a suitable combination of parameters is found, they will be used to carry out the full evaluation.

Following this, the WGAN-GP is implemented, using the hyperparameters outlined by the authors only. This was chosen for limitations of the available computational power. As it was found that the WGAN-GP was more resource intensive and harder to train, some preliminary tests run failed and thus, a single working implementation was carried out. This can still be compared with the results from the DCGAN to see whether improvements are obtained. Finally, once the two are compared, based on which model performs better, a large sample of images is generated for evaluation, using the evaluation metrics previously explained and some samples are converted into MIDI format for the listening test.

5 Results

5.1 Restricted Boltzmann Machine

This section contains the results for the tests run on the restricted Boltzmann machine. The first part involves the tuning of hyperparameters by trying out different combinations of parameters, while the second makes use of the more suitable parameters to then sample the RBM.

5.1.1 Hyperparameter Tuning

As stated in the previous section, the main metric used to evaluate the suitability of the hyperparameters is the reconstruction error. Due to the computational power available for the project, a balance between the number of hidden features and number of examples in the training dataset was required. Based on the rule of thumb of using a number of hidden units at most one order of magnitude less than the number of input units, the number of hidden units required was between 1000-10000. The exact combinations tested are shown in Table 5.1:

Table 5.1 - Hyperparameters tested for RBM

Hyperparameter	Values
Number of hidden units	2560, 5120, 6400
Steps in contrastive divergence (k)	1,5,10
Minibatch size	10,32,64
Learning rate	0.001,0.01,0.1

The figures below shows the final reconstruction error for different parameter combinations. For a given number of sampling steps (k) and batch size, the different learning rates are plotted. The figures shown are for a fixed number of features:

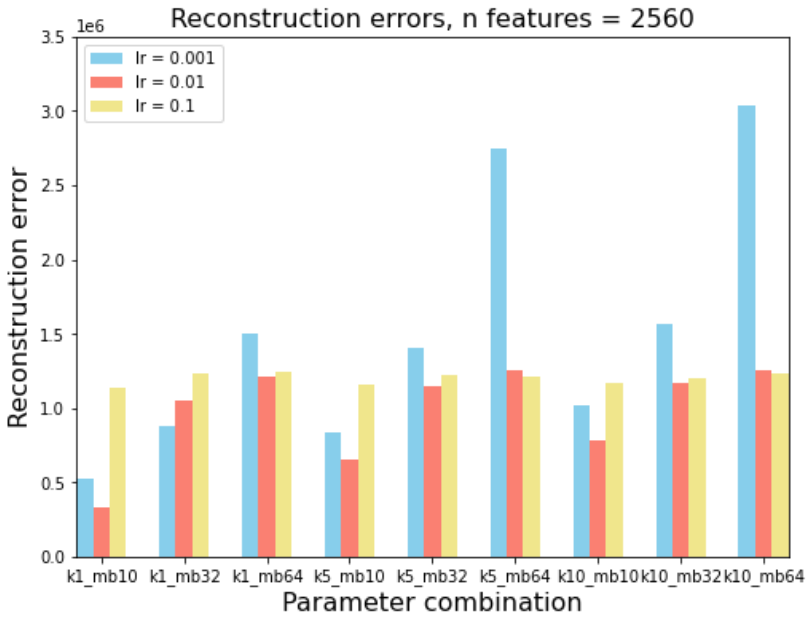

Figure 5.1 – Final reconstruction errors for 2560 hidden units

Figure 5.1 shows that all three of the non-fixed hyperparameters shown are dependent on each other. The learning rate that yields the lowest reconstruction error depends on the minibatch size. For small

minibatches of 10 and 32, a learning rate of 0.1 gives the largest error. However, on minibatches of 64, the opposite effect is obtained, where the smallest learning rate gives the highest error. The learning rate of 0.01 seems to be the most consistent, and often gives a very similar reconstruction error as the learning rate of 0.1, but still performs well on the larger batches. In fact, the learning rate of 0.01 gives the lowest error in all cases except for a single contrastive divergence step and a minibatch size of 32. The number of steps of contrastive divergence has no significant effect on the reconstruction error if a learning rate of 0.001 or 0.01 is used. For the smaller learning rate, more steps gives a higher error. This could be due to the fact that for multiple steps, there is actually a divergence and an increase in error with successive steps, because the learning rate is too small to learn the gradients. However, it is worth noting that increasing the steps in contrastive divergence generally has the effect of improving the quality of samples. This highlights the fact that while the reconstruction error can be used to detect issues with the training, it does not give a direct measure of the quality of the samples. Although for lower number of sampling steps the error was slightly lower, the actual samples themselves may have less variety than for higher values, which would be an undesired property of the model. Similar patterns are observed for the other values of number of features/hidden units:

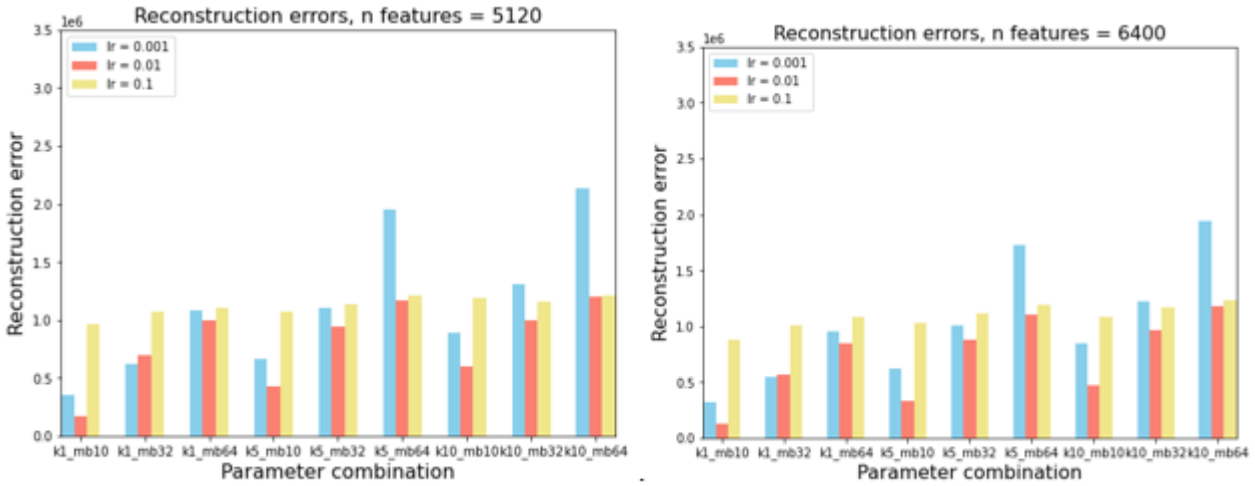

Figure 5.2 - Final reconstruction errors for 5120 (left) and 6400 (right) hidden units

The general trend in all cases is that the reconstruction error increases as the batch size increases except for the learning rate of 0.1, and increasing the number of steps in contrastive divergence also increases the error. The learning rate of 0.01 mostly gives the lowest error, and comparing the different figures for the same settings, it is evident that increasing the number of features/hidden units significantly reduces the error. This is shown more clearly in the figure below:

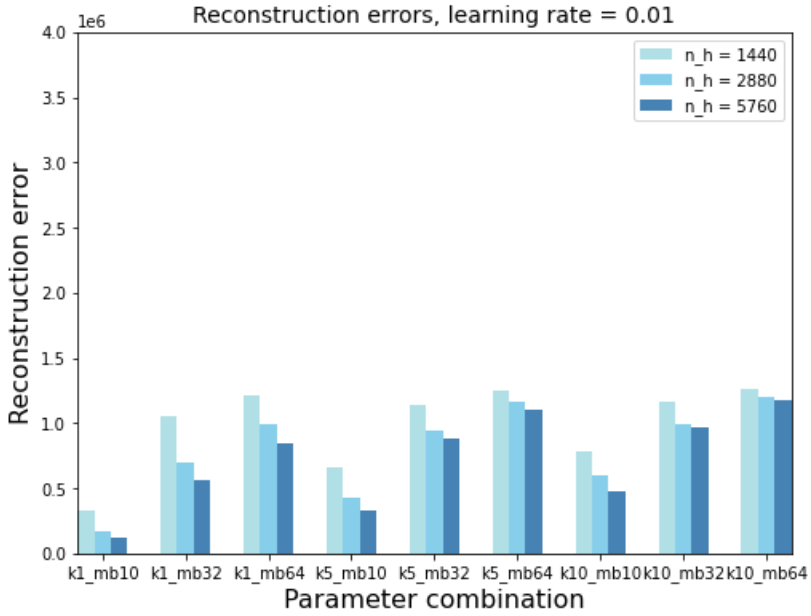

Figure 5.3 - Final reconstruction errors for different numbers of hidden units and parameter combinations

While these graphs so far consider the final error only, it is interesting to plot the reconstruction error per epoch to observe how the learning process took place. Figure 5.4 shows the reconstruction errors for a fixed number of hidden units of 6400 and a single step of contrastive divergence.

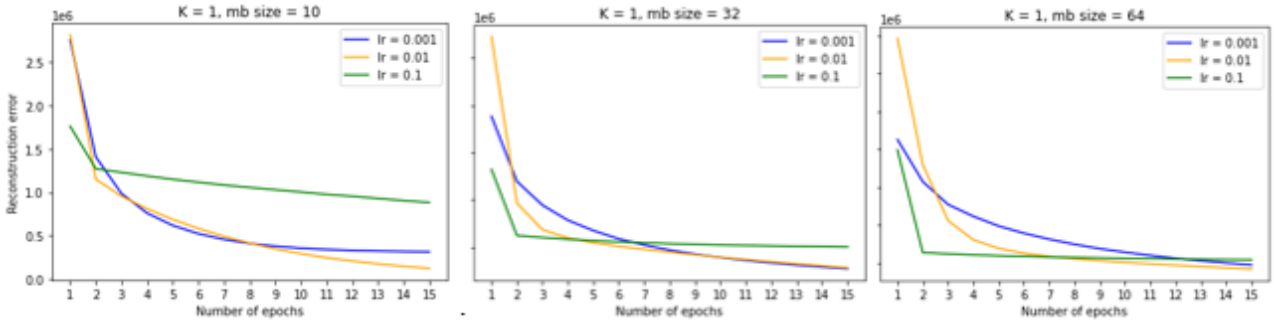

Figure 5.4 - Plots of reconstruction errors per number of epochs for different hyperparameter combinations

For a batch size of 10, the curve of the smallest learning rate follows the expected profile for the reconstruction error in an RBM at each epoch. The expected result is a curve which rapidly decreases and plateaus after a few epochs. For the learning rate of 0.1, a batch size of 10 might be too small as the reconstruction error decrease is a combination of two linear segments. For the larger learning rate, a value is reached at the second epoch, after which it decreases very gradually in a linear way. The learning rate might be too large, meaning that after reaching a value for the reconstruction error upon the first weight update, the result is that the weights become fixed at their value, and cannot improve further. The same trend is observed for different values of k and different numbers of hidden units. For these reasons, the final hyperparameters chosen in order to sample the RBM are a learning rate of 0.01, on a batch size of 32. While using a batch size of 10 reduces the overall error, the learning process appears to be smoother at a batch size of 32, and therefore is the one used. Additionally, since this did not yield a significantly larger error, 5 steps of contrastive divergence are used, based on the theoretical notion that increasing the number of steps allows to learn the data distribution more accurately. The number of hidden units chosen is 6400.

5.1.2 Generated samples from the restricted Boltzmann machine

The technique for sampling from the RBM involves two main strategies. As the aim of the project is to generate a polyphonic music piece from scratch, the ideal method for sampling would not require the user to input a pre-existing melody or piece, partial or otherwise, to obtain a new sample. Hence, a random binary noise vector of the dimensions of the input image is used, with a number of "on" units corresponding roughly to the average number of notes in each image of the training data. The second method involves using the trained restricted Boltzmann machine to reconstruct a partial image. The partial images used are obtained from the GMP dataset, although are not part of the training set of 2000 images used to train the model. A common application for RBMs is to reconstruct partially obstructed images, as the input sample's active pixels or units can be clamped to the observation, with the aim of activating the relevant hidden units or features for reconstruction. [69] Thus, the generated image will contain the original obstructed image in addition to the reconstructed part.

Figure 5.5 shows an example of an image from the dataset and an image sampled from the RBM using the random noise vector as an input, and the reconstruction.

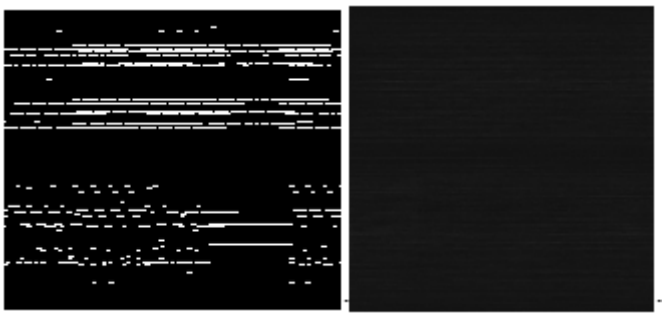

Figure 5.5 - Example of a training data (left) and randomly sampled (right) image

The image is almost black, meaning the majority of pixels were assigned a very low probability. In order to transform the generated images into musical pieces, a thresholding is required. The exact value was chosen by obtaining the 90^{th} percentile of pixel values and activating them, while deactivating the rest. The threshold was chosen to be high in order to include only those pixels which received a relatively high probability of being activated. Additionally, the image is reshaped into the original dimensions of the piano roll of 128x320.

