

**BIRKBECK, UNIVERSITY OF LONDON**

Department of Computer Science and Information systems



---

**Using Deep Learning to create music in a specific genre: an attempt to assist songwriters in overcoming writer's block.**

---

A project report submitted in partial fulfilment of the requirements for the

**MSc in DATA SCIENCE**

Ivan Sanz

(Part-time cohort 20-22)

Supervisor: Cen Wan

October 3, 2022

# Declaration

At Birkbeck, coursework is monitored for plagiarism and if detected may result in disciplinary action. In submitting this piece of work, I hereby confirm that I have read Birkbeck's plagiarism guidelines and on this basis declare that this coursework is free from plagiarism.

# Acknowledgements

This work would have not been possible without the backing of my partner, family, friends, tutor, mentor and supervisor. You know who you are, and I do appreciate your support and encouragement.

Thank you all.

# Abstract

Composing Music is a non-linear process that requires a very specific skill set. A proficient songwriter or a composer must have a vast array of talents: ranging from knowledge of music composition techniques or playing an instrument to writing lyrics and having people's skills. All of this without forgetting key factors like creativity and inspiration. The latter can be considered inherently human qualities that are known to have a substantial weight in the songwriting process. That's why it might result somehow strange to think that a computer might be able to compose music in the same fashion as Metallica, Muse or Eminem.

This report will explore a specific case: Use Deep Learning Models in an attempt to support a songwriter that is in a rut and struggling to compose more music.

## **Keywords:**

Deep Learning, Neural Networks, Music Composition, Music Generation, Writer's block.

# List of Figures

Figure 1 Main Hypothesis workflow .....	10
Figure 2 Robot-Piano (aremenko-Sergii) .....	11
Figure 3 Human's perception of sound .....	15
Figure 4 Frequency and amplitude .....	18
Figure 5 A real soundwave (piano sound) .....	18
Figure 6 Fourier Transform for S .....	19
Figure 7 S1 and S2 .....	19
Figure 8 Fourier Transform for S .....	20
Figure 9 How to get a Spectrogram .....	21
Figure 10 Audio pipeline .....	21
Figure 11 Unsupervised learning .....	22
Figure 12 ANN .....	23
Figure 13 The Artificial Neuron .....	23
Figure 14 Leaky ReLU .....	25
Figure 15 Training for ANN .....	26
Figure 16 Convolution .....	27
Figure 17 Zero padding .....	28
Figure 18 Pooling example .....	29
Figure 19 Pooling example, feature learn .....	29
Figure 20 Minmax function .....	31
Figure 21 Minmax High Level .....	31
Figure 22 Training cycle on a GAN .....	32
Figure 23 DCGAN Generation Audio .....	33
Figure 24 Final project's methodology .....	36
Figure 25 High level view of CAW .....	37
Figure 26 Up sampling process .....	38
Figure 27 Training cycle, 3000 epochs .....	39
Figure 28 Generator .....	40
Figure 29 Discriminator .....	40
Figure 30 Wav vs MP3 .....	43
Figure 31 Human Range Frequency .....	44
Figure 32 GAN dataset files .....	48
Figure 33 CAW Dataset .....	50
Figure 34 Mel Spectrogram GAN dataset .....	52
Figure 35 Training main parameters (source: author's notebooks) .....	54
Figure 36 New samples .....	57
Figure 37 Example questionnaire .....	69
Figure 38 Final questionnaire submission .....	75

# List of Acronyms

DL	Deep Learning
NN	Neural Networks
GAN	Generative Adversarial Network
CNN	Convolutional Neural Network
MC	Middle C (256 Hz)
MIDI	Music Instrument Digital Interface
Db	Decibel
FT	Fourier Transform
STFT	Short Time Fourier Transform
ANN	Artificial Neural Network
AN	Artificial Neuron
ReLU	Rectified Linear Unit
DCGAN	Deep Convolutional GAN
CAW	Catch A Waveform
WAV	Waveform Audio file format
GPU	Graphics Processing Unit
A/D	Analog to Digital converter
CD	Compact Disc file system
DAW	Digital Audio Work station
UMG	Unconditional Music Generation
MS	Main Songwriter

## Table of Contents

<b>Declaration</b>	<b>1</b>
<b>Acknowledgements</b>	<b>2</b>
<b>Abstract</b>	<b>3</b>
<b>List of Figures</b>	<b>4</b>
<b>List of Acronyms</b>	<b>5</b>
<b>1 Introduction</b>	<b>8</b>
1.1 Motivation	9
1.2 Background	11
1.3 Objectives	12
1.4 Project Report Outline	13
<b>2 Theory</b>	<b>14</b>
2.1 Musical Concepts	14
2.1.1 What are Music and Sound?	15
2.1.2 Rhythm and Pitch	15
2.1.3 Melody and Harmony	16
2.1.4 Timbre and Loudness	17
2.2 Visual representation of sounds	18
2.2.1 The Fourier Transform	19
2.2.2 Frequency Domain	20
2.2.2 From Frequency domain to spectrograms	20
2.3 Deep Learning and Neural Networks	22
2.3.1 The artificial Neuron	22
2.3.2 Activation and Cost functions	24
2.3.3 Training our model (Gradient Descent and Backpropagation)	25
2.4 Convolutional Neural Networks (CNNs)	27
2.5 Generative Adversarial Networks (GANs)	30
2.5.1 GAN-Basics	30
2.5.2 GANs-Training issues	32
2.6 GAN models for audio generation	33
2.6.1 Catch a Waveform (Unconditional Music Generation, Variations)	37
2.6.2 Method	37
2.6.2 Training	39
2.6.4 Architecture	40
2.6.4 Other parameters	41

<b>3 Data</b>	<b>42</b>
3.1 Audio characteristics	42
3.2 Analog vs Digital	43
3.3 Graphical representation of Audio	45
3.4 From L'Anima Music dataset to GAN/CAW dataset	45
3.5.1 Artistic/ Aesthetic curation	46
3.5.1 Technical Perspective/ Final Dataset	49
3.6 Data Manipulation	50
<b>4 CAW for Audio Generation</b>	<b>53</b>
4.1 Working Environment/ tools	53
4.2 Implementation	53
4.3 Training	54
4.4 Results	55
4.4.1 Newly Generated Audio Files	55
4.4.2 Metrics	57
4.5 Other Models (Jukebox, SampleRNN)	66
4.6 Comparison	66
<b>5 User Study</b>	<b>68</b>
5.1 Methodology	68
5.2 Results	70
5.2.1 Unconditional Music Generation (addressing issue 1)	70
5.2.2 Variation (addressing issue 2)	72
<b>6 Conclusion and Future Directions</b>	<b>76</b>
<b>List of References</b>	<b>77</b>
<b>Appendix</b>	<b>80</b>
Link to GitHub:	80
Links to datasets in Google Drive:	80



# 1 Introduction

Data Science is a ubiquitous term (VanderPlas, 2017), encapsulating a wide range of skills that will help the Data Science practitioner to solve a particular problem. The nature of this problem might be commercial, academic or scientific but the important concept to take on board is that there is a problem that needs a solution. This first step towards a solution is clearly defining the problem by asking the right questions. Secondly and after some research, a series of hypotheses, models and actions will be formulated as a possible solution. Finally, the solution or solutions will be evaluated in order to probe their efficacy or not. This is the method that we will follow here with regard to the problem we have introduced in the abstract.

## 1.1 Motivation

As stated in this project's proposal, my band has a problem that needs to be addressed: the band's main songwriter suffers from writer's block. Although able to compose individual fragments and develop them in a coherent musical way, the inability to crystalize the sum of all those fragments into fully fledged songs has become quite apparent. This situation has gotten to the point that the rejection folder of his computer contains twice as many projects as the finished material folder. As we can see, the problem is not the lack of ideas but the temporal inability to find a common thread that will 'connect' all those. For instance, some songs have a clearly defined Intro/ verse/ Chorus structure but there is not a bridge section that will join it with the end of the song. Other songs feature a verse and two bridges but there is no chorus. Some other songs do not have an ending just yet. Finally, there are some songs on the rejections folder that although finished, didn't make the cut. The songwriter's subjective judgement states that they are not good enough as a final product. As we can see, we have two types of 'rejected' material: On one side, incomplete material; songs that are missing one or more pieces of the puzzle. On the other side, finished songs that are 'missing something' and would need to be reworked/ reshuffled to make the cut. The question now is what can we do to help?

My proposed solution for issue number 1 would be as follows: to generate new original material that could fit within the inner structure of the songs and therefore contribute to its completion. This material does not need to be a whole song, we need 'chunks' of maybe 20 or 25 seconds of different sections commonly used in songs like verses or choruses. For example, there is a song called 'Before the world' that would benefit from this approach (Identified as S4 in the training dataset) if we could generate material to help with the composition of a bridge for after the solo and an ending; we would have accomplished our mission.

For issue 2 my proposed solution would be as follows: generating variations of specific parts of those songs that have been identified as not 'working 100%'; with the hope that this action can lead to a successful rework of the song's structure/shape and therefore, become part of the folder of accepted material. An example of this would be the song called "The King" (Song S13 of the training dataset). This process has already been followed by the main composer of the band, but none of the different final versions of the song has passed the cut yet.

Both solutions are based on the fact the generated 'chunks' are not going to be final within the song. Although this premise will also help later on with the technicalities of our Deep Learning Model (especially in terms of duration and quality of the samples), the main aim of the generation of these 'chunks' is to trigger a "Eureka effect" on the main songwriter of our band. As per its definition, a Eureka effect is "a moment of sudden, triumphant discovery, inspiration, or insight"<sup>1</sup> that leads to solving a problem or situation that was at an impasse. The most obvious example is Archimedes itself when

---

<sup>1</sup> <https://www.merriam-webster.com/dictionary/eureka%20moment>

shouting “Eureka! Eureka!” after having the epiphany that led to him finding the solution of the famous “Archimedes principle”<sup>2</sup>. This situation does happen in Popular Music as well. An example would be me attending a drumming masterclass in 2005<sup>3</sup> and listening to Jazz legend percussionist Bill Summers explain how Herbie Hancock had one of those Eureka moments after listening to him improvising a melody with a bottle<sup>4</sup>. This led to the composition of one of Herbie Hancock’s most famous songs to date: “Watermelon Man” [2]. Trying to recreate this kind of event would be our main hypothesis for our solution: by presenting as many chunks of newly generated music to our main songwriter, we hope that, after listening to some of them, a Eureka moment will be triggered and that will ‘inspire’ our songwriter to ‘connect the dots’; This will result on re-starting the creative process of an unfinished song, allowing for successful completion. In essence, we are trying to provide an AI muse that will seek to de-randomize the eureka effect or at least accelerate it, if possible.

This hypothesis could also be formulated as a case of ‘reverse prompt engineering’<sup>5</sup>; with the machine trying to generate and refine the best musical prompt possible to obtain the desired output from the human.

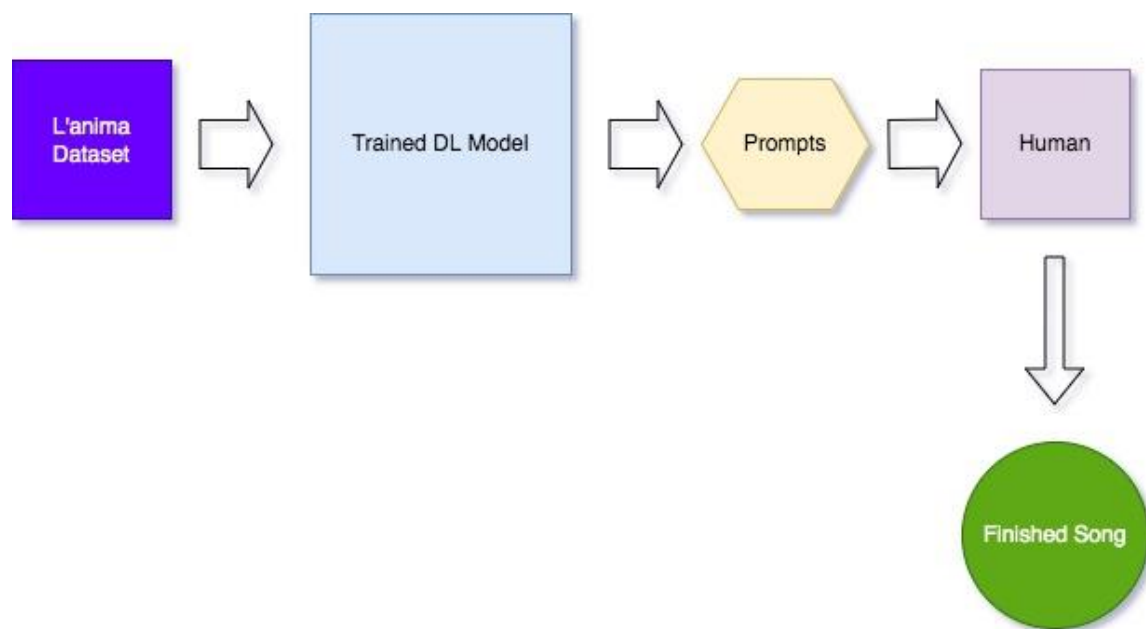


Figure 1 Main Hypothesis workflow

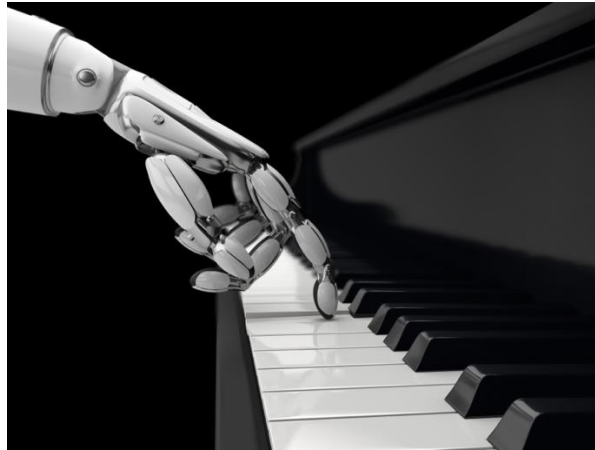
<sup>2</sup> <https://www.britannica.com/science/Archimedes-principle>

<sup>3</sup> <https://www.youtube.com/watch?v=b1hNVeo51g>

<sup>4</sup> <https://www.youtube.com/watch?v=u3z8n5tAVXY>

<sup>5</sup> <https://dallery.gallery/the-dalle-2-prompt-book/>

## 1.2 Background



*Figure 2 Robot-Piano (aremenko-Sergii)*

In attempting a solution to our problem, we have identified the need of generating original audio clips that ‘resemble’ the music of L’Anima. That is, new music that keeps the sonic integrity of the original ‘data’ in terms of rhythm, pitch and timbre; This will give us a chance to trigger the Eureka effect. Therefore, we need Deep Learning models capable of achieving a significant level of ‘Music translation’ (Huzaifah, 2020) without forgetting another key requirement: the main songwriter of my band is already able to do what generative DL models do; that is, to ‘learn’ from unsupervised data (Foster, 2019) like music. He already knows what are the sonic trademarks that need to be included in a successful song of his band; he has that training already. Therefore, if our generated audio clips are to work as ‘prompts’, our songwriter should perceive them as music from his band. In essence, we must consider him as a ‘Discriminator’ first and foremost. Therefore, we need to recognize that there is an adversarial element happening even before the new music can even be considered a prompt.

This is the main reason why we should consider GANs as our main DL framework to implement the solution for our problem. Generative Adversarial Networks (Goodfellow et al, 2014) are based on an adversarial game where A tries to fool B by generating fakes so good that are perceived as real. These fakes are usually images but at present, there are state-of-the-art GAN models that work well for raw audio generation (Donahue et al., 2019), (Engel et al, 2019).

Other DL models can generate raw audio, like autoregressive models and autoencoders.

Sample RNN (Merhi et al., 2016) is one of them and it has been the foundation for two other models that have proven successful at generating audio. The first one is PRISM-SampleRNN<sup>6</sup>, developed by the Royal Northern College of Music<sup>7</sup> and the second one is authored by CJ Carr and Zack Zukowski. This is more interesting for us as their papers [9] show a method that has worked for modern styles of music like Math Rock or Black Metal. Styles of music with a certain sonic resemblance to progressive metal, the style with L'Anima<sup>8</sup> is associated.

Autoencoders have also proven successful for raw audio generation: Wavenet (Aaron et al., 2016) and more recently RAVE (Caillon, 2021). Again, the latter could be an interesting option to explore, but as the literature states [12] both Autoencoders and Auto-regressive models need huge amounts of data and they are slow at the training stage. The GANSynth paper [13], also offers some reasoning in favour of the use of GANs over other models:

- GANs do present a better waveform generation and better audio quality, surpassing WaveNet on human acceptance.
- Timbre translates better with GANs. This is key if we are to preserve 'resemblance' to the original material.
- GANs can generate new music at a faster rate. Less computational resources are needed.

To summarize, as Ian Goodfellow states in his paper there are challenges when implementing GAN models. But given all the previous reasoning, implementing our solution using a GAN-based model seems to be the best option as we are maximizing our options of having realistic generated audio samples, that somehow capture most of the original data distribution. More precisely we will be looking into several models but the one we intend to work on is Catch a Waveform (Greshler et al, 2019)

## 1.3 Objectives

The main objective of this project is to use my own dataset to train a CAW model that will master our own band's specific music style, in an attempt to support the main writer of L'Anima to finish more music that can make the cut into the band's second album. This objective could be broken down as follows:

- Create the L'Anima-GAN-dataset. This dataset should contain a set of samples that will help in addressing issues 1 and 2 (as mentioned in the intro). Samples for addressing issue 1 should

---

<sup>6</sup> <https://github.com/rncm-prism/prism-samplernn>

<sup>7</sup> <https://www.rncm.ac.uk/research/research-centres-rncm/prism/prism-collaborations/prism-samplernn/>

<sup>8</sup> <https://open.spotify.com/album/4g6PFelIE5nDGtBIWw5JPg5>

be taken from unfinished songs whereas samples for issue 2 should be taken from finished but rejected songs.

