Universidad Zaragoza

1542

Master Thesis

GPianoroll: a Deep Learning System with Human Feedback for Music Generation

Author

Miguel Marcos Gil

Directors

Rubén Martínez-Cantín

Lorenzo Mur

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2023

ACKNOWLEDGEMENTS

First, I want to thank my tutors, Rubén and Lorenzo, for their help, guidance and understanding through this thesis' development. Despite being a somewhat tangent topic with respect to the contents of the course, we managed to find a middle point where I could work on a topic I enjoy while making use of the knowledge acquired from the master.

Second, thanks to all the master's faculty, PhD students and other people who collaborated through classes, talks and answers to my questions along the course, and special thanks to those who took part in the experiments presented in this paper.

Finally, thanks to my friends, family and partner for all their support.

GPianoroll: a Deep Learning System with Human Feedback for Music Generation

ABSTRACT

Generative deep learning models are valuable assets for synthetic content creation and dataset generation, and are growing in popularity due to their capacity to produce massive amounts of high quality data. However, the generative potential of these models comes with some setbacks. It can be difficult to generate a certain unique sample among all the possible outputs, or it may not be possible to directly use a function to measure its adequacy for a task as some distance or probability.

Our work covers one such case: Music. Musical scores can be understood as different functions of frequency over time. These functions, like many others in science, can be composed and treated in countless ways. However, no typical function can measure how 'good' (or any other subjective criterion) a song is. Moreover, songs present problems when comparing them. Depending on the representation, a single melody played at several scales could be completely disjoint from the original, while still being pretty similar. And if we were to change a single note from that same melody, it could result in something completely different. However, there's no objective scale to measure how similar or different these new melodies are, so there's no telling how any change can affect a listener's opinion.

In this work, we propose an alternative evaluation method that leverages human feedback as our quality measurement to avoid relying on automatic numerical functions for these subjective criteria. Our purpose is to find the song that better fits a person's taste among a space of possible songs by using Bayesian Optimization. Leveraging this technique's sample efficiency and function agnosticism, we aim to find an optimum in a user's taste. We sample an artificial low-dimensional latent space, which is later mapped to the high-dimensional latent space of the state-of-the-art musical score generator MuseGAN. We play the output, and ask the final user to evaluate how they like the output of the model. After few iterations, our method can find which point of the latent space is more to the subject's liking. Our work includes the results obtained after performing the optimization process with several subjects, the evaluations said test subjects gathered through a Likert scale survey, and an exploration of the limits of classic Deep Learning techniques applied on the field of music.

Index

Chapter 1

Introduction

1.1 Motivation

Content generation makes an important part of machine learning advances. From image, to video, to text or semantics, every project needs a collection of data to train with, and the larger and more complete it is, the better. As such, generative models receive an important deal of attention due to their capability to assist with this task, and to this day many different architectures [1, 2, 3, 4, 5] have risen to it. They can be useful in many domains, and among them, we find an specially interesting one: Music.

Music is a great part of human culture and life. A musical piece played during a key moment in a movie or play can shift the listener from their sit to the time and place the action occurs in. Musicians and singers fill stadiums and theaters with audiences from all around the globe who travel distances just to see some minutes of their craft first hand. Moreover, the impact of music does not limit itself to culture. It also reaches other fields, like medicine. For example, listening to certain sounds can improve a person's sleep patterns, and music therapy can help with certain illnesses and treatments. In short, music is present both as a medium itself and as part of many other scenarios, but whatever the case, it plays an important role on how humans interpret and react to all the information they receive through all their senses. However, when treating it as data, music presents some interesting qualities and challenges that make it different to other types of data, like images.

On one hand, though music is usually regarded as an art, it also has a solid mathematical base that is studied by musicians like any other scientific field. Sound is a vibration heard by a listener, waves that follow physical laws which are already known and established, and as such it can be interpreted mathematically. Each style has one or some characteristic rhythm patterns, and notes in a score have an assigned standard frequency. The relation between these rhythms and frequencies is what makes a melody feel aesthetically pleasing or shockingly dissonant. These properties present some domain-specific structural challenges when applying Deep Learning techniques on music, for example when comparing them. The same melody played one octave higher or lower can cause its representation to be completely different, but it should be still considered pretty close, if not the same. Similarly, if we displace one of the notes of that same melody, we could create a completely new piece, while their representations would probably still be almost identical. A smaller modification may not yield more similar results than a big transformation, and there's no defined scale to measure how large the difference is for these cases.

On the other hand, it is also undeniable that music has a subjective part that depends on the listener. The same piece of music, no matter if it follows a set of musical rules or not, may or not be regarded equally by two different people. One may prefer a musical style to other, or be more fond of a bass playing a certain melody over a guitar. This preferences may not be exclusively decided based on the physical properties of the sounds played in a song, and as such they are hard, or rather impossible, to evaluate at the time of writing it. Something close to a method to write down the subjective components of music would be musical notation terms such as 'lacrimoso' or 'leggiadro' (Italian for 'tearful' and 'graceful') often included in musical scores next to the dynamics, which tell the feeling that should be evoked when playing. But even these terms may mean a different thing to the composer who wrote the piece, the musicians playing it, and the listeners.

This kind of subjective aspects of music are hard to convey to machines. If one were to make a computer compose music on command, it would be desirable for it to comprehend the intention or feelings behind a human's request. This would require either a different algorithm for each occasion and purpose, which is impractical, or an artificial intelligence able to learn what makes a song sad, jolly, or any adjective one may want to add to a composition. In the end, the problem of selecting a song out of a collection of possible creations can be abstracted as extracting a sample from a distribution given a condition, and many solutions exist for this kind of problems. For the case of deep learning, the solutions involve capturing these distributions as series of combinations of an input carried out across consecutive layers. In generative tasks such as this one, the model is asked to hallucinate samples given an input, which can be a description of the objective in the shape of an encoding or just random noise. These inputs form what is called the model's latent space. The model is iteratively provided with real samples as examples, and its parameters adapt each time (hence the 'learning') to maximize the probability of its creations to come from the same distribution as the real samples. As a model grows in layers (hence the 'deep'), the more complex the distributions it can represent become, and combining layers in different ways results in many different architectures for different tasks, like Generative Adversarial Networks or Large Language Models.

Deep learning focused works on the music domain approach these preferences by annotating their data with text tags, or grouping it by author or genre, and building their latent spaces around these groups. A model may, for example, take a text prompt such as “a pop song played in Mozart’s style” [6, 7] or the lyrics to a song [8] and output an audio fragment that resembles such request. However, these approaches require great effort for building a classification that will be just as arbitrarily complete as the tags’ vocabulary and as representative as the data permits. Others [9, 10] choose not to focus on a final audible piece, but on composition. This gives more control over the generated pieces and reduces the variables to take into account for judgement, but this abstraction makes it harder to judge them, as they lack several steps of production and can vary depending on how they are played.

But in the end, no matter if a piece of music is human or machine-made, no tag can tell if a song is ‘good’. Even if classified by large amounts of people, the song most listened world-wide may not be up to someone’s taste. A listener may think a piece is bad despite the composer’s confidence on their work, or *vice versa.* Even with the latest advances in deep learning, with language models and generative AIs now present in many browsers, sites and other projects, music remains a challenge for machines. Musical style may be teachable to a certain extent to a model, but whether its generations are ‘good’ or ‘bad’ is still out of its reach, as it still needs a listener to judge. The conclusion is that, ideally, a model made to create music should adapt to each listener whenever asked to create a song in order for its creations to be considered good. However, modelling a person’s taste is no easy task. The already existing music supposes a challenge to judge thoroughly not only because of quantity, but also because of variety. Even if one supported their decisions in already assigned tags, music genres are not mutually exclusive nor homogeneous. A person may like or dislike a song for a presumably infinitely long list of possible reasons, or even for one they might not be able to clearly express in words or for no reason at all. So, providing a model with enough real and precise examples to perfectly capture the distribution of the songs a person likes and dislikes would take a prodigious amount of the person’s time and patience, and it would be necessary to invest that time again with each and every person that uses it. Instead of proceeding with that plan, we propose an alternative.

1.2 Objectives and scope

Our work aims to create a music generation model that takes into account individual taste without requiring huge amounts of resources and time. We aspire to achieve this using Bayesian Optimization. Bayesian Optimization (BO) is a global optimization technique that is sample efficient and function agnostic, meaning it can approximate a function of unknown shape in few iterations. Our intuition is that, leveraging this technique, we could greatly reduce the number of times a user is queried about their taste. We propose a system with human-machine interaction and BO as our main tools to find what a user considers to be the best song out of the possible generations of a given model. By asking them to grade model-generated pieces, we pretend to approximate the user's taste to find the best point in the model's latent space for generating a 'good' composition for each subject.

To achieve this, we will first conduct a study of the current scene of deep learning for music generation. We will visit the pros and cons of each model, focusing in those most compatible with our vision of the final system. We will look for a model able to generate pieces of moderate length in a short amount of time in order to make the process as agile as possible, and that provides accessible implementation to make the necessary adjustments to it if they were needed. After that, we will implement the chosen model, we will replicate its results and proceed to build the human-machine interaction system.

The system should play the generated tracks for the users, and they should be able to give some kind of feedback. Since we already stated that words are unreliable, we propose a numeric feedback: A classic score, from zero to ten. That way, users can still contribute to the betterment of the model without having to argue why they like the song or not. Since no work has attempted this approach with music before, we will validate it designing and performing a pilot test with users and gather their opinion on both the efficacy of BO for finding a song they like and their own ability to give scores in this manner.

So, in summary, the objectives of this work are:

– Exploration of State-of-the-Art deep learning models for music generation.

– Recreation of a music generation model suitable for Bayesian Optimization.

– Implementation of an optimization system based on BO and human-machine interaction.

– Validation of BO as an optimization approach for music and human judgement.

1.2.1 Resources and tools

The following are the tools used for the development of this Master's Thesis.

Most of the implemented material for this thesis will be written in the Python programming language and interpreter. We chose this language for its vast collection of libraries, among which we find PyTorch, our choice for developing and testing deep learning models. For coding and performing Bayesian Optimization, we will use the BayesOpt library, which is built in C++ and counts with a Python interface. This way, we can leverage C++'s speed as a compiled language and still maintain a unified program in Python.

Platform-wise, we will use Google Colab (also known as Colaboratory) and Visual Studio Code. The first will be used for the development of the deep learning model. Since we want to focus on light and fast models, we think a cloud environment is ideal for easy deployment and reproducibility. As for Visual Studio Code, it is a fairly extended IDE choice that permits development in several operative systems and counts with extensions to easily execute Python code. Since we need a way to locally execute code for running BayesOpt and run the pilot tests, Visual Studio is our choice for a local environment.

Finally, we will also use GitHub, an open-source project sharing platform. This serves two purposes. First, it will grant us access to the open implementations of music generation models, facilitating us with material for our first objective. Second, we will publish the results and code for our work in a public repository, for all interested people to see and reproduce, contributing to the Bayesian Optimzation scene for music.

1.2.2 Project planning

This project corresponds to 30 ECTS in the Master in Robotics, Graphics and Computer Vision. This equals a total of 750 hours of work. Table 1.1 contains our initial estimation for dedicated time for each of the objectives. Table 1.2 and Figure 1.1 contain a summary of the different tasks and the actual time invested on each of them during the project's span.

Objective	**Estimated time**
SotA exploration	2 weeks (80h)
Music generation model implementation	3 weeks (120h)
Results evaluation	3 weeks (120h)
Optimization model development	5 weeks (200h)
Pilot experiments	4 weeks (150h)
Document writing	2 weeks (80h)
Total	19 weeks (750h)

Table 1.1: Initial planning for the previewed project phases.

Tasks		Hour count	
Documentation	Related work search	53	87
	State of the art analysis	24	
	New tools training	10	
Implementation	Music generation model	134	357
	Preference model	197	
	Evaluation interface	26	
Evaluation	Generated results analysis	127	265
	Human feedback experiments	138	
Writing	Project report	92	92
Total			801