Figure 5.6 - Randomly sampled image from RBM after threshold and reshaping

From visual inspection, it is already evident that the image now resembles the original input data. However, it can be already seen on first glance that there seems to be less structure than the training data, with a higher presence of sparse single-timesteps notes activated, corresponding to those units which had been activated and passed the threshold. Furthermore, even though random noise vectors

are used for sampling, the resulting generations are all almost identical. This suggests that the RBM did not learn a varied distribution. For many different inputs, it maps the visible units to the same reconstruction. In order to obtain some varied results, an input image into the model is required for reconstruction. For the second sampling technique, an example of an obstructed image and the corresponding reconstruction are shown in Figure 5.7:

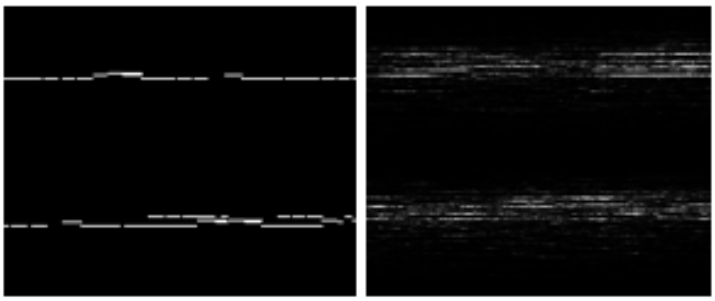

Figure 5.7 - Obstructed image (left) used for sampling RBM and reconstructed image (right)

Compared with the random sample, it is already evident that even the raw output is more defined. The obstructed image, in this case a simple monophonic melody consisting of approximately the same note throughout, results in an image which follows the input's structure but has more variety, and compared with the random image, it has brighter pixel values meaning that the visible probabilities were in general higher. The reshaped image with a threshold applied to remove noise is shown in Figure 5.8:

Figure 5.8 - Reconstructed image after threshold and reshaping

Compared with the random sample, it appears to be less noisy. For a section of the image, it has the same structure as the input image without any reconstruction, but aside from that the image is evidently different compared to the input. The evaluation of the generated samples will be performed with the final evaluation of different models.

5.2 Generative Adversarial Networks

5.2.1 Experiment 1 – Dataset sizes

Given the computational demand of running the model with different hyperparameters, both in terms of the available memory and GPU use time allowed, the first experiment involves determining an acceptable size for the training dataset. By reducing the dataset size, more experiments can be run on different settings, as well as in less time. However, by using less examples, there is the risk of the model not learning an acceptable variety of different features, or possibly not being able to learn

relevant features. Therefore, the model was run on three different training dataset sizes, of 1000, 9850, and 29550 examples (referred to as the small, medium and large datasets respectively). The small and medium datasets are th**e first 16 seconds of the song,** while for the large dataset, multiple 16-second segments were extracted from the same song. The result is that some examples in the large dataset may be very similar to each other which might introduce some bias, but the number of examples is large enough that overall, it is expected to obtain more variance.

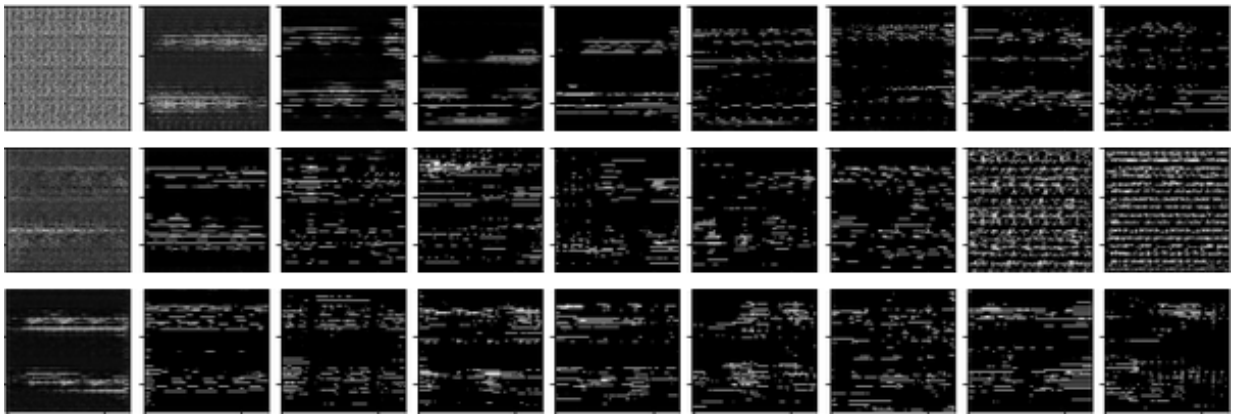

Figure 5.9 - Generated images from the training process for the small (top), medium (middle) and large (bottom) datasets

Figure 5.9 shows the generation of images at different epochs for the three different sizes of datasets. It can be seen that the small dataset takes comparatively a longer time in order to learn a distribution which is similar to the input images, as the image is at first very blurry. There is also seemingly less variety in the successive images until the very end. For the other datasets, there is immediately more variety in terms of how the notes are distributed in the image. For the medium dataset, the last images begin to become very noisy to the point of resembling static. This indicates that the learning process eventually failed, and the generator eventually not being able to keep up with the discriminator's classifying ability. The plots for the generator and discriminator losses of the three datasets are shown in Figure 5.10:

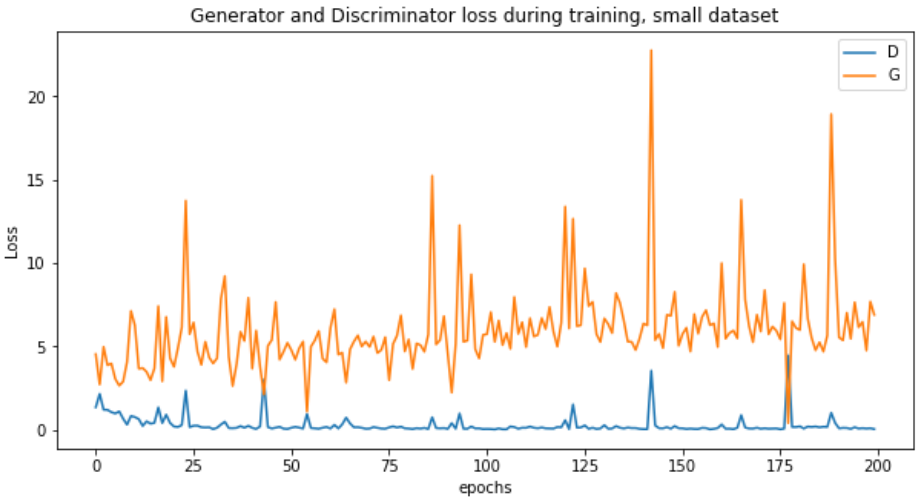

Figure 5.10 - Generator and discriminator loss for small dataset

For the small dataset, convergence is achieved throughout the training process. The discriminator loss initially drops rapidly and stabilizes at an approximately constant value. The generator loss experiences a more significant fluctuation, but the average remains fixed at approximately the same

value throughout. There is a slight trend towards the loss increasing, suggesting that the models could diverge given more epochs.

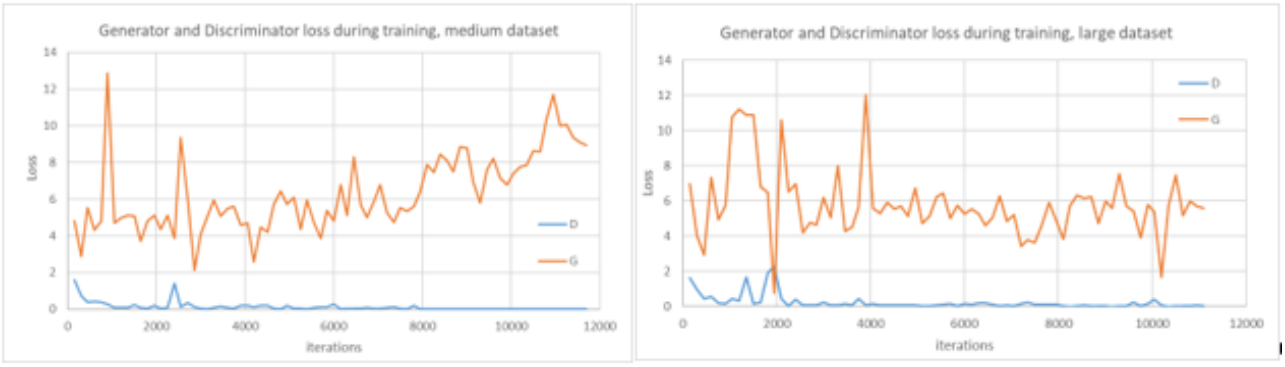

Figure 5.11- Losses during training for medium (left) and large (right) datasets

For the medium and large datasets, the process of the generator is slightly different. In both cases, the range and frequency of the fluctuations is less than for the small dataset. This could be due to the fact that the larger amount of training data allows the generator to learn the overall probability distribution more easily, being able to generalize, leading to a more stable training. However, for the medium dataset, it is interesting to note that towards the end of training, the generator loss steadily increases while the discriminator loss approaches zero. At this stage, the discriminator can identify all the fake examples which means that the generator cannot learn further and begins to output noisy images. This can be seen more clearly by observing the plot of the discriminator loss only:

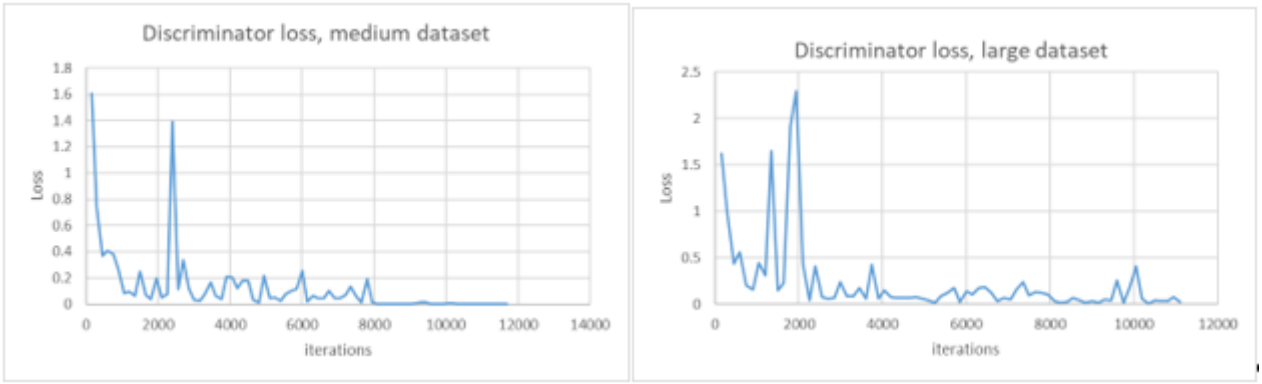

Figure 5.12 - Discriminator loss for medium (left) and large (right) datasets

Compared with the discriminator loss for the large dataset, which tends to zero but shows fluctuation and occasional spikes, the discriminator loss for the medium dataset eventually reaches a stable value of zero. This corresponds to the point at which the images become noisier. For the small dataset, the discriminator loss is shown in Figure 5.13:

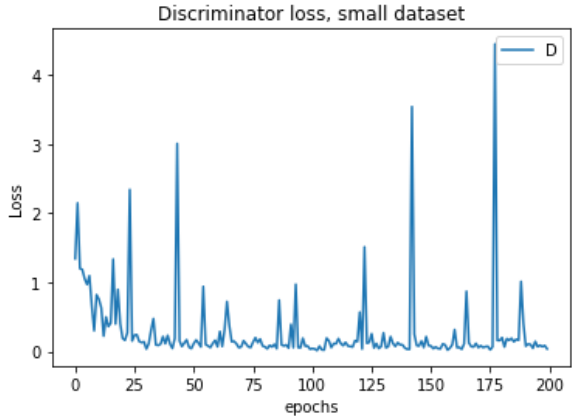

Figure 5.13 - Discriminator loss for small dataset

The value decreases towards zero but has frequent and significant spikes. The spikes suggest that throughout the entire learning process, the generator is able to create samples which the discriminator mistakes as real. However, this could be due to the fact that the discriminator cannot learn as well as the other models what the probability distribution is and is therefore unable to accurately classify samples. To investigate if the musical pieces generated using the small data are acceptable, some evaluation metrics need to be applied.

Table 5.2 - Results of feature extraction on generated samples

Feature	Score for generated images		
	small	medium	large
Empty ratio	0.21	0.09	0.06
Occupation rate	2.08	3.15	3.77
Polyphonic rate	0.71	0.82	0.88
Qualified Notes	0.49	0.54	0.58
Qualified Rhythms	0.54	0.55	0.53
Mean note duration	5.41	6.39	6.46
Pitch range	44.90	55.22	59.90
Pearson correlation with key profile	0.66	0.72	0.73
Average information rate	2.32	2.33	2.32

Table 5.2 shows the results for the features of the generated images using the small, medium and large datasets. As expected, the small dataset exhibits more features that are closer to those of a random, unmusical image. The value for qualified notes is lower, meaning that it is more fragmented, also supported by the lower mean note durations. The occupation rate is lower, meaning that it is more sparse, and the pitch range is much lower meaning that it learned less variety across the vertical axis. In addition, the Pearson correlation with the key profiles is significantly lower, indicating that the result is less musical. Interestingly, the features for the medium and large datasets are very similar for each feature. This means that both the large and medium exhibit many of the same characteristics, and whether using the larger dataset produces more interesting compositions could be determined only via a listening test.