- Create a GAN-based generative model that will address the need to generate new material with a high resemblance to the music of L'Anima. CAW seem to be the main candidate.
- Experiment with different settings and hyperparameters of the model until enough material is ready to be presented for human evaluation.
- To test the material with the main songwriter of L'anima and ascertain whether our main hypothesis can be a valid solution for the initial problem.
- To get feedback from the main songwriter to evaluate the project and set the scope of future work

## 1.4 Project Report Outline

The succeeding chapters will discuss the following:

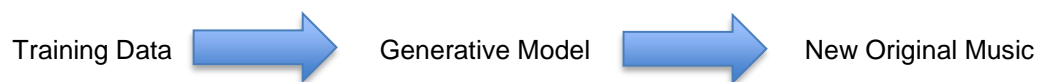
- Chapter 2 will address theoretical aspects of music and Neural Networks. The basics will be covered to then explain how everything works in the context of this project.
- Chapter 3 will address all the aspects related to the data for this project. Pre-processing and other relevant actions will be discussed.
- Chapter 4 will cover all the aspects related to our GAN generative model and our working environment.
- Chapter 5 will cover the human evaluation process and hypothesis validation.
- Chapter 6 will feature our conclusion for this project and future work

## 2 Theory

This chapter is dedicated to introducing the main concepts that are the foundation of the work presented in this report. Section 2.1 will cover some music concepts and how they can be interpreted more technically. Section 2.2 will give some information on how music can be treated as 2D and 3D data so it can be fed to DL models. Section 2.3 will introduce Neural Networks and their basic principles. Finally, the three final sections of this chapter will cover Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs), the DL models used for this project.

### 2.1 Musical Concepts

In the field of Deep Learning, and as Foster suggests [14], unsupervised learning performed by generative models to unstructured data allows for the generation of new music. The schema would be as follows:



It turns out that DL models are quite good at ‘Music Translation’ [15] and ‘representation learning’ [16]; that is, understanding what are the underlying patterns that make our data unique and specific; to then generate new original data that still maintains those relationships, that integrity. In musical terms, we would be talking about a system that can understand how timbre, rhythm, harmony, pitch or melody are organized as well as intertwined with each other in an original audio sample; to then produce a new audio clip that still preserves that initial sonic structure and ‘resemblance’. The question is: what is timbre, rhythm and so on, and more importantly, in what form do we need to present those concepts to a DL model so it can ‘learn’ from them?

## 2.1.1 What are Music and Sound?

If you look at the dictionary, music would be defined as ‘*sounds that are arranged in a way that is pleasant or exciting to listen to. People sing music or play it on instruments*<sup>9</sup>, that is, a collection of different sounds that put together make sense and are not perceived as noise. So, what is important to us is to understand what sound is. We can define sound as variations of pressure over time that we humans ‘perceive’ vibrating around us with different levels of intensity. We can also visualize it as if it was a soundwave that spreads around and is picked up by our ears. This concept of a soundwave is quite important as it will provide us with the starting point to encode sound in a way that can be fed to DL models.

### Sound wave

---

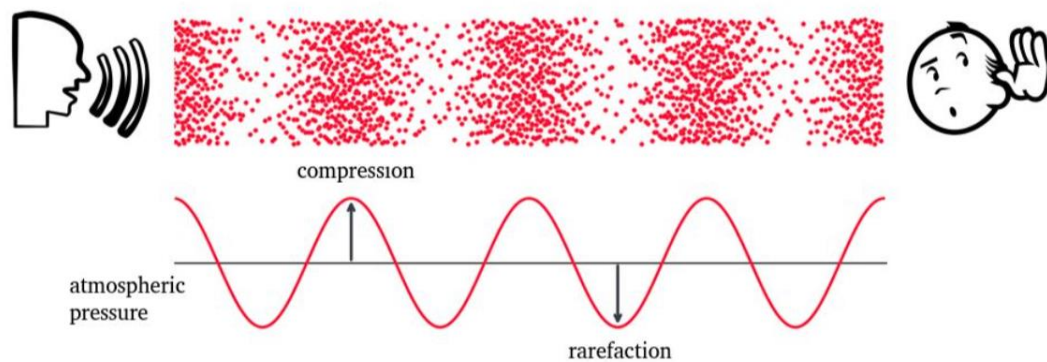


Figure 3 Human's perception of sound

(source: <https://thesoundofai.slack.com>)

## 2.1.2 Rhythm and Pitch

The fundamental unit of Music is the note. A note is a sound that vibrates at a certain frequency and has a specific duration. These notes can be organized over time in certain groups or cadences to create

---

<sup>9</sup> <https://www.oxfordlearnersdictionaries.com/definition/english/music>



patterns. That's what we call rhythm, and there are three important features of rhythm that we need to know:

- Duration: the notes of our patterns can be long or short.
- Tempo: the notes of our patterns can be played at a faster rate or a slower rate. This is measured by the number of beats per minute (BPM)<sup>10</sup>.
- Meter: the notes of our patterns can be organized according to specific groupings or cadences. An example would be the Tango pattern, where 3 notes of equal duration are grouped to produce a ternary cadence when accepting the first one. **1 2 3 1 2 3 1 2 3 1 2 3 1 2 3** and so on.

In Popular Music, a song is usually comprised of a set of rhythmic patterns that repeat cyclically at a steady tempo, following a specific meter. We need our model to learn about these rhythmic patterns in our dataset but not only that, also how they interact with other features.

Pitch is the way we humans perceive the sounds produced by vibrations. If the pitch is high, it means that there are a lot of vibrations per second (high frequency) whereas if the pitch is low the number of vibrations per second is small. Western music is based on 12 notes, each one with its own frequency and consequently, its specific vibrating frequency. For instance, middle C's root frequency is 256 Hz.

## 2.1.3 Melody and Harmony

Melody is a sequence or pattern of single notes that when played together do make sense. All the notes have a specific frequency and being a number, it is possible to establish different mathematical relationships between them. We are not going to go deep on this subject, we only need to know the fact that not all frequencies work well together in a specific pattern or scale<sup>11</sup>. The ones that do, are the ones that we put together in melodies. The same effect happens when we have sequences or patterns consisting of two or more notes played at the same time: some work and some others don't. All those relationships are established by what we call harmony. Harmony and Melody are comprised of a set of mathematical relationships that tend to be constant over time in the context of a song. That's also what we need to be able to capture with our DL models.

---

<sup>10</sup> <https://theonlinemetronome.com/blogs/13/what-does-bpm-mean>

<sup>11</sup> <https://www.simplifyingtheory.com/music-scales/>

## 2.1.4 Timbre and Loudness

Timbre is what we call the ‘colour of sound’. We can be hitting the same minor chord<sup>12</sup> with a piano and an electric guitar, but we will perceive both differently. The piano chord will be a clean sound with some touches of sadness, whereas the same chord played with an electric guitar will be perceived as a fairly distorted sound, angrier than sad. We are playing the same notes, but we are not only perceiving that, we also get its ‘colour’; which will depend on a series of factors like instrumentation or effects.

This is also what our model should be able to capture, and that’s why we will be using raw audio generation instead of symbolic representation (scores) or MIDI<sup>13</sup> (generate sequences of MIDI sounds). Zukowski and Carr [17] state in their second paper that those methods fail to translate the timbre associated with modern productions. Audio examples from their website<sup>14</sup> do suggest that using raw audio with DL models is a more valid way to maintain a good level of sonic translation and capture all that timbre-associated nuances (distorted guitars, screaming vocals, whispering vocals etc.) that otherwise would be lost with a score or MIDI file.

Finally, loudness is the intensity of the vibrations associated with a sound. Some vibrations carry more energy than others and it is measured in decibels (Db). This can be seen visually in the form of a bigger waveform. The measure that gives us that information is called amplitude. For us humans, the bigger the amplitude, the more volume we perceive and vice versa. We will visualize it alongside Frequency in the next chapter.

---

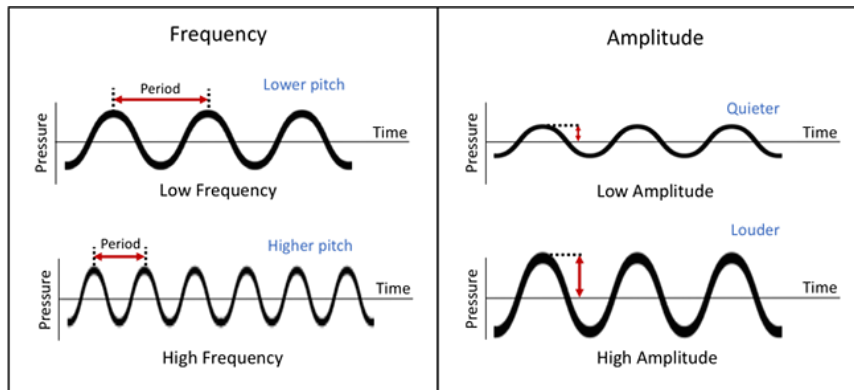
<sup>12</sup> <https://www.fender.com/articles/play/guitar-chords-minor-chord>

<sup>13</sup> [MIDI - Computer Science GCSE GURU](#)

<sup>14</sup> <https://dadabots.com/music.php>

## 2.2 Visual representation of sounds

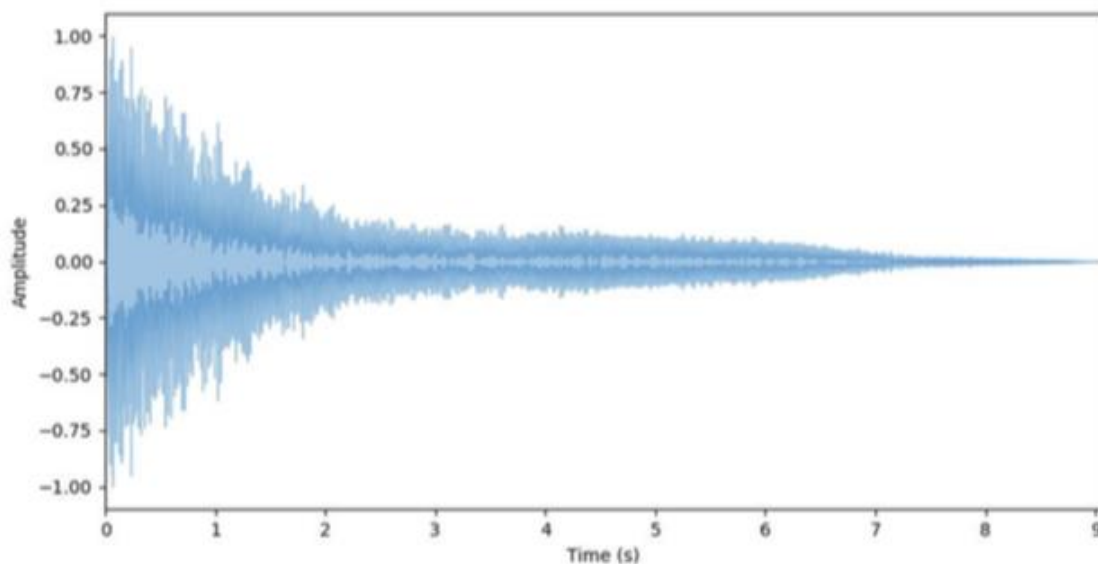
The image below clearly represents sound at its main features in a very visual way, easy to understand.



*Figure 4 Frequency and amplitude*

(source: <https://thesoundofai.slack.com>)

But a real sound looks a little bit more like this image:



*Figure 5 A real soundwave (piano sound)*

(source: <https://thesoundofai.slack.com>)

Although a simple piano stroke, it is possible to appreciate all the complexity of a real sound. This is a problem for two main reasons: how can we infer patterns from this representation? how can a DL model learn from this representation? The answer is in the Fourier Transform.

## 2.2.1 The Fourier Transform

The Fourier Transform is a mathematical transformation that allows us to decompose a complex soundwave as the sum of a set of simple waves that oscillate at different frequencies.

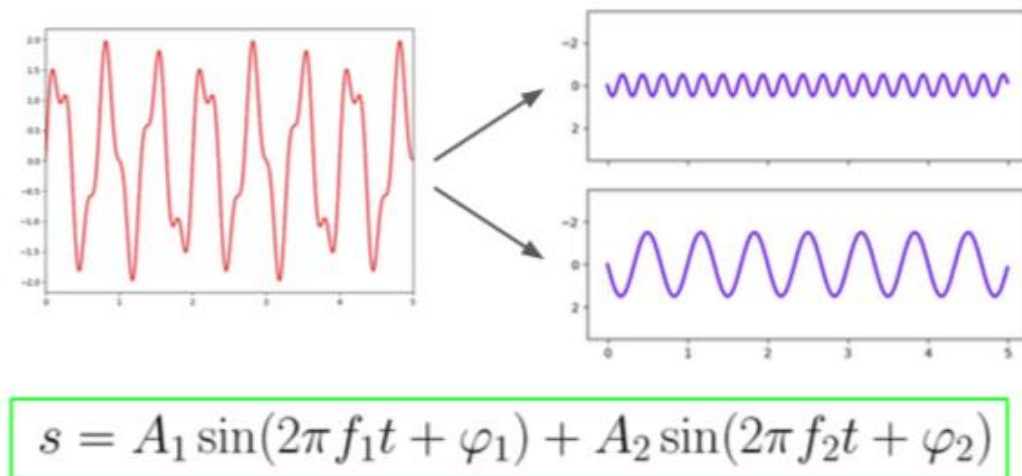


Figure 6 Fourier Transform for S

(source: <https://thesoundofai.slack.com>)

So, in our example, the complex wave S is now the sum of two simpler soundwaves with parameters that we already recognise and can quantify (Amplitude, Frequency and Phase). Thanks to this transformation we can find out how much every single soundwave contributes to the overall sound; this will be quantified by a set of parameters inherent to each simple wave that over time will create repetitive patterns and combinations.

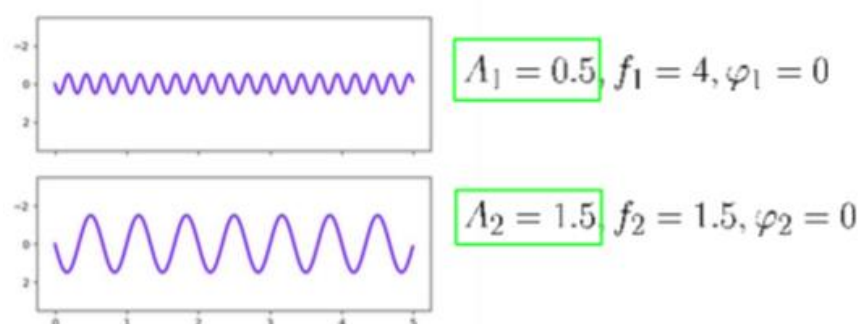


Figure 7 S1 and S2

(source: <https://thesoundofai.slack.com>)

## 2.2.2 Frequency Domain

If we generalise the previous example to a real soundwave, we obtain what is called a spectrum:

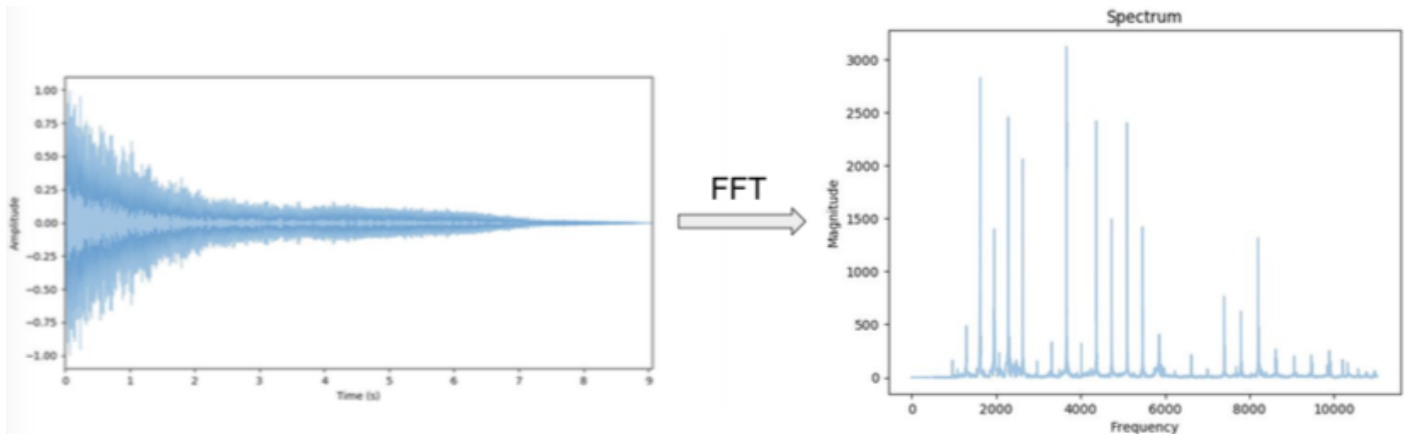


Figure 8 Fourier Transform for S

(source: <https://thesoundofai.slack.com>)

This way we can decompose a complex sound and understand how different frequencies are contributing to it. The spectrum gives us the amplitude as a function of frequency, moving from the time domain to the frequency domain. This is great for the reasons previously exposed but there is a problem: this is a static representation of the sound, a snapshot that lasts for 9 seconds. But sound does not work like that since it is a time series, therefore we want to know how things change over time. Something that we would be missing with an FT. So, what's the solution? The solution is called Short Time Fourier Transform, STFT in short.