Table 1.2: Hour assignment breakdown for each task

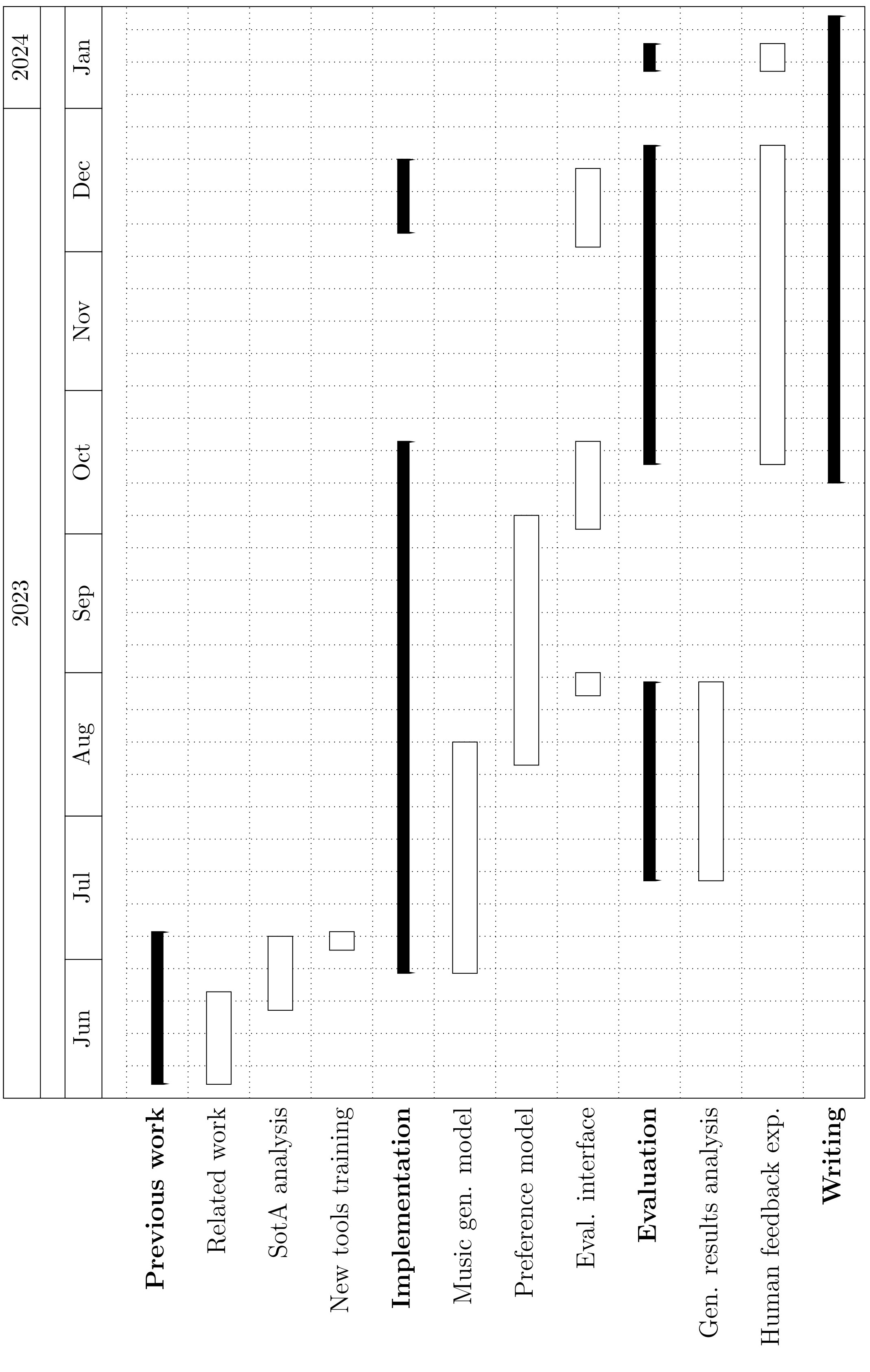


Figure 1.1: Gantt diagram for the project

1.3 Contribution

By the end of this project, we have achieved the goals proposed in the previous section. After reviewing the state of the art in music generation, we selected a gold standard model: MuseGAN [10], a Generative Adversarial Network able to compose piano rolls for several instruments. We modified the architecture of the model to better suit an optimization problem, and then train it over the Lakh Pianoroll Dataset [11], made of thousands of rock music piano rolls. Our modifications include dimensionality reduction of the input for faster optimization, which we achieve using REMBO [12], a Bayesian Optimization oriented technique to map high-dimensional spaces to low-dimensional ones.

During our development phase, we also collected a series of trials for other dimensionality deep learning oriented reduction techniques, like reducing the model's input by adding new layers or creating other models that used the pre-trained model as a backbone. However, we found that this approached led to underperformance of our model. The results of this exploration create a view of the various setbacks these techniques have when used in this particular case, which should be avoided in case future works use this one as a base.

Finally, we developed an optimization system along with an interface for catering our model to several users' taste. We optimize the black-box function that maps the reduced latent space across the model's generative process and all the way to the user's evaluations using the BayesOpt library and the Python programming language. This allows for a user-driven, individual optimization of the model in a reduced number of iterations. To validate our method, we asked users to listen and evaluate songs generated by our custom model and find an optimal sample they like the most. We provide examples of optimal compositions found by the users, as well as the aggregated results of a Likert scale survey, that prove the qualitative improvement of the generations after performing Bayesian Optimization.

All of the developed code is available at GitHub[1] for open access. This includes the tools to recreate the model used in our experiments completely online, and the code for running the experiments is available for download.

[1]https://github.com/Mikceroese/GPianoroll

Chapter 2

Related Work

2.1 Generative models in Deep Learning

A generative model has the task to capture the distribution of a set of data and then be able to create plausible samples within said distribution. This kind of models has a wide range of possible applications, both on its own or as a smaller part in other projects, like generating larger datasets for other Deep Learning models.

Some examples within this category are Generative Adversarial Networks [1] (GANs) and Variational Autoencoders [2, 3] (VAEs), both mentioned in later chapters. GANs and their later successors, such as the Wasserstein GANs [4] (WGANs), use two Neural Networks. A generator G and a discriminator D face each other in a minmax game, where G creates 'counterfeit' data samples and D is asked to tell real samples apart from fakes. In the original GAN, D outputs a probability of the sample being real. For WGANs, D becomes a critic C, and it gives a scalar score as low as possible for fakes and as high as possible for true samples. If trained correctly, G is able to map its latent space onto the original samples' distribution with enough quality for D or C to not be able to distinguish them.

Similarly, Autoencoders [2, 3] have an encoder and a decoder that work together. The encoder must extract features from the data and the decoder must reconstruct the original data from the extracted features. The model trains to minimize the difference between the reconstructions and the originals, and ends assigning a different encoding to each sample. The extracted features form an ordered latent space where similar samples lie close together. But this kind of model alone can produce sparse latent spaces, where some points are not mapped to any content. Their generative capability shines the brightest when made Variational. Instead of letting the model completely decide how to organise the latent space, a Gaussian prior is imposed on it. This way, the decoder can then be used to generate new data by sampling from the latent space. This method is also useful for performing interpolation between real samples by obtaining their encoded features, interpolating them, and then decoding the resulting features onto new samples.

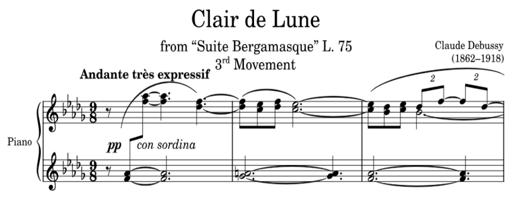
Finally, the Transformer architecture [5] exploits attention mechanisms, which factorize data distributions as the product of several smaller distributions using the chain rule of probability. This mechanism allows models to consider context when generating pieces of data. This is the case of the Generative Pre-trained Transformers [13] (GPTs), models trained in vast amounts of text data using unsupervised learning used for natural language processing tasks, among which we can find text generation. GPT models generate text word by word, usually in response to a prompt, by passing the parts of the text they have already generated through their learned attention maps and using this contextual information to choose the next word. This models usually undergo some fine-tuning for specific tasks or to avoid certain undesirable biases product of the use of unsupervised data, which is complemented with reinforcement learning from Human Feedback [14] (RLHF). This technique (which aligns with our work's perspective) defines a reward model that defines which output texts are preferred based on human judgement, elevating some answers and punishing others. On a final note, though most models condition the answer on a textual prompt, they can also be used for other inputs and outputs. For example, some recent works [15] have utilized the GPT architecture to caption medical images, and Diffusion models[16] also use the Transformer architecture to generate images from a prompt and random noise by iteratively denoising each step conditioning the model on the prompt.

2.2 Generative models for music

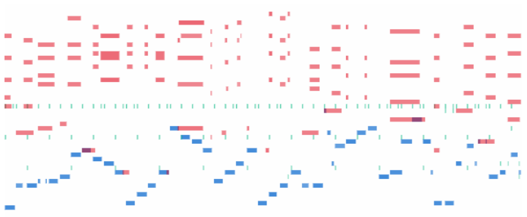
During the last few years, music has progressively been gaining interest as a field onto which to apply the most recent advances in Deep Learning, and the different models show the same range in format as sound itself. Figure 2.1 shows examples of different graphic representations of music, which are referenced and discussed along the following paragraphs.

However, artificial music composition and generation date back to before Deep Learning. The first algorithmic composition systems tried to predict the next note to be played in a melody based on transition tables. Neural Networks used for composition [17, 18] started by extending these functionalities with MultiLayered Perceptrons.

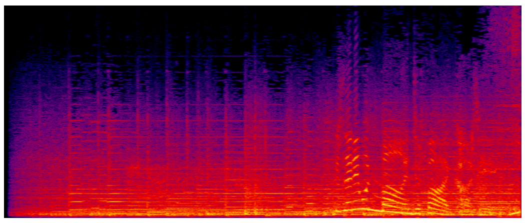
During the works that followed, the use of Neural Networks evolved from MLPs to Long Short-Term Memory Networks [19, 20] and eventually to Generative Adversarial Networks [10, 21]. Among the latter group, we encounter MuseGAN [10], a WGAN trained to create piano rolls (see Figure 2.1 (b)) with multiple instruments: Drums,



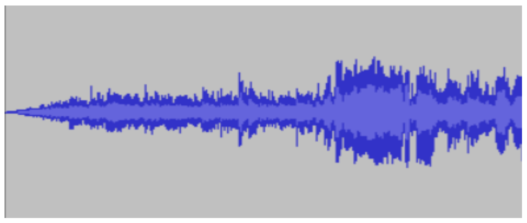
(a) Piano sheet



(b) Piano roll



(c) Spectrogram



(d) Waveform

Figure 2.1: Different graphic representations of music. Image sources: (a) from Claude Debussy's 'Claire de Lune' (public domain) at MuseScore, (b) from 'html-midi-player' by user *cifkao* at GitHub, (c) and (d) by the author.

Piano, (bowed) Strings, Guitar and Bass. The architecture of the model is built to extract features of the piano rolls both along time, along tone and both, as well as for each instrument individually and jointly. The output of the model is a piano roll with a track per instrument. This output can later be converted into MIDI format.

Coconet [9], a later model that also works on composition, approaches the process trying to imitate that of actual human composers. Instead of writing a whole score following the direction of time like previous models, the authors favor iterating and rewriting, like a human composer. They achieve this by training an instance of the NADE [22] density estimator to fill gaps in Bach chorales and sampling the space to fill in a way that is not tied to any causal direction.

At this point, music generation models started to focus on audio generation, abstracting the composition process. MelNet [23], a model contemporary to Coconet, works with Mel-Spectrograms. Spectrograms are an alternative representation of sound, which consist of a 2D decomposition of the soundwave on different frequencies that vary in amplitude over time. These spectrograms are converted to Mel scale, a quasi-logarithmic scale that places frequencies at the distances humans perceive them (i.e. tones). MelNet trains over these inputs and is able to recreate realistic piano solo performances.

Wrapping up, two of the latest works focus on the generation of music in raw audio format and use the Transformer architecture. AudioCraft is a combination of three models: EnCodec [24], a neural audio compressor, and two text-to-sound generators, AudioGen [6] for general sound and MusicGen [7] for music, which rely on both a text prompt and the feature space of the first model to finally create a 15 seconds long final waveform. Jukebox [8] takes the lyrics of a song without any temporary alignment and a text description of the music. The model then outputs the final waveform with both the instrumental and sung part of the music, which can last up to several minutes. It is worth mentioning that the timing of the lyrical part of the result is completely up to the model, which can lead to some surprising results such as deciding to repeat a line to make it match the music. The performance of these two models is outstanding, as the use of raw audio (see 2.1 (c) and (d)) as output allows for the capture of many interesting sound effects. However, this comes with a loss of control with respect to models that work solely on composition, and any desirable change resulting in a need for heavy prompt engineering.

Coming back to our own work, we need a baseline model to perform optimization on. After reviewing the distinct related works, we chose to select a composition model as baseline rather than an audio generation model. We considered that selecting a composition model would suit better a human-feedback approach, since raw audio generation adds too many variables non-exclusive to music to take into account (e.g. noise, timing...). Out of the presented composition models, MuseGAN was the one we considered most adequate for the task at hand. This model holds the 'most implemented' distinctive in Papers With Code under the 'Music Generation' category at the time of writing this thesis. Despite not being the latest model, MuseGAN is still a gold standard due to its capacity for managing and composing for multiple instruments, which can provide enough variety in the generation for us to explore without giving up control.

2.3 Bayesian Optimization

Bayesian Optimization (BO) is a global optimization technique specially useful for optimizing black-box functions, which is of great interest for machine learning algorithms and other fields. The principle of this technique is the approximation of an unknown function using a surrogate model (namely a Gaussian Process, GP) based on previous observations, and the use of an acquisition function to look for maxima in this approximation. This yields advantages such as sample efficiency and global optimality. The objective function is not assumed to be of any particular shape, and its derivatives are not needed for optimization. Other optimization methods, such as Gradient Descent, are not global or need these derivatives for optimization, and other search methods, like grid or random searches, rapidly fall behind when evaluations are expensive to make.

Bayesian Optimization can be used in many applications [25, 26, 27], including robotics, machine learning, and more. For the case of robotics, BO has been used for adding robustness to policy search agents [27]. In their work, Garcia et al. envelop the execution of a path finding policy in a black-box function and add noise to the control. The use of BO allows the agent to learn a safe policy even when facing imprecision when following it.

We find another case of use for BO in deep learning. In this case, BO can be used as a hyperparameter tuning technique [28]. The objective function in this case is a mapping from the hyperparameters (e.g. learning rates, batch sizes, multiple loss weights...) to the final loss of the model (or some other evaluation metric). Each of the observations would consist of a complete training period of a given model using these hyperparameters, which can be quite long depending on the architecture and the training data. Leveraging the sampling efficiency of BO, one can greatly reduce the number of evaluations needed to find the optimal combination of hyperparameters for a model.

Another interesting example is the use of BO in the training of AlphaGo[29], an artificial intelligence capable of playing the board game Go at professional level, and the first to beat a world champion at it. Though deep learning is also involved in this case, BO was not used to tune hyperparameters. Instead, the authors used BO to tune the parameters for Monte Carlo tree search, which are related to game state exploration and play selection. With this technique, the authors were able to raise the win rate of AlphaGo when playing against itself from 50% to 66.5%.