In terms of the probability distributions learned, the medium and large datasets learn similar distributions when compared with the small dataset. An interesting result is that the small dataset

seems to exhibit more variance in terms of the feature distances, suggesting that they are more spread out.

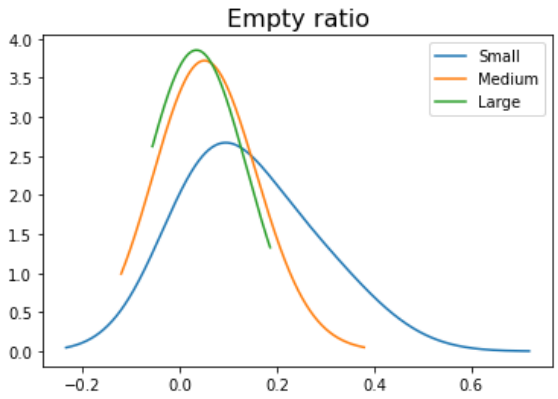

Figure 5.14 - PDFs, empty ratio

For the empty ratio (the ratio of empty to non-empty timesteps), for example, the shifted curve for the small dataset indicates a higher mean, and the broadness of the peak shows the variance. For the empty ratio, the distributions for the medium and large datasets are very similar. This was found to be the case for most features. Regarding the Pearson correlation with the key profiles, all datasets show a similar distribution, though the large dataset tends to obtain a higher correlation. This suggests that by using many examples, the model is able to imitate training examples more accurately, as the Pearson correlation is a direct indicator of the "musicality" of the generated content. However, the small and especially medium datasets obtain a similar correlation, meaning that it is still possible to generate new samples which are more or less correctly on key using much smaller datasets.

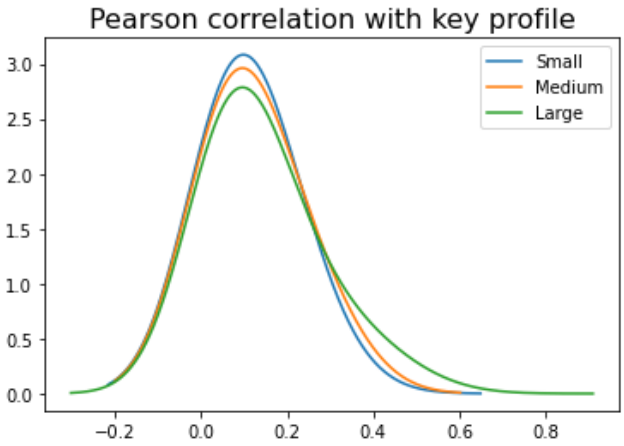

Figure 5.15 - PDFs, Pearson correlation

5.2.2 Experiment 2 – Hyperparameter Tuning

The image dimensions chosen were 160x160. This allows for 2 tracks with a full span of 80 pitches. To account of the different dimensions, a different kernel size is used in one of the layers in both the generator and discriminator. A total of 18 different hyperparameter combinations were tested. These are summarized in Table 5.3:

Table 5.3 - Hyperparameter combinations for DCGAN

Hyperparameter	Values
Batch sizes	32, 64
Learning rates (generator, discriminator)	(0.0002, 0.0002), (0.0005, 0.0002), (0.0002, 0.0001)
Number of features (generator, discriminator)	(64,64), (128, 64) (160, 40)

Both the learning rates and the number of features in the generator and discriminator have an important effect in the minimax game of the two networks. The values chosen are such to ensure that the generator can learn effectively to pass examples which trick the discriminator. As the initial images generated by the generator are very noisy, as it has not yet seen many examples, a scenario in which the discriminator instantly reaches a point of classifying the fake and real examples perfectly can easily be achieved, and in this case the generator cannot learn further. This was also taken into consideration when choosing the number of features in the network. By giving the generator more features, it can improve its learning process with respect to the discriminator, meaning the discriminator has a harder time distinguishing between the real and fake examples.

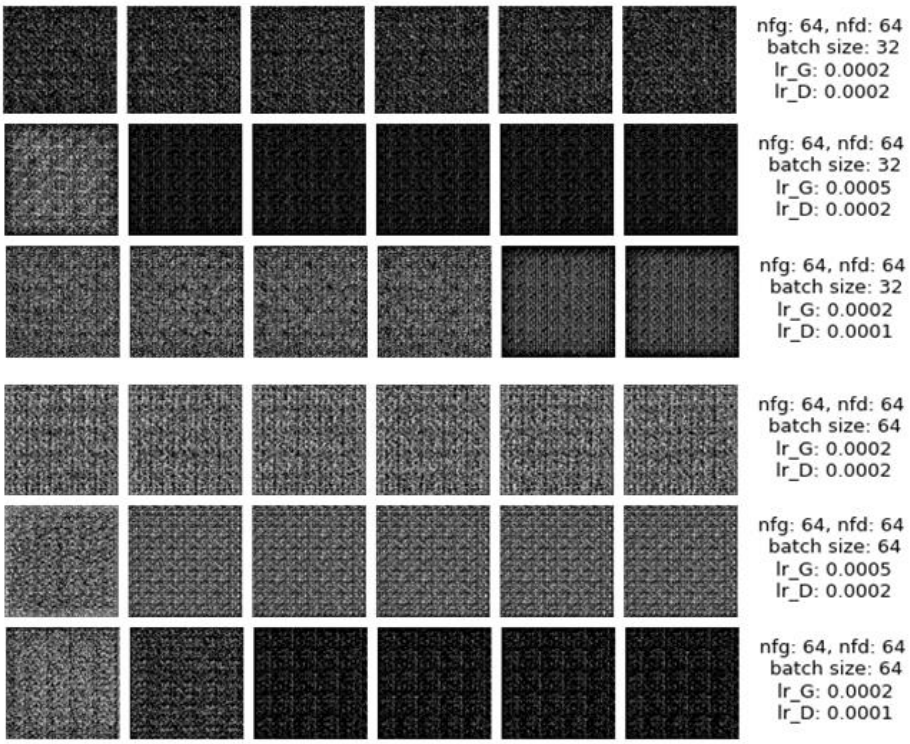

Figure 5.16 - Generated images, epochs 0 – 50

Figure 5.16 shows generated images every 10 epochs for a feature maps size of 64 for both the generator and discriminator. As shown, for both groups of batch sizes, the generated images are noisy in all cases. In most of the hyperparameter combinations, the loss for the discriminator went instantly to 0, meaning that the generator loss kept increasing and could therefore not learn any distribution. An example for a batch size of 32 and equal learning rates for the generator and discriminator is shown in Figure 5.17:

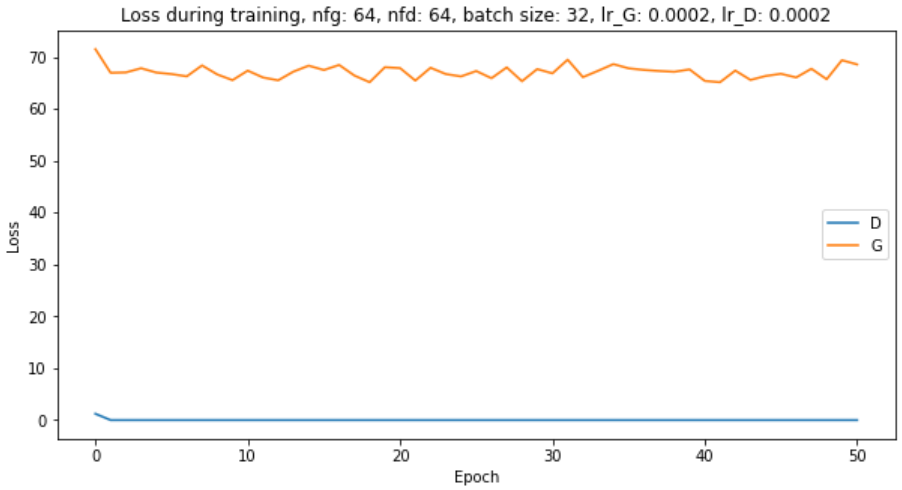

Figure 5.17 - Loss during training

As shown, the difference between the losses is very large, and most importantly, the loss of the discriminator remains fixed at zero. Reducing the learning rate of the discriminator does not lead to improvements in results. Figure 5.18shows the effect of reducing the learning rate for the discriminator optimizer, and as seen, there is no significant difference. The only exception is that there is a sharp increase in the generator loss.

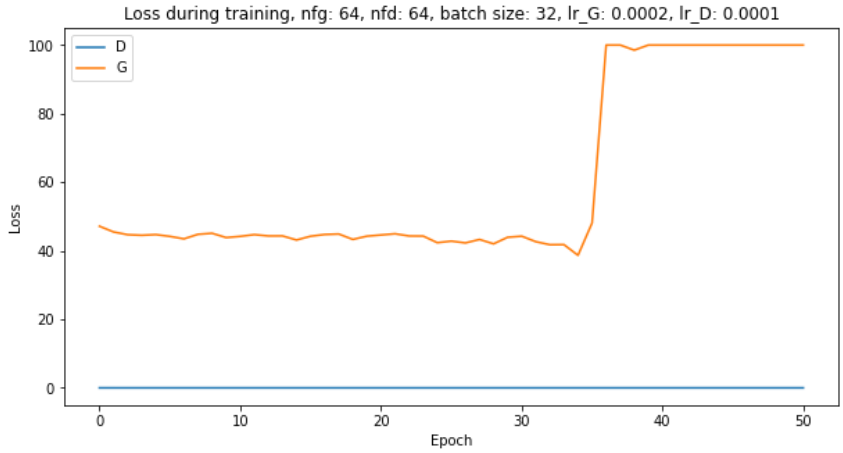

Figure 5.18 - Loss during training

Increasing the learning rate of the generator also does not have a desirable effect, and in fact this leads to mode collapse. The plot of the errors for an increased generator learning rate is shown in Figure 5.19:

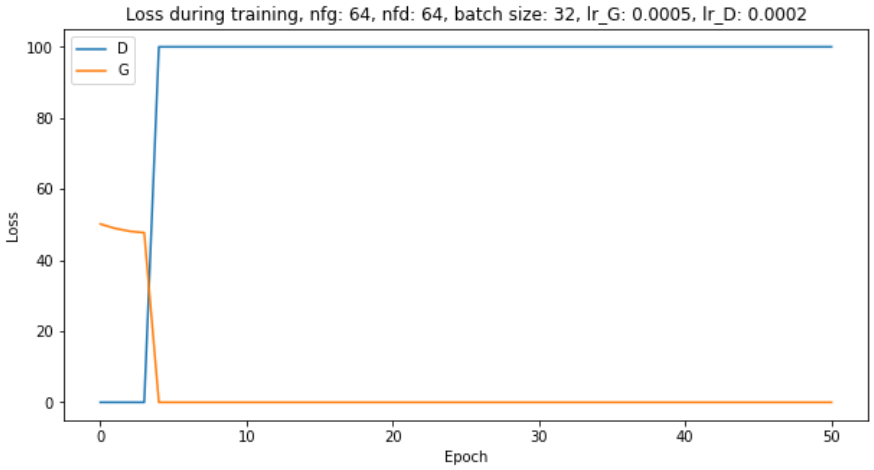

Figure 5.19 - Loss during training

Here, the generator loss quickly reduces to a stable value of 0, while the discriminator loss is also fixed at the maximum value. In this case, the learning rate for the generator is too high – it was able to quickly learn an acceptable image, and ends up replicating the same image constantly, to the point where the discriminator cannot learn further to classify the different examples. While the images produced in mode collapse seem like random noisy, the mode collapse is evident when comparing the images generated at epoch 20 and 50:

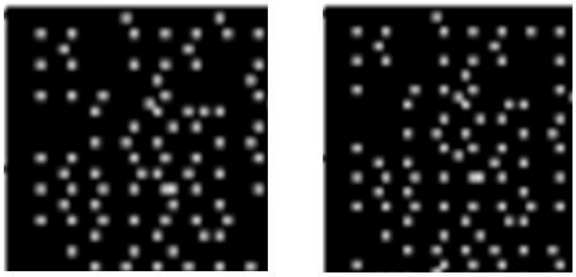

Figure 5.20 - Details of images generated after 20 (left) and 50 (right) epochs

Observing two details of the top-left corner of the images generated for epochs 20 and 50 in Figure 5.20, it is evident that the network is outputting very similar images, which the discriminator labels as real and thus does not improve, meaning the generator can keep outputting the same image.