## 2.2.2 From Frequency domain to spectrograms

What STFT does is compute FFTs at different intervals or points in time; by doing so, it preserves information about time and how a particular sound evolves over it; 9 seconds in our example. If we look at the picture, the piano sound is quite strong at the beginning but decays over time. The FFT would have missed that information completely, whereas an STFT, by iteratively performing a number of FFTs over the 9 seconds will effectively capture the behaviour of the sound on three axes: time, frequency and amplitude. The result is a spectrogram, which is a representation of amplitude as a function of frequency and time. Visually it looks like this:

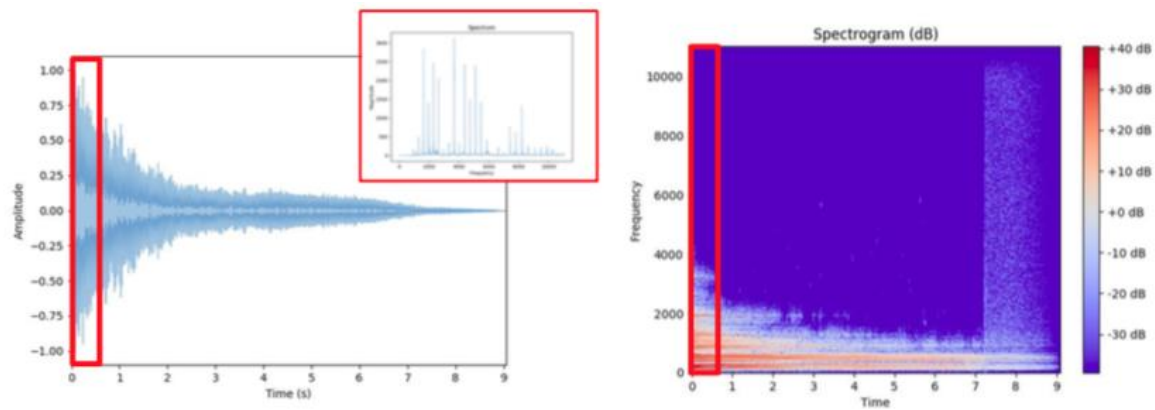


Figure 9 How to get a Spectrogram

(source: <https://thesoundofai.slack.com>)

As we can see, the spectrogram keeps a certain resemblance with the original soundwave, as energy fades in time. Now, in theory, we could have this information processed by DL models as we now can trace how certain key parameters work and their relationships over time. But we are still one step away: As previously mentioned, it is key to collect information about another important element of sound which is its colour, its timbre. How do we do that? By using Mel Spectrograms. We will talk in more detail about this particular feature in the third chapter. For now, we will focus our attention on the fact that we have a method that allows us to translate sound into information that DL models can understand. The resulting pipeline would be as follows:

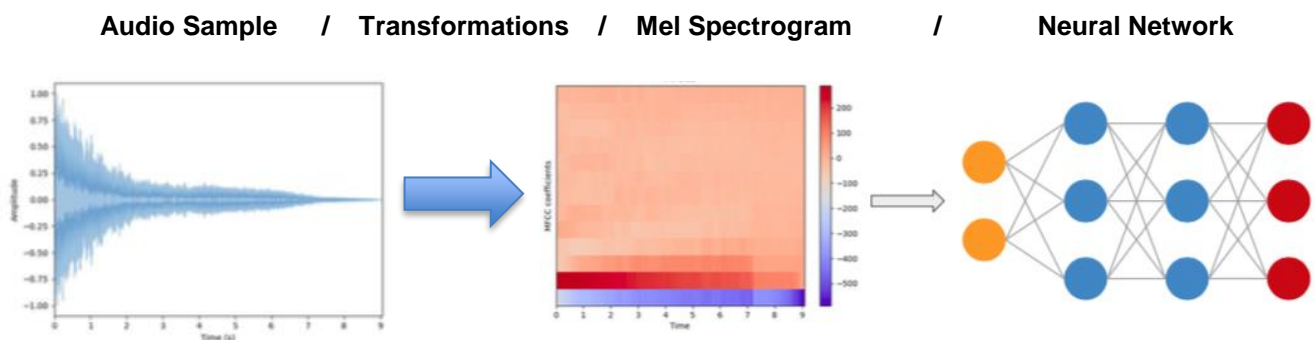


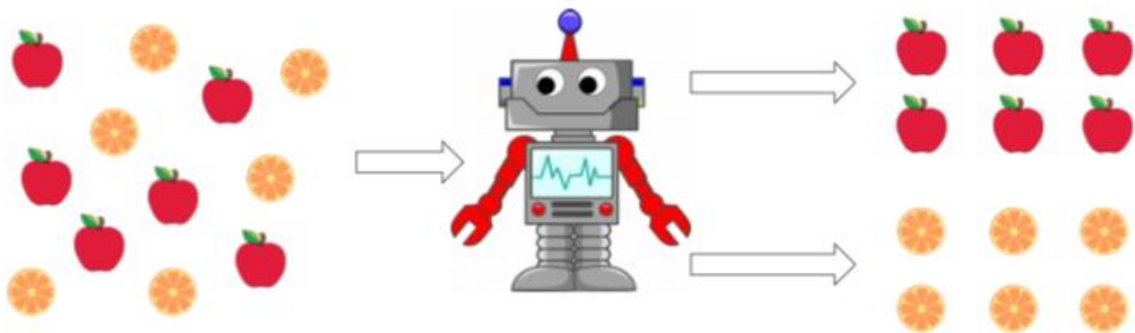
Figure 10 Audio pipeline

(source: <https://thesoundofai.slack.com>)

We have a new concept at the end of the pipeline. The next chapter will be dedicated to exploring Neural Networks, the foundation of Deep Learning models.

## 2.3 Deep Learning and Neural Networks

In previous sub-chapters, we have been talking about Deep learning and how it is possible to generate new music using a specific DL model. Now it is time to explain this concept and get the right context. We will start with the following questions: What is Deep Learning and how do we do generative modelling, i.e., generate new music? According to Foster [18], Deep Learning is a type of Machine Learning algorithm with more than one hidden layer that by trying to imitate how the brain works, can learn and identify high-level features present on unstructured data (non-tabular) like music or images. Traditional Machine Learning is mostly based on learning about the data to then make predictions; whereas with Deep learning algorithms, the focus is shifted to 'learning' what are the underlying features within our data to then generate something that has never existed before but still will look like the original data. A great example of this is the website <https://this-person-does-not-exist.com/en>.



*Figure 11 Unsupervised learning*

(source: <https://thesoundofai.slack.com>)

As stated above, these algorithms are based on how the brain works, hence the name of Artificial Neural Networks (ANN) although more commonly referred to just like Neural Networks (NN). This is the paradigm change that allows computers to 'learn' from the data as opposed to performing very quick calculations with it, for instance. How does this work?

### 2.3.1 The artificial Neuron

The fundamental unit of an ANN is the neuron and a NN features a significant amount of them, all set in layers. The data travels through all these units, starting on the input layer first. The information is processed when passing through the remaining hidden layers, where we keep on improving our knowledge of the data until reaching the output layer; There, a meaningful output will be displayed. For example, the probability for the inputted audio clip to be a rock song.

## Artificial Neural Network (ANN)

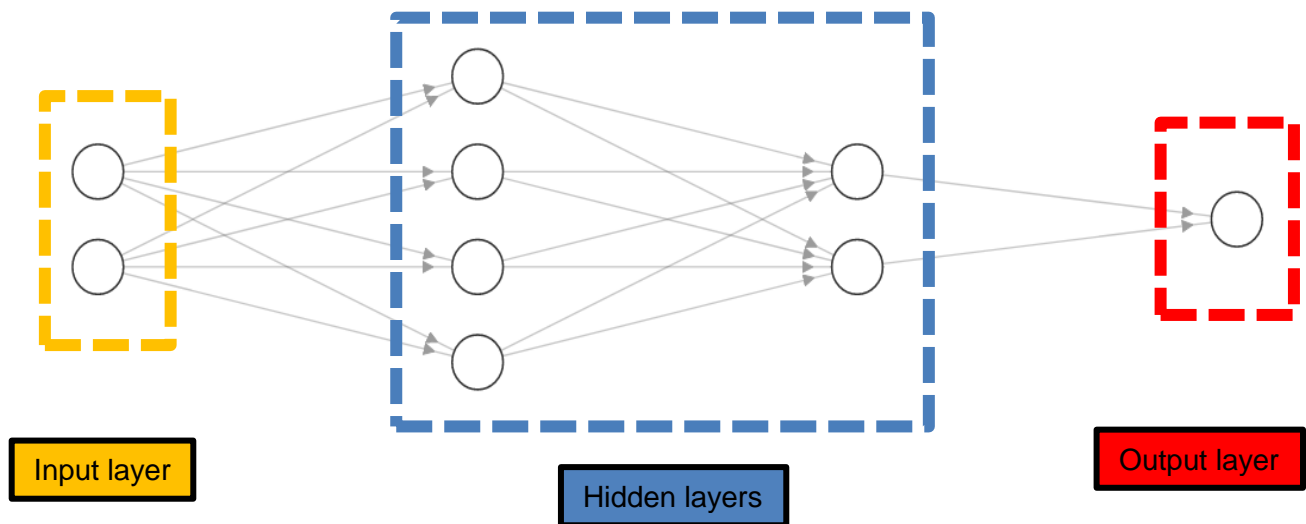


Figure 12 ANN

(created using: <http://alexlenail.me/NN-SVG/index.html> , modified by author)

This is a high-level explanation but if we look at it more mathematically, we can find the following inner structure, the Artificial Neuron itself:

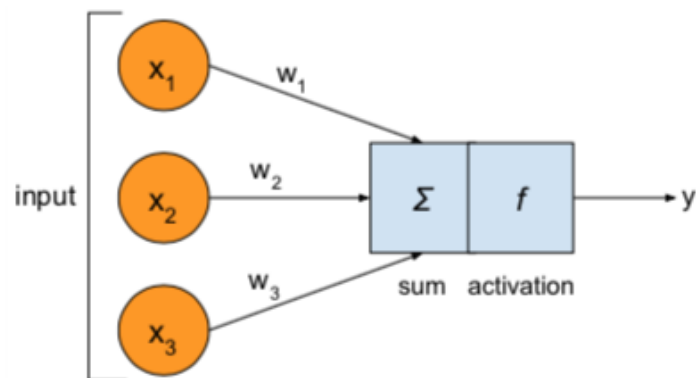


Figure 13 The Artificial Neuron

(created using source: <https://thesoundofai.slack.com> )

This would be a simplified version of a biological neuron (dendrites, cell body, etc). The AN features a series of inputs ( $x_1$ ,  $x_2$ ,  $x_3$ ) with certain weights associated with them ( $w_1$ ,  $w_2$ ,  $w_3$ ). Then we have the neuron itself that performs two computations (Sum, Activation); and finally, the output of that



computation. The first computation is the sum, comprising all the inputs multiplied by their weights. This would be where we learn and create H, the 'learning part'.

$$H = \sum_i X_i * W_i = X_1 * W_1 + X_2 * W_2 + X_3 * W_3$$

Then we have the activation function where we pass H. This would be the 'insight' we need to pass on, as we can see it is a function of H,

$$y = f(H) = f(X_1 * W_1 + X_2 * W_2 + X_3 * W_3)$$

### 2.3.2 Activation and Cost functions

Activation functions are very important for NN because thanks to them we achieve non-linearity in the functions. If there were only linear functions, complex data, like the one we need to use for our project, would not be properly modelled and it would be oversimplified; effectively defusing the point of having layers that go deeper and deeper. It is also key, as this function will provide us with the activation or not of the neuron; controlling how the information is passed on to the output layer.

For our problem, Sigmoid and ReLU are commonly used functions. Sigmoid's equation is as follows:

$$f(x) = \frac{1}{(1 + \exp^{-x})}$$

This equation produces values in the range of [0,1]. This is quite useful as output if an image is fake or real, for instance. This function is quite common for GAN frameworks as it will output 0 for fakes and 1 for real images.

Another function that is quite common for GAN frameworks is Tanh or tangent hyperbolic function. This could be seen as an extension of sigmoid but with a higher range: [-1,1], instead of [0,1]. Depending on our type of NN we will use one or the other, as both present their own advantage and disadvantages<sup>15</sup>.

ReLU is another activation function that can be used in the layers of our model. It is a function that is linear with positive values and non-linear with negatives. It is described by the following equation:  $f(x) = \max(0, x)$  or using python as `def ReLU(x)` where we would return `max(0, x)`. As we can see it would not be very computationally heavy to implement, as well as having a stable behaviour whilst featuring

---

<sup>15</sup> <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>

linear elements on it<sup>16</sup>. However, for certain DL models such as GANs, the use of Leaky ReLU is more favoured as it will improve stability. Leaky ReLU is a variation of ReLU with the function also accepting negative values:

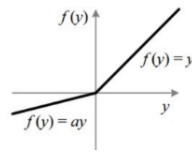


Figure 14 Leaky ReLU

(source: [https://paperswithcode.com/method/leaky-](https://paperswithcode.com/method/leaky-relu#:~:text=Leaky%20Rectified%20Linear%20Unit%2C%20or,is%20not%20learnt%20during%20training.)

[relu#:~:text=Leaky%20Rectified%20Linear%20Unit%2C%20or,is%20not%20learnt%20during%20training.](https://paperswithcode.com/method/leaky-relu#:~:text=Leaky%20Rectified%20Linear%20Unit%2C%20or,is%20not%20learnt%20during%20training.) )

Finally, we have loss functions. Its purpose is to offer a measurement of how good our ANNs work by computing the difference between the real value and the produced by the model. There are a number of them, for example Cross entropy. You can see this video of Sebastian Rachka (DL expert and professor) who mentions it when building his own GAN model<sup>17</sup>. There is another important loss implementation that has a very important role in ensuring that the training in GANs is stable; we are talking about the Wasserstein loss with gradient penalty<sup>18</sup>. Thanks to this implementation many of the training issues with GANs are minimized; this will be an important factor for our future model as well.

### 2.3.3 Training our model (Gradient Descent and Backpropagation)

When we train our models, we want to obtain the best possible outcome. The algorithms mentioned above will allow us to do so by minimizing the loss function. This means that eventually we will obtain the best possible result and there is no need to train our model any longer. We do this by feeding some inputs to our network and then looking at the predictions. The difference between those predictions and the expected value will be the loss function. We will use that information to then iteratively adjust the weights until the best loss function is achieved. The iterative process would be as follows:

On the first pass, we feed forward the data and then a prediction is achieved. The loss function will be calculated, and the error will be fed back to the initial layers; it will be backpropagated. On the backpropagation process we calculate the gradient of the error in relation to the weights; that's the information that we need to update the parameters of the weights. The point of using gradient descent

<sup>16</sup> [https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/#:~:text=The%20Leaky%20ReLU%20\(LReLU%20or,Neural%20Network%20Acoustic%20Models%2C%202013.](https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/#:~:text=The%20Leaky%20ReLU%20(LReLU%20or,Neural%20Network%20Acoustic%20Models%2C%202013.)

<sup>17</sup> <https://youtu.be/cTlxZ1FO1mY>

<sup>18</sup> <https://paperswithcode.com/method/wgan-gp>

is to ensure that the weights are going to be updated in a way that will improve the prediction and therefore minimize the loss.

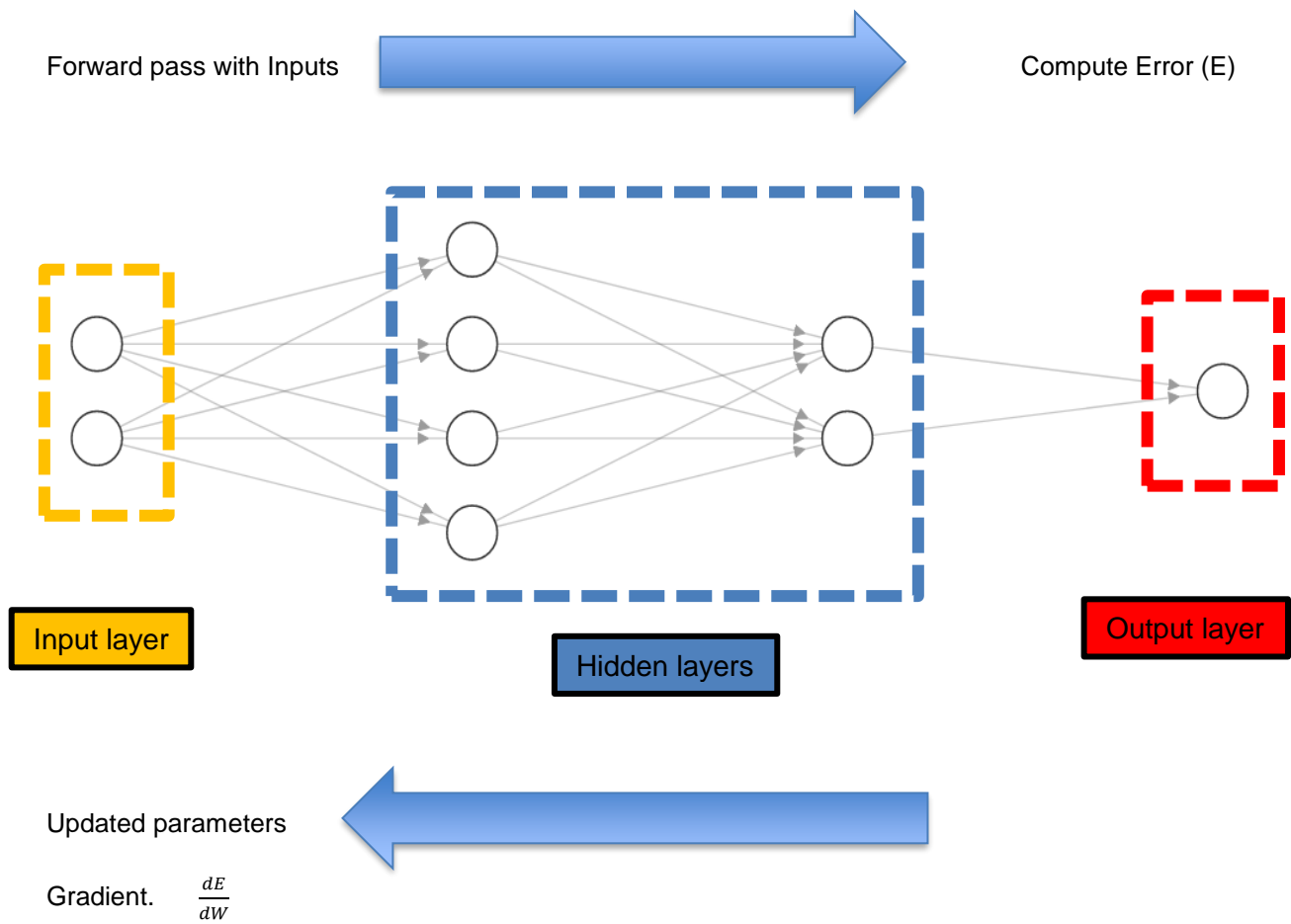


Figure 15 Training for ANN

(created using: <http://alexlenail.me/NN-SVG/index.html> modified by author)

## 2.4 Convolutional Neural Networks (CNNs)

As seen in figure 2.11 of the previous section, all the neurons are connected to each other. But that's not always the case. Convolutional Networks (LeCun, 1989) are an example of a different internal structure where not all neurons are fully connected. CNNs were developed for processing images and it turns out that they have fewer parameters and perform better at this task than the Multilayer Perceptron, for instance. Since we already can transform our audio data into images, we now find ourselves with a powerful neural network that is able to process images with good performance results. Let's find out what is the process behind it, assuming that our audio is an image already.

CNNs try to emulate the way human vision perceives images. They do so by extracting different types of features like edges, vertical bars etc. using two main components: Convolution and pooling.

Convolution is the process where we apply kernels (seen as a grid of weights) whose function is to 'filter' when applied to the image (when we have it as a grid of numbers/ pixels from 0 to 255).