Other works focus on Bayesian Optimization as a way to take into account human judgement. An example of this application is encountered in the field of computer graphics. Brochu et al. propose an evaluation system for procedural smoke animations based on user preference [30]. Users are shown several simulations of smoke particles following curly velocity fields generated with the parameters to be tuned by the optimization, and are asked to give them scores in order to find one that, in their opinion, looks the best. In that same work, the authors later propose an alternative evaluation system based on preference of some samples over others, instead of individual evaluations. A similar approach is used in other works also in the graphics domain, where the authors use this technique to model material appearance [31]. This particular form of BO, where the evaluation does not yield a scalar but rather a relation between samples, receives the name of Preferential Bayesian Optimization (PBO).

This technique solves some issues that arise when working with human judgement such as *drift*, where opinion changes scale over time, or *anchoring*, as the first experiences influence the next ones. However, it also comes with a series of limitations. For once, the surrogate model becomes more complicated to elaborate, and more comparisons are needed in order to get a model that is representative enough. This is a notable setback taking into account that one of the main advantages of BO is its efficiency.

As last case worth of special mention, there is one work where PBO is used for the optimization of music generation. In [32], the authors build an interface for melody composition, onto which the user can create a melody and then select a section for the system to propose variations. By choosing their preferred variation of the selected fragment, the user feeds the Bayesian Optimization algorithm with several comparisons to build the surrogate model.

Chapter 3

Method

3.1 Music generation model

Our method's foundation is a music generation model. Among the candidates for this task, we selected MuseGAN, a WGAN capable of generating MIDI-like pieces, commented in the previous chapter.

Formally, a GAN is composed of two agents, a generator G and a discriminator D, facing each other a minmax game

$$min_G max_D V(G, D) = E_{X\sim P_x}[\log D(x)] + E_{Z\sim P_z}[1 - \log D(G(z))] \tag{3.1}$$

where G tries to minimize and D tries to maximize the value function V. G takes an input z from a latent space Z that follows a distribution P_z, and creates counterfeit samples $G(z)$ that should emulate the real samples X from the real distribution P_x. Meanwhile, D examines both the real and counterfeit samples, and outputs a probability $D(x)$ and $D(G(z))$ for each of them to be real. Thus, a perfect D has for outputs $D(x) = 1$ and $D(G(z)) = 0$, and the target scenario for G is $D(x) = D(G(z)) = 0.5$.

For a Wasserstein GAN, the game is reformulated as

$$min_G max_C V(G, C) = E_{X\sim P_x}[C(x)] - E_{Z\sim P_z}[C(G(z))] \tag{3.2}$$

where D becomes a critic C, and is no longer constraint to output a probability (i.e. can output values below 0 and above 1). This new version of the problem prevents both C and G from reaching a dead end before being able to learn meaningful features.

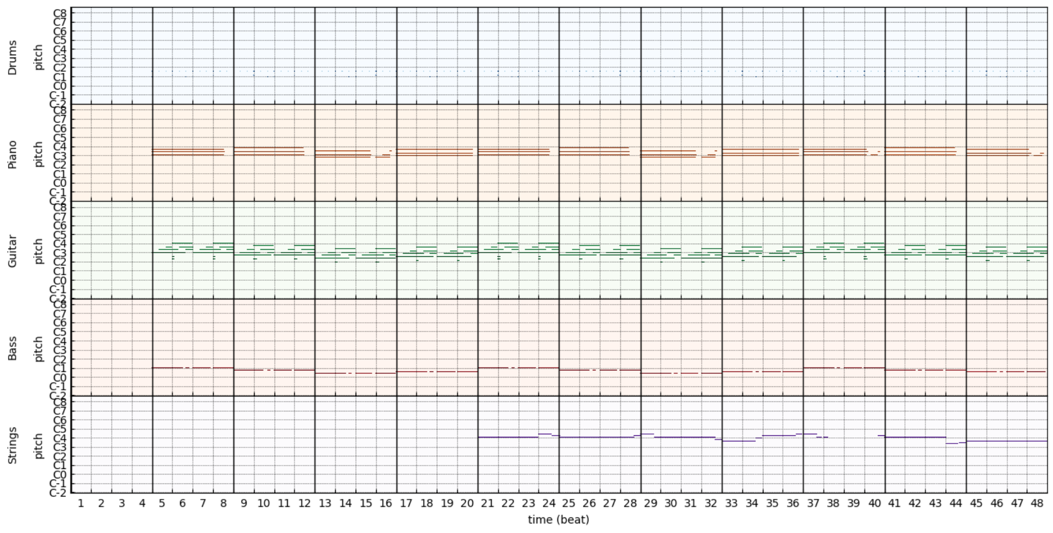
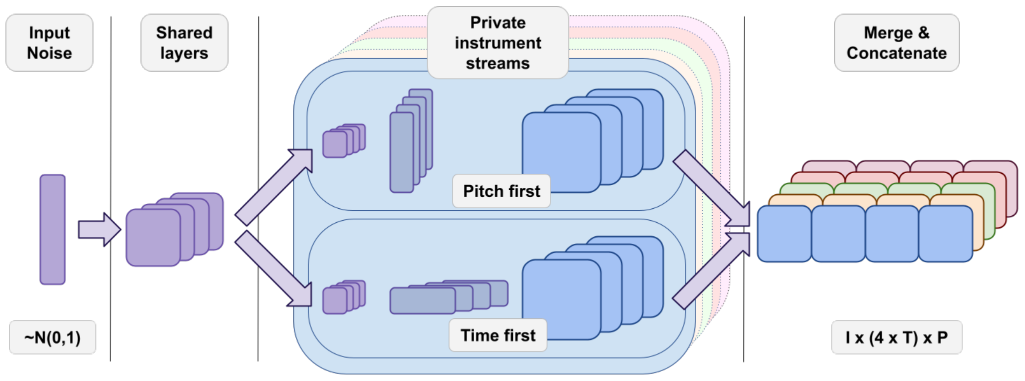


Figure 3.1: Example 12-bar long piano roll from the Lakh dataset. The horizontal axis represents time and the vertical axis represents pitch. Bars are separated by black vertical lines. Different colored bands correspond to different instruments. Top to bottom: Drums, Piano, Guitar, Bass and Strings.

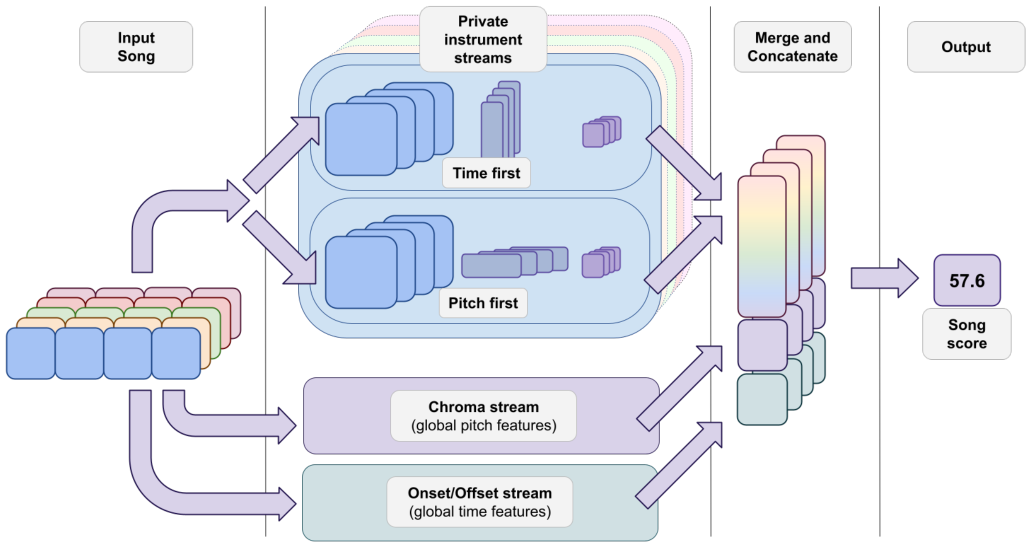
In our case, Z follows a multivariate Gaussian distribution. The generator takes Gaussian noise as input and produces 4 bars (also referred as measures) of music with 5 different instrument tracks. Each track consists of a two-dimensional matrix representing time and pitch, as shown in Figure 3.1. Each bar is divided into beats, and beats are divided into timesteps. An array of binary values indicates whether each tone is playing or silent at each timestep. For the true dataset, this values are binary. The generated pieces take a value between zero and one, and then they are binarized using thresholds. The discriminator takes said matrices, extracts features from them and then outputs a scalar score for the whole composition. The discriminator aims to maximize the score difference between the dataset samples and the generator's, while the generator aims to minimize it.

3.1.1 Baseline versions

The MuseGAN repository provides a TensorFlow pretrained implementation of the latest model, as well as a notebook with PyTorch code to implement it from scratch, though the latter lacks some of the latest additions. The models show differences in the dimensionality of the input and the output, as well as the architecture. Figure 3.2 shows a diagram featuring the distinct parts of the generator and the critic for the Tensorflow architecture, which is the most complete and close to the final selected model.



(a) Generator architecture. The model takes Gaussian noise, runs it through several common 3D convolutions and then replicates and splits the data for each instrument. When every instrument stream has produced 4 bars of music, bars are concatenated and instruments are stacked. The output is a 3D matrix, where the dimensions represent Instrument, Time and Pitch.



(b) Critic architecture. The model takes the input song, runs each instrument through a convolutional stream, and concatenates the result with global features extracted from other two streams. The resulting features are run through Fully Connected layers and a scalar score is returned for the song.

Figure 3.2: MuseGAN architecture. The generator's pitch-first stream, and the critic's time-first, Chroma and Onset/Offset streams are exclusive to the Tensorflow architecture.

Regarding the time and pitch resolution, the TensorFlow implementation uses bigger sizes for the data and the output. The generated songs are 4 bars long for both, but the TensorFlow implementation divides them into 48 timesteps each, while the PyTorch implementation uses 16. Finally, the PyTorch implementation ignores the lowest scale of the 7 scales of the pitch dimension, with a total of 72 pitch values instead of 84.

	Kernel	Stride	Output	Channels
Shared network	(4,1,1)	(4,1,1)	(4,1,1)	512
	(1,4,3)	(1,4,3)	(4,4,3)	256
	(1,4,3)	(1,4,2)	(4,16,7)	128
Private pitch-first	(1,1,12)	(1,1,12)	(4,16,84)	32
	(1,3,1)	(1,3,1)	(4,48,84)	16
Private time-first	(1,3,1)	(1,3,1)	(4,48,7)	32
	(1,1,12)	(1,1,12)	(4,48,84)	16
Private merged	(1,1,1)	(1,1,1)	(4,48,84)	1

Table 3.1: Layer breakdown showing the convolutional layers of the Tensorflow implementation generator. The shared network is common for all instruments, while all the rest of the layers are replicated for each instrument. Private pitch-first and time-first layers are parallel and then merged with a 1x1 convolution.

	Kernel	Stride	Output	Channels
Shared network	(4,1,1)	(4,1,1)	(4,1,1)	256
	(1,4,1)	(1,4,1)	(4,4,1)	128
	(1,1,4)	(1,1,4)	(4,4,4)	64
	(1,1,3)	(1,1,1)	(4,4,6)	32
Private time-first	(1,4,1)	(1,4,1)	(4,16,6)	16
	(1,1,12)	(1,1,12)	(4,16,72)	1

Table 3.2: Layer breakdown showing the convolutional layers of the PyTorch implementation generator. The shared network is common for all instruments, while all the rest of the layers are replicated for each instrument.

Architecture-wise, both the generator and discriminator are multi-stream 3D Convolutional Networks in both cases. Data is represented as a tensor of batch size B, I instruments, M measures, T timesteps and P pitch values. I is considered the channel dimension, so only M, T and P are affected by the convolutions, as depicted in Tables 3.1 and 3.2. The generator's architecture consists of an initial part shared among instruments that splits into private streams for each instrument. In the TensorFlow case, each instrument has two private streams that are merged at the end to produce each instrument's track, while the PyTorch implementation uses only one. For the first case, each private stream of each instrument expands the dimensions of the data in different order, pitch-first and time-first. For the second, convolution order is time-first only. Despite the data being 3D, convolutions are always 1 or 2-dimensional in practice. At least one of the dimensions is fixed at every layer, and the M measures are separated at the beginning of the architecture for both models.

Also, though measures and timesteps are merged into a single time dimension in the end, they are treated separately. This allows the model to break continuity when a new measure starts, fitting the beat better.

The critic's architecture is similar, mirroring the generator's. The TensorFlow version adds two new streams that extract features from the whole score. The original authors named these streams the 'Chroma' stream and the 'Onset/Offset' stream.

The Chroma stream folds each track into scales (bands of 12 pitch values) summing the values of coincident tones in different scales. This feature provides the model with information about the different chords of the composition, while focusing on non-empty scales. The Onset/Offset stream extracts the difference between consecutive timesteps of each track. This feature relates to the instructions present in MIDI compositions, which tell when each note should be turned on or off. Both features are passed through convolutional layers before merging with the instrument streams.

3.1.2 Model modifications

After reviewing the different versions of the model, we arrived at the conclusion that five instruments, times the temporal and tonal resolution of the complete model, was too complex for humans to evaluate thoroughly. However, it was desirable to keep the architecture additions of the Tensorflow implementation, as the complete model produced pieces that better resembled the original dataset and, according to the authors, had higher quality. Thus, we decided to use the PyTorch implementation and modify the model and the dataset.