Changing the number of features for the generator and discriminator to be 128 and 32 respectively yielded much better results. The images generated every 10 epochs are shown in Figure 5.21:

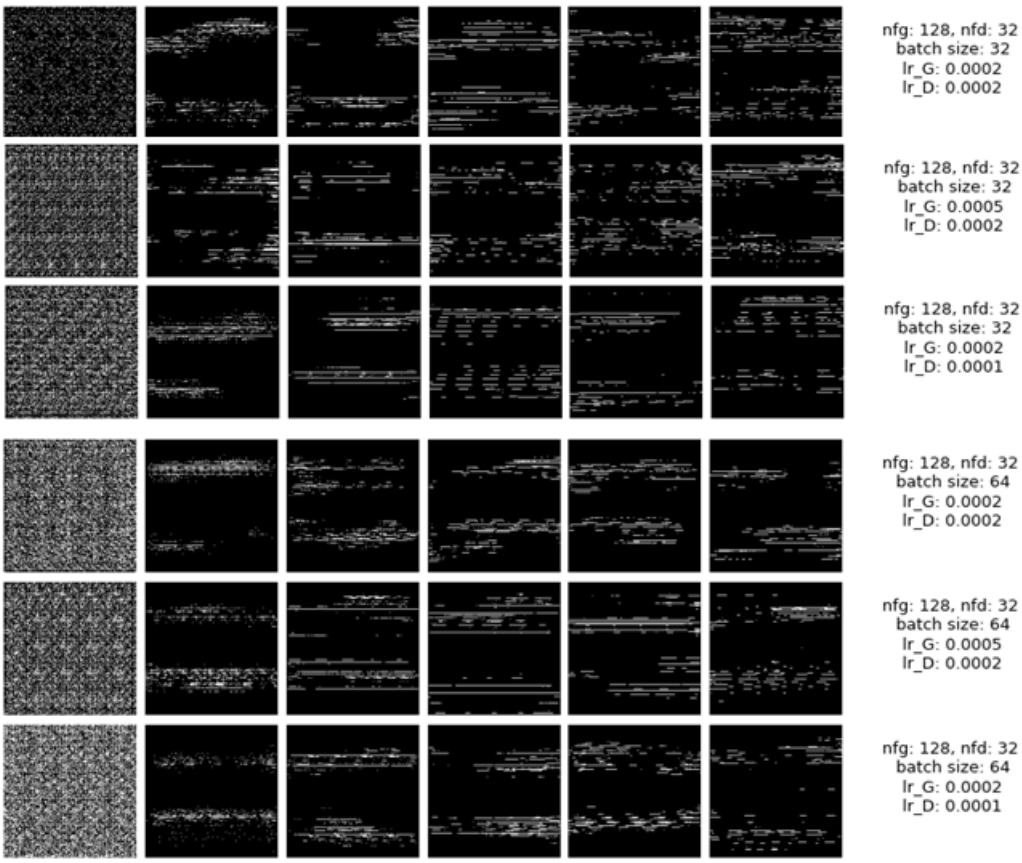

Figure 5.21- Images generated every 10 epochs for different hyperparameter settings

In this case, every combination of hyperparameters yielded images which with a visual inspection, seemed to resemble the training data. You can see two distinct tracks, mostly with spacings that would suggest the generation of chords or melodically coherent melodies. With the equal values of learning rates, as suggested in the original GAN paper, the losses obtained are shown in Figure 5.22:

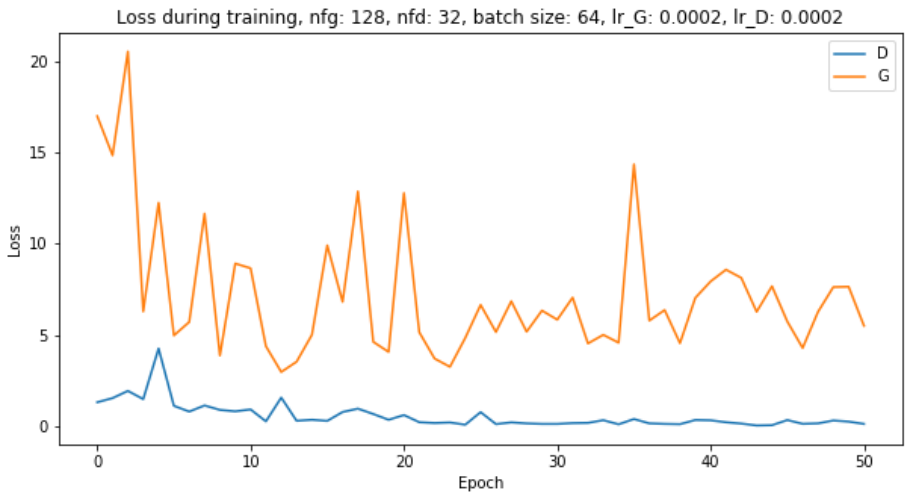

Figure 5.22 - Generator and discriminator loss, default hyperparameters

The generator loss oscillates at a value which is significantly lower than the previous plots shown, and the discriminator shows more oscillations and while it tends towards zero, it does not actually reach zero so the generator can learn throughout the whole training process. Decreasing the learning rate of the discriminator results in a slower decline of the discriminator loss, suggesting that it adapts

slower and the generator is able to deceive the discriminator better. This also could explain the more frequent oscillations of the generator loss, and the fact that the overall value for the loss is lower.

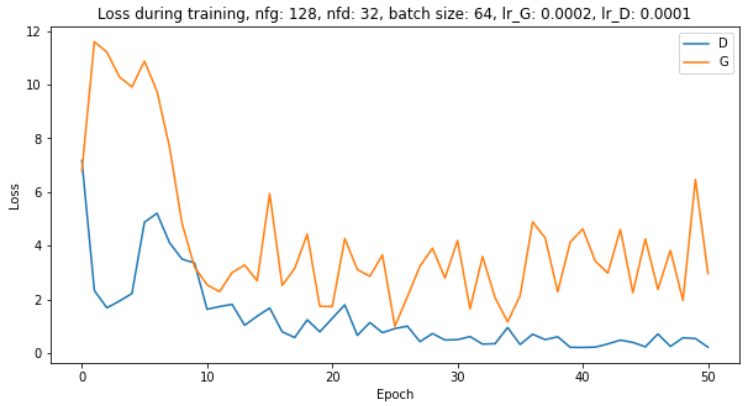

Figure 5.23 - Discriminator and generator loss, reduced learning rate for discriminator

Similarly, increasing the learning rate of the generator results in the difference between the losses being reduced:

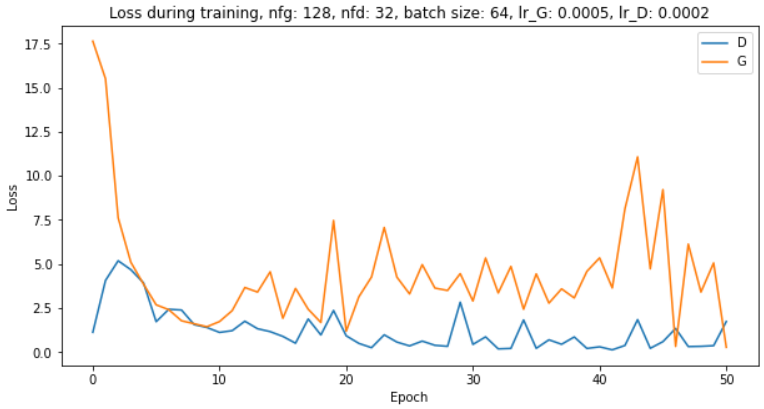

Figure 5.24 - Discriminator and generator loss, increased learning rate for generator

In this case, the value for the discriminator loss seems to be steadier compared with the previous plot, in which a slow decline is observed, while in this case, the average loss seems to be fixed, with spikes throughout the learning process. Towards the end, it is interesting to note that the generator loss is approaching zero. This could indicate that the generator can learn too quickly and given more epochs, mode collapse would occur. For the smaller batch size, the results were very similar. Regarding the last feature maps setting, of 160 and 40 for the generator and discriminator respectively, the training was overall more unstable compared with 128 and 32. For some settings of the learning rate and batch sizes, some good results were obtained, however for most of the settings the images obtained were noisy.

To select the final configuration of parameters, an indication of the exact values for the discriminator error is useful. Ideally, this should be around 0.5, meaning that it it has a good discriminative ability but still makes some mistakes, allowing the generator to learn. Observing the error plots for the second half of training, it is unclear which setting is better – while the line for the discriminator learning rate of 0.0001 is approaching a value of zero, suggesting mode collapse as mentioned previously, it could still be simply a fluctiation in value.

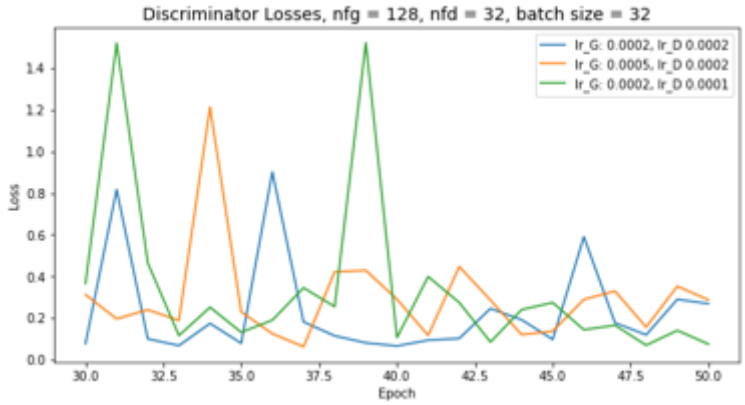

Figure 5.25 - Discriminator losses during training

For a batch size of 64, the losses are comparable:

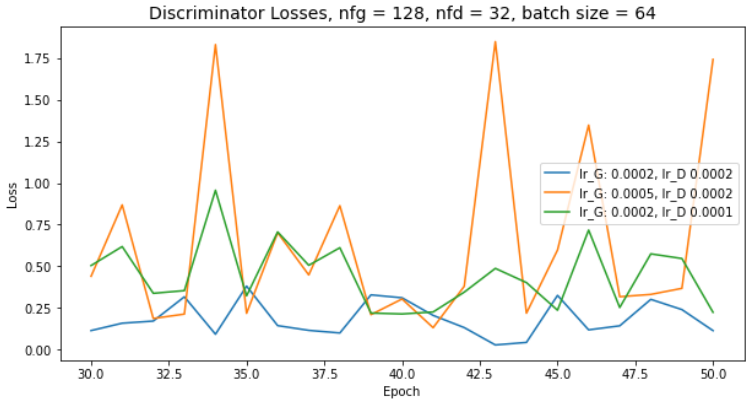

Figure 5.26 - Discriminator losses during training

Interestingly, for the feature maps of 160 and 40, the range of results was wider:

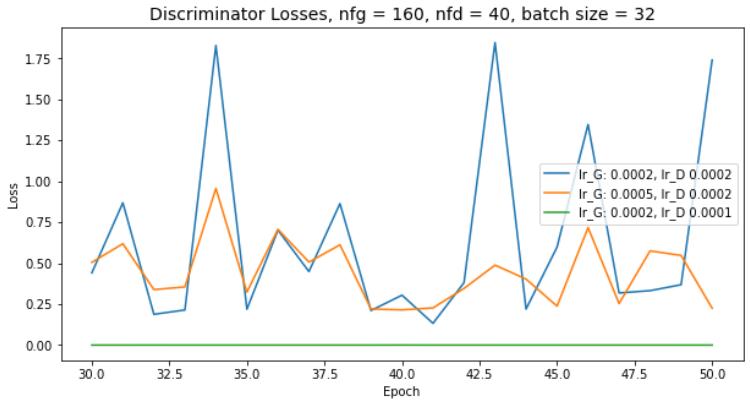

Figure 5.27 - Discriminator losses during training

When the learning rate of the discriminator was reduced, the network could not learn any distribution, However, for the case where the learning rate of the generator was increased, this led to a stable fluctuation between 0.20 and 0.50, which is a good value for the loss, and better than those for the other feature map dimensions.

As no significant improvements were found by altering the hyperparameters, the final configuration chosen is the one suggested by the initial DCGAN paper – a learning rate of 0.002 for both generators

and using the feature map sizes of 128 and 32. However, the batch size is changed to 32 as it seemed to give slightly higher loss in the discriminator.

5.2.3 Experiment 3 – WGAN-GP

For the WGAN-GP, several different combinations of parameters and image sizes were attempted. However, these failed to converge. In addition, the training times for WGAN-GP and the memory requirements were such that not many tests were attempted fully. However, one test using the same image size as with the DCGAN, and the same dataset size of 1000 examples was carried out, in order to directly compare the performance of the two models and see whether using the WGAN GP gives better results. The WGAN GP was trained for 200 epochs.

Figure 5.28 shows the discriminator loss, with a smoothed curve superimposed to show the trend of the loss values with each training steps. As shown the general trend is a decrease. The loss is quite noisy, though the steady decrease indicates that the training is stable. For the WGAN, the loss value is negative, so for visualization the modulus of the loss was taken. In an ideal training scenario, the loss should initially be high and gradually decrease towards 0, without actually reaching the value. While the loss stabilizes quickly, it still has a downward trend, indicating that more training iterations would have improved results further. This can also be seen in the change in the Wasserstein distance, which is also expected to decrease, as the objective of the WGAN is to minimize the Wasserstein distance between the real images and the generated images.