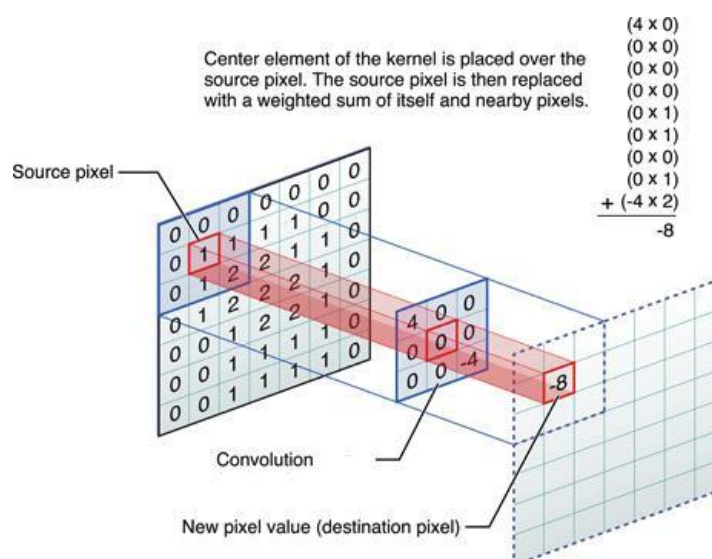


Figure 16 Convolution

(source: <https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411> )

As seen in the figure, a forward pass over all the pixels will happen and the dot product between the value of the pixel and the values of the kernel will be computed. The output will be a grid of values of the same size as the original image. The final values will depend on the type of kernel/ filter applied. What those values represent is the way that we learn features as we apply different types of filters.

There is an issue: if we do this procedure as shown in the image, the edges will not get a number. There are two solutions for this problem: One would be to ignore them, resulting in a loss of information, or the other one would be what is called applying zero padding. This entails surrounding the original

images with an edge full of zeros to enable the dot product to happen and maintain the integrity of the image.

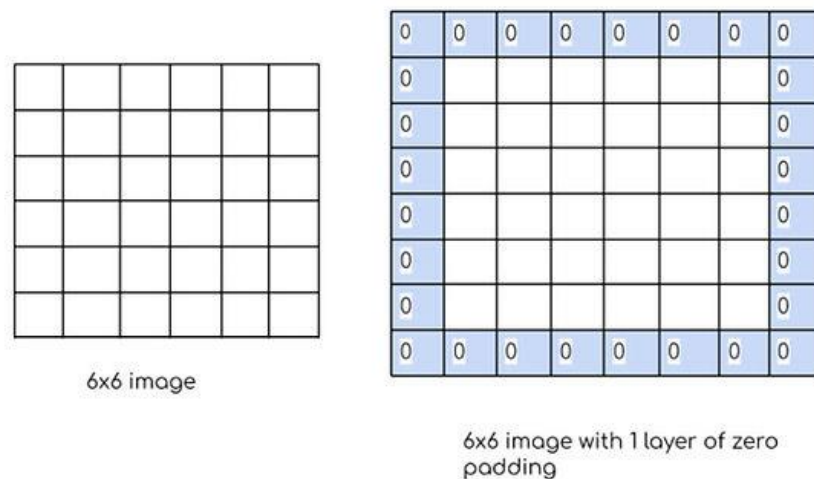


Figure 17 Zero padding

(source: <https://www.numpyninja.com/post/how-padding-helps-in-cnn> )

Some other parameters shape the output of a convolutional layer such as Filter, Depth, Stride or number of kernels. But what we really need to take on board about a CNN is that will learn what kernels (the numbers) apply, as our weights will be the variable the whole process. If we perceive kernels as feature detectors, then, when the CNN learns about what kernels to use, the actual shapes and features that are present on the input image will be processed until the best possible result is achieved.

Pooling is the other process that happens in a CNN. It is essentially a downsampling of the image with a protocol quite similar to what happens in convolution but without any weights that learn over time. There are some other parameters that control how the process (Stride, grid size) is done but the important takeaway here is knowing that the most common pooling function for DL is Max Pooling<sup>19</sup>.

---

<sup>19</sup> <https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/#:~:text=A%20convolution%20is%20the%20simple,input%2C%20such%20as%20an%20image.>

3	9	7	4	7	5
0	6	7	3	1	2
2	4	5	0	3	2
3	7	5	0	2	1
1	5	0	7	3	6
8	9	2	5	1	8

9	7	7
7	0	3
9	7	8

Figure 18 Pooling example

(source: <https://programmatically.com/what-is-pooling-in-a-convolutional-neural-network-cnn-pooling-layers-explained/> )

If we look at the example, it keeps the biggest number. This means that preserves the most important information of a specific feature. This could be compared to a compressor in music, where the signal is squashed so we hear everything in a more compact way.

To sum up, a CNN uses pooling and convolution following a specific architecture/ configuration in order to learn important features (Low level/ high level) from the input. The order is as follows: Convolution--ReLU---Pool, and can go as deep as we want. After that, it usually goes to a fully connected layer before reaching the output layer. Depending on the problem the output layer will be a sigmoid or a SoftMax function, for instance. The example below will feature a SoftMax as the final layer, outputting the probability distributions for the input image to be a car, a bicycle etc. Obviously, the most probable outcome will have the highest percentage.

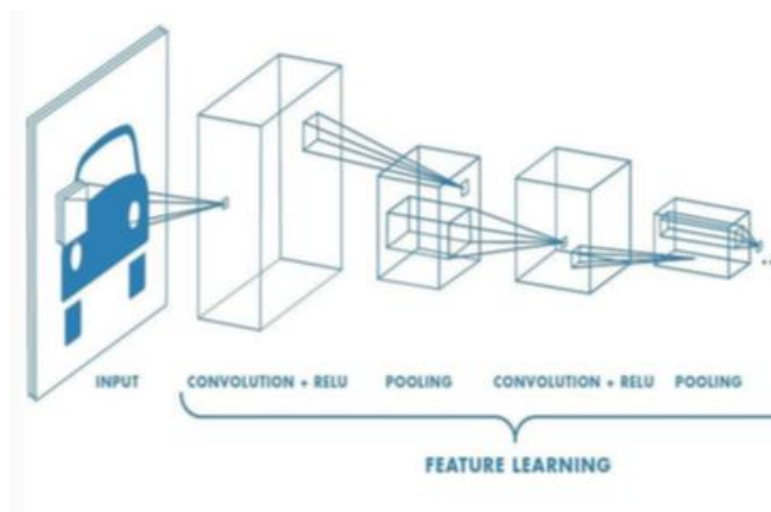


Figure 19 Pooling example, feature learn

(source: <https://thesoundofai.slack.com>)

## 2.5 Generative Adversarial Networks (GANs)

So far, we know that we are able to transform our audio files into images and we have CNNs specifically designed for images, notably good at extracting features. That means that we are in a good position to create a model that can classify our music, for instance. Our problem is not that one, we need to generate new images that will be translated back into audio. How do we do that then? As we specified in our background chapter, we are going to use GANs as the main model for our solution. What is a GAN?

### 2.5.1 GAN-Basics

In the paper Generative Adversarial Nets, published in 2014, Ian Goodfellow originally introduced the GAN. It is a DL framework able to generate new data after 'learning' the distribution of the original training material. The adversarial element gets its name due to the nature of the interaction between the two models that together make a GAN: the generator and the discriminator. These two models 'play' an adversarial game quite similar to the one that a counterfeiter would play with the secret service in the EEUU. Following this analogy, the generator's job as the counterfeiter is to create fakes that will be as good as the original ones. The job of the discriminator as in the secret service is to ascertain whether the notes that he/ she oversees are fake or real. At the training stage, the generator will try to iteratively create better and better fakes whereas the discriminator will work to its best at detecting them, no matter how good they are. The counterbalance in this game will be reached when the generator's fakes are so perfect that the discriminator's ability to predict is impaired and it is only capable of outputting only a 50/50 chance of detecting the fake. That is what is called 'convergence' in DL terms.

As per Goodfellow's paper<sup>20</sup>, we can address this adversarial situation using the following notation:

- $x$  is an image belonging to the original dataset.
- $D$  is the discriminator, a neural network whose output  $D(x)$  will be the probability of  $x$  being an image of the original dataset or a fake. The output will be high when it's real whereas the output will be low if otherwise. This part of the framework is a binary classification problem. We could use the label 0 for fake and the label 1 for real.
- $z$  would be a random vector that will be mapped through the function  $G$  into a data distribution  $P_g$ . eventually to achieve the highest possible resemblance to the original data distribution  $P_{data}$
- $G$  will be the generator and  $G(z)$  will be the generated sample or fake.

---

<sup>20</sup> <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>

From this, we can infer that  $D(G(z))$  will be the probability of the output coming from the generator of being real. This is where the game comes into place with  $D$  trying to maximize his chances of spotting fakes as in  $\log(D(x))$  and  $G$  trying to minimize the probability of  $D$  spotting what the generator outputs as fake (represented as  $\log(1 - D(G(z)))$ ). This results in the following formula:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Figure 20 Minmax function

(source: <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf>)

A visual of the training cycle would look as follows:

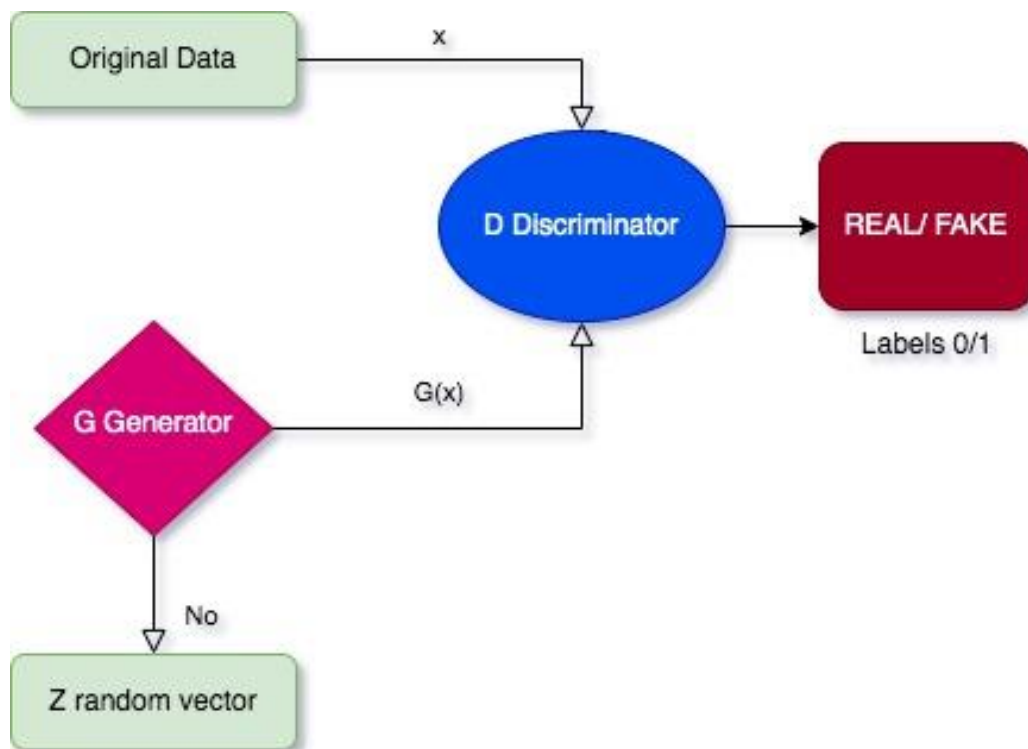


Figure 21 Minmax High Level

(source: author using draw.io)

As both  $D$  and  $G$  are neural networks this will work until the best outcome is achieved (It also can be a zero-sum game, as one has to lose). That will be done by assessing the loss function; Its information will be backpropagated and  $G$  will assess how often was caught by the discriminator whereas  $D$  will assess how often was outsmarted by  $G$ . Functions and weights will be updated and the process will be



repeated until convergence is achieved. The figure below also offers a more in-depth view of the process:

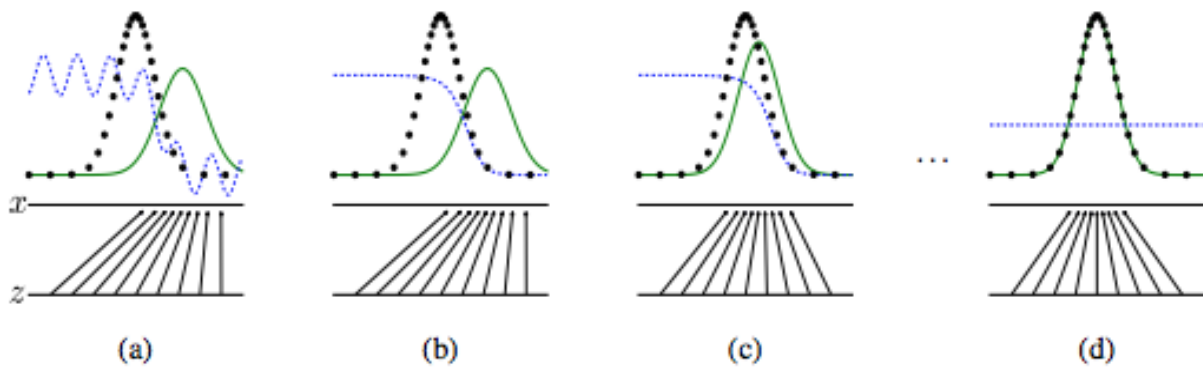


Figure 22 Training cycle on a GAN

(Source: <https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf> )

In figure A (First phase of training) we can see the real distribution of the data (black dotted line), the initial data distribution of the random noise vector (green) and the discriminator mapping only a few of the real data properly. Figure B and C show how training develops: how the generator is improving the quality of his fakes; at the same time, the discriminator improves at classifying both fakes and real data. Eventually, convergence is achieved, and the discriminator can only classify at  $D(x) = 1/2$ , hence the parallel line. This is the point where  $P_g = P_{data}$  as figure D shows (convergence is achieved).

## 2.5.2 GANs-Training issues

GANs are problematic and they are prone to generate issues in training, not only in images or music. The most relevant problems are:

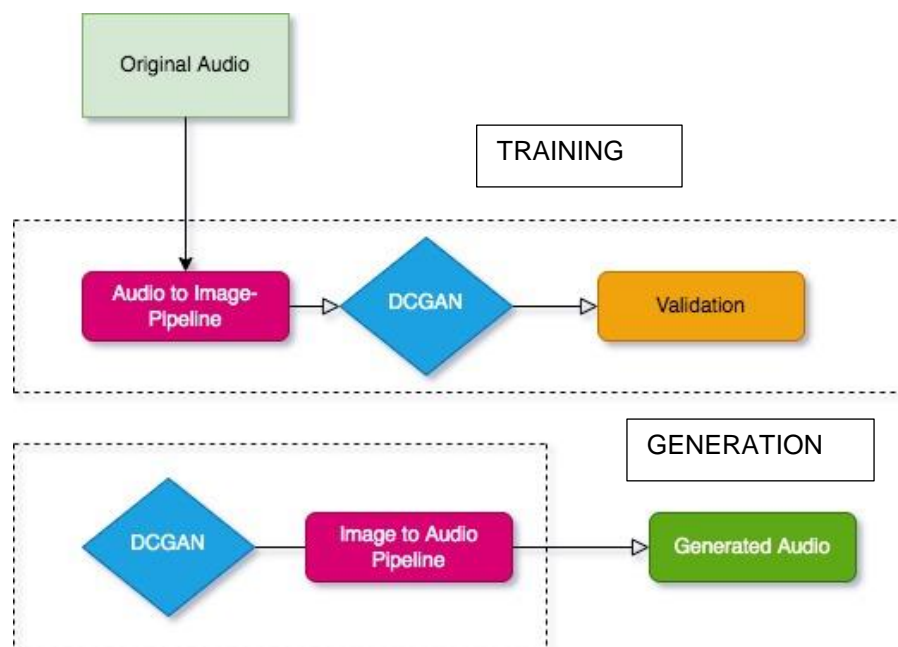
- Non-convergence: Situation D of figure 2.16 is not achieved, the model is unstable.
- Mode collapse: generator drops some parts of the original data and fails to generate meaningful output. An example would be only generating 1's when doing training on the MNIST dataset.
- Vanishing gradient: inability to update the weights during learning iterations as there might be a 'perfect' discriminator.

These could be considered the main issues but also care should be taken when selecting different sets of hyperparameters for training or ensuring that both discriminators are balanced. There is plenty of

literature discussing all these issues and their solutions. Both articles<sup>21</sup>, papers<sup>22</sup> and GitHub<sup>23</sup> repositories tackle these issues. As far as this report goes, this is not such a relevant issue, as over the years researchers have been able to tackle this issue and develop stable and well-performing GAN frameworks.

## 2.6 GAN models for audio generation

Introducing the DCGAN (Radford, Alec, et al, 2016) was a major step toward achieving a stable GAN model and also a way to introduce CNNs into the GAN structure<sup>24</sup>. As we can remember CNNs are very effective at extracting features therefore this model should help us with our problem in terms of achieving a stable, reliable and accurate model for our solution. Therefore, now we would be in a position of generating audio samples, by implementing the following workflow:



*Figure 23 DCGAN Generation Audio*

(Source: author)

21 <https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-adversary-networks-819a86b3750b>

22 <https://arxiv.org/pdf/1606.03498.pdf>

23 <https://github.com/soumith/ganhacks>

24 <https://paperswithcode.com/method/dcgan>

This workflow would produce new audio clips and could be coded using Pytorch<sup>25</sup> alongside Pytorchaudio<sup>26</sup> or Librosa<sup>27</sup> as auxiliary modules for the audio/ image pipelines. This approach would imply using Mel spectrograms as discussed in previous chapters, to then generate audio by using a phase reconstruction algorithm like Griffin-Lim [21]. This method has been implemented in autoregressive models such as Tacotron [22] or MelNet [23] with meaningful results. As for GANs and although very stable, there are clear disadvantages from our problem's perspective:

- It is indeed an old GAN framework. At the moment of writing this report there are more advanced models such as WaveGAN or GANsynth.
- DCGAN still will require a big enough dataset for training purposes. As previously stated, we have sufficient material, and we could be able to provide enough data for training and validation. However, in order to get decent enough results, you must train the model for at least 10000 epochs<sup>28</sup>, using a GPU. After consultation with my supervisor and getting validation on the number of epochs required, this method was scraped as a possible solution due to the lack of computing resources to undertake the task.