Our version of the model has a reduced number of 3 tracks, cutting out Strings and Piano, to make it easier to evaluate the tracks. We implemented the pitch-first and time-first streams of the generator and the critic, as well as the Chroma and On/Offset stream of the TensorFlow architecture into PyTorch. Finally, we decided to keep the reduced resolution of the PyTorch model, adapting all the new streams to the new output sizes. All the differences between our model and the original implementations are gathered in Table 3.3. It is worth noting that we also attempted to reduce the dimension of the input to make optimization easier. However, as we will comment on the next chapter, results experienced a downgrade in variety when attempting this.

	MuseGAN (complete)	**MuseGAN (from scratch)**	**Ours**
Framework	Tensorflow	PyTorch	PyTorch
Num. of tracks	5 (D,B,G,S,P)	5 (D,B,G,S,P)	3 (D,B,G)
Input dimension	128	128	128
Output dimension (per track)	(4,48,84)	(4,16,72)	(4,16,72)
Generator streams	2*N (10)	N (5)	2*N (6)
Critic streams	2*N+2 (12)	N (5)	2*N+2 (8)

Table 3.3: Feature comparison between all models of our baseline. First column corresponds to the pre-trained Tensorflow implementation, second column to the PyTorch notebook implementation, and third column to our custom version. The letters in the 'Num. of tracks' row are the initials of the present instruments (Drums, Bass, Guitar, Strings and Piano), and N equals the number of tracks

3.2 Latent space reduction model

When performing optimization on a limited budget, like our case, it is desirable for the domain of the problem to be small to explore it thoroughly and fast. As in our case the amount of samples we can take within an acceptable time frame is limited, Optimizing our model's 128-dimensional input space using human evaluations is simply out of the question. In fact, most Bayesian Optimization scenarios focused on small sample sizes do not exceed 20 dimensions to optimize. In addition to this issue, a Generative Adversarial Network's latent space is not necessarily smooth, unlike other generative models. This means that two similar samples can be separated in the latent space. Considering this, a reduction of the GAN's latent space to a smaller, smoother one is in our best interest.

Our approach to this issue was building a Variational Autoencoder that used the pre-trained GAN as part of the architecture. The GAN's generator would act as the decoder, and the critic as the encoder (eliminating the critic's output layer). The intuition behind this is that, after being trained, the critic should be able to extract meaningful features out of the piano rolls in order to give its scores to the pieces. Then, adding trainable layers between its tail and the generator's input would suffice to map these features to a small, ordered latent space, and then back to the generator. Figure 3.3 shows a diagram depicting how the MuseGAN model can be salvaged into this new architecture.

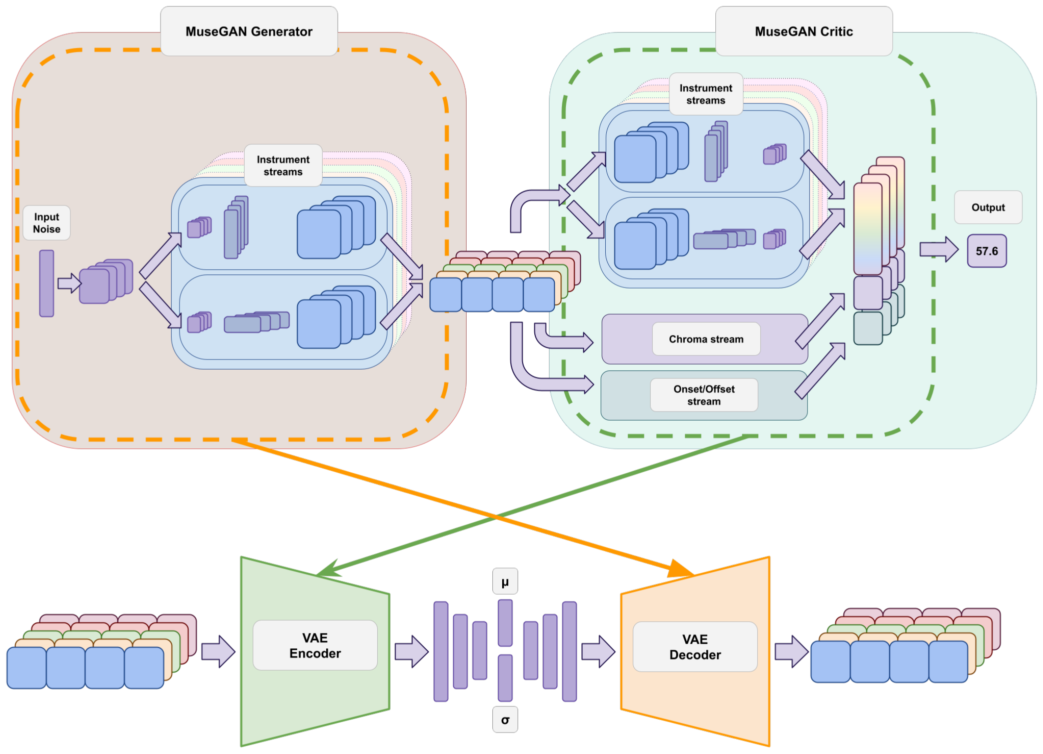


Figure 3.3: Diagram depicting how the MuseGAN architecture can be transformed into a Variational Autoencoder. The pretrained generator and critic from the GAN can be used as the decoder and encoder of the VAE, respectively. For the encoder, the critic is beheaded and a middle layer is used as output. Additional middle layers are included in between to reduce dimensionality and impose the Gaussian prior on the new latent space.

VAEs are trained to minimize a loss function such as

$$L_{VAE}(x, x') = E(x, x') + \beta_{KL} * KL(Z, N(0, 1)) \tag{3.3}$$

where $E(x, x')$ is the reconstruction error between the real sample x and the reconstructed sample x' and and KL is the Kullback-Leibler divergence between the encoded latent space distribution Z and a multivariate standard Gaussian distribution, times a weight factor β_{KL}. In other words, minimizing this loss means reducing the reconstruction error while imposing the Gaussian prior on the latent space. The choice of a reconstruction error function should be affected by the domain of the problem and the resulting model can change depending on this decision.

Following on this line of work, we also implemented a VAEGAN [33], a combination of both a Variational Autoencoder and a Generative Adversarial Network. The VAEGAN is composed of an encoder, a decoder and a critic. The encoder and decoder take the place of a GAN's generator, and at each pass, the critic must tell the original samples from their encoded and decoded versions.

Training is performed in pairs of modules. The decoder and critic are trained following the original GAN's minmax game, while the encoder and decoder are also trained with a learned similarity measure, which instead of measuring the reconstruction error like a VAE, it measures the difference between the features extracted by an intermediate layer of the critic.

As our last approach, we also tried another dimensionality reduction technique oriented to problems solved Bayesian Optimization: Random Embedding Bayesian Optimization (REMBO)[12]. REMBO assumes that the effective dimensionality of the problem is smaller than that of its domain, and thus a low-dimensional random embedding of the space can contain the global optimum, which will be easier to find if optimizing over the embedding rather than over the whole domain. We explain the method more in depth in the next section.

3.3 Bayesian Optimization

3.3.1 BO Algorithm

Bayesian Optimization is our tool of choice for exploring the latent space of the developed model. BO approximates an unknown function f using a probabilistic surrogate model $P(f)$, commonly a Gaussian Process (GP). A Gaussian Process is a collection of random variables such that every finite subset of them has a multivariate normal distribution, and it is defined by its mean $\mu(.)$ and covariance $k(.,.)$ functions.

The GP is built on a collection $\mathcal{D}_n$ of n previous observations of f, and its posterior probability is updated every time a new observation is acquired. The posterior model allows predictions y_{pred} at query points x which are normally distributed $y_{pred} \sim N(\mu_n(x), \sigma_n^2(x))$ such that

$$\begin{aligned} \mu_n(x) &= \mu(x) + \boldsymbol{k}^T(K + \mu^2 I_n)^{-1}\boldsymbol{y} \\ \sigma_n^2(x) &= k(x,x) - \boldsymbol{k}^T(K + \mu^2 I_n)^{-1}\boldsymbol{k} \end{aligned} \tag{3.4}$$

where $\boldsymbol{y}$ is the vector of evaluation results, $\boldsymbol{k}$ is a vector with $k_i = k(x, x_i)$ and K is the kernel matrix with $K_{ij} = k(x_i, x_j)$ for $i, j \in \{1, ..., n\}$. There exist multiple choices for $k(.,.)$ but among the most popular we can find the squared exponential (SE) kernel and the Matèrn kernel:

$$\begin{aligned} k_{SE}(x,x') &= \exp\left(-\frac{1}{2}r^2\right) \\ k_{M,v}(x,x') &= \frac{2^{1-v}}{\Gamma(v)}(\sqrt{2vr})^v K_v(\sqrt{2vr}) \end{aligned} \tag{3.5}$$

where Γ is the gamma function, v is a tunable parameter that controls the derivability of the GP and $r^2 = (x - x')^T \Lambda(x - x'))$ with some positive semidefinite Λ. Frequently, $\Lambda = diag(\theta_l^{-1})$ where θ_l are the lengthscales of the kernel, which measure smoothness. These lengthscales can be the same for all or different for each dimension. When each optimized dimension has its own lengthscale, the Matèrn kernel is given the surname of automatic relevance determination (ARD).

In addition to the surrogate model, BO also uses an acquisition function $a(x, P(f))$ for efficiently choosing the next evaluation point x_{t+1} at each step t of the algorithm, commonly following a greedy approach

$$x_{t+1} = \operatorname*{argmax}_x a(x, P(f|\mathcal{D}_t)) \tag{3.6}$$

As for examples of $a(x, P(f))$, one of the most extended ones is the Expected Improvement (EI) function

$$\begin{gathered} EI(x) = \mathbb{E}_{p(y|x,\theta)}[I(x)] = (\rho - \mu(x))\Phi(z) + \sigma(x)\phi(z) \\ z = (\rho - \mu(x))/\sigma(x) \end{gathered} \tag{3.7}$$

where ρ is the current best result among the observations of the GP and ϕ and Φ are the Gaussian PDF and CDF, respectively, and $\mu(x)$ and $\sigma(x)$ are obtained from Equation 3.4.

The algorithm starts fitting a surrogate model to the initial collection $\mathcal{D}$ of obtained evaluations of the objective function. Once the posterior of the GP is computed, the algorithm iterates for a number of iterations (the 'budget'). The budget is fixed before starting the algorithm and depends on the problem and the available resources, but it is presumed to be small. We compute the acquisition function based on this posterior, and look for the point that maximizes it to decide where to sample the next observation. Every time a new observation is sampled, it is added to the model and the posterior is updated. After the algorithm runs out of budget, the historic best point x^* within the model is returned.

In our case of use, the objective function f corresponds to human evaluations of the generated songs, to which we refer as 'taste', and $\boldsymbol{y}$ are numerical scores given by the users. As for initialization methods, the preliminary design can follow any desired strategy. Some examples are uniform sampling, Latin Hypercube Sampling or Sobol initialization, with the latter two being among the most popular.

Algorithm 1 Bayesian Optimization

```
Require: Objective function $f$, acquisition function $a$, budget $N$
1: Initialize data set $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$ with the data points $\boldsymbol{X} = \{x_1, x_2, \ldots, x_I\}$ and their
   responses $\boldsymbol{y} = \{y_1, y_2, \ldots, y_I\}$, using a preliminary design.
2: for $t = 1, 2, \ldots, N$ do
3:     Fit the surrogate model to $\mathcal{D}$: $P(f|\boldsymbol{X}, \boldsymbol{y}) \propto P(\boldsymbol{X}, \boldsymbol{y}|f)P(f)$
4:     $x_t = \text{argmax}_x\, C(x|P(f|\boldsymbol{X}, \boldsymbol{y}))$
5:     $y_t = f(x_t) + \epsilon$ //Where $\epsilon$ represents observation noise
6:     $\mathcal{D} = \mathcal{D} \cup (x_t, y_t)$
7: end for
8: Return $\text{argmax}_x(y|(x, y) \in \mathcal{D})$
```

3.3.2 Random Embeddings

As stated in the previous section, Bayesian Optimization does not escalate well with high dimension problems. This problem rises from the need for cover of the domain. The number of needed samples grows exponentially with the problem's dimension, which in turn makes the size of the K covariance matrix to also grow exponentially. Several works have addressed this issue over time and BO is applicable to high dimensions when combined with some other techniques.

One of these techniques is Random Embedding Bayesian Optimization (REMBO) [12]. REMBO works on the assumption that the objective function depends only on a subspace whose effective dimension is lower than the domain of the function, as pictured in Figure 3.4.

By stating that there is a vector $\boldsymbol{z}^* \in \mathbb{R}^d$ and a matrix $\boldsymbol{A} \in \mathbb{R}^{D\times d}$ such that the optimum x^* of the original latent space yields an observation $f(x^*) = f(\boldsymbol{A}\boldsymbol{z}^*)$, the dimensionality of the problem is reduced from the original D to a reduced dimensionality d. Our hypothesis is that a user's taste on music depends on less variables than the model's latent space (from now, S^D) is built on, at least when working on the limited range of pieces the model can create. Thus, we can create a random embedding to map an artificial low-dimensional space (from now, S^d) to the model's and find an optimum in this low-dimensional space instead.

In order to use REMBO, we need to ensure that the result of the product of any vector $\boldsymbol{z} \in S^d$ and matrix $\boldsymbol{A}$ follows a standard multivariate normal distribution, the same as S^D. However, BO assumes a uniform distribution of the domain, which we choose to bound between 0 and 1. Since $\boldsymbol{A}$ is constant, each row-by-column product yields a sum of d variables $v_i \sim U(0, A_{ij})$ for $i \in \{1, ..., D\}$ and $j \in \{1, ..., d\}$. Since this variables are not equally distributed, its sum does not necessarily yield a Gaussian distribution, so we cannot guarantee the restriction above holds.