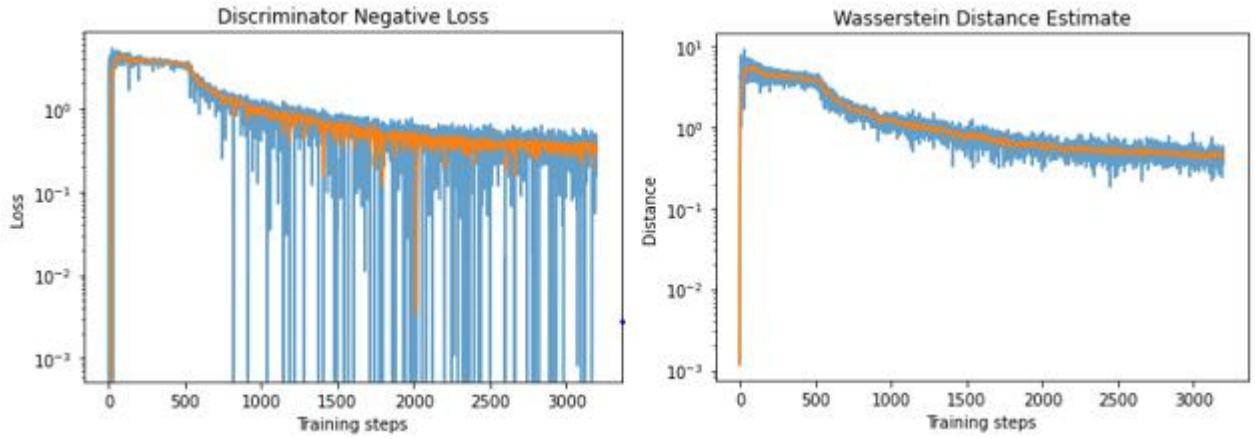

Figure 5.28 - Discriminator loss (left) and Wasserstein Distance (right)

On the other hand, the generator loss tends to increase:

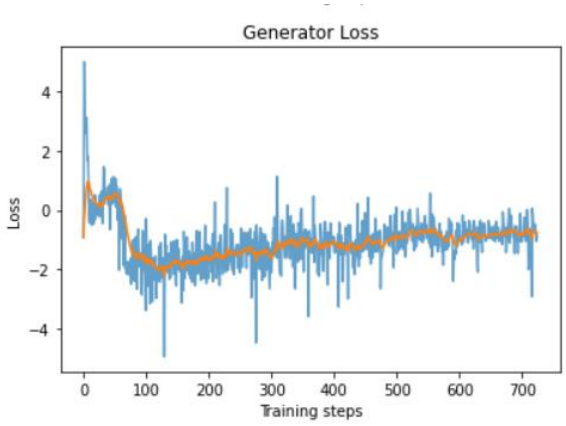

Figure 5.29 - Generator loss

An expected result for stable training is an initial rapid drop in the generator loss, which then should stabilize around zero. While the loss is increasing, it is still below zero, meaning that after more iterations, the value could either stabilize or keep increasing. The quality of the generated images also seems acceptable by observing the generations for every 20 epochs:

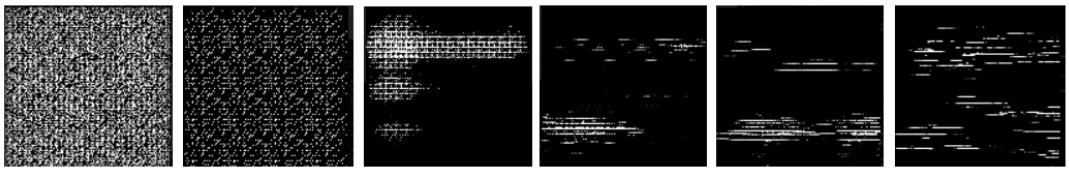

Figure 5.30 - images generated by WGAN-GP every 20 epochs

One thing to note is that the WGAN apparently takes more time to achieve a good quality image. The last image shown, obtained after 100 epochs, is the only one that fully resembles the input dataset, while after 40 epochs the image is still very noisy. In contrast, the DCGAN generated at least vaguely acceptable images after just 5-10 epochs. However, while this visual assessment can give an indication of how the learning process takes place in both types of GAN, the two results must be compared. Therefore, the evaluation metrics will be applied.

Table 5.4 - Extracted features for training dataset, DCGAN (middle) and WGAN-GP (right) generated images

Feature	Score for generated images		
	GMP	DCGAN	WGAN-GP
Empty ratio	0.08	0.21	0.23
Occupation rate	4.04	2.08	1.82
Polyphonic rate	0.87	0.71	0.67
Qualified Notes	0.67	0.49	0.42
Qualified Rhythms	0.45	0.54	0.63
Mean note duration	10.87	5.41	5.35
Pitch range	42.22	44.90	48.68
Pearson correlation with key profile	0.81	0.66	0.68
Average information rate	2.35	2.32	2.27

The results show that the generated samples from the WGAN-GP are more sparse compared with those of the DCGAN (Table 5.4). This is seen comparing the empty ratio, occupation rate and

polyphonic rates. However, the average values for each feature tend to be more similar between the two sets of generated images than with the original data, suggesting that the learned distributions in each of the models have similar properties. It is difficult to ascertain which of the models generates better samples based on the features alone. The Pearson correlation was slightly higher for the WGAN-GP, suggesting that the WGAN learns the vertical distribution of the notes better, and can recreate the spacings and proportions of each pitch class more accurately than the DCGAN. The information rate was lower for the WGAN however, meaning that overall, the generated compositions of the WGAN have less structure.

The final estimation of which model can approximate best the distribution of the training data can be made using the Kullback-Liebler divergence of the intra-set distances of each of the features.

Table 5.5 - KL divergence with training data features intra-set distances

Feature	KL divergence	
	DCGAN	WGAN
Empty ratio	0.03	0.03
Occupation rate	-	-
Polyphonic rate	0.21	0.44
Qualified Notes	0.90	3.23
Qualified Rhythms	0.16	0.10
Mean note duration	-	-
Pitch range	0.25	1.07
Pearson correlation with key profile	0.10	0.03
Average information rate	0.03	0.01

Table 5.5 shows the KL divergence values obtained for each feature for the two models. The KL divergence is calculated using the original dataset's intraset distances for each feature as the reference distribution. The idea behind using the KL divergence is that, if the general distribution of the training data is modelled exactly, the resulting samples should only show a very small divergence, as the probability density functions for the feature distances should be approximately the same. However, it can be seen from the table that each model performs better than the other on certain features. For the DCGAN, it achieves lower KL divergences in features such as the empty ratio, the polyphonic rate and the qualified notes. This suggests that the DCGAN can more accurately recreate the rhythmic qualities of the training data. With regard to higher level musical structure, however, the WGAN is able to capture the distribution better, obtaining a significantly lower divergence for the Pearson correlation and information rate. It is worth mentioning that comparing the two Tables, it can be seen that the fact that the mean value is more similar does not necessarily indicate that the distribution was recreated accurately. While the mean values can give information on the quality itself, it does not necessarily mean that the distribution was learned properly.

5.2.4 Experiment 4 – DCGAN transposed and RGB datasets

The final experiment involves collecting samples and evaluating these using the measures described previously. In particular, this section of the evaluation aims to estimate whether the generated samples have learned the distribution of the data, using the musically-related features that describe characteristics of structure, rhythm and melody. Three different configurations of the DCGAN are evaluated, as well as the reconstructed samples from the RBM described previously. The configurations for the GAN are:

1) 160x160 images, A minor/C major dataset (1335 training samples, DCGAN-1)
2) 160x160 images, all keys dataset (2000 training samples, DCGAN-2)
3) 128x128 images, RGB , A minor/C major dataset (1335 training samples, DCGAN-3)

These, along with the generated samples from the RBM, are compared with the overall distribution of the GMP dataset. To make the results more comparable, a random sample of 1500 examples was extracted from the GMP dataset for calculations of features and probability density functions. In each case, 100 generated samples were created, for which the features are then extracted. For each, the intra-set distances of features is calculated, and the inter-set distances with the training data. By calculating the Kullback-Liebler divergence between the intra-set and inter-set distances, a measure of how well the distribution of the data was modelled will be obtained. Table 5.6 shows the average values for each feature:

Table 5.6 - Extracted features for different generative models

	GMP	DCGAN-1	DCGAN -2	DCGAN -3	RBM
Empty ratio	0.11	0.10	0.10	0.08	0.11
Occupation rate	3.42	2.84	3.36	2.86	3.76
Polyphonic rate	0.86	0.80	0.82	0.81	0.79
Qualified Notes	0.63	0.47	0.57	0.49	0.46
Qualified Rhythms	0.44	0.55	0.53	0.59	0.52
Mean note duration	8.85	5.80	7.35	5.31	7.88
Pitch range	46.04	59.35	60.49	54.49	33.07
Pearson correlation with key profile	0.81	0.83	0.76	0.85	0.76
Average information rate	2.53	2.44	2.52	2.34	1.91

All models are able to learn the approximate ratio of empty to non empty time steps. The rhythmic characteristics are similar for all models. Observing the qualified notes, meaning the ratio of notes of at least 3 time steps, none of the models are able to obtain a value as high as the dataset. This suggests that all the models create songs which are more fragmented or noisy. The RBM in particular creates the most fragmented compositions. However, the mean note duration for the RBM was the highest, which contradicts the idea that its compositions are more fragmented. However, this is due to the reconstruction technique in sampling the RBM. The input obstructed images contain a series of long notes which last many timesteps; these are incorporated into the final song generated by the RBM, raising the mean note duration. The polyphonic rate is also similar in all models and comparable to the training dataset. The fact that this value is not too high suggests that the models do not generate an excess of pixels: there are still occasional timesteps in which a single melody is playing, which is a good result. This is confirmed by the fact that the occupation rate is similar to the training data – the occupation rate is the average proportion of notes playing at any timestep. A value too low suggests that the composition is not polyphonic, but if the value were much higher than the training

dataset, it would mean that the compositions are too cluttered, with too many melodies happening simultaneously, or chords with an excessive number of notes playing. Interestingly, the qualified rhythms value is higher than the training dataset for all of the models. The qualified rhythms indicate the proportion of notes which have a duration that fits within the standard beat divisions. The higher value in the generative models is explained by the fact that while the training dataset has notes of a wide range of durations, the ones which classify as a qualified rhythm occur overall more frequently than the non-qualified note durations. Therefore, the networks observe overall a high proportion of notes to fit within the qualified rhythms length and end up outputting more than in the original training data. For the pitch range, the DCGAN tends to overestimate the average pitch range of each composition, and each model gave a wider range. This could be in part due to the fact that the images are somewhat noisy. While the vast majority of notes may occur within a certain range, even one noisy pixel at a very high or low pitch is sufficient to increase the pitch range substantially. For the RBM, the pitch range was very limited. This is because using the reconstructions, only the pixels in closer proximity to the input obstructed image are activated, and those further away do not.

Regarding the Pearson correlation, all models except for DCGAN-2 used a training dataset in a single key. The result is evident: using data from a single key leads to a higher Pearson correlation with the song's key, meaning that the distribution of pitches is learned more accurately. In fact, the DCGAN-1 and DCGAN-3 obtain a Pearson correlation which is even higher than that of the training dataset, meaning that they produce less melodies which are off-scale compared with the training data. From a musical point of view, this is not necessarily a good thing, as occasional off-scale melodies or key changes can make a composition more interesting, but from the point of view of learning the distribution, a high value for the Pearson correlation is optimal. Furthermore, while a composer might deliberately include an off-scale section, it is difficult to do so and make it sound pleasant or appealing. For the generative models, going off-scale would most likely mean including a note or small melody which is completely off-scale and sound discordant. It is interesting to note however that the DCGAN-2, which used music from all keys, did not obtain a significantly lower score for the Pearson correlation. The resulting generated songs, while possibly in a range of different keys, still follow a coherent musical distribution, albeit less so than those trained on the dataset in A minor/C major. This points towards the fact that the DCGAN can efficiently learn translation-invariant patterns. Transposing a song into a different key involves raising all notes by the same number of semitones – that means that in all keys, the distances between each pixels on the vertical axis remain the same. Hence, the DCGAN-2 generates songs which maintain an appropriate distribution along the vertical axis, despite the fact that it sees examples from a variety of different keys. For the restricted Boltzmann machine, the Pearson correlation was lower than with the other models, or the training dataset, even though it was trained using the dataset in A minor/C major. This could be due to two reasons. Firstly, a decrease in the average Pearson correlation by 0.05 could be statistically insignificant. This could also explain why the other models obtain a Pearson correlation which is higher by a similar amount. Secondly, the RBM could have learned which pitch classes are the most likely to be activated, therefore creating songs which are in the key of A minor/C major, but without following the actual distribution in terms of the frequency of each pitch class. For example, it could underestimate the presence of a particular pitch class that doesn't occur too frequently in the scale, and by underestimating the presence, a lower Pearson correlation with the scale is obtained, despite the fact that the song is still in the correct scale. For the information rate, the DCGAN that made use of all keys was able to obtain a value closest to that of the training dataset. For the other versions of the model, this value was slightly lower. Again, this decrease could be statistically insignificant. Furthermore, the information rate gives a score on the self-similarity of the piece, meaning that a high score is obtained if there is a good balance between repeating chords and melodies, and variations.

Therefore, a low score could indicate that the generated song is either too repetitive or too random, and to determine which of the two it is requires the listening test. However, for the RBM, based on a visual inspection of the images and listening to the samples, the lower IR is most likely due to the fact that it contains certain notes, corresponding with the input obstructed image, that have a very long duration. This would cause the IR to decrease. The reduced pitch range, and the fact that it plays specific notes for a long duration and repeatedly, leads to the lower IR.