Another possible solution could be WaveGAN, a DCGAN-based model introduced by Chris Donahue et al. in 2019. This is a significant improvement from the DCGAN approach stated in the figure. The GitHub repository<sup>29</sup> features improvements such as the possibility of training WaveGAN only using raw audio samples as input, multiple channels (not limited to mono input/ 1 channel) or new compatibility for python 3. Not having an audio processing pipeline, either way, is really a significant advance, since the Griffin Lim algorithm will turn generated images into audio files, but it will not achieve the same quality as the original input file<sup>30</sup>. That is a noteworthy improvement, but we have a serious issue with this model. WaveGAN is capable of generating audio samples whose maximum length is four seconds (at 16Khz). That is nowhere near the number of seconds that we need for our 'chunks' to address issues 1 and 2 of our problem. As stated, we need a minimum of 25 seconds to feature at least one verse or one chorus (or both). That is, a meaningful musical structure that can be recognized as such by our songwriter. Therefore, WaveGAN cannot be part of our solution.

Finally, there are two more viable options that can be explored as our GAN framework: GANSynth [24] and Catch a Waveform (Greshler et al. 2021). The first one was key in helping us choose a GAN framework as our main element for our solution. As a paper, it clearly states the reasons why autoregressive models are not an efficient solution for our problem, however, GANSynth is not a viable

---

<sup>25</sup> [https://pytorch.org/tutorials/beginner/dcgan\\_faces\\_tutorial.html](https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html)

<sup>26</sup> [https://pytorch.org/tutorials/beginner/audio\\_preprocessing\\_tutorial.html](https://pytorch.org/tutorials/beginner/audio_preprocessing_tutorial.html)

<sup>27</sup> <https://librosa.org/doc/latest/advanced.html>

<sup>28</sup> <https://towardsdatascience.com/gans-generative-adversarial-networks-an-advanced-solution-for-data-generation-2ac9756a8a99>

<sup>29</sup> <https://github.com/chrisdonahue/wavegan>

<sup>30</sup> Note from the author: this can be demonstrated empirically by taking an original wav file, transforming it to a Mel Spectrogram and then reversing the process. This can be done either with Librosa or Pytorch and regardless, the resulting audio will have less qualified than the original.

option either. Although the resulting generated file is an audio file, the whole training process is MIDI based<sup>31</sup>. This is something that contradicts our aim of maintaining the nuances of the original music. Translating the L'Anima dataset into MIDI would be the same as using an MP3 file instead of a WAV: we would lose key information and the final result would be seriously compromised in terms of quality. This argument will be fully explained in the next chapter.

The final model is CAW<sup>32</sup>, a model presented in a paper whose main aim is to address an issue that is quite common across the field of audio generation: not having enough data to train your model. This is not our main problem per se. When perusing the paper, you can find that this model can perform several tasks: Unconditional Generation, Music Variations, Bandwidth Extension, Inpainting and Denoising; will all of them needing from 20 to 100 seconds to produce valid and relevant results. As for Unconditional generation, they state that their model does generate samples that keep sonic coherence with the original signal, and it only needs 20 seconds of audio to generate meaningful results. They also state that it is possible to create consistent and intelligible variations of a song, adding extra sections. The examples of these two case studies on their website (please see 30) are indeed proof that the previous statements are true. These premises fit very well our needs as a possible solution for both issue 1 and 2 of our problem. Also, it deals with audio files as initial input and although the output files are at 16Khz and mono; it is enough quality to be used as a prompt. The examples of their website are self-explanatory. It would be a different matter if we were to use them as final elements but again, we believe they are fit for purpose when used as prompts. We also have seen how training time and computational resources are decisive factors. Running models for 3000 epochs is the amount that they have used to achieve the results shown on their website. This equals 10 hours of training per model and sample, using a GPU. This is something we can achieve with the means at our disposal.

This is a summary of the advantages of choosing CAW over other GAN models (and autoregressive for that matter):

- Once trained on a sample, CAW is able to generate audio clips in the style of the training sample, keeping timbre and semantic coherence.
- No vast dataset is needed for training. Surgical training can be done (train on specific samples).
- Computationally viable.
- Audio generated clips at 16Khz, enough quality for human evaluation.

After reviewing all the possibilities at hand and balancing all the pros and cons, we are now in a position to establish our final methodology for this project (please see below):

---

<sup>31</sup> [https://colab.research.google.com/notebooks/magenta/gansynth/gansynth\\_demo.ipynb](https://colab.research.google.com/notebooks/magenta/gansynth/gansynth_demo.ipynb)

<sup>32</sup> <https://galgreshler.github.io/Catch-A-Waveform/>

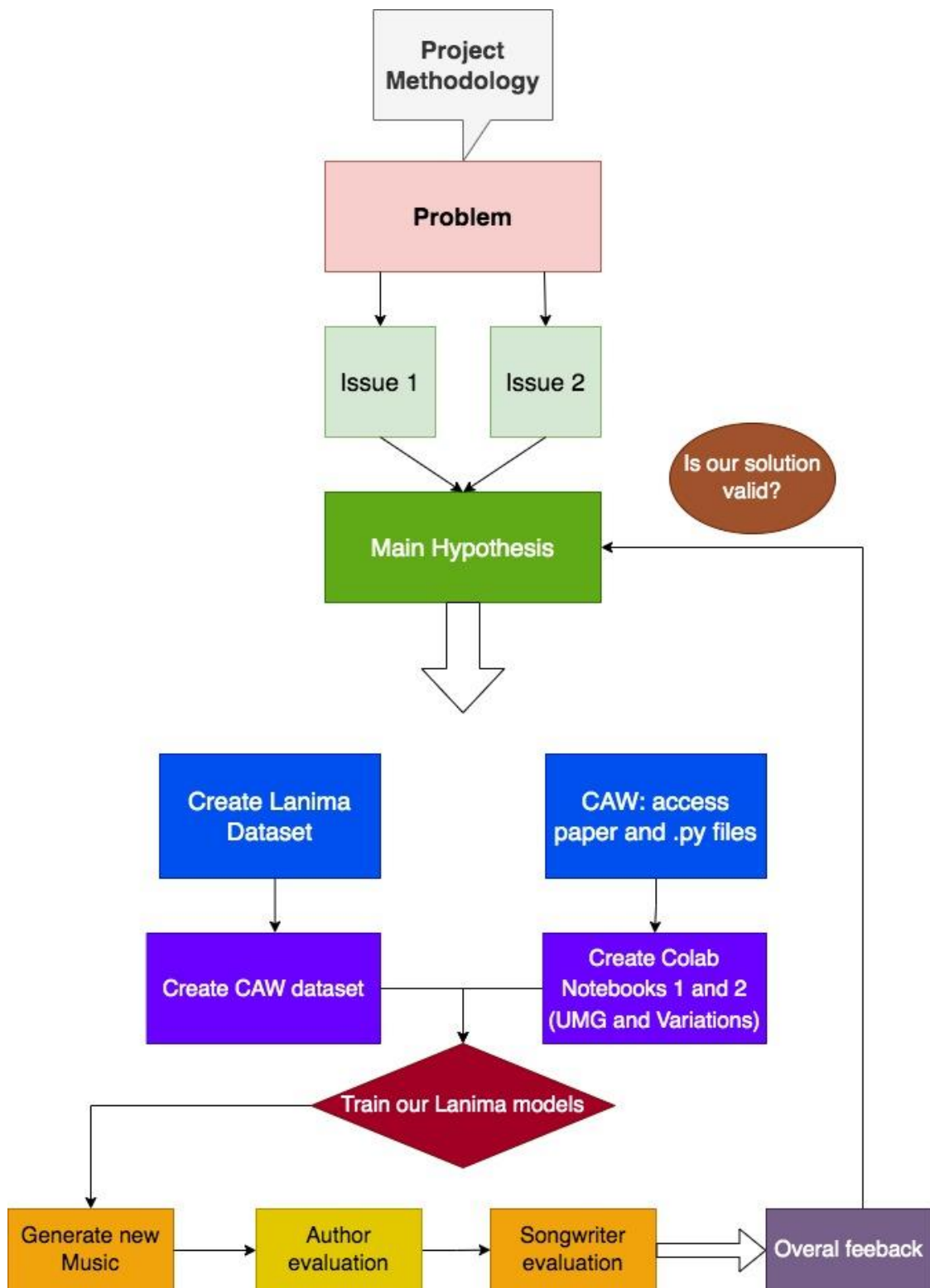


Figure 24 Final project's methodology

(Source: author)

### 2.6.1 Catch a Waveform (Unconditional Music Generation, Variations)

As previously stated, CAW is a model that can be trained using a single raw audio file as input. Also, there is no need for any other pre-training and it is unsupervised. This means that once trained on a specific sample it will be able to generate semantically similar audio files to the original, adding newly 'composed' elements and structures [26]. In Music terms, this could be seen as creating different 'improvisations' over the same motive. What does this mean for us in terms of our solution? We have an initial number of unfinished songs for which we could train this model and obtain new 'improvisations' for specific sections that need reworking or still do not fit well. Therefore, and although computationally costly, it opens the way to an iterative process of experimentation where potentially it could be possible to rework all the sections of a given song by providing 'improvisations' of them. We can iterate through the all L'Anima songs and identify suitable samples to then generate a number of future prompts that can potentially help to finish the 'problematic' songs. The question now is to see whether this works empirically or not, as stated in our main hypothesis.

### 2.6.2 Method

If we look at figure 2.19, we can infer that it is a multi-scale GAN featuring a series of generators and discriminators trained in a sequential manner, starting at the lowest sample rate and finishing at 16KHz.

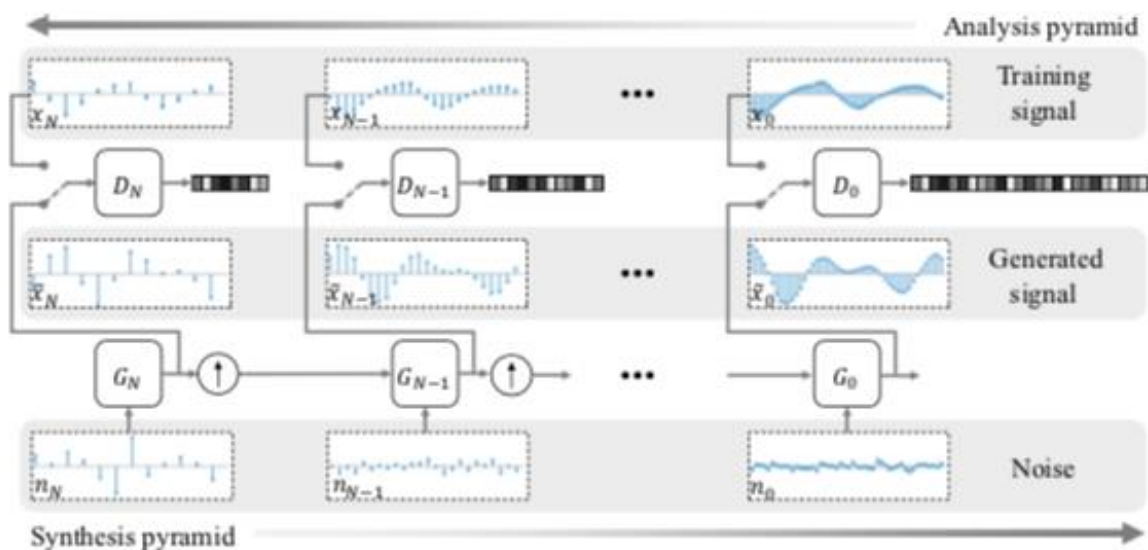


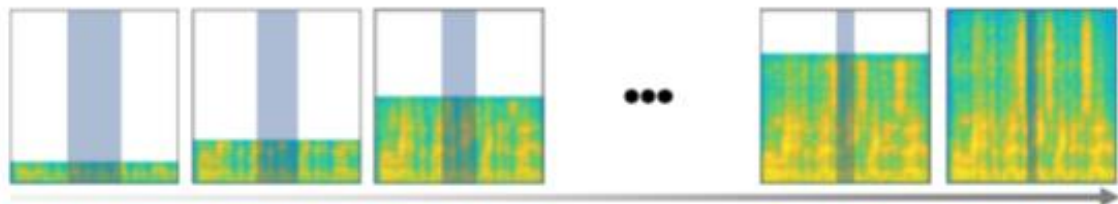
Figure 25 High level view of CAW

(Source: CAW paper)

The initial sample scales are as follows:

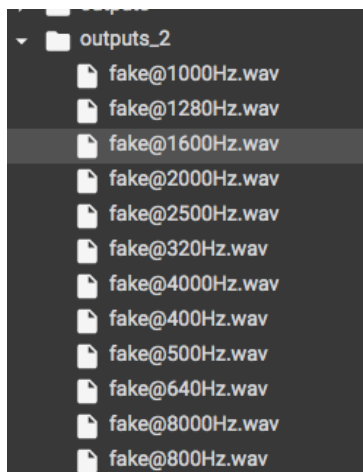
```
fs_list = [320, 400, 500, 640, 800, 1000, 1280, 1600, 2000, 2500, 4000, 8000, 10000, 12000, 14400, 16000]
```

That is what the paper calls the synthesis pyramid, where for each layer, noise is fed to the generator. This will prompt it to generate a fake sample that will be fed to the discriminator alongside a downsampled version of the original sample. Then the discriminator will decide which one is real or fake. Then we train the generator as much as possible with the aim of fooling the discriminator. The important point here is using a downsampled version of the original as the real sample to feed the discriminator. By doing so we ensure that variations of the original sample are created and incorporated throughout the process, otherwise, we would end up with the original sample. Once the first discriminator is trained, we produce a fake and take it to the new frequency band and use it as input together with some new noise to feed the next generator. Then repeat the same adversarial process as before but with the original sample again downsampled to its correspondent frequency band. This process will be sequentially executed a total of 16 times in a course to fine fashion. Throughout the whole process, the receptive field is kept constant in the generators to ensure that at the lowest levels it captures the long-range dependencies that happen on the lower frequencies. It will be mostly drums and bass to then build the process up when adding the rest of dependencies (guitars, overtones etc.) on every pass through the frequency bands.



*Figure 26 Up sampling process*

(Sources: CAW paper and code from author)



As per the figure, we start with the lowest sampling rate and then keep on adding frequency bands to the signal generated on the previous layer. That is, overlapping distributions at different scales until the final sample is fully fledged. The image on the left shows the fakes that are generated at every stage.



## 2.6.2 Training

As stated in the previous sub-chapter training is sequential and it uses a convolutional G and D per every stage. The most important aspect of training is the fact of having two different losses per every scale. Firstly, we have the adversarial loss for which they use Wasserstein GAN loss (Arjovsky et al, 2017) alongside gradient penalty (Gulrajani, Ishaan et al, 2017). This is to ensure that convergence is achieved, and the model is stable. Then we have reconstruction loss, which works at each layer ensuring that if there was no noise inputted, the model still would map out the original sample. This is important because given the pyramidal nature of the model we need to enforce a reconstruction process that ends up in a fully-fledged sample.

```
|***** Scale 15 (320 [Hz]) *****|
receptive_field = 6378[msec] (31.7% of input)
signal_energy = 0.0148
noise_amp: 1.000000
Total time in scale 15: 176[sec] (0.06[sec]/epoch on avg.). D(real): -0.136814, D(fake): -0.444169, rec_loss:
0.0159, gp: 0.0403
|***** Scale 14 (400 [Hz]) *****|
receptive_field = 5102[msec] (25.3% of input)
signal_energy = 0.0193
noise_amp: 0.084654
Total time in scale 14: 259[sec] (0.09[sec]/epoch on avg.). D(real): 0.189550, D(fake): -0.333089, rec_loss:
0.0199, gp: 0.0642
|***** Scale 13 (500 [Hz]) *****|
receptive_field = 4082[msec] (20.3% of input)
signal_energy = 0.0230
noise_amp: 0.000608
Total time in scale 13: 312[sec] (0.10[sec]/epoch on avg.). D(real): 0.739244, D(fake): 0.111347, rec_loss: 0.0230.
gp: 0.1776
|***** Scale 12 (640 [Hz]) *****|
receptive_field = 3189[msec] (15.8% of input)
signal_energy = 0.0252
noise_amp: 0.000461
Total time in scale 12: 422[sec] (0.14[sec]/epoch on avg.). D(real): 1.241792, D(fake): 0.989646, rec_loss: 0.0219.
gp: 0.0433
|***** Scale 11 (800 [Hz]) *****|
receptive_field = 2551[msec] (12.7% of input)
signal_energy = 0.0265
noise_amp: 0.000363
Total time in scale 11: 549[sec] (0.18[sec]/epoch on avg.). D(real): 1.221243, D(fake): 0.676117, rec_loss: 0.0232.
gp: 0.1938
|***** Scale 10 (1000 [Hz]) *****|
receptive_field = 2041[msec] (10.1% of input)
signal_energy = 0.0271
noise_amp: 0.000257
Total time in scale 10: 698[sec] (0.23[sec]/epoch on avg.). D(real): 0.652178, D(fake): 0.296320, rec_loss: 0.0212.
gp: 0.1161
|***** Scale 9 (1280 [Hz]) *****|
receptive_field = 1594[msec] (7.9% of input)
signal_energy = 0.0278
noise_amp: 0.000264
Total time in scale 9: 882[sec] (0.29[sec]/epoch on avg.). D(real): 0.404019, D(fake): 0.001139, rec_loss: 0.0202.
gp: 0.1122
|***** Scale 8 (1600 [Hz]) *****|
receptive_field = 1275[msec] (6.3% of input)
signal_energy = 0.0283
noise_amp: 0.000220
Total time in scale 8: 1100[sec] (0.37[sec]/epoch on avg.). D(real): -0.290176, D(fake): -0.633771, rec_loss:
0.0202, gp: 0.0317
|***** Scale 7 (2000 [Hz]) *****|
receptive_field = 1020[msec] (5.1% of input)
signal_energy = 0.0290
noise_amp: 0.000267
Total time in scale 7: 1317[sec] (0.44[sec]/epoch on avg.). D(real): -0.565926, D(fake): -0.844825, rec_loss:
0.0201, gp: 0.0372
|***** Scale 6 (2500 [Hz]) *****|
receptive_field = 816[msec] (4.1% of input)
signal_energy = 0.0297
noise_amp: 0.000257
Total time in scale 6: 1642[sec] (0.55[sec]/epoch on avg.). D(real): -0.637351, D(fake): -0.917101, rec_loss:
0.0219, gp: 0.0324
|***** Scale 5 (4000 [Hz]) *****|
receptive_field = 510[msec] (2.5% of input)
signal_energy = 0.0304
noise_amp: 0.000271
Total time in scale 5: 2468[sec] (0.82[sec]/epoch on avg.). D(real): -0.841339, D(fake): -0.891798, rec_loss:
0.0229, gp: 0.0129
|***** Scale 4 (8000 [Hz]) *****|
receptive_field = 255[msec] (1.3% of input)
signal_energy = 0.0316
noise_amp: 0.000339
Total time in scale 4: 4749[sec] (1.58[sec]/epoch on avg.). D(real): -0.669727, D(fake): -0.762075, rec_loss:
0.0289, gp: 0.0190
```

Figure 27 Training cycle, 3000 epochs

(Source: code from author)



## 2.6.4 Architecture

The architecture for this model is a variation for audio of SinGAN (Shaham, Tamar, et al., 2021) with a major difference in having 8 fully convolutional layers as opposed to 5 in both the generator and the discriminator.