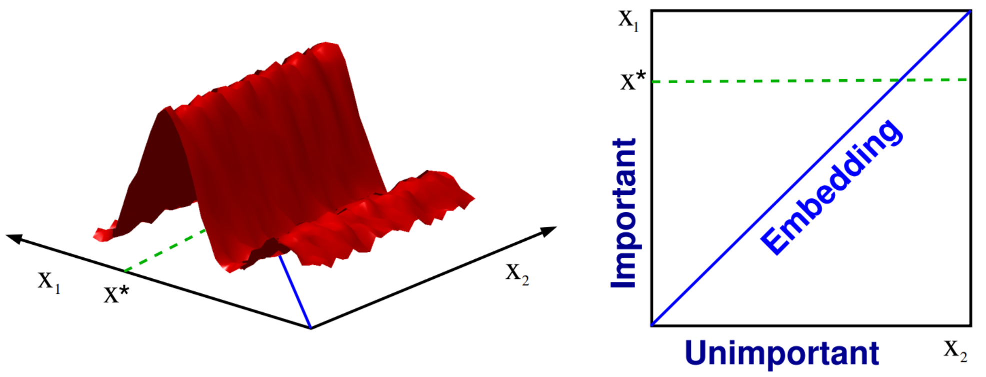


Figure 3.4: An example of a function over a 2-dimensional domain and an one effective dimension. Searching over the 1-dimensional embedding is more efficient than searching over the whole domain. Image sourced from [12].

This issue is solved by applying the Box-Muller transform to the $\boldsymbol{z}$ vector, transforming it into $\boldsymbol{z'} \sim N(0, I)$. Thus, each row-by-column product yields a sum of d variables $v_i \sim N(0, A_{ij}^2)$, which also follows a normal distribution, and can be standardized dividing by $\sqrt{\sum_{j=1}^{d} A_{ij}^2}$, which can be precomputed right after generating $\boldsymbol{A}$.

On top of enabling the mapping from the low-dimensional latent space to the high-dimensional one, the use of Box-Muller also helps covering the domain of the problem. Though the whole process is rather more complicated to explain, it is based on taking pairs of uniformly distributed data points and using them to sample polar coordinates, which in the end are converted to normally distributed Cartesian coordinates.

This has the effect of mapping a certain subspace formed by every polar coordinate with zero module to the origin, as depicted in Figure 3.5.

This means we can sample indefinitely within said subspace, gather the user's evaluation once, and replicate the result of one evaluation over every point we sample. We take advantage of this to further extend the initial dataset $\mathcal{D}$ before starting iterating and improve the model.

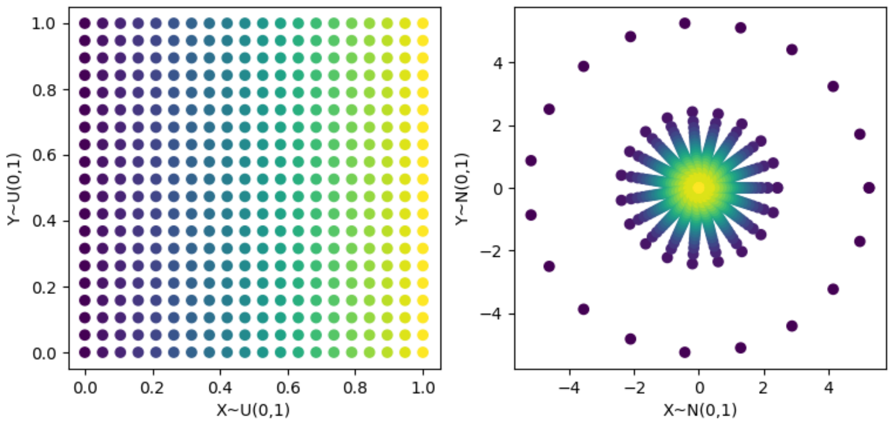


Figure 3.5: A 2-dimensional example of the Box-Muller transform. The uniformly distributed data (left) ends up following a Gaussian distribution (right) after being transformed. All data points with X=1 in the uniform distribution end up converging to the origin when becoming Gaussian.

Chapter 4

Experiments and Validation

In this chapter we report our training protocol for our custom piano roll generation model, as well as the evaluation protocol and results for our human-driven experiments. For the model, we present evaluation metrics selected from the ones used in the baseline's original paper. We showcase the Empty Bar Ratio, Used Pitch Classes, and Tonal Distance of the applicable tracks of the model, compared to both the dataset's metrics and the baseline's. Next we showcase the obtained results when trying to reduce the dimension of our problem, providing results and our gained insight on the limitations they have. Lastly, for the experiment, we present an explanation of how we built the interface for the experiments, the script used for carrying out the evaluation and optimization, examples of the results of user-generated pieces and the aggregated results of a survey presented to the participants with Likert-scaled answers regarding the quality of the generated pieces and their own opinion about their capacity to evaluate said pieces.

4.1 Music generation

4.1.1 Training protocol

Our custom model was trained over 4-bar-long fragments of music, which at a reasonable speed of 100 beats per minute, equals roughly ten seconds. The fragments are extracted from the piano rolls present in the Lakh Pianoroll Dataset. This dataset is the result of the baseline authors' preprocessing of the Lakh MIDI Dataset[11]. The dataset is composed of 174.154 multitrack pianorolls. All the songs that made it to this curated dataset had the 'Rock' tag in the original dataset and 4/4 tempo. The original instruments in the MIDIs are matched to one of the five selected instruments (Drums, Piano, Guitar, Bass and Strings) in the piano rolls.

The authors provide code to extract 4-bar fragments from the dataset with the desired time resolution and pitch range. Each composition is divided in 4-bar-long fragments, which are uniformly sampled (without repetition) until a target number of samples per song is reached or no more fragments remain. We chose to extract a maximum of 18 songs. Our protocol includes an additional preprocessing step where we delete the Piano and Strings tracks, and also make sure that the resulting fragment is not completely silent, to avoid accidentally creating empty fragments by deleting these tracks. If it occurs, we select fragment until we find an appropriate one, or we run out of candidates. In total, our version of the dataset contains 57,201 fragments. For most models, this dataset would be divided into training, validation and test subset. However, GANs do not need this distinction. Since we are only interested in the generator, which doesn't use the real samples directly, the whole dataset was used for training.

Training was performed entirely on Google Colaboratory, using a NVIDIA T4 GPU environment, and is fully reproducible with the code provided in the GitHub repository referenced in Chapter 1. The final version of our model was trained over 25000 steps with a batch size of 32. The first ten training steps consisted of fifty critic passes per generator pass, and each one onward consisted of five critic passes per generator pass. A head start for the critic prevents the generator from taking advantage in early stages of training and improves training stability. We also placed a Gradient Penalty constraint to prevent exploding or vanishing gradients to appear when using the WGAN. Both the generator's and the critic's parameters are optimized using Adam [34] with a learning rate of 0.001, and parameters $\beta_1 = 0.6, \beta_2 = 0.95$.

4.1.2 Model evaluation

To validate the training of our custom model, we used a selection of the metrics used in the original MuseGAN paper:

- Empty Bar Ratio (EB%) measures the percentage of bars that are completely silent. It ranges from 0 to 1. Applies to all tracks.
- Used Pitch Classes (UPC) measures the number of different notes used in each pitched track. The same note played in different scales counts as only one note. It ranges from 0 to 12. Applies to Guitar and Bass.
- Tonal Distance [35] (TD) measures harmonic relations between pitched tracks. The lower the TD, the stronger the relation. Applies to the Guitar-Bass pair.

		EBR (%)			**UPC**		**TD**
		D	**G**	**B**	**G**	**B**	**G-B**
MuseGAN	Dataset	8.06	19.4	8.06	3.08	1.71	1.57
	Model	2.33	18.3	6.59	3.69	1.53	1.56
	Abs. Error	5.73	1.1	1.47	0.61	0.18	0.01
Ours	Dataset	6.4	20.07	8.74	3.43	2.25	1.27
	Model	8.5	18.75	6.53	3.22	2.01	1.36
	Abs. Error	2.1	1.32	2.21	0.21	0.24	0.09

Table 4.1: Metric comparison between the used dataset and the closest values reported in evaluation, both for the MuseGAN baseline and our final model. Values of the 'Model' row are better the closer to the 'Dataset' row. Notice that both datasets show different metrics due to random sampling, track trimming and sample rejection. For the columns, 'D', 'B' and 'G' correspond to Drums, Bass and Guitar.

We decided to not take into account the other two metrics used in the original paper: Qualified Note Ratio (QN) and Drum Pattern (DP). QN measures the percentage of notes that last longer than a 32nd note. Any note shorter than this is cosidered undesirable. Since the time resolution of our output is reduced and each bar is divided in 16 timesteps, the shortest note our version of the model can generate is a 16th, which is twice long as a 32nd. Because of this, our model is incapable of generating such undesired notes, and the calculation of the metric is trivial. DP measures the percentage of bars with an 8- or 16-beat drum pattern. For the same reason than QN, the variability of drum patterns in our model is considerably smaller than in the original, so we decided to discard this metric as an evaluation objective.

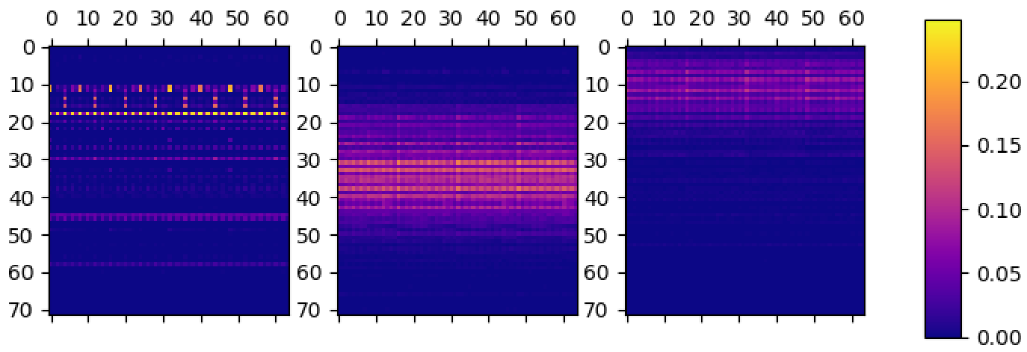
Note that the resolution and sampling changes affect the metrics of the dataset itself, so instead of trying to replicate the results reported in the original paper, we aim for the generated pieces to report values as close as possible to our extracted dataset. Metrics for our generator were obtained evaluating 5000 generated samples, which equals the number used in the original paper.

Table 4.1 shows the values reported by the dataset for each metric next to the ones obtained by the model. For reference, we also include the values for the dataset in the MuseGAN paper and the closest values in their evaluation. The reported values indicate that our model was trained correctly, obtaining metrics similar in scale and distance for the dataset and model, and performing specially well for the Drums, where we obtain a far smaller difference for EBR.

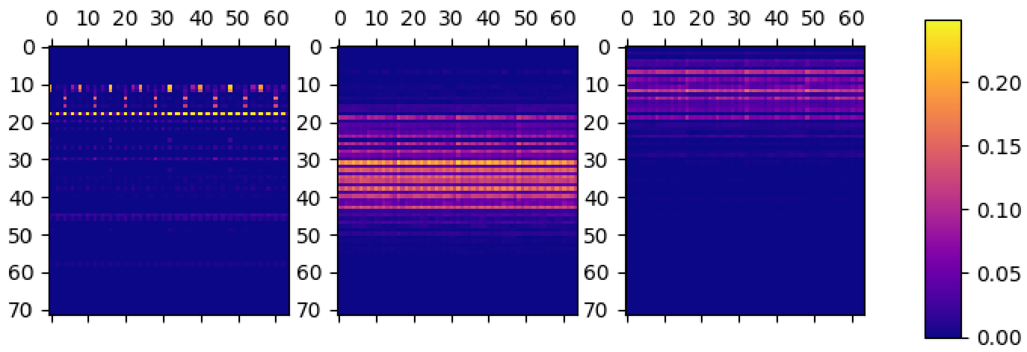
4.2 Latent space reduction

4.2.1 GAN input reduction

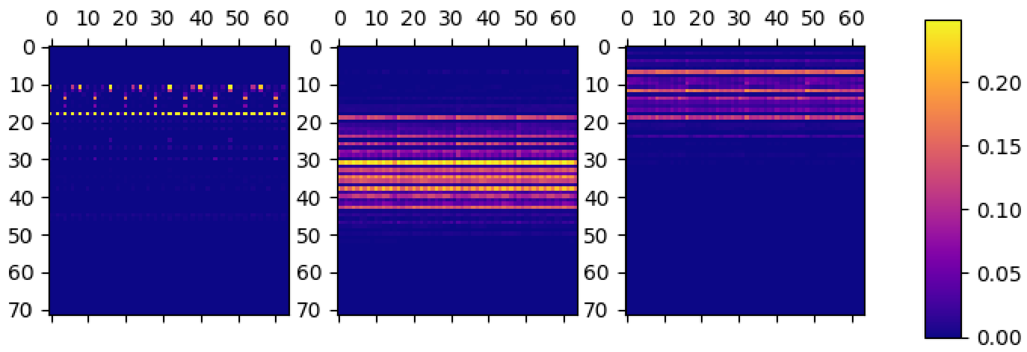
Our first attempt to reduce our model's latent space was to explicitly shrink the input of the model by adding more layers at the beginning of the architecture for upsampling until the data matches its former size. We attempted to reduce it to 64 and 32 dimensions.



(a) Sample variance for 128-dimensional input model.



(b) Sample variance for 64-dimensional input model.



(c) Sample variance for 32-dimensional input model.