The intra-set distances can be used to understand how the features were distributed. Table 5.7 shows the mean values for the intra-set distances for each feature

Table 5.7 - Intra-set distances for each feature

	GMP	DCGAN-1	DCGAN-2	DCGAN-3	RBM
Empty ratio	0.12	0.06	0.07	0.05	0.21
Occupation rate	1.47	0.55	0.81	0.57	2.4
Polyphonic rate	0.13	0.07	0.08	0.07	0.27
Qualified Notes	0.34	0.07	0.12	0.08	0.1
Qualified Rhythms	0.11	0.06	0.06	0.06	0.06
Mean note duration	6.19	1.01	1.51	1.17	2.54
Pitch range	13.43	11.43	11.26	12.14	11.33
Pearson correlation with key profile	0.12	0.1	0.11	0.1	0.1
Average information rate	0.6	0.34	0.32	0.32	0.79

In this case, the mean value represents how varied the samples were. A low value for the mean suggests that for most of the samples, the features were the same. It can be seen that the training dataset has a higher mean value for the intra-set distance on almost every feature over the generative models, with the exception of the occupation rate for the RBM. As shown, the mean for the occupation rate of the RBM is very high compared to the other models, and the training data. In most cases, the RBM showed a much higher variety, and was more similar to the training data. This, however, highlights a flaw in using the distances as a metric to see the quality of the samples themselves. While it may seem that the DCGAN generates samples with little variety, and the RBM mimics the training data's variety, closer inspection of the generated samples reveals that the distance is high for some features because some reconstructions failed. Below is an example of an image generated by the RBM (Figure 5.31). On the left, an example of a successfully sampled image from the RBM is shown. The input image triggers many different hidden units and upon reconstruction, an image with many activated visible units is returned. To the right, an image in which barely any units activated is shown. This explains the very high mean distance for the occupation rate, as well as polyphonic rate and information rate. For the DCGANs, while the errors were monitored to ensure that mode collapse did not take place, the error alone cannot indicate if the generator is creating very similar if not identical images. The values for the mean intra-set distance for the DCGANs, which are much lower than for the GMP, suggest that while the images generated are more similar, there is still some variety. In some cases, such as the Pearson correlation, the mean was comparable with the training data.

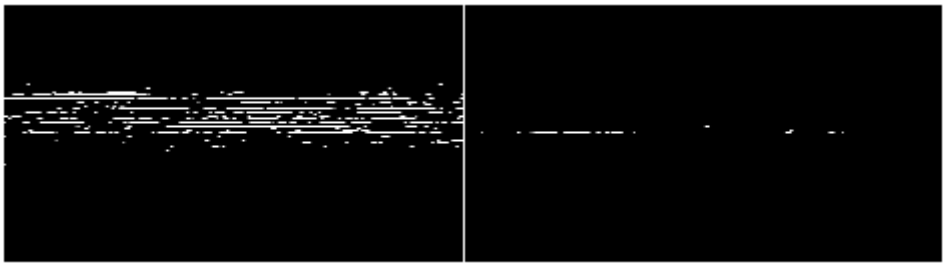

Figure 5.31 - Two samples generated with the RBM

The intra-set distances can be visualized using a probability density function. This can be done to gain a better understanding of how similar the distributions are. For the polyphonic rate, for example, the probability density functions are very similar for all datasets, except for the RBM which has a mean skewed to the right. This is shown in the table, but the plot also gives an indication of the general distribution. In most cases, the distributions of the DCGANs are almost identical, with the DCGAN-2 being closer to the GMP distribution.

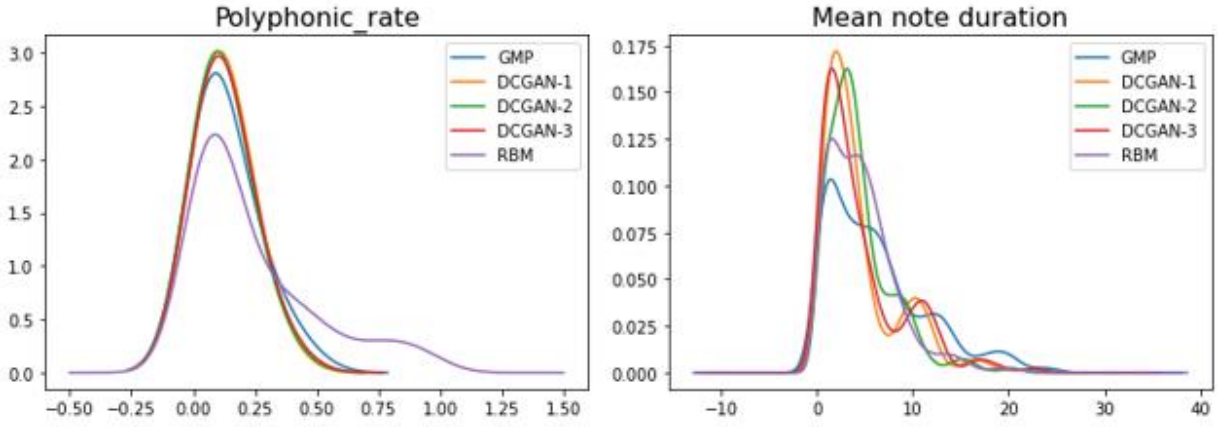

Figure 5.32- PDFs, polyphonic rate (left) and mean note duration (right)

It is worth noting that the plots which differ in the amount of distortion are in part due to the automatic bandwidth selection based on Scott's rule, which can lead to different smoothings. In the distribution for the mean note duration, the RBM closely follows the GMP distribution. Below, the plot for the average information rate is shown (Figure 5.33).

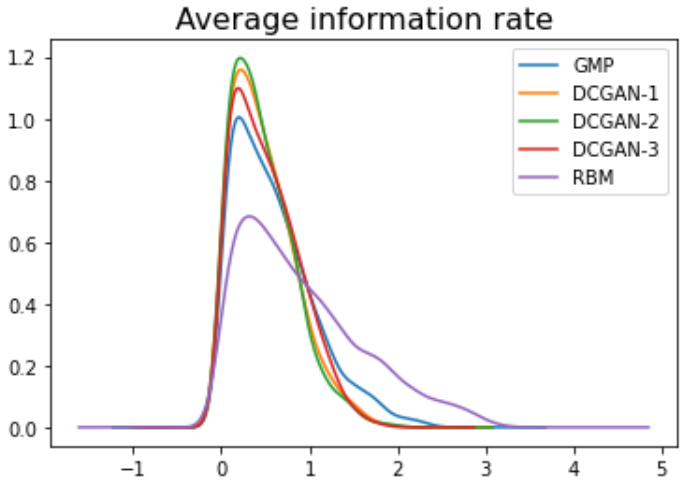

Figure 5.33 - PDFs, average information rate

For the final assessment of the models, the Kullback-Liebler divergence between the probability density functions of the intra-set distances and inter-set distances with the original dataset were calculated. As shown (Table 5.8), the distances for the dataset (GMP) are zero, as the inter-set and intra-set distances are the same. The closer to zero, the more similar the distributions are. Interestingly, the RBM models most of the features in a way which is closer to the training dataset. This was not evident from the plots of the pdfs. However, it is worth highlighting that the KL divergence is measured between the intra-set and inter-set distances. This was done because while two intra-set distributions may be identical (the distances between the generated samples are in the same distribution) using the inter-set distance makes use of additional information, i.e the exact values. For two identical intra-set distributions, the inter-set can still be different, and therefore is a better estimate of similarity.

Table 5.8 - KL divergences

	GMP	DCGAN-1	DCGAN - 2	DCGAN -3	RBM	Best performing model
Empty ratio	0	0.04	0.038	0.066	0.036	RBM
Occupation rate	0	0.35	0.133	0.315	0.072	RBM
Polyphonic_rate	0	0.08	0.055	0.074	0.017	RBM
Qualified Notes	0	0.85	0.412	0.793	0.588	DCGAN-2
Qualified Rhythms	0	0.18	0.101	0.341	0.092	RBM
Mean note duration	0	1.04	0.733	0.813	0.264	RBM
Pitch range	0	0.14	0.163	0.023	0.165	DCGAN-3
Pearson correlation with key profile	0	0.01	0.003	0.015	0.009	DCGAN-2
Average information rate	0	0.09	0.115	0.171	0.029	RBM

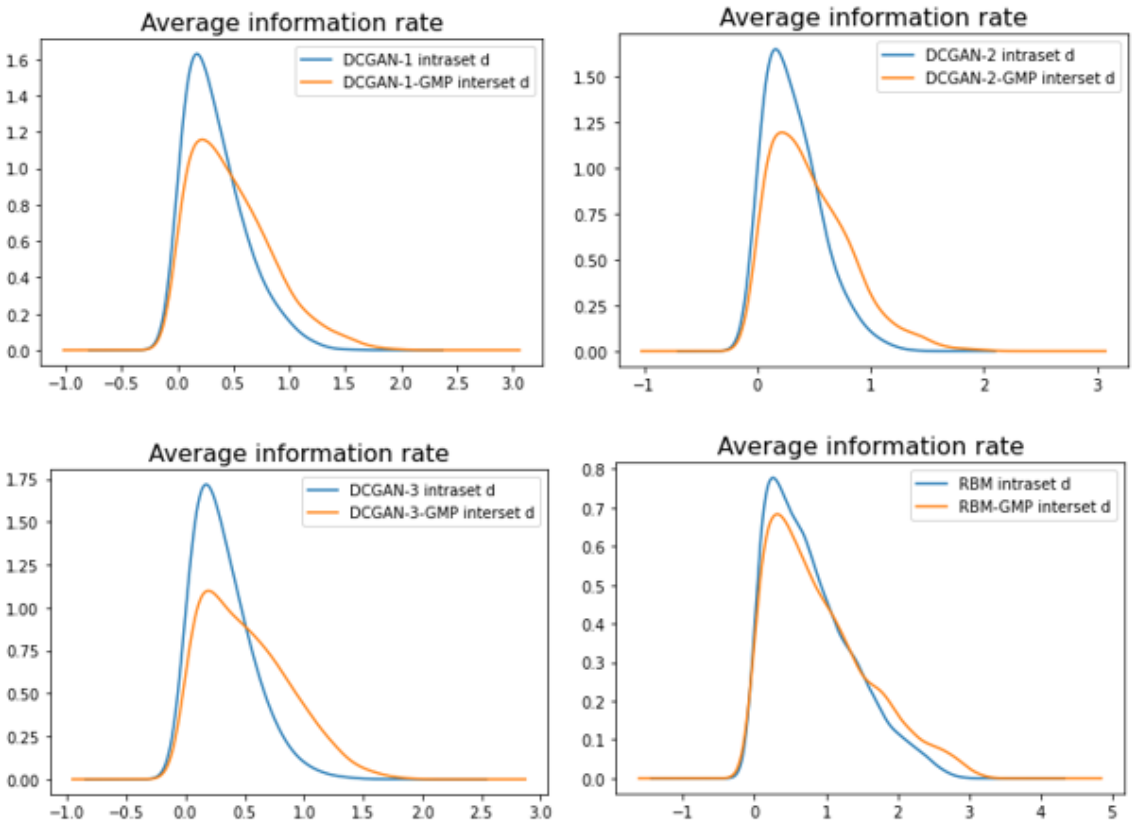

Figure 5.34 - PDFs of intra-set and inter-set distances between training data and generated samples

This point is illustrated taking into account the average information rate. From the previous plot, it would seem that the DCGANs are more similar. However, when plotting the inter-set distances with the intra-set distances (Figure 5.34), the RBM is shown to have the most similar distribution, reflected by the fact that the lowest KL divergence was obtained.

The evaluation metrics proposed act as indicators of certain features that a musical dataset possesses. It is important though to highlight the fact that while these measures can give an indication on similarity, there may be other reasons why an apparently good modelling was produced for that feature. If the generative model reproduced closely resembles the distribution of the training data, then it is expected that any properties of the training data should be found in the generated samples. Therefore, in general, these results show that to some extent, the models were successful in recreating images that have a similar distribution to the training data and confirm the feasibility of using this method of representing music as a piano-roll image to generate new examples. However, while the distribution may have been modelled, it is still to be determined whether this translates to actually enjoyable compositions. For that purpose, the listening test provides insight into the quality of the samples from a human perspective.

5.3 Listening Test

The set of samples used in the evaluation metrics was also used in the listening test. Not all of the experiments performed were then evaluated on the listening test, just the most promising ones. While

using different dataset sizes makes the comparisons less robust, the choice of varying the sizes was made out of both necessity but also to investigate whether more or less examples would be beneficial.

To summarise, the following models were tested:

- RBM – random sampling, reconstruction of incomplete image (dataset with songs in Am/Cmaj)
- WGAN-GP – GMP dataset consisting of 2000 examples, all keys
- DCGAN
    - Dataset consisting of around 2000 examples, all keys
    - Dataset with songs in Am/Cmaj
    - Dataset with songs in Am/Cmaj, RGB encoding
    - Full dataset with around 9500 examples

The listening tests were carried out using 15 participants, who had no formal background in either music or artificial intelligence. The samples used in the listening test are available on the project repository. [78]

5.3.1 Turing test

The Turing test involved showing participants samples generated by the models, with occasional training data examples and also randomly generated songs using a rule-based algorithm. The evaluation for each sample is in the range (1-5) and the score represents the confidence that the song was composed by a human, with a value of 5 meaning 100% confidence that it is a human composition. The scores allow for a better comparison compared with a simpler pass/fail test. The order of the samples was randomized for each participant. For each model, the average of all the sample results is shown in Table 5.9.

Table 5.9 - Average scores for Turing test

Samples	Average Score
Randomly generated song	1
RBM – random sampling	1.2
RBM – image reconstruction	1.5
WGAN	1.6
DCGAN – 2	2.4
DCGAN – 3	2.9
DCGAN – full dataset	3.2
DCGAN – 1	3.6
GMP dataset (human-composed)	4.9

As shown, most of the samples did not pass the Turing test. The music samples generated with the RBM had the worst performance in the Turing test. As expected, the Wasserstein GAN performed worse than the DCGAN. While in theory, the WGAN has been shown to be an improvement on the DCGAN, due to issues with training, finding a suitable configuration of hyperparameters and model architecture was not possible. The WGAN was trained using the dataset that contained samples from all musical keys. For that same dataset, the DCGAN did not perform substantially better, with the samples on average being ranked as probably not composed by a human, and as expected from the comparison of the two models carried out previously, the samples generated by the DCGAN were of

better quality. On the other hand, not all results from the Turing test are in accordance with the evaluation metrics. The DCGAN-2 had performed better than the other models in terms of the KL divergence between inter-set and intra-set distances, yet performs worse than other DCGAN implementation on the Turing test. Additionally, the RBM models, which had achieved the best KL divergence, performed worse than all other models, being easily identifiable as computer-made compositions.