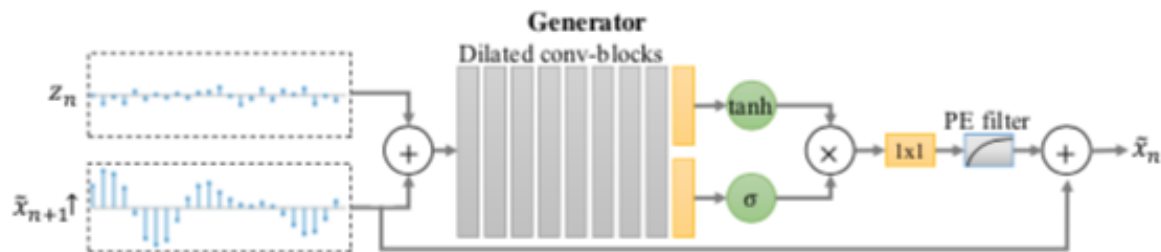


Figure 28 Generator

(Source: CAW paper)

As per the image, we have eight dilated convolutional layers with the first seven layers featuring Batch-Norm and LeakyReLU (slope 0.2). Layer eight is just convolutional. The use of dilation is to achieve a higher receptive field in the coarser layers (we get the most features out of the low end-good for rock and metal). You can appreciate this in figure 2.20. A Gated activation function comes after (Tanh and a sigmoid) to finish with a  $1 \times 1$  Conv layer and a filter.

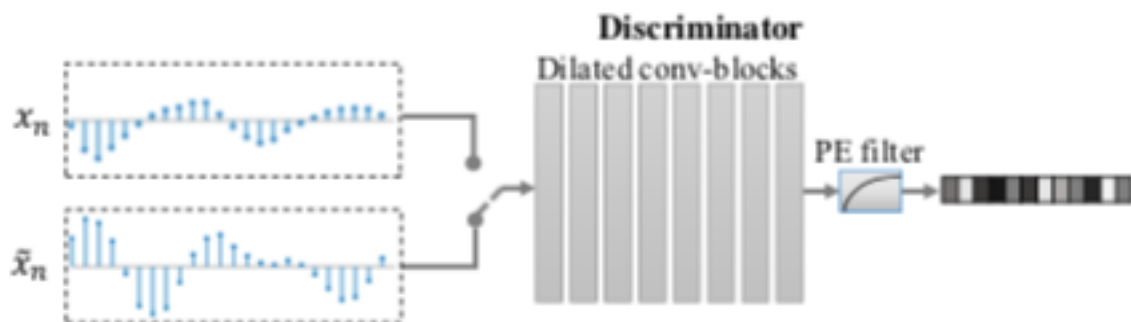


Figure 29 Discriminator

(Source: CAW paper)

The discriminator is quite similar to the generator but without the activation unit. Notice that both the generator's output and the real signal have the same length. To ensure this there are two actions, the first one can be seen in figure 22 when injecting noise with the same length. The second is using zero padding as input too. This results in the final generated sample having the same start and ending as

the original. This is something that we will have to bear in mind when we present the final samples for human evaluation as we want to avoid the feeling of being the same sample over and over. Trimming it by 200 Ms should suffice to get rid of this effect.

## 2.6.4 Other parameters

This framework uses an Adam optimizer to update the weights. Since the introduction of DCGAN by Radford et al., it seems to be the default optimizer instead of using Stochastic Gradient Descent.

Other parameters for training will be using 3000 epochs per scale and a learning rate of 0.0015. These will be our initial values for the training.

## 3 Data

One of the disadvantages of GANs applied to audio is the lack of datasets that can pave the way for meaningful research on the audio field; especially if compared with image generation. MNIST and CIFAR-10/ 100 datasets are standard and available for download in frameworks such as Pytorch whereas very few audio datasets are obtainable, especially for creative use. In our case and given the nature of our problem/ solution we do not need any standard dataset. What we have done is a curation process of the extensive collection of projects that the main songwriter of our band has created for our second album. Some other data has also been curated from the demos that we both have interchanged throughout the composition process as well as some rough mixes of music that did pass the cut. But before we discuss the dataset any further, let's look at some more characteristics of audio that are relevant to our project.

### 3.1 Audio characteristics

Audio can be considered a time-series type of data as it changes over time; it can have one channel (mono), two (stereo) or multiple (quadrophonic, ambisonic, surround etc.). For the purpose of this project, we will consider Stereo as our first choice, especially when configuring our dataset. This will make our process simpler since Librosa<sup>33</sup> and SoundFile<sup>34</sup> will convert our file to mono once it uses the `load()` or `read()` function. The type of file that we will be using is .WAV, a type of uncompressed audio file that is standard in the Music field. Other models like WaveGAN<sup>35</sup> are able to incorporate other types of files, like MP3s or OGGs. For this project, we will use uncompressed audio as our original source. If we look back at our problem, we do need to maximize the options of prompting the Eureka effect on our main songwriter; therefore, will have a better chance for that to happen if the newly generated samples maintain as much of the original distribution of the data as possible. This is something that would not be possible using MP3 files, as they are compressed files that feature less information. Having said that, are some drawbacks with Wav files; it has to do with its size, with an average song taking up to 100 MB. We already discussed using up to 25-second audio clips. Consequently, we consider the trade-off size/ quality as acceptable; quality over quantity is needed in this particular case. Given the circumstance, we could train even bigger files, as this is a computational issue when training with GPU. Training 25/30 seconds with take up less than 16G of RAM if we use a Tesla V100 as GPU.

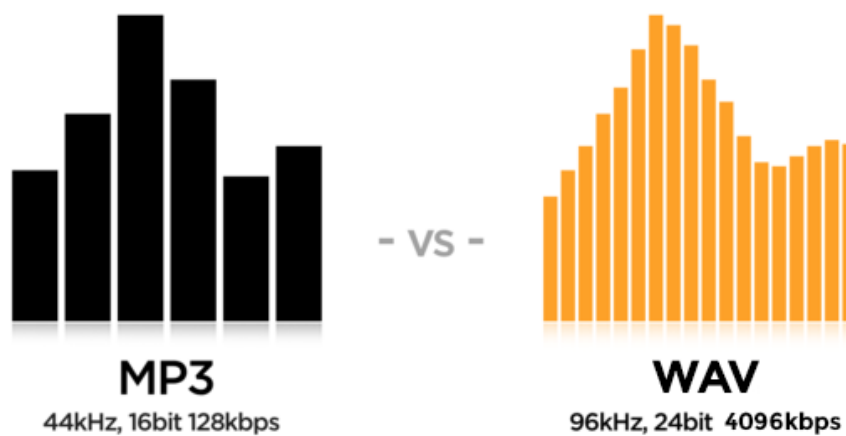
The process of creating the audio files will be done using the DAW (Digital Audio Workstation) Logic Pro X, capable of producing multiple types of audio files as output, including Wav and MP3.

---

33 librosa — librosa 0.9.2 documentation

34 <https://pysoundfile.readthedocs.io/en/latest/>

35 <https://github.com/chrisdonahue/wavegan>



*Figure 30 Wav vs MP3*

(source: <https://vox.rocks/resources/wav-vs-mp3> )

## 3.2 Analog vs Digital

The above diagram represents the process of sampling, which is to transform a musical event such as guitar playing or singing into digital. The sound waves produced by real instruments are a continuous variation of pressure over time. We can represent either frequency or amplitude as a time-based variable which would generate a continuous signal. If we visualize this as a soundwave like figure 2.1, we could also infer that throughout that wave we could obtain infinite values of amplitude if we were as we move over the time axis. This is what we call an analog system and it is something that computers cannot understand. For instance, a computer can't record and store vocal sounds without using a mic and an interface. The mic would translate our soundwaves into electrical impulses and then the interface or A/D converter would translate those electrical impulses into a series of 0's and 1's. This process is done by discretising the signal and it's called again sampling. By sampling, we mean taking a series of discrete points from the soundwave to create a digital image of it. The more of those points we take the better the representation of the soundwave will be. Figure 3.1 gives us a high level view of the process where the more bins we feature the better contour of a soundwave we will obtain and of course, better quality.

This is the way of encoding an analog signal into digital, where we take sample points of the soundwave. The value of the samples is in the range of -32768 and +32767, which equates to the range of numbers that can be represented using a 16-bit configuration (note: 24-bit does not increase the quality, it allows for better music editing as it prevents distortion of the signal).

Now that we have established the range of values, we need to set the number of samples that we will take per second. If we listen to a CD it will feature music that has been encoded by taking 44100 samples per second. Why that number? This has to do with the Nyquist-Shannon Theorem<sup>36</sup> that establishes the following: when sampling an analog signal at double the rate of the highest possible value, it is possible to effectively reproduce it back from the sampled values without any issues. In Music terms what it means to us is that by sampling at 44100 samples per second, digital audio devices can reproduce frequencies up to 22050Hz which covers more than all the spectrum that humans can listen to (up to 20Khz).

Now the question would be, if our output files for the CAW model are going to be at 16Khz, how is this going to be enough quality is the maximum range that we will appreciate will be 8KHz?

The answer is again a trade-off. We have established that CAW is the best GAN framework that we can use for this project for a series of reasons. But we also note that the other options also feature output audio at 16KHz. Therefore, it is still the better model.

Secondly, if we look at the figure below, we will see the frequency spectrum division:

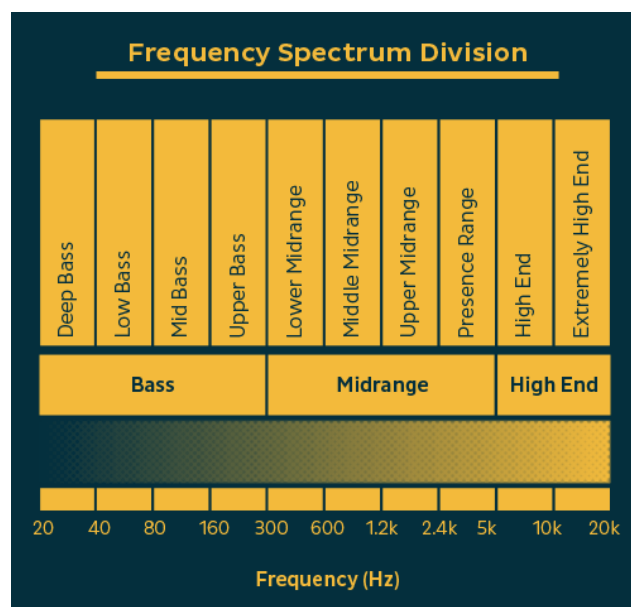


Figure 31 Human Range Frequency

(source: <https://flypaper.soundfly.com/produce/equalization-101-everything-musicians-need-know-eq/> )

The great majority of the instruments that are going to be featured in our audio clips will have their fundamental notes and a fair number of overtones within the bass and midrange categories. This includes drums, bass, guitars and vocals. Whereas some of the overtones of these instruments will fall into the high-end. The high end starts at around 5Khz and will go up to 12 Khz. This is the area where the shine and some elements of articulation of the different instruments like vocals, pianos or synths lie. If we have a cut-off point at 8khz we will be missing some of that shine and some fine detail will be

36 <https://www.allaboutcircuits.com/technical-articles/nyquist-shannon-theorem-understanding-sampled-systems/>

lost. But again, that is an acceptable trade-off as our music is mostly guitar driven and pianos or synths are marginal. Vocals will not sound as crispy and they might be perceived as a bit artificial and but again, this is not the final product, so it is acceptable. As for the frequencies over 12Khz, what mixing engineers call “air”, it is important to have them at the beginning of the training process to start with. They need to be part of the initial data distribution as we want our model to train by being exposed to the whole spectrum of frequencies, even if the output is going to be limited afterwards. This could be a developmental point for the future: try to configure the model to output files at 22050 or 44100, as well as using two channels (stereo) as input and output.

### 3.3 Graphical representation of Audio

As stated in previous chapters it is possible to represent audio in several ways. The first one would be as a soundwave and the second as a spectrogram (see previous chapters). Both representations will be used at a later stage when comparing original and generated signals to provide graphic analysis and facilitate human evaluation. As per our methodology, human evaluation will be the key element and the auditory element will be the first method of analysis. But since we are not going to include other metrics like the inception score or the similarity matrix (out of scope); we have considered effective to include some image comparison to visually evaluate how the data distribution has been learned for different audio samples. It will be interesting to see what are the areas with more energy after the generation, and if the models have respected the initial settings.

### 3.4 From L’Anima Music dataset to GAN/CAW dataset

The starting point for our data is a folder that contains all the projects that have been part of the composition of the second album of our band. Twenty-eight Logic X<sup>37</sup> folders containing a total of 60 Gb of information, needed to be filtered in order to create a dataset for our project. The process will be described below.

---

<sup>37</sup> <https://www.apple.com/uk/logic-pro/>

### 3.5.1 Artistic/ Aesthetic curation

The search for meaningful data can have a broad interpretation; in our case, the first part of the filtering has to find segments of information that were meaningful musically. So, we went through all the folders, listened to the music and had a first pass, making notes of what were the most relevant areas to look at: interesting sections and passages that could be used as starting points. After that, a second pass was done and audio clips were extracted from each song, taking into account duration as well. Some audio clips are shorter and others are larger, depending on their musical content and meaning but also looking at their duration. A total of 153 files from a range of 28 songs were extracted and placed onto three main datasets: training, validation and testing. Please see below:

```
paths = []
size = 0
for root, dirs, files in os.walk(path):
    for file in files:
        if (file.endswith(".wav") and (not (file.startswith(".") or file.startswith("noise")))):
            paths.append(os.path.join(root, file))
            size += os.path.getsize(os.path.join(root, file))

print(f'We have {len(paths)} .Wav Files with {size/1024**2:.2f} Mb in size')
```

☐ We have 153 .Wav Files with 1822.51 Mb in size

```
results = []
duration = []
labels = []

pathTr, durTr = [], []
pathTs, durTs = [], []
pathVa, durva = [], []

for i in paths:
    d = int(librosa.get_duration(filename=i))
    duration.append(d)
    if "Training" in i :
        pathTr.append(i)
        durTr.append(d)
        labels.append("Training")

    elif "testing" in i :
        pathTs.append(i)
        durTs.append(d)
        labels.append("Testing")
    else :
        pathVa.append(i)
        durva.append(d)
        labels.append("validation")
```

```
dic = {"paths":paths, "annotation": labels, "duration": duration}
Train = {"paths":pathTr, "annotation": ["Training" for i in range(len(pathTr))], "duration": durTr}
Test = {"paths":pathTs, "annotation": ["Testing" for i in range(len(pathTs))], "duration": durTs}
Valid = {"paths":pathVa, "annotation": ["Validation" for i in range(len(pathVa))], "duration": durva}

df = pd.DataFrame(dic)
trainDf = pd.DataFrame(Train)
testDf = pd.DataFrame(Test)
validDf = pd.DataFrame(Valid)
```

```
trainDf.head()
```

	paths	annotation	duration
0	/content/drive/MyDrive/GAN dataset/Training/fo...	Training	56
1	/content/drive/MyDrive/GAN dataset/Training/fo...	Training	126
2	/content/drive/MyDrive/GAN dataset/Training/fo...	Training	28
3	/content/drive/MyDrive/GAN dataset/Training/fo...	Training	24
4	/content/drive/MyDrive/GAN dataset/Training/fo...	Training	186

```
testDf.head()
```

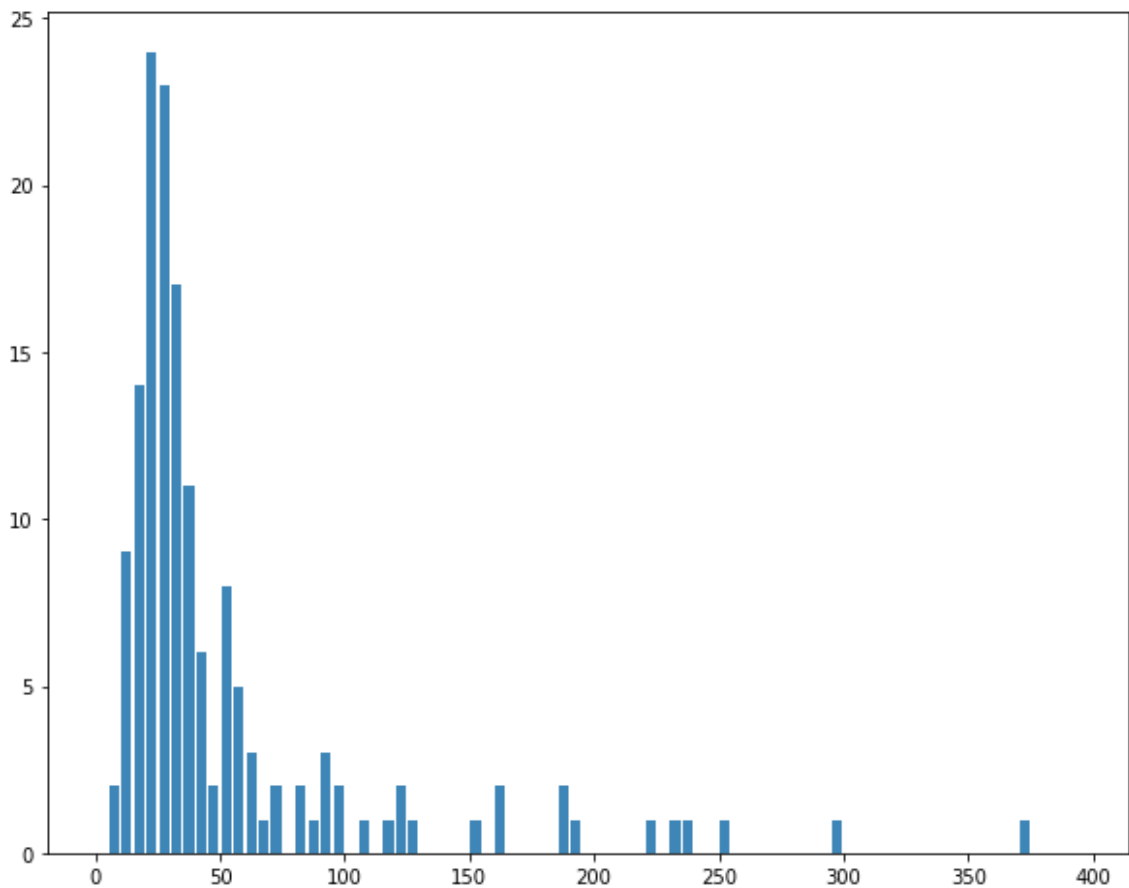
	paths	annotation	duration
0	/content/drive/MyDrive/GAN dataset/testing/fol...	Testing	25
1	/content/drive/MyDrive/GAN dataset/testing/fol...	Testing	27
2	/content/drive/MyDrive/GAN dataset/testing/fol...	Testing	34
3	/content/drive/MyDrive/GAN dataset/testing/fol...	Testing	27
4	/content/drive/MyDrive/GAN dataset/testing/fol...	Testing	44

```
validDf.head()
```

	paths	annotation	duration
0	/content/drive/MyDrive/GAN dataset/test.wav	Validation	21
1	/content/drive/MyDrive/GAN dataset/100_epochs.wav	Validation	21
2	/content/drive/MyDrive/GAN dataset/validation/...	Validation	26
3	/content/drive/MyDrive/GAN dataset/validation/...	Validation	26
4	/content/drive/MyDrive/GAN dataset/validation/...	Validation	32

(Source of the code: authors' notebooks)





*Figure 32 GAN dataset files*

(source: author)

The distribution shows that majority of the audio clips are in the range of 0 to 50 seconds. This iterative process of selection was successful as the trade-off between musicality and duration had to be upheld. GANs are computationally intensive but also long segments could be problematic in terms of 'learning' the data distribution and could lead to problems like the model 'forgetting' some key elements/ sections. Having said that some 'outliers' were kept in the dataset for their intrinsic musical value.