Figure 4.1: Variance of 5000 sampled songs for several versions of our model with varying input dimension, for Drums (left), Guitar (middle) and Bass (right). For each plot, the horizontal axis corresponds to time and the vertical axis corresponds to pitch. Each timestep-pitch pair is treate as a Bernoulli distribution.

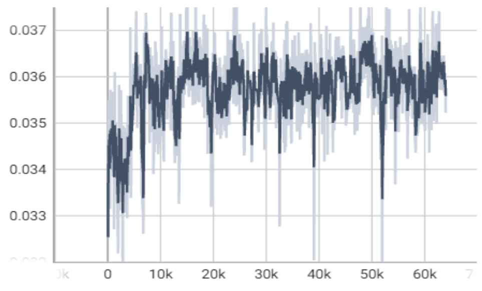
For both cases we added the needed layers as Fully Connected layers with the desired dimension as input and the next size (i.e. from 64 to 128 and from 32 to 64 channels).

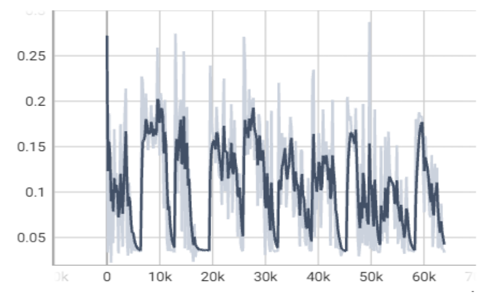
The evaluation metrics showed no notable changes, with little difference with respect to the original dimension of the problem. However, the model showed a qualitative downgrade regarding to variation when generating their tracks. To measure this, we sampled 5000 songs, interpreted each of the notes at every timestep as a Bernoulli distribution and plotted the variance. This variances are obtained as $p(1-p)$, where p is the probability of a positive value, which for our case equals the average of all 5000 samples. Figure 4.1 shows a plot of this variance for the different versions of the model.

Reducing the dimension of the input makes the variance of the resulting songs to group onto some notes over others, meaning that only a reduced number of notes has a tendency to change between songs, and most other stay the same (either always playing or always silent). After obtaining this results, we came to the conclusion that reducing the problem size in this way would not be an adequate method for our task. We considered that the variety of samples was needed for the users to find a suitable optimum, and since the latent space would continue to not be smooth either way, the tradeoff between problem size and variety was not worth to assume.

4.2.2 Variational Autoencoders

For the case of VAEs, we implemented several architectures. For all of them, the encoder was built on our GAN's critic and the decoder on the generator. We added middle Fully Connected layers to (potentially) reduce the dimension, impose the Gaussian prior, and upsample back to the generator's input dimension. For our starter models, we maintained a 128-dimensional input. Based on previous observations of the GAN model, we decided to first focus on smoothing the latent space before trying to reduce it. All models described onward were trained on the same conditions: an Adam optimizer with a learning rate of 1, parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and using a weight decay of $1e-4$. The models were trained using the same dataset as the WGAN, with the exception of selecting and separating a random tenth of the data for validation. Our first choice for a loss function was measuring the reconstruction error using the Mean Absolute Error between the real and reconstructed samples. We also varied β_{KL} periodically during training [36], increasing and being reset at several points. This lets the model focus on learning a meaningful starter encoding before imposing the prior.


(a) Training MAE


(b) Validation MAE

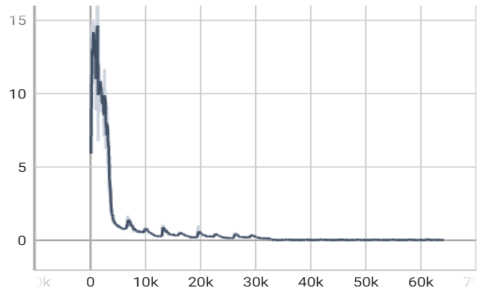

(c) KL divergence

Figure 4.2: MAE curves and KL divergence (vertical axes) evolution along training steps (horizontal axes). While KL divergence is reduced, MAE shows no improvement.

After analyzing the loss curves in Figure 4.2 we arrived at the conclusion that the VAE was not training properly. Despite KL divergence being visibly reduced, the MAE shows no sign of being minimized, except for periodic peaks that correspond to steps where β_{KL} is reset to 0. There's also an important difference between training and validation losses. The obtained reconstructions were not cut out for the task, either, with visually obvious errors. We elaborated several hypotheses for this.

Firstly, the pretrained generator could not be used as the decoder of the VAE as is, since GANs tend to not replicate their training data [37] as it grows in size and complexity. If the generator was left frozen, the implemented model showed any improvement from training, and it was unable to reconstruct any of the input piano rolls. These results suggested that the data was indeed too complex to be imitated by the generator, and that, should it be used, it should be unfrozen at some point. Our second hypothesis was Fully Connected layers being insufficient for ordering the features. In response, we started comparing all future models using FC layers and Self Attention layers. Our third hypothesis was underperformance being due to a bias in the dataset distribution. In our case, negative values, or zeroes, correspond to silence, and since notes in the piano roll are sparse, silence becomes an easy bet for the model. The obtained results suggested that reconstruction quality couldn't be properly evaluated using Mean Square Error, and other attempts with any similar element-wise function reported similar results.

We tried to address this by using a more complex loss: ASL [38]. This Asymmetric Loss rewards true positives above true negatives, and punishes false negatives over false positives. The loss formula is computed as

$$\begin{gathered} ASL = \begin{cases} L_+ & = (1-p)^{\gamma_+} \log(p) \\ L_- & = (p_m)^{\gamma_-} \log(1-p_m) \end{cases} \\ p_m = max(p-m, 0) \end{gathered} \tag{4.1}$$

where L_+ and L_- are the losses for positive and negative values, which are summed to form the final loss, p is the model's output for each value, and γ_+, γ_- and m are tunable hyperparameters. m is a negative clipping parameter, with any values lower than m being set to zero. γ_- and γ_+ are the asymmetric focus parameters that control how much negative predictions affect positive values and vice versa.

The authors recommend to set $\gamma_- > \gamma_+$. After experimenting with the hyperparameters, we settled for $m = 0.05$, $\gamma_- = 16$ and $\gamma_+ = 4$. We supported this decision by defining a series of cases, listed on Table 4.2, and ordering them in decreasing desirability.

After unfreezing the generator and applying the new loss, the model showed a sudden and heavy improvement, converging in few steps as shown in Figure 4.3. Self Attention layers also seemed to positively affect the model, smoothing the curve and reaching lower losses. However, the designed loss did not reach the global minimum. Further tests confirmed that changing the moment of unfreezing the model led to different final losses, what next to the qualitative underperformance of the reconstructions suggested that the model was falling for local minima.

Despite trying several techniques to address this problem (e.g. label smoothing [39], cyclical KL annealing [36] ...) the model seems to still converge to these minima. We distinguished that the reconstructed pieces featured notes that followed repeating incorrect patterns around the general location of the notes, as pictured in Figure 4.4. After considering possible reasons for it, we ended attributing it to VAEs' tendency to blur their reconstructions. This is not desirable when working with binary outputs like piano rolls. Despite having a spatially close disposition of the notes, the reconstructed samples sound completely different from the originals. In the end, piano rolls are 2D matrices with values within a [0,1] interval, much like other common-use datasets like MNIST, and they can be regarded as images, if we equate the instrument tracks to color channels.

Case	Case description	ASL	MAE
Perfect	Notes are 1, silences are 0	0.5	0
Wrong note	Notes are 1, all silences are 0 but one	0.52	7e-5
Low uncertainty	Notes are 0.9, silences are 0.1	0.75	0.1
High uncertainty	Notes are 0.6, silences are 0.4	2.74	0.4
Mean	All values are 0.5	4.61	0.5
Random	All values follow ∼U(0,1)	9.79	0.5
Silence	All values are 0	12.67	0.02

Table 4.2: List of cases to validate the Assymetric Loss function parameter selection, next to their ASL and MAE losses for a batch of songs. Cases are ordered in descending correctness. The values for ASL show a growing order, while the values for MAE show undesirable values for the Random and Silence cases.

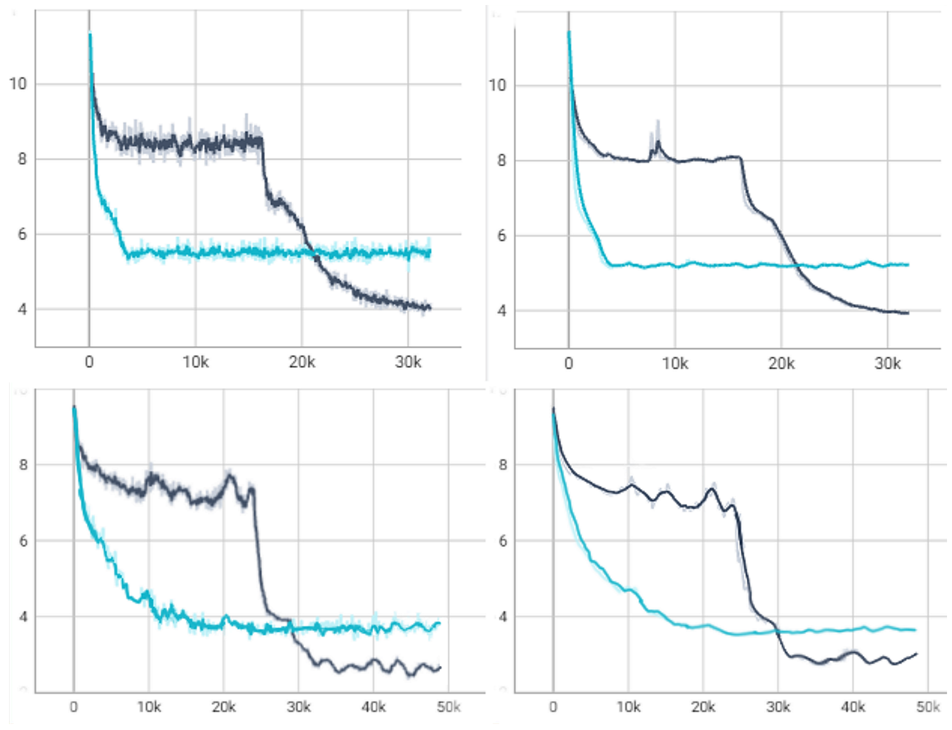
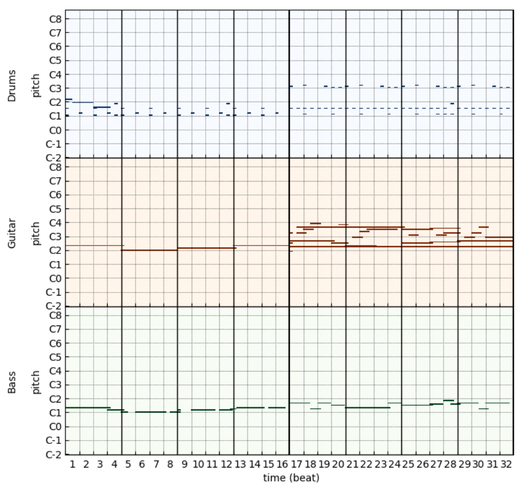


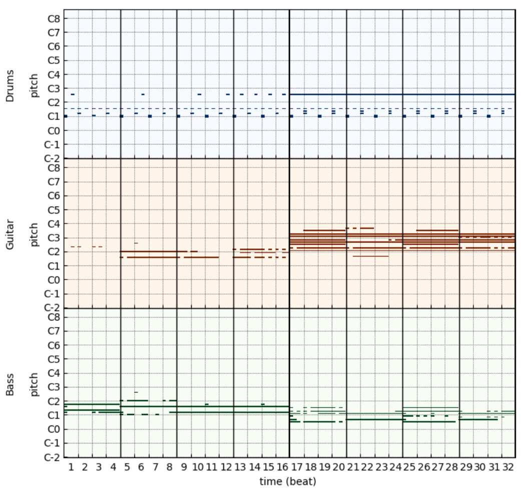
Figure 4.3: Training (left) and validation (right) Asymmetric Losses for Variational Autoencoders with Fully Connected (top) and Self-Attention (bottom) middle layers. The model is unfrozen at the beginning (light blue line) or at the second half (dark blue line) of training.

However, small changes to single values don't affect both the same way. In a last attempt to address this issue, we followed former works [40] that found this same problem in other domains and defined a perceptual loss function based on features.

Specifically, we used the MAE between the ones extracted by the critic's streams (frozen and apart of the VAE) for both the original and the reconstruction. Results with this loss function resemble that of the first model trained with MAE over the piano rolls, without any sign of the loss being reduced.



(a) Original



(b) Reconstruction

Figure 4.4: Two example 4-bar-long pieces before and after being encoded and decoded by the Self Attention VAE. The general location of the notes is close, but the model misplaces some of the correct notes and adds new, incorrect ones.