The reason for this could be explained that passing the Turing test is not a direct indication of quality. It is a useful metric in determining the success of the overall project, but when comparing the scores for different tests, it is not as informative. Firstly, the order in which the examples are shown affects the result. A sample played directly after one of the random, unmusical samples, led to a higher score on average, and conversely, if the sample is heard right after a real example is played, the difference between the two becomes more apparent, and a lower score is assigned. Secondly, passing the Turing test depends on many factors. For the RBM, the samples were completely in the right key, which confirms that the distribution was learned correctly. However, in terms of the rhythm and melodies, it had a more artificial quality – there was very little repetition in melodic themes or recurring chords or melodies, and the overall result was quite static. For the other models, the DCGAN-2, trained on all keys, had more samples which contained notes off-key. In a musical piece, even a single dissonant passage or note is enough to significantly decrease the confidence score, as these do not occur at all in the human-made compositions. In general, the DCGAN trained on all keys with a dataset of 2000 samples tended to have more dissonant sounds, suggesting that the various distributions of pitches from the different keys were not discerned completely, and suggesting that the learning in not totally translation-invariant. In fact, the DCGAN trained on the smaller dataset, but with examples from a single key, learned better patterns and chords, and performed best in the Turing test. The next test gives a more nuanced perspective on the quality of the samples.

5.3.2 Survey

The survey makes use of the same participants and samples. In this case, however, the songs are to be rated based on four categories, and not in reference to a human composition. The ratings are in the range (1-5) with 5 being the most positive rating. The categories represent the following attributes of the pieces:

- Harmonies – indicate whether the notes are distributed on key, and the chords are pleasing/not dissonant
- Rhythm – whether the rhythms of the pieces are divided correctly and in an interesting way
- Structure – whether the pieces have certain recurring melodies or chords, or patterns of melodies which vary in a realistic way, or even a chord progression that sounds pleasing
- Overall – score based on general feeling of liking the song

Table 5.10 - Results of the survey

	Pleasant harmony	Rhythm	Structure	Overall
RBM - random sampling	3.87	2	1.33	2
RBM - image reconstruction	4	2.27	1.95	2.33
WGAN	3.98	0.8	2.13	2.27
DCGAN -1	4.87	3.4	4.34	4.8
DCGAN - 2	3.01	3.33	3.6	3.47

DCGAN - 3	4	3.2	4.73	3.87
DCGAN - full dataset	3.6	4.73	4.35	4
DCGAN – 1	3.6	0.009	0.003	0.015

The results, shown in Table 5.10, give some interesting information on the generative abilities of the models. In general, participants found the RBM compositions to have very pleasant harmonies, but assigned a low score to the rhythms, which were more random and unpredictable, and a very low score for the structure. This is reflected in the Turing test, where the RBM compositions are the most realistic. These do not follow a musical structure but are mostly a random sequence of notes in a particular learned scale. The overall result is quite poor. When using the image reconstruction method, the structure is higher, which is to be expected as it is based on the structure of the input obstructed image. The WGAN also performed very poorly in some categories. It produced a very poor score on rhythm, though the generated samples have more structure compared with the RBM. Nonetheless, the DCGAN models performed best, with the DCGAN trained on the A minor dataset achieving the best performance. While the DCGAN trained on the RGB dataset (DCGAN-3) made use of the same dataset as DCGAN-1 (encoded differently), it achieved poorer results. However, the participants found the samples from the DCGAN-3 to have a slightly better structure. This suggests that there is merit to the encoding method using RGB, whereby using overlapping pixels of different colours, repetitions of notes, chords and melodies, divided in rhythmically "acceptable" timesteps is enforced. The overall result though was poorer, which could be due to the added complexity of training the DCGAN with three channels instead of one.

Overall, the results of the survey are promising. Some of the models implemented produced songs which received a high score. This means that they do recreate the musical qualities of the training data with some degree of success. On the other hand, the compositions are still not at the level of human compositions, lacking some of the higher-level aspects of music that ultimately makes it enjoyable.

6 Discussion

In this section, a broader discussion of the generative models developed with respect to the project objectives is carried out.

The RBM model developed produced some acceptable results, though in terms of the final quality, it performed worse compared with the generative adversarial networks. While it was able to recreate some of the evaluation features better than all other models, the listening test showed that ultimately the musical pieces generated with the RBM were more flawed. Furthermore, the random sampling technique using the RBM gave poor results, and different random images would always activate the same pixels, and even then, the pixels had a low activation probability. This means that overall, the RBM is not suitable for the task of music generation from scratch. Unlike with GANs, a random input vector cannot be used to generate samples, and the reconstruction method must be used instead. This falls short of the objective of creating a composition from scratch, as it requires the use of a modified pre-existing musical piece. The hyperparameter tuning phase for the RBM was useful in selecting an architecture without having to manually inspect all samples on a subjective basis, to rule out combinations that were unlikely to yield good results. The results suggest that the most important hyperparameter to reduce reconstruction error is the number of features, and the values chosen were based on the upper limit of computational power available, beyond which the model could not be run. RBMs have been applied to algorithmic composition, but usually in combination with other methods such as convolutions [46] or recurrent neural networks. [44] The original reason for applying RBMs to this task was to then expand the model into a convolutional RBM, but this was not carried out because ultimately, the literature on GANs is much vaster and allowed to concentrate the focus of the project in adapting the models for the music generation task as opposed to the implementation. So, once it was clear that RBMs alone could not generate high quality musical samples with the computational power available, the focus of the project was moved to GANs.

As expected, the GANs performed more strongly compared with the RBM. All configurations achieved superior results on the listening test, though could not model certain features as accurately as the RBM. This highlights the issue of determining some robust estimators for quality in generative models, especially those which are used for computational creativity tasks such as music generation. Music generation is particularly difficult to evaluate since the attributes that relate to musical quality are themselves poorly defined. The features used are necessary, yet not sufficient, properties of a good quality musical piece. Furthermore, the values for these features that do lead to a piece of high quality is itself subjective. If a generative model is designed to generate images of realistic objects, then visual inspection of the generated samples can be enough to determine the success of the model, with the possibility of then using a classifier on the resulting generated images to obtain a quantitative measure of the GAN's performance. With music generation this is not possible. Still, as these features are elements present in the training data and do relate to characteristics of music, generating pieces which closely match the training data with respect to these features indicates a degree of success. In particular, aside from the more basic features such as the polyphonic rate and occupation rate, the Pearson correlation with the key profile and the information rates were robust methods for estimating the degree to which the model successfully generated music. This is shown by the fact that the DCGAN trained on the dataset with examples from the same key (DCGAN-1), obtained a much higher Pearson correlation than the one trained on all keys, and was also the one that performed best in the listening test.

Regarding the tests with the DCGAN, some interesting results were found. While it was expected that increasing the number of training examples would improve results, the method of using multiple segments from the same song was not successful. The results obtained were not an improvement over the medium dataset, and while samples from the large dataset were not included in the final listening test, no improvement were heard compared with the medium dataset. This could be due to the fact that using consecutive song segments may introduce a lot of bias in the data, since often sections within a song are repetitive. This would mean that many samples are duplicates, introducing bias into the dataset and thus negatively affecting the learning process. The hyperparameter tuning experiment was limited by the available computational power, so relatively few combinations were tested. This meant that ultimately, no significant improvements on the original formulation of the DCGAN were found, with the exception of a modification to the minibatch size. With the DCGAN, the losses were used as an estimate of the training performance which led to the identification of several failure modes. Even though the initial output of the DCGANs are images, visual inspection of the generated samples was also useful at initially determining how the training took place. Similarly, testing on the WGAN-GP led to some issues. Initial tests on different hyperparameters failed to converge and would require a more powerful GPU to perform using the same datasets used for the other tests. On some of the features, the WGAN performed better than a DCGAN trained with the same dataset but had the worst performance out of the generative adversarial networks in the listening test.

Finally, the two preprocessing techniques used led to strong results. Using a dataset which was reduced in size but only contained examples from a single major and a single minor key, led to the best outcomes on the listening test. The result was not as realistic as the songs generated using the full dataset, which achieved the best score in the Turing test, but was rated the highest among the participants in terms of how enjoyable the musical pieces were. This result did not translate to the Kullback-Liebler divergence with the training data evaluation where in that case, it was the RBM which performed best. This stresses the importance of the listening test in the final evaluation. As previously stated, most systems for algorithmic computation mainly make use of a listening test for evaluation, instead of more objective measures. Measuring the KL divergence between feature probability density functions has proven to be a somewhat limited measure as the results of the KL divergence and listening test are not always in accordance. Nonetheless, the method used with the KL divergence was the most in-depth method for analysis of algorithmic composition systems, [77] and within that framework incorporated additional higher-level estimators for musical quality. Whether a more robust method for evaluating algorithmic composition models is required depends on the final objective of the model. If the only aim of the model is to develop music that people will enjoy, then a listening test is sufficient. In the case of this project, part of the objectives explicitly involved implementing a less subjective way of assessing the results, which proved to be useful but not definitive.

7 Conclusions

7.1 Conclusions

In conclusion, two distinct generative models have been applied to the task of generating polyphonic music from scratch. The initial part of the project involved identifying a suitable representation for the musical data and converting this in a format acceptable for generative models. Compared with other types of generated content, music in particular has a very wide range of possibilities in terms of how it is encoded for algorithmic composition, and the choice will depend both on the resources available and the final objective. A data cleansing and preprocessing step was also applied once the dataset was chosen to remove outliers and improve the generative capabilities of the models.

In terms of the experiments performed in this project, two main approaches were used for each of the two types of model (RBM and GAN). The first approach was more grounded in the theoretical aspects of the models and made use of the training errors to assess the quality of training and from that infer how well the model learned the training data distribution. This was also done to test improvements on the model's performance by changing hyperparameters. The evaluation of the training of both RBMs and GANs was carried out, with an assessment on how modifying the hyperparameters affected the network. However, this process did not ultimately lead to the identification of a model architecture that was clearly superior to the default settings by the original authors. Nonetheless, as the input images used were very sparse, meaning that the majority of pixels are off, due to the fact that for each timestep no more than 5-6 notes play simultaneously. This is much more than typical training datasets used for machine learning problems, so the use of such a sparse type of image was a concern in terms of the effectiveness of the models. This meant that it was important to try out different combinations of hyperparameters to determine whether at least some improvement could be achieved on this type of image.

The second approach was focused on the generated samples. The issue of identifying suitable evaluation metrics was discussed at length and different metrics were implemented. The development of these quality measures, and application of these to the generated sample proved to be very useful both in evaluating the performances of the models without having to exhaustively perform listening tests for all models, and to get a measure of the model's ability to recreate the data's distribution. The Kullback-Liebler divergence was helpful in establishing how similar the generated samples were to the training data. However, as most of the features extracted were lower-level features, describing very basic rhythmic and melodic characteristics, the model being able to match these closely did not translate to a positive score on the human listening test. For the listening test, while not many participants were available, a large set of samples was generated and rated by the participants, with some models producing pieces which have been rated as enjoyable. The Turing test results were not as positive but as mentioned before, it is very easy to identify a piece as a computer composition as even a single mistake will reveal the fact that the composition was not made by a human, but also because the AI models developed cannot create a composition that is expressive and has higher level, more subtle features that give music its overall quality. On the other hand, the fact that the pieces did not pass the Turing test and were yet considered to be enjoyable by the participants is a promising result for algorithmic composition as it shows that the models are able to generate music that is enjoyable even without needing it to be completely realistic, and that people do not necessarily need to find the music realistic to be enjoyable.

7.2 Future Work

In the project scope, the identification of future work to continue for this was outlined as an objective. The project has led to several interesting points that could be further expanded to achieve stronger results and has introduced some ideas that could be interesting if carried out further.

Firstly, in terms of the models developed, the hyperparameter tuning steps could be expanded to try out a much wider range of combinations. Doing so could improve the results significantly. Not only this, but with increased computational power available the models could be run for many more epochs and using a much larger dataset. The dataset used in this project was one of the largest publicly available in the domain of music tasks with AI, though for most experiments, its full size could not be used.

Aside from future work that would mainly overcome practical limitations of this project, the use of the RBM has shown to be promising. The distributions learned were the most robust, and while the listening test was the least successful, its modelling capabilities, especially in being able to accurately capture the musical keys of the training data, could be applied to good use. The task for this project was composition from scratch – in that regard the RBM was not suitable, due to the requirement of the input obstructed image which negates it being from scratch. However, this could be a very successful method for conditioning tasks. By using a desired chord progression or melody as an input, the RBM could be trained to compose an accompaniment, or an improvisation based on that. This would involve some slight modifications to the RBM used in this project and to the data representation, but the results could be interesting.