### 3.5.1 Technical Perspective/ Final Dataset

As previously stated, the DCGAN framework and others were scrapped as part of our solution in favour of CAW. This resulted in a new iterative process to choose the right files to create the final CAW dataset with samples taken out of sections of the following songs: S1, S3, S4, S5, S7, S9 and S11. The dataset features two samples per each song that has been chosen: one per each issue that needs addressing. That is, if S1 has been chosen, there will be one Sample S1\_UMG for unconditional generation, in order to create variations of the original sample. There will be another sample S1\_V the variation aspect, meaning that an extra section will be added to the resulting musical sample. Please see below:

```
path = "/content/drive/MyDrive/CAW_training_final"

[4] paths = []
size = 0
for root, dirs, files in os.walk(path):
    for file in files:
        if (file.endswith(".wav") and (not (file.startswith("S1_UMG") or file.startswith("S1_V")))):
            paths.append(os.path.join(root, file))
            size += os.path.getsize(os.path.join(root, file))

print(f'We have {len(paths)} .Wav Files with {size/1024/1024} Mb in size')

We have 13 .Wav Files with 31.56 Mb in size
```

```
df.head(20)
```

	paths	annotation	duration
0	/content/drive/MyDrive/CAW_training_final/S11_...	Final training samples	24
1	/content/drive/MyDrive/CAW_training_final/S4_U...	Final training samples	23
2	/content/drive/MyDrive/CAW_training_final/CAW_...	Final training samples	30
3	/content/drive/MyDrive/CAW_training_final/S9_u...	Final training samples	22
4	/content/drive/MyDrive/CAW_training_final/S11_...	Final training samples	24
5	/content/drive/MyDrive/CAW_training_final/S3_V...	Final training samples	21
6	/content/drive/MyDrive/CAW_training_final/S7_m...	Final training samples	35
7	/content/drive/MyDrive/CAW_training_final/S1_U...	Final training samples	15
8	/content/drive/MyDrive/CAW_training_final/S1_V...	Final training samples	21
9	/content/drive/MyDrive/CAW_training_final/S7_V...	Final training samples	24
10	/content/drive/MyDrive/CAW_training_final/S7_U...	Final training samples	35
11	/content/drive/MyDrive/CAW_training_final/S5_V...	Final training samples	21
12	/content/drive/MyDrive/CAW_training_final/S5_U...	Final training samples	20

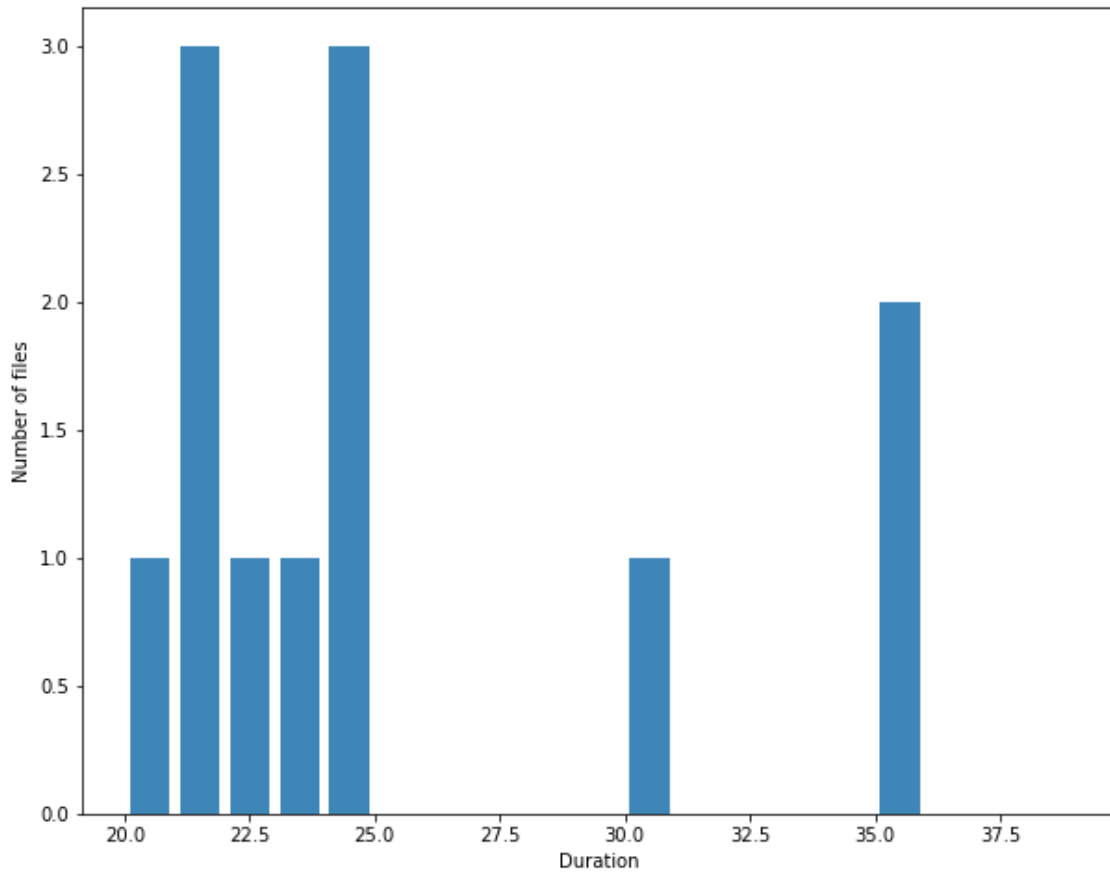


Figure 33 CAW Dataset

```
df["duration"].value_counts()
```

24	3
21	3
35	2
23	1
30	1
22	1
15	1
20	1

(Source of the code: authors' notebooks)

## 3.6 Data Manipulation

One of the technical advantages of using CAW as part of our solution is the fact that it uses raw audio as input. This only requires one extra manipulation to ensure that the data is ready for training: all files need to be mono. Our GAN dataset was comprised of stereo files at 48KHz and 16 bits. This parsing process is quite straightforwardly done by using Logic X and was the last step in the iterative process of creating the CAW dataset.

In order to produce some of the Mel spectrograms to help with visual feedback during the human evaluation process, some of the files will also undergo some extra manipulation. For this process we will re-use some of the audio pipelines created for the DCGAN model, please see below (please note that spectrograms can be produced instead at a later stage):

```
class AudioUtil():
    # -----
    # Load an audio file. Return the signal as a tensor and the sample rate
    # -----
    @staticmethod
    def open(audio_file):
        sig, sr = librosa.load(audio_file)
        sig = np.expand_dims(sig,0)
        return (sig, sr)

    @staticmethod
    def pad_trunc(aud, max_ms):

        sig, sr = aud
        num_rows = sig.shape[0]
        sig_len = sig.shape[-1]

        max_len = sr//1000 * max_ms

        if (sig_len > max_len):
            # Truncate the signal to the given length
            sig = sig[:, :max_len]

        elif (sig_len < max_len):
            # Length of padding to add at the beginning and end of the signal
            pad_begin_len = random.randint(0, max_len - sig_len)
            pad_end_len = max_len - sig_len - pad_begin_len

            # Pad with 0s
            pad_begin = np.zeros((num_rows, pad_begin_len))
            pad_end = np.zeros((num_rows, pad_end_len))

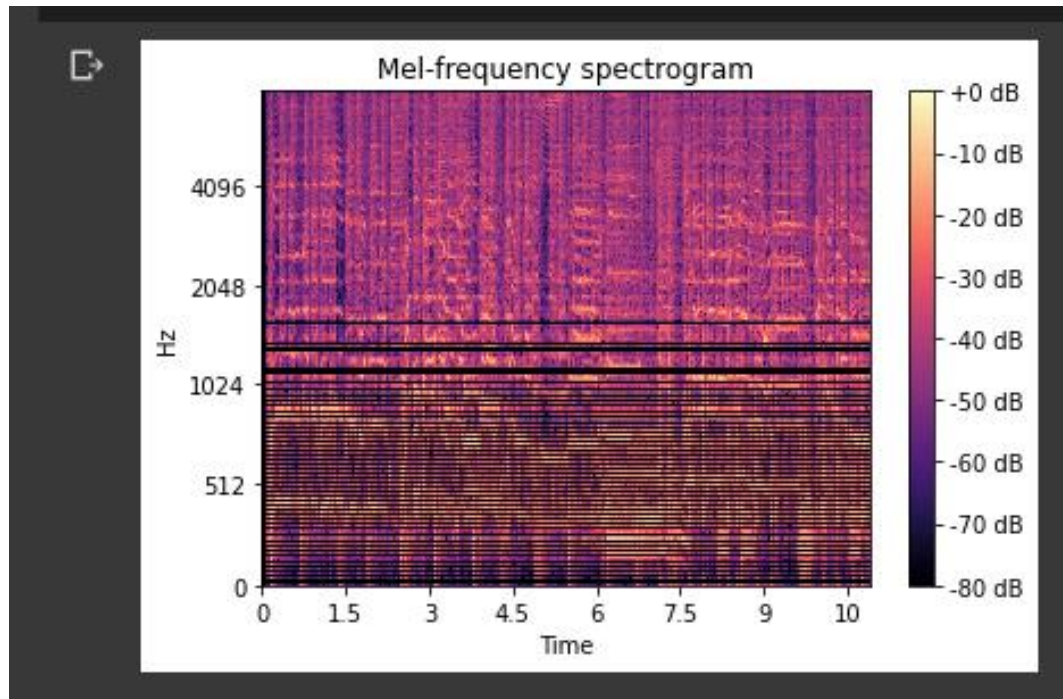
            sig = np.concatenate((pad_begin, sig, pad_end), 1)

        return (sig, sr)

    @staticmethod
    def spectro_gram(aud, n_mels = 10, sr = 48000):
        y = np.squeeze(aud[0])
        spec = np.expand_dims(librosa.feature.melspectrogram(y, sr = sr, n_mels=n_mels, fmax=9000),0)
        #spec = torch.from_numpy(spec)
        return (spec)
```

(Source of the code: authors' notebooks)

Once instantiated this class will be able to produce Mel spectrograms like the one below:



*Figure 34 Mel Spectrogram GAN dataset*

(source: [author](#))

In order to ensure that all the Mel spectrograms are standardized and therefore suitable for comparison, the pipeline will ensure that all files have the same length.

## 4 CAW for Audio Generation

This chapter will provide insights into the process of producing the newly generated samples and the decision-making behind them. We will also be describing our working environment and what tools are we using for the whole process.

### 4.1 Working Environment/ tools

As stated in figure 2.18, we will be using some of the code provided on CAW's GitHub to build our models to address issues 1 and 2. This uses Python 3, and it is officially implemented on Pytorch. All the necessary modifications/ additions to the code will consequently be done in python 3.

We will be translating all the code from .py files to Colab notebooks as our working environment will be Google Colab for training and G drive for storing all the project's datasets, alongside the newly generated material. We have chosen Colab for its access to GPUs, more precisely, the paid option Google Colab Pro +. Thanks to this fact we are able to use faster GPUs, utilize more memory when training and enjoy longer runtimes with almost no idle time. The idea is to be able to train our model for at least 10 hours; that is the minimum time required if we want to train our model at 3000 epochs.

All our data is stored in G drive, Colab uses G drive to save all the notebooks there, so it only makes sense to store our data in the same location to have everything centralized. Also, G drive is optimized to work with Colab and offers an easy interface to facilitate the workflow. It is also possible to use Bash commands to move files around, so overall it is a practical workspace for our project. It is also possible to open the notebooks when at different locations so I should be able to do the training from both home and work at the same time. That is, be able to open from home a notebook whose training started at work and vice versa. This is quite handy as we are going to train for at least two hundred hours, also allowing that flexibility that is key to managing the whole process.

### 4.2 Implementation

We have created two Colab notebooks as the starting point for our training process. The first one is called Unconditional Music Generation and features all the code necessary to address issue 1. The workflow is as follows:

- Import and install all necessary dependencies.
- Access G drive and copy over a file on Colab's workspace for training purposes.
- Perform the training process.
- Generate new samples, and save them on G drive.

The second one is called Variations and features all the necessary code to address issue 2. The workflow is as follows:

- Import and install all necessary dependencies.
- Access G drive and copy over a file on Colab's workspace for training purposes.
- Perform the training process.
- Generate new samples, and save them on G drive.

After working on both the code and structure of the Colabs, both have been tested, the code works 100% and both are able to produce the required outcome as well as handle the file workflow as per above.

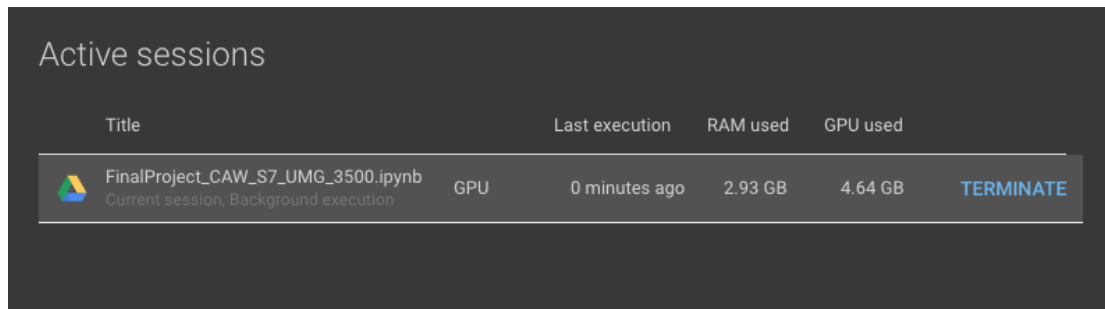
## 4.3 Training

Once both Colabs were fully operational the training stage started, and it has been a process of more than 200 hours with a total of 13 samples as input. Please see the notebooks provided on GitHub for further info. Although the paper suggests training at 3000 epochs, we have tried different values to effectively establish that the suggested number of epochs was a valid reference. We have trained samples at 100, 1500, 2000, 3000 and 3500 and then analysed the resulting material based on quality and time that takes training. 100 epochs are obviously something not to consider as it produces mere noise. Samples produced at 1500 and 2000 could potentially be a solution as the time to train is significantly lower than 3000 epochs; finally, we decided not to pursue this option. The difference with 3500 is not that significant in terms of quality but it is relevant in terms of training time. Therefore, after some try-outs, it was decided that the training would be at 3000 epochs. Other parameters used in the training are as follows:


```
] #parameters
inpainting_indices= [0, 1]
is_cuda = torch.cuda.is_available()
gpu_num = 0
manual_random_seed = -1
input_file = 'S4_U.wav'
segments_to_train = []
start_time = 0
init_sample_rate = 16000
fs_list = [320, 400, 500, 640, 800, 1000, 1280, 1600, 2000, 2500, 4000, 8000, 10000, 12000, 14400,
max_length = 25
run_mode = 'normal' #['normal', 'inpainting', 'denoising']
num_epochs = 3000
learning_rate = 0.0015
scheduler_lr_decay = 0.1
betal = 0.5
speech = False
num_layers = 8
output_folder = 'outputs'
filter_size = 9
set_first_scale_by_energy = True
min_energy_th = 0.0025
hidden_channels_init = 16
growing_hidden_channels_factor = 6
plot_losses = False
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
initial_noise_amp = 1
noise_amp_factor = 0.01
```

Figure 35 Training main parameters (source: author's notebooks)

Note that the main model for our training is #normal as we are using the unconditional music generation and the variations features of CAW. The length of the files is limited to 25 seconds for computational reasons, but it could go up to 100 seconds or even more.



The screenshot shows the 'Active sessions' panel in JupyterLab. It contains a table with the following data:

Title		Last execution	RAM used	GPU used	
 FinalProject_CAW_S7_UMG_3500.ipynb Current session, Background execution	GPU	0 minutes ago	2.93 GB	4.64 GB	<a href="#">TERMINATE</a>

Fs\_list features the values of the scales for the different up-sampling phases. Finally, the learning rate is 0.0015; the rest of the parameters are mostly related to the helper functions of the model. We intended to experiment with the learning rate as well but given the time constraints, we have prioritized the production of musical samples for evaluation. Finally, there was a clear conclusion after the first try-outs: samples with vocals are to be left out of the training as the initial results are very unrealistic. It might be an option only to train just vocals so to get some inspiration for potential new melodies, but full instrumental band music has been prioritized. The main reason for this is the way that we work as a band. The last element is the vocals, once the music has been composed. This fact and the lack of realism in the generated samples with vocals prompted us to follow the training with instrumental samples only.

## 4.4 Results

We have been successful at generating a range of audio samples to address both issue 1 and 2. Please see below.