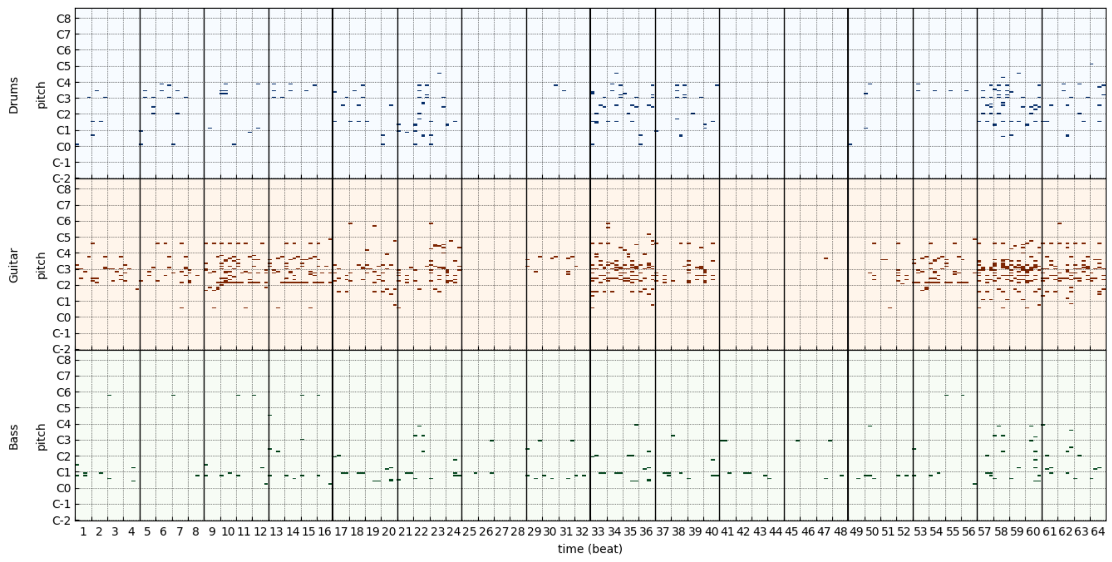


Figure 4.5: Example piece generated by sampling the VAEGAN's latent space. The pieces do not resemble those of the dataset.

Our final model was a VAEGAN, that used this loss at the same time that the GAN was trained, rather than using it with a pretrained model, in order to see if combining the GAN and VAE losses would mean any benefit. However, the loss curves continued to not show signs of convergence and the produced pieces were not resembling those of the dataset. An example is showed in Figure 4.5. With this last attempt, we ceased our efforts to build a dedicated model for dimensionality reduction.

4.3 Human-interaction-based BO

Our final goal is to validate the use of Bayesian Optimization to find a Deep-Learning-generated song that best caters human users' taste. For this, we designed an experiment in which we generate several songs, play them for the user, and gather their opinion.

4.3.1 Model-BO integration

In order to run the experiments, we developed an application based on BayesOpt and several MIDI-related libraries.

We worked using the Python interface for BayesOpt, which provides a wrapper for defining custom optimization problems. The definition of a problem mainly includes the number of dimensions of the domain, the boundaries for each of them, the initial dataset size and iteration budget, and the black-box function to be optimized. Parameters related to the GP like the kernel function used for the Gaussian Process, acquisition function or the initialization method, can also be modified. In our case, we decided to reduce the size of the problem to 10 dimensions (which will later be mapped to the original 128 with REMBO) and each dimension is contained in the [0,1] interval. For the kernel function, we chose a Matèrn kernel with ARD. Our acquisition function is Expected Improvement.

Keeping in mind that each fragment takes 10 seconds to be reproduced, and that obtaining an evaluation from the user will add an overhead to the acquisition, we estimated that an initial dataset size of 22 samples and a budget of 42 iterations would fit a 30 minute time limit, which is our intended duration for the experiment. From the 22 initial samples, the first one is the 'middle sample', shown in Figure 4.6. This piece is the one generated when the model is provided an all-zero input. This sample is also the result of any of the latent vectors contained in the origin subspace of the Box-Muller transform, so the score given by the user is saved and then replicated for a collection of latent vectors uniformly sampled within said subset. In total, we build the initial dataset with 42 samples. 21 of these samples from Box-Muller's origin subset and 21 samples obtained sampling the entire latent space with Sobol sampling, a quasi-random series that ensures better coverage of the domain than uniformly sampling it. Afterwards, we run the algorithm for 42 iterations. The model ends up having 84 data points with only 64 songs being generated.

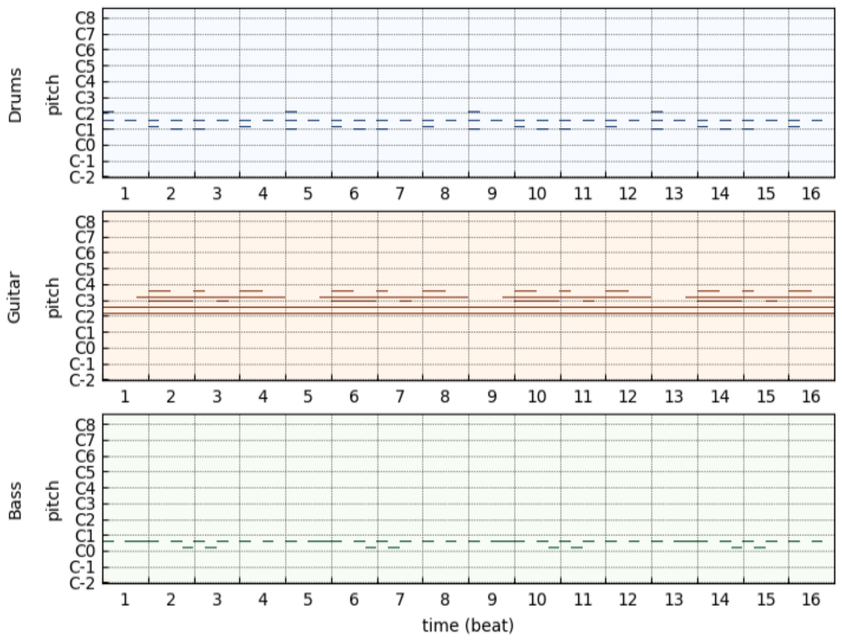


Figure 4.6: Middle sample of our model, obtained when provided with an all-zero input.

Each problem is defined as an instance of a Python class, and the black-box function is provided in the form of a Python function defined for said class. In our case, this function takes a query over our low dimensional latent space, performs the Box-Muller transform, maps it to MuseGAN's latent space by multiplying the transformed query with REMBO's $\boldsymbol{A}$ matrix (which is generated and saved at the beginning of the problem), and uses the resultant high-dimensional vector as input for the model. The model returns the generated piece, which is then saved in MIDI format and reproduced. The function awaits for the user to cast their evaluation, normalizes the results to bound it between 0 and 1, and adds the result to the surrogate model. This function is be called by the library at each iteration, and the inner implementation of the library takes care of fitting the GP posterior and finding the next point to be queried.

4.3.2 Experiment setup

In order to have an agile way to reproduce the samples and recording the user's evaluations, we complemented our optimization system with a graphic user interface, implemented by us, which enables a person to reproduce the samples and introduce the score. This interface is depicted in Figure 4.7. At all moments, the interface displays the generated song at each timestep and the historic best sample, along with its score.

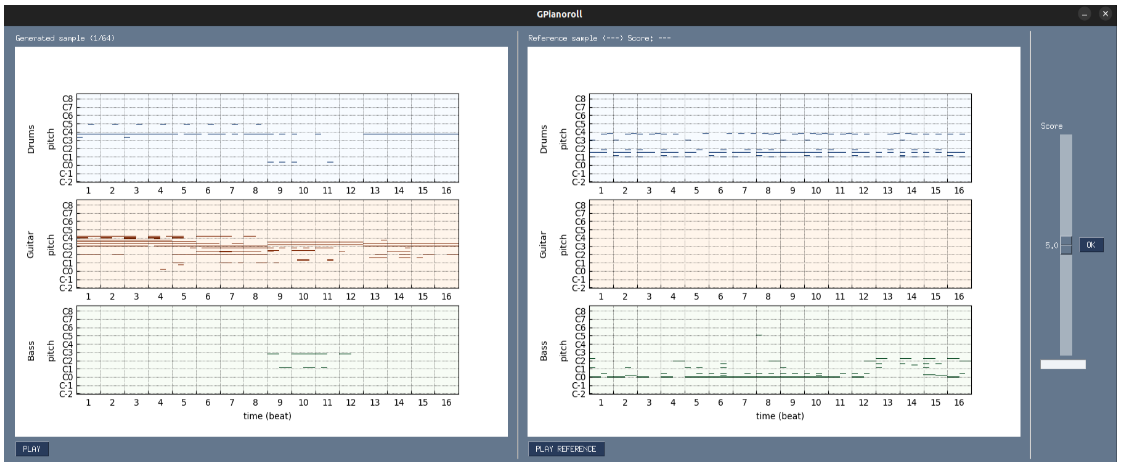


Figure 4.7: Application interface for the experiments. The supervisor can see the generated pieces' piano rolls and play the resulting MIDIs using the buttons below the piano rolls. Score can be provided via the slider to the right or the textbox beneath it.

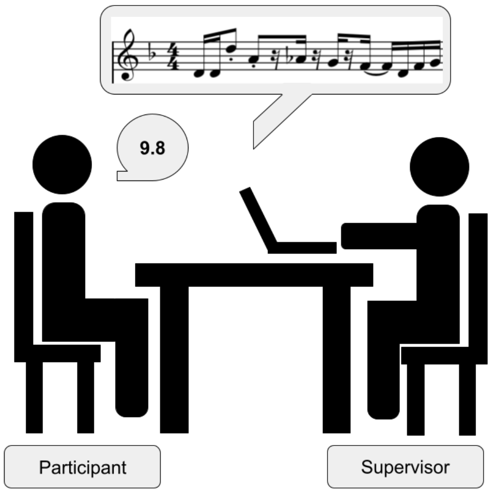


Figure 4.8: Diagram of the experiment setup. Only the supervisor sees and interacts with the interface. Participants can only listen to the played pieces and give their scores.

The setup for the experiments consisted of the supervisor's personal laptop, on which they had the interface running. Interaction occurs only between the participants and the supervisor, never between the participants and the hardware or software. Users give their scores aloud, and the supervisor enters them in the system themselves. Supervisor and participant are alone in a room with no other people nor other auditive distraction. The supervisor is sit in a chair with the laptop facing them. Users are asked to sit across the table, with no visual clue of the songs that will be played, as depicted in Figure 4.8. This setup design is intended so the user does not condition their evaluation on the visual aspect of the piano roll before listening to the piece.

Other than that, no constrain is posed upon the volunteers. Short 15 second breaks are evenly distributed after 10, 21, 43 and 54 samples are listened to, and a longer 30 second break is planned after 32 samples, when the experiment reaches half its duration. This breaks are intended to reset the stimulation of the user's hearing.

4.3.3 Experiment procedure

First off, it is worth mentioning that our experiments were conducted according to the ethical recommendations of the Declaration of Helsinki. All users were volunteers and gave their written consent to participate upon entering the room of the experiment. Our protocol purely involves the gathering of opinion data, with no risk for any participant, and the only personal data collected is whether or not they have a musical background.

After collecting the volunteer's consent, they are told the following pieces of information:

– They will be listening to several computer-generated, 10-second long musical compositions. The objective is to find the song that is best catered to their taste among the possible generations.

– They will be giving a score to each of the fragments, from 0 to 10 and with up to 0.1 precision. They are a moment to correct themselves in case they think they misjudged, but once they confirm their decision, the scores can't be changed. Scores are said aloud to the supervisor.

– They can ask the supervisor to replay a fragment, before giving the score, if they need to. They can also ask to replay the fragment they gave the best score to, and be reminded of what the score was.

– The experiment lasts for approximately 30 minutes, assuming they don't abuse the replay, and accounting for short breaks every few songs. The supervisor will tell them when a break starts and stops.

Then, the supervisor generates and plays two random songs. The user is told not to give a score for these songs. This serves two purposes: First, letting the participant know how the generated songs sound, to control expectations. This is our approach to dealing with anchor bias, avoiding giving a score to the first song the users hear. Second, giving them the chance to adjust the volume. The user is intended to turn it up enough so they can clearly hear the music but not as much as to be painful to hear.

After that, the supervisor starts the BO algorithm, and a total of 64 samples are played throughout the duration of the experiment. After the participant is done giving scores, they are presented a survey asking them on their opinion about the songs and their own capacity to give scores.

4.3.4 Results and final survey

A total of eight people participated in our experiments. Five out of eight participants had a musical background, category under which we considered either having official musical education, playing an instrument or having any music-related line of work. The results of their optimization process are showcased in Figure 4.9 and are reproducible online at our website[1]. All obtained samples show great difference between them, which seems to indicate that our method works when attempting to find specific regions of the latent space. The reported metrics of the resulting samples also show variety, as depicted in Figure 4.10. All songs with a Bass track had less or a very close to the mean number of Used Pitch Classes, and the same metric is higher than average for the Guitar track, which also shows great variation among subjects. The results for subject 4 are of special significance, since they got the model to generate a piece with no Guitar nor Bass (as in the tracks being completely silent). This supposes a corner case of the model, given the EBR metrics of the respective tracks, and still the optimization system was able to reach the region where the final latent sample lied.

The survey states four affirmations and users are asked to tell how much they agree with them. The survey points say as follows:

- Q1: ”Overall, I like the generated songs”
- Q2: ”I like the song I gave the best score to”
- Q3: ”I like the song I gave the best score to better than the first song”
- Q4: ”After listening to all the songs, I am alright with the scores I gave to the best and first songs”

All four points can be answered using Likert scale answers: “Strongly disagree”, “Disagree”, “Neither agree nor disagree” (onward “No preference”, for short), “Agree” and “Strongly agree”. Figure 4.11 shows the aggregated results of the survey and Table 4.3 shows each of the individual answers.