The application of GANs to algorithmic composition has been shown both in this project and other implementations surveyed in the literature review to be possible and to yield potentially good results. In this project specifically, the possibility of representing the musical pieces and each segment as a different channel could be explored further, as this specific method has not been used in other systems that were surveyed in the literature review and could be a positive contribution to the set of methodologies used in algorithmic composition. The results give a preliminary indication that by overlapping images in different channels, some level of repetition of melodies and chords is achieved. With some improvements, this method could achieve much stronger results.

8 References

[1] H. GM, M. K. Gourisaria, M. Pandey and S. S. Rautaray, "A comprehensive survey and analysis of generative models in machine learning," *Computer Science Review,* 2020.

[2] A. Alankrita, M. Mamta and B. Gopi, "Generative adversarial network: An overview of theory and applications," *International Journal of Information Management Data Insights,* vol. 1, no. 1, 2021.

[3] J. D. Fernandéz and F. Vico, "AI Methods in Algorithmic Composition:A Comprehensive Survey," *Journal of Artificial Intelligence Researc,* pp. 513-582, 2013.

[4] J.-P. Briot, G. Hadjeres and F.-D. Pachet, Deep Learning Techniques for Music Generation, Computational Synthesis and Creative, Springer,, 2019.

[5] C. Ames, "Automated Composition in Retrospect: 1956-1986," *Leonardo,* vol. 20, pp. 169-185, 1987.

[6] L. Hiller and L. Isaacson, "Musical composition with a high-speed digital computer," *Journal of Audio Engineering Society,* pp. 154-60, 1958.

[7] L. Hiller and C. Ames, "Computer Cantata: A Study of Compositional Method," *Perspectives of New Music,* vol. 3, no. 1, p. 62, 1964.

[8] H. Brun, "From Musical Ideas to Computers and Back," in *The Computer and Music,* 1964.

[9] M. &. M. Pearce and G. David & Wiggins, "Motivations and Methodologies for Automation of the Compositional Process," *Musicae Scientiae,* vol. 6, 2002.

[10] R. L. d. Mantaras and J. L. Arcos, "Ai and Music: From Composition to Expressive Performance," *Science Direct Working Paper,* vol. S1574, 2018.

[11] S. Ji, J. Luo and X. Yang, "A Comprehensive Survey on Deep Music Generation:," 2020.

[12] D. Lidov and J. Gabura, "A melody writing algorithm using a formal language model," *Computer Studies in the Humanities and Verbal Behavior,* vol. 4 , no. 3-4, pp. 138-148, 1973.

[13] G. M. Rader, "A method for composing simple traditional music by computer," *Communications of the ACM,* vol. 17, no. 11, pp. 631-638, 1974.

[14] S. R. Holtzman, "Using generative grammars for music composition," *Computer Music Journal,* vol. 5, no. 1, pp. 51-64, 1981.

[15] M. J. Steedman, "A generative grammar for jazz chord sequences," *Music Perception: An Interdisciplinary Journal,* vol. 2, no. 1, pp. 52-77, 1984.

[16] M. Chemillier, "Towards a formal study of jazz chord sequences generated by Steedman's grammar," *Soft Computing - A Fusion of Foundations, Methodologies and Applications,* vol. 8, no. 9, pp. 617-622, 2004.

[17] J. Gillick and K. Tang, "Learning jazz grammars," *Proceedings of the Sound and Music Computing Conference,* p. 125130, 2009.

[18] S. Gill, "A technique for the composition of music in a computer," *The Computer Music Journal,* vol. 6, no. 2, pp. 129-133, 1963.

[19] S. Schwanauer, "A learning machine for tonal composition," in *Machine models of music,* Cambridge, The MIT Press, 1993, pp. 511-532.

[20] G. Widgmer, "Qualitative perception modeling and intelligent musical learning," *Computer Music Journal,* vol. 16, no. 2, pp. 51-68, 1992.

[21] R. R. Spangler, *Rule-Based Analysis and Generation of Music,* California Institute of Technology, 1999.

[22] E. Morales and R. Morales, "Learning musical rules," in *Proceedings of the International Joint Conference on Artificial Intelligence,* 1995.

[23] M. Grachten, "JIG: Jazz improvisation generator," in *Proceedings of the Workshop on Current Research Directions in Computer Music,* 2001.

[24] B. H. D. V. Y. Manaris, "Monterey mirror: Combining Markov models, genetic algorithms, and power laws.," in *Proceedings of the AAAI National Conference on Artificial intelligence*, 2011.

[25] P. Todd, "A connectionist approach to algorithmic composition," *Computer Music Journal,* vol. 13, no. 4, pp. 27-43, 1989.

[26] M. Duff, "Backpropagation and Bach's 5th cello suite," in *Proceedings of the International Joint Conference on Neural Networks*, 575, 1989.

[27] M. Mozer, "Connectionist music composition based on melodic, stylistic and psychophysical constraints," in *Music and Connectionism,* Cambridge, The MIT Press, 1991, pp. 195-211.

[28] J. P. Lewis, "Creation by refinement and the problem of algorithmic music composition," in *Music and Connectionism,* Cambridge, The MIT Press, 1991.

[29] N. Shibata, "A neural network-based method for chord/note scale association with melodies," *NEC Research and Development,* vol. 32, no. 3, pp. 453-459, 1991.

[30] M. I. Bellgard and C. P. Tsang, "Harmonizing music using a network of Boltzmann machines," in *Proceedings of the Annual Conference of Artificial Neural Networks and their Applications*, 1992.

[31] M. Nishijimi and K. Watanabe, "Interactive music composer based on neural networks," *Fujitsu Scientific Technical Journal,* vol. 29, no. 2, pp. 189-192, 1993.

[32] P. Toiviainen, "Modeling the target-note technique of bebop-style jazz improvisation: an artificial neural network approach," *Music Perception: An Interdisciplinary Journal*, vol. 12, no. 4, pp. 399-413, 1995.

[33] J. Franklin, "Multi-phase learning for jazz improvisation and interaction," in *Proceedings of the Biennial Symposium on Arts and Technology*, 2001.

[34] H. Hild, J. Feulner and D. & Menzel, "HARMONET: a neural net for harmonising chorales in the style of J.S. Bach," in *Proceedings of the Conference on Neural Information Processing Systems*, 1992.

[35] C. Goldman, D. Gang, J. Rosenschein and D. & Lehmann, "NETGEN: a hybrid interactive architecture for composing polyphonic music in real time," in *Proceedings of the International Computer Music Conference*,, 1996.

[36] B. Sturm, J. Santos, O. Ben-Tal and I. Korshunova, "Music transcription modelling and composition using deep learning," 2016.

[37] E. Waite, *Generating long-term structure in songs and stories*, 2016..

[38] G. Hadjeres and F. Nielsen, *Interactive music generation with positional constraints using anticipation-rnns*, 2017.

[39] A. Roberts, J. Engel, C. Raffel, C. Hawthorne and D. Eck, *A hierarchical latent vector model for learning long-term structure in music*, 2018.

[40] I. Yamshchikov and A. Tikhonov, *Music generation with variational recurrent autoencoder supported by history*, SN Appl. Sci, 2020.

[41] G. Hadjeres, F. Nielsen and a. F. Pachet, " Glsr-vae: Geodesic latent space regularization for variational autoencoder architectures," in *IEEE Symposium Series on Computational Intelligence* , 2017.

[42] L. Yu, W. Zhang, J. Wang and Y. Yu, " Seqgan: Sequence generative adversarial nets with policy gradient," in *Thirty-first AAAI conference on artificial intelligence*, 2017.

[43] K. Wang and X. Wan, " Sentigan: Generating sentimental texts via mixture adversarial networks.," in *IJCAI*, 2018.

[44] N. Boulanger-Lewandowski, Y. Bengio and P. Vincent, "Modeling temporal dependencies in highdimensional sequences: Application to polyphonic music generation and transcription," in *International Conference on Machine Learning*, 2012.

[45] Q. Lyu, Z. Wu and J. Zhu, "Polyphonic music modelling with lstm-rtrbm," in *Proceedings of the 23rd ACM international conference on multimedia*, 2015.

[46] S. Lattner, M. Grachten and G. Widmer, *Imposing higher-level structure in polyphonic music generation using convolutional restricted boltzmann machines and constraints*, 2016.

[47] D. D. Johnson, "Generating polyphonic music using tied parallel networks," in *International conference on evolutionary and biologically inspired music and art*, 2017.

[48] E. Lousseief and B. Sturm, "Mahlernet: Unbounded orchestral music with neural networks," in *Nordic Sound and Music Computing conference*, 2019.

[49] Z. Wang and Y. Zhang, "Pianotree vae: Structured representation learning for polyphonic music," in *ISMIR*, 2020.

[50] Y.-S. Huang and Y.-H. Yang, *Pop music transformer: Generating music with rhythm and harmony*, 2020.

[51] H. Chu, R. Urtasun and S. Fidler, "Song from pi: A musically plausible network for pop music generation," in *ICLR*, 2017.

[52] J. Bonada and X. Serra, "Synthesis of the singing voice by performance sampling and spectral models," in *IEEE signal processing magazine*, 2007.

[53] C. Donahue and H. H. Mao, " Lakhnes: Improving multi-instrumental music generation with cross-domain pre-training," in *ISMIR*, 2019.

[54] J. Ens and P. Pasquier, *Mmm: Exploring conditional multi-track music generation with the transformer*, 2020.

[55] H.-W. Dong and W.-Y. Hsiao, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *AAAI*, 2018.

[56] F. Guan, C. Yu and S. Yang, " A gan model with self-attention mechanism to generate multi-instruments symbolic music," in *International Joint Conference on Neural Networks (IJCNN)*, 2019.

[57] G. Chen, Y. Liu, S.-h. Zhong and X. Zhang, " Musicality-novelty generative adversarial nets for algorithmic composition," in *Proceedings of the 26th ACM international conference on Multimedia*, 2018.

[58] Y. Li-Chia and C. Szu-Yu, "Midinet: A convolutional generative adversarial network for symbolic-domain music generation," in *ISMIR*, 2017.

[59] N. Trieu and R. Keller, "Jazzgan: Improvising with generative adversarial networks," in *MUME workshop*, 2018.

[60] H. H. Tan, "Chordal: A chord-based approach for music generation using bi-lstms," in *ICCC*, 2019.

[61] Google, "Improv rnn," 2018. [Online]. Available: github.com/tensorflow/magenta/tree/master/magenta/models/.

[62] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves and N. Kalchbrenner, *Wavenet: A generative model for raw audio*, 2016.

[63] J. Engel and C. Resnick, "Neural audio synthesis of musical notes with wavenet autoencoders," in *International Conference on Machine Learning*, 2017.

[64] A. Défossez and N. Zeghidour, " Sing: Symbol-to-instrument neural generator," in *Advances in Neural Information Processing Systems*, 2018.

[65] S. Mehri and K. Kumar, "Samplernn: An unconditional end-to-end neural audio generation model," in *ICLR*, 2017.

[66] C. Donahue, J. McAuley and M. Puckette, "Adversarial audio synthesis," in *ICLR*, 2019.

[67] J. Engel and K. K. Agrawal, "Gansynth: Adversarial neural audio synthesis," in *ICLR*, 2019.

[68] T. Aldwairi, D. Perera and M. A. Novotny, "An evaluation of the performance of Restricted Boltzmann Machines as a model for anomaly network intrusion detection," *Computer Networks,* vol. 144, pp. 111-119, 2018.

[69] A. Fischer and C. Igel, *An Introduction to Restricted Boltzmann Machines,* 2012.

[70] V. Upadhya and P. Sastry, "An Overview of Restricted Boltzmann Machines," *Journal of the Indian Institute of Science,* vol. 99, 2019.

[71] G. E. Hinton and R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," *Science,* vol. 313, pp. 504-507, 2006.

[72] a. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, *Generative Adversarial Nets,* 2014.

[73] A. Radford, L. Metz and S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,* 2016.

[74] M. Arjovsky, S. Chintala and L. Bottou, *Wasserstein GAN,* 2017.

[75] I. Gulrajani, F. A. M. Arjovsky, V. Dumoulin and A. Courville, *Improved Training of Wasserstein GANs,* 2017.

[76] D. Temperley, "What's Key for Key? The Krumhansl-Schmuckler Key-Finding Algorithm Reconsidered," *Music Perception,* vol. 17, no. 1, pp. 65-100, 1999.

[77] L.-C. Yang and A. Lerch, "On the evaluation of generative models in music," *Neural Computing and Applications,* vol. 32, 2020.

[78] F. Loyola, "Project repository - "TFM - Polyphonic music generation using neural networks"," [Online]. Available: https://github.com/FedericoL-UPC/TFM-Polyphonic-music-generation-with-NNs.

[79] vishnubob, "Python-midi," 8 July 2015. [Online]. Available: https://github.com/vishnubob/python-midi.

[80] Q. Kong, B. Li, J. Chen and Y. Wang, *GiantMIDI-Piano: A large-scale MIDI dataset for classical piano music,* 2020.

[81] G. Hinton, "A Practical Guide to Training Restricted Boltzmann Machines," in *Neural Networks: Tricks of the Trade*, 2012.

[82] M. Lucic and K. Kurach, "Are GANs Created Equal? A Large-Scale Study," in *32nd Conference on Neural Information Processing Systems (NIPS)*, Montréal, 2018.

[83] I. Goodfellow, *Improved Techniques for Training GANs,* 2016.

[84] I. Goodfellow, *NIPS 2016 Tutorial: Generative Adversarial Networks,* OpenAI, 2016.

[85] J. Brownlee, "Machine Learning Mastery," 10 July 2019. [Online]. Available: https://machinelearningmastery.com/tour-of-generative-adversarial-network-models/.