[1]https://mikceroese.github.io/GPianoroll/

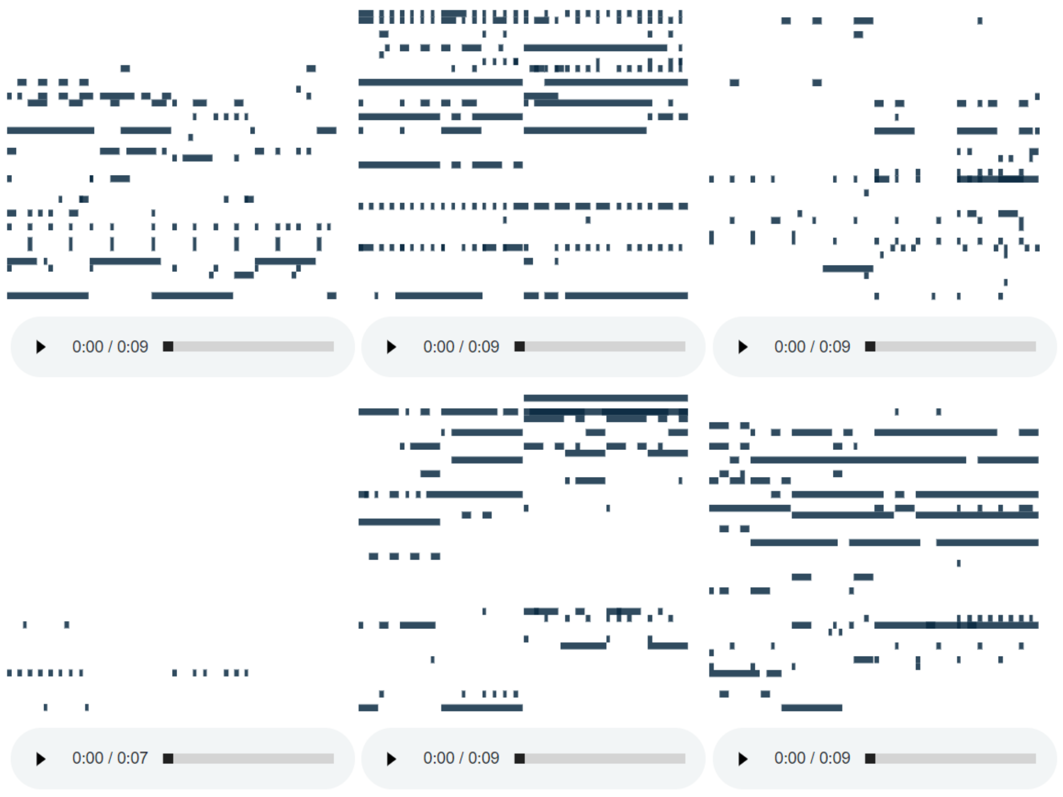


Figure 4.9: Examples of optimal compositions found by our volunteers, which granted each of them the highest scores in their respective experiments. The generated pieces show great difference between the middle sample and among them, hinting the success of our method. The samples are reproducible online at our website.

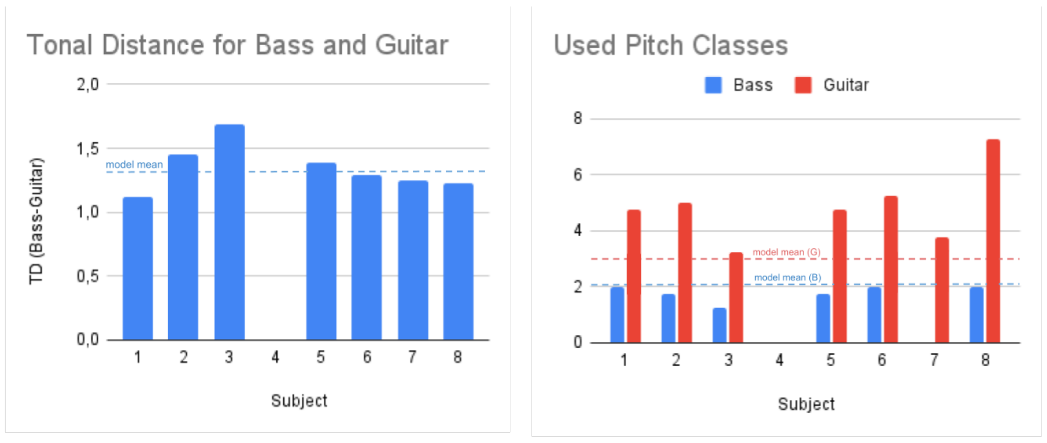
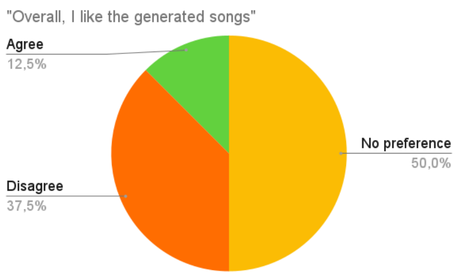
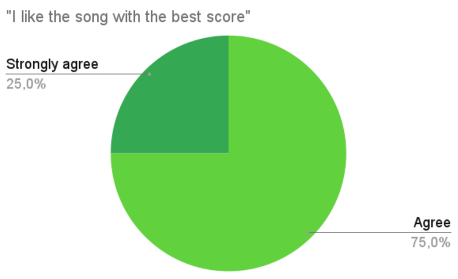


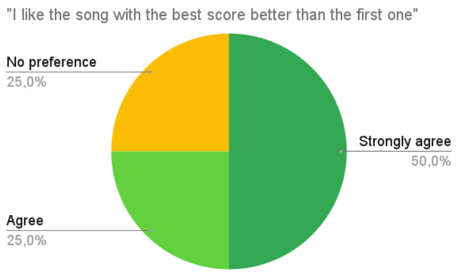
Figure 4.10: UPC and TD metrics for our eight participants' selected songs. Participants are dsitributed along the horizontal axis and the respective metrics are represented by the vertical axis. The average metrics are indicated with dashed lines.



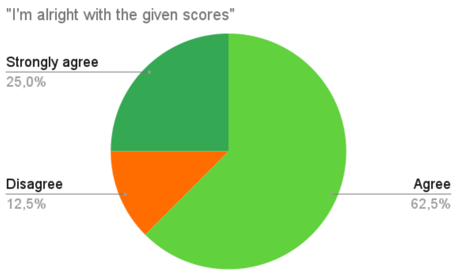
(a) Q1



(b) Q2



(c) Q3



(d) Q4

Figure 4.11: Pie charts showcasing the aggregated results of the Likert survey. All charts show overall positive responses (orange and yellow for Q1, yellow and green for Q2, Q3 and Q4). No participant answered any of the questions with ”Strongly disagree”.

Question \User	**U1**	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**
Q1	NP	D	NP	D	NP	A	D	NP
Q2	A	A	SA	A	A	SA	A	A
Q3	SA	SA	A	NP	A	SA	NP	SA
Q4	A	D	A	SA	A	A	SA	A

Table 4.3: Color coded table showing the responses of individual users for each question.

The results show that the optimization process helps most users find a song they like. Just one out of the eight participants agreed they liked the generated music in general. However, all answers to Q2 are positive, with six our of eight agreeing and two of of eight strongly agreeing. As for Q3, no participant stated that their selected song was worse than the first song. These are fantastic results for the optimization system.

Regarding the capacity of users to give a numeric score, all eight participants admitted at some point of the experiment that ’it is very hard to give a number’. However, the answers to Q4 show that all users but one were fine with the given scores. This answers indicate that the numerical evaluation is indeed reliable, and that optimization can be carried out even with a non-preferential model. With the current results, we venture to say that non-preferential Bayesian Optimization is indeed a valid way to take into account human judgement for music generation.

Chapter 5

Conclusions

Our work's goal was to further validate the use of Bayesian Optimization in the generation of musical pieces using Deep Learning. As stated in previous chapters, generative models should adapt their output to the taste or needs of their final users, to the point of individually adapt to different users in different situations, while also constraining the time they take to do so within a sensible margin. We proposed Bayesian Optimization as a way to take into account this individual taste, emphasizing its sampling efficiency and global optimality. By combining a gold standard model for music composition, MuseGAN, with a human-driven Bayesian Optimization system, we achieved to provide a way to find the most likeable regions of the music generation model's latent space, including humans on the generative process.

Despite the high dimensionality of the initial problem, we we able to construct a system able to reduce its size down to ten dimensions using REMBO, under the hypothesis that the effective dimension of the problem was smaller. The results we obtained confirmed that this method did indeed work. We were able to gather a total of eight subjects to participate in our pilot experiments. In just half an hour, and without any prior knowledge of their musical taste on our part, all of them were able to find a piece they considered to be of their liking. All of them admitted being fond of their final selected song, even if, overall, the generated pieces weren't up to their taste.

Along the way, we also provided examples of how building a representative latent space from which to sample using our selected baseline was challenging. We presented a collection of model architectures and loss functions that failed to capture meaningful features onto an ordered latent space. This remains as a possible line of future work. However, we finally solved this problem by using Random Embeddings Bayesian Optimization, technique that we leveraged jointly with the Box-Muller transform on an artificial uniform and reduced latent space to efficiently sample and optimize the taste of subjects, which also yielded an improvement in our ability to gather observations for our model.

As for future work, we have yet to prove how different models behave when combined with Bayesian Optimization. Our intuition is that, given Bayesian Optimization's function agnosticism, the same process should be applicable to any other model.

Finally, we have gathered all our source material within an open repository on GitHub for anyone wanting to reference or extend our work, with some of it even reproducible completely online.

Chapter 6

Bibliography

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014.

[2] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[3] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1278–1286, Bejing, China, 22–24 Jun 2014. PMLR.

[4] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223. PMLR, 06–11 Aug 2017.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[6] Felix Kreuk, Gabriel Synnaeve, Adam Polyak, Uriel Singer, Alexandre D'efossez, Jade Copet, Devi Parikh, Yaniv Taigman, and Yossi Adi. Audiogen: Textually guided audio generation. *ArXiv*, abs/2209.15352, 2022.

[7] Jade Copet, Felix Kreuk, Itai Gat, Tal Remez, David Kant, Gabriel Synnaeve, Yossi Adi, and Alexandre Défossez. Simple and controllable music generation, 2023.

[8] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music, 04 2020.

[9] Cheng-Zhi Anna Huang, Tim Cooijmans, Adam Roberts, Aaron C. Courville, and Douglas Eck. Counterpoint by convolution. In *International Society for Music Information Retrieval Conference*, 2019.

[10] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and yi-hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. 02 2018.

[11] Colin Raffel. Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching. 2016.

[12] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, and Nando Freitas. Bayesian optimization in a billion dimensions via random embeddings. *IJCAI International Joint Conference on Artificial Intelligence*, 55, 01 2013.

[13] Gokul Yenduri, Ramalingam Murugan, Chemmalar Govardanan, Supriya Y, Gautam Srivastava, Praveen Reddy, Deepti Raj, Rutvij Jhaveri, Prabadevi B, Weizheng Wang, Athanasios Vasilakos, and Thippa Gadekallu. Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions, 05 2023.

[14] Paul Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. 06 2017.

[15] Alexander Selivanov, Oleg Y. Rogov, Daniil Chesakov, Artem Shelmanov, Irina Fedulova, and Dmitry V. Dylov. Medical image captioning via generative pretrained transformers. *Scientific Reports*, 13(1):4171, Mar 2023.

[16] Jonathan Ho, Ajay Jain, and P. Abbeel. Denoising diffusion probabilistic models. *ArXiv*, abs/2006.11239, 2020.

[17] MICHAEL C. MOZER. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3):247–280, 1994.

[18] Peter M. Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

[19] D. Eck and J. Schmidhuber. Finding temporal structure in music: blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756, 2002.

[20] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. Deepbach: a steerable model for bach chorales generation, 2017.

[21] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 2852–2858. AAAI Press, 2017.

[22] Benigno Uria, Iain Murray, and Hugo Larochelle. A deep and tractable density estimator. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 467–475, Bejing, China, 22–24 Jun 2014. PMLR.

[23] Sean Vasquez and Mike Lewis. Melnet: A generative model for audio in the frequency domain. *ArXiv*, abs/1906.01083, 2019.

[24] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression, 2022.

[25] Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, 4, 06 2012.

[26] Ruben Martinez-Cantin. Funneled bayesian optimization for design, tuning and control of autonomous systems. *CoRR*, abs/1610.00366, 2016.

[27] Javier Garcia-Barcos and Ruben Martinez-Cantin. Robust policy search for robot navigation. *IEEE Robotics and Automation Letters*, 6(2):2389–2396, 2021.

[28] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[29] Yutian Chen, Aja Huang, Ziyun Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. Bayesian optimization in alphago. *ArXiv*, abs/1812.06855, 2018.

[30] Eric Brochu, Tyson Brochu, and Nando Freitas. A bayesian interactive optimization approach to procedural animation design. pages 103–112, 07 2010.

[31] Javier González, Zhenwen Dai, Andreas Damianou, and Neil D Lawrence. Preferential bayesian optimization. In *International Conference on Machine Learning*, pages 1282–1291. PMLR, 2017.

[32] Yijun Zhou, Yuki Koyama, Masataka Goto, and Takeo Igarashi. Generative melody composition with human-in-the-loop bayesian optimization, 10 2020.

[33] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. Autoencoding beyond pixels using a learned similarity metric. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1558–1566, New York, New York, USA, 20–22 Jun 2016. PMLR.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[35] Christopher Harte, Mark Sandler, and Martin Gasser. Detecting harmonic change in musical audio. 10 2006.

[36] Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Celikyilmaz Asli, and Lawrence Carin. Cyclical annealing schedule: A simple approach to mitigating. pages 240–250, 01 2019.

[37] Qianli Feng, Chenqi Guo, Fabian Benitez-Quiroz, and Aleix Martinez. When do gans replicate? on the choice of dataset size. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6681–6690, 2021.

[38] T. Ridnik, E. Ben-Baruch, N. Zamir, A. Noy, I. Friedman, M. Protter, and L. Zelnik-Manor. Asymmetric loss for multi-label classification. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 82–91, Los Alamitos, CA, USA, oct 2021. IEEE Computer Society.

[39] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016*

IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2818–2826, 2016.

[40] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. volume 9906, pages 694–711, 10 2016.

[41] I.M Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

List of Figures

List of Tables