### 4.4.1 Newly Generated Audio Files

We have a new dataset with a total of 77 full newly generated samples for the following songs: S1, S3, S4, S5, S7, S9 and S11. Some samples are the product of doing some testing during the training whereas other samples are the result of a conscious training process. This is a sufficient amount to iterate through them and curate a range to be submitted for human evaluation. Please see below some information about the dataset and its samples:



```

path = "/content/drive/MyDrive/CAW_new_generated_music"

paths = []
size = 0
for root, dirs, files in os.walk(path):
    for file in files:
        if (file.endswith(".wav") and (not (file.starts
            paths.append(os.path.join(root, file))
            size += os.path.getsize(os.path.join(root,

print(f'We have {len(paths)} .Wav Files with {size/1024*

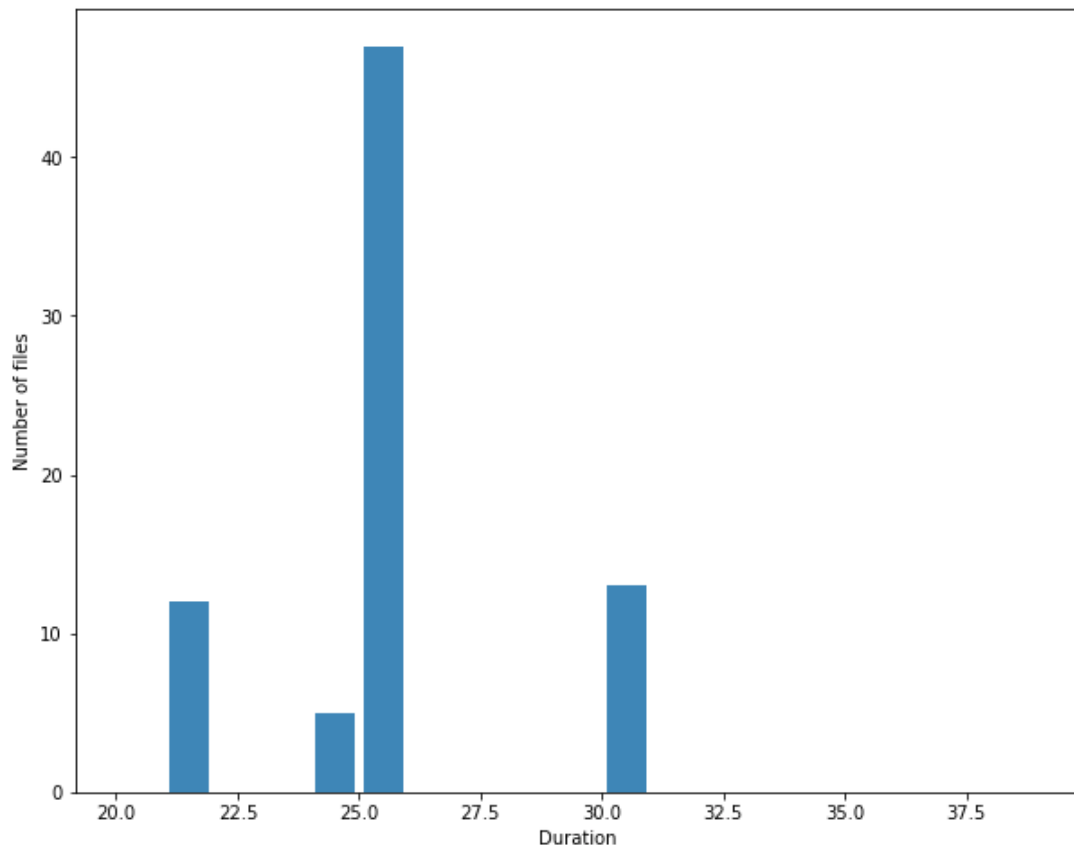
We have 77 .Wav Files with 59.21 Mb in size

```

1 to 25 of 77 entries

index	paths	annotation	duration
0	/content/drive/MyDrive/CAW_new_generated_music/Generated/S7_U_100_epochs/S7_100_U.wav	generated	25
1	/content/drive/MyDrive/CAW_new_generated_music/Generated/S11_U_2000_epochs/S11_U_2000.wav	generated	25
2	/content/drive/MyDrive/CAW_new_generated_music/Generated/S9_U_2000_epochs/S9_2000_U.wav	generated	25
3	/content/drive/MyDrive/CAW_new_generated_music/Generated/S1_V_3000_epochs/S1_V_0.wav	generated	21
4	/content/drive/MyDrive/CAW_new_generated_music/Generated/S1_V_3000_epochs/S1_V_1.wav	generated	21
5	/content/drive/MyDrive/CAW_new_generated_music/Generated/S1_V_3000_epochs/S1_V_2.wav	generated	21
6	/content/drive/MyDrive/CAW_new_generated_music/Generated/S1_V_3000_epochs/S1_V_3.wav	generated	21
7	/content/drive/MyDrive/CAW_new_generated_music/Generated/S3_V_2000_epochs/S3_2000_U.wav	generated	21
8	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_11.wav	generated	25
9	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_1.wav	generated	25
10	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_2.wav	generated	25
11	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_3.wav	generated	25
12	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_4.wav	generated	25
13	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_5.wav	generated	25
14	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_6.wav	generated	25
15	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_7.wav	generated	25
16	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_8.wav	generated	25
17	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_10.wav	generated	25
18	/content/drive/MyDrive/CAW_new_generated_music/Generated/S5_U_3000_epochs/S5_U_0.wav	generated	25
19	/content/drive/MyDrive/CAW_new_generated_music/Generated/S7_U_3500_epochs/S7_U_0.wav	generated	25
20	/content/drive/MyDrive/CAW_new_generated_music/Generated/S7_U_3500_epochs/S7_U_1.wav	generated	25
21	/content/drive/MyDrive/CAW_new_generated_music/Generated/S7_U_3500_epochs/S7_U_2.wav	generated	25
22	/content/drive/MyDrive/CAW_new_generated_music/Generated/S7_U_3500_epochs/S7_U_3.wav	generated	25
23	/content/drive/MyDrive/CAW_new_generated_music/Generated/S7_U_3500_epochs/S7_U_4.wav	generated	25
24	/content/drive/MyDrive/CAW_new_generated_music/Generated/S7_U_3500_epochs/S7_U_6.wav	generated	25

Show 25 per page 1 2 3 4



*Figure 36 New samples*

```
df["duration"].value_counts()
25    47
30    13
21    12
24     5
```

(Source of the code: authors' notebooks)

#### 4.4.2 Metrics

As discussed, and given the nature of the project, we are going to use mostly human evaluation as it fits our objective better. Inception score, similarity matrix or any other metric were therefore deemed not relevant when comparing them against human evaluation. The latter will be done at two levels: first, it will be the author as also a member of the band who will provide an overall view of the auditory quality of the samples and will conclude if they are fit for purpose. The criteria for that will be mostly musical, where we will explore whether sufficient translation has been achieved and what new structures have been created. For that, I will iterate through the dataset, listen to every sample and then curate a range of samples for each case (issue 1, issue 2) to analyse them in more depth. Some visuals will also be

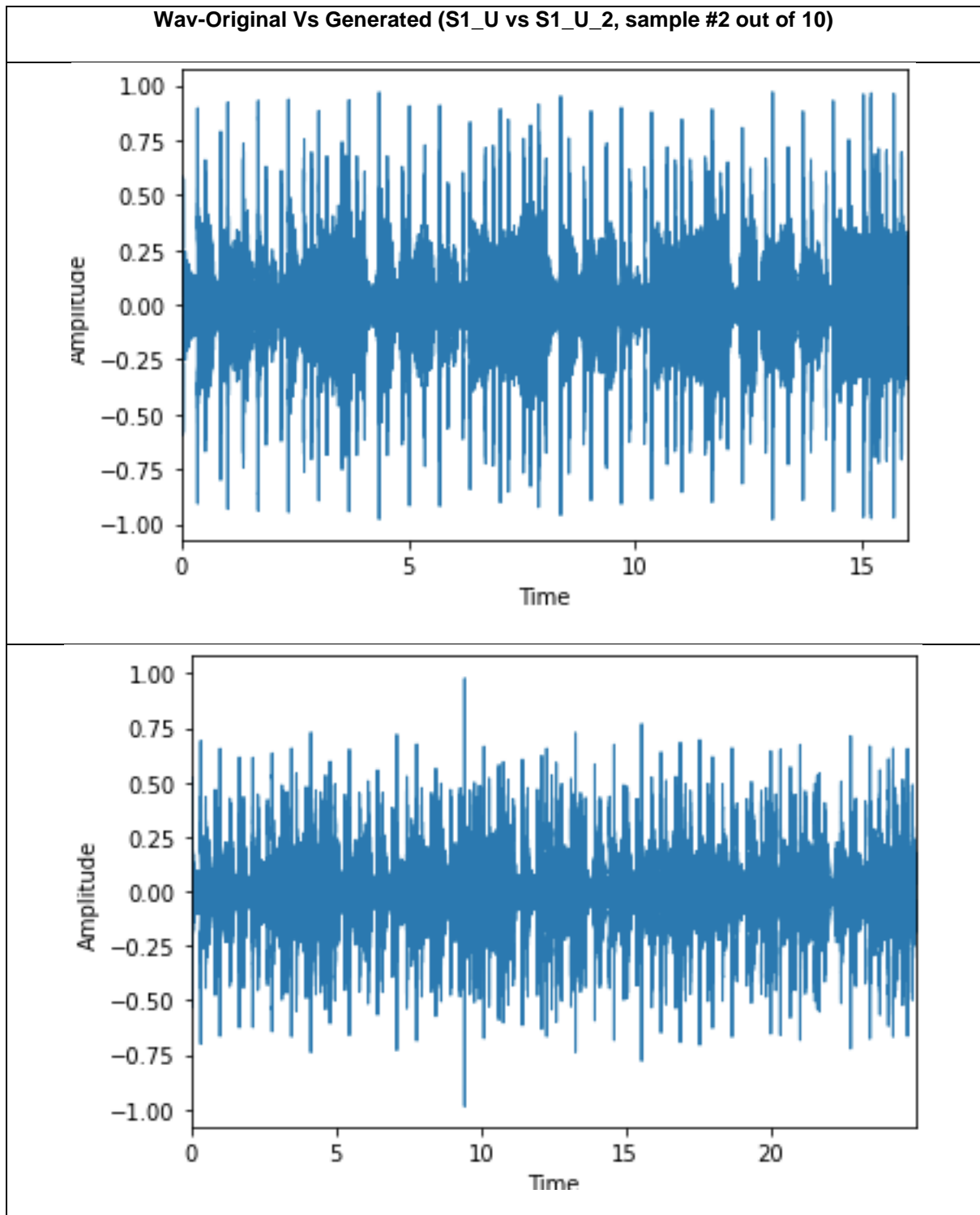
provided with concrete examples. The second phase will entail another process of curation of all the newly generated samples, to create two mini-datasets of 10 samples each. Those will be handed to the main songwriter of our band alongside a questionnaire. This process will be addressed in the next chapter.

After the perusal of the whole dataset of newly generated samples and the analysis of a range of curated samples (the best and most coherent), we can establish the following conclusions: there is good translation overall, and semantic coherence is achieved. The model successfully translates both rhythm, pitch and timbre. The latter is perhaps the most important feature, and we consider this a great achievement, as this contributes to maintaining the style of the initial recording. It is clearly music from L'anima as the new samples feature its guitar tone, a key feature for bands with electric guitars. This is something that cannot be achieved with the use of MIDI or symbolic representation. Good timbre translation is also noticeable on drums, whereas the bass is a bit unmasked.

When listening to several samples from the same original it is possible to appreciate that variations have been created. It is clear that the model is able to create 'improvisations' of the same initial music motive. Long range dependencies are picked up for the whole range of frequencies. However, those variations are naïve. This means that when the model attempts to create alternatives, it lacks the 'musical' knowledge required to perform the high level 'operations' that are required for this type of composition work, and artefacts and weird sounding passages do appear as a result of that. It is like someone who has been playing piano for a few months trying to improvise Jazz: it sounds very much like the original and the 'improvisations' are forced and flimsy. One can notice that there is not enough training material to allow enough 'learning' to happen and propel the creation of more coherent musical structures. Having said that, the generated material has some good qualities to it, including the fact of containing some random elements, that although in my opinion it detracts from the final result, it can definitively be an inspiration for our songwriter. We know him and he tends to find inspiration in weird sounds and artifacts as he believes that from ugliness, beauty can be distilled. Obviously, this is a completely subjective opinion based on the songwriter's experience and approach to composition. Therefore, it is impossible to measure it or predict it, if you will. We do hope though, that some of these samples will contribute towards the solution of our problem, as I believe that these samples contain enough elements to potentially trigger a Eureka moment. Especially if we take into account the fact that most samples feature no vocals in them. As previously discussed, vocals have been discarded from training purposes as they do not translate well in our case. Sometimes having vocals (as in weird and ethereal sounding) on the new samples adds an interesting effect but overall, it detracts from the original intent. The model seems to work well by recreating vocals only, but in our experiments, we have found that they get in the way of the music. Could this be the GAN failing to train some vocal qualities or a specific set of frequencies? Are there too many elements to learn? Do vocals get somehow thrown out of the final mix? It is not the place to answer those questions, but we do think it was a good decision to train with instrumental samples only.

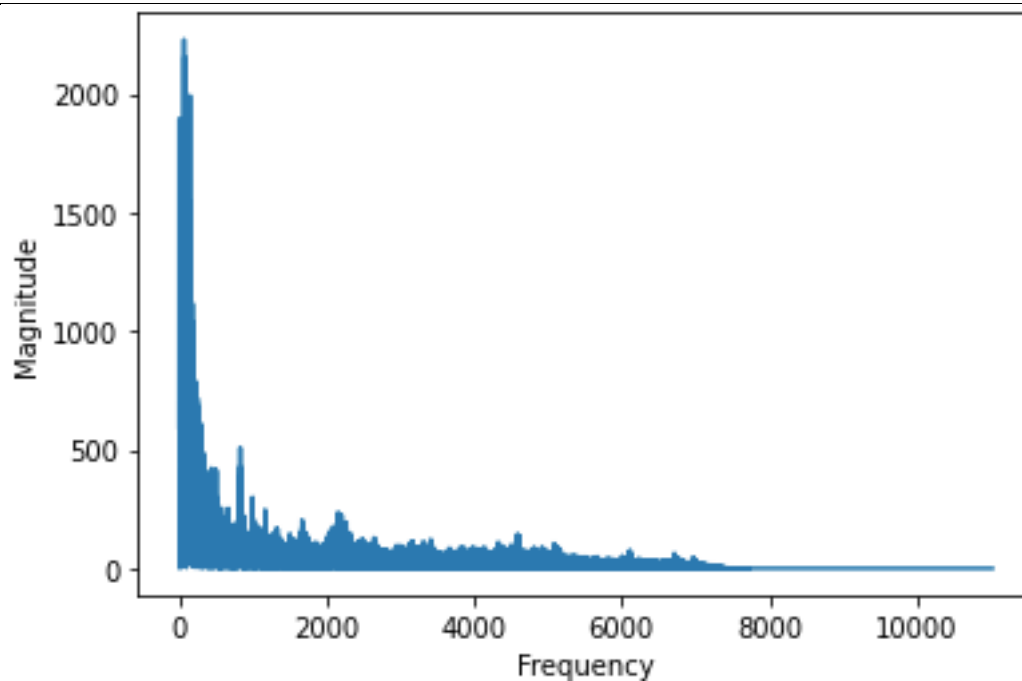
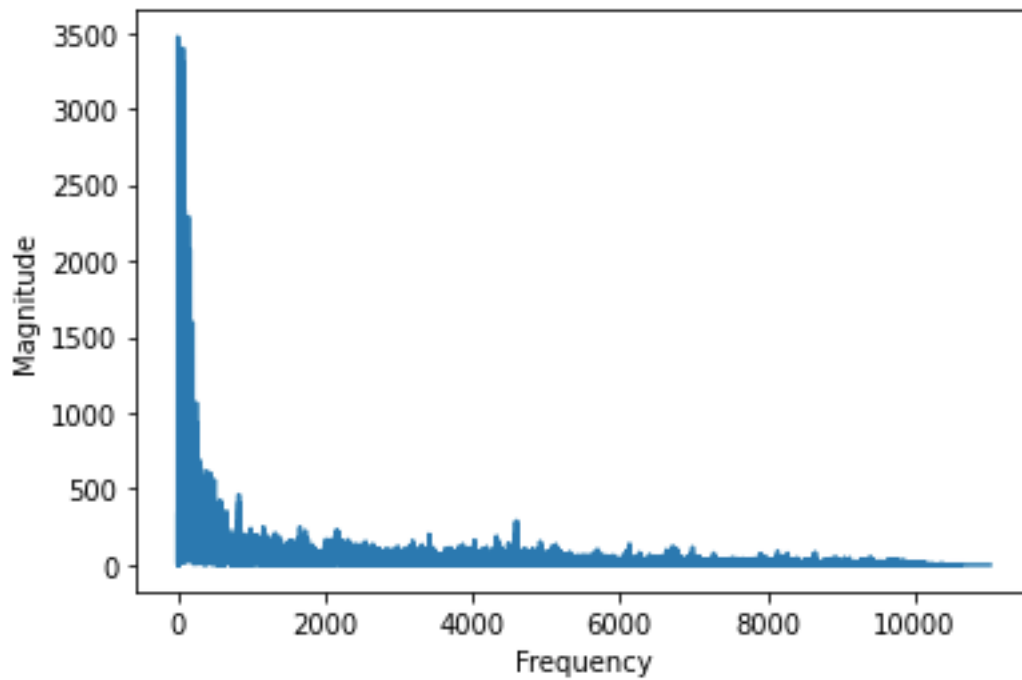
Finally, and to sum it up, it is quite clear that although capable of achieving good auditory results, training on a single sample clearly limits the ability of the model to generate new structures. Those new structures are somewhat naïve and restricted to what the model does know; that is, the information provided for only one sample.

The training results for S1 are the perfect example of everything that we have discussed above. For S1 I have chosen two passages that include a strong guitar riff. This is part of a bridge that needs rework as the different ideas do not flow together. When listening to the generated samples It is quite clear that it is a L'Anima sample and it is indeed quite similar sonically. However, the structure is quite similar to the original and its variations naïve; it seems as if the model is trying to cut and paste or superimpose certain chunks after others. This results in weird sounding sections that might work but this would be more of a random effect. Interesting results might be achieved if you generate a big number of new samples as it would increase the probability for the model to create a sample with more 'musical' coherence. But again, it would be more of a random approach than anything. In our case, generating 10 samples is not going to produce that many variations, given the model's limited prior knowledge. This is perfectly seen in S1 generated samples, providing a clear example of the overall trend. Below we can see some visuals of S1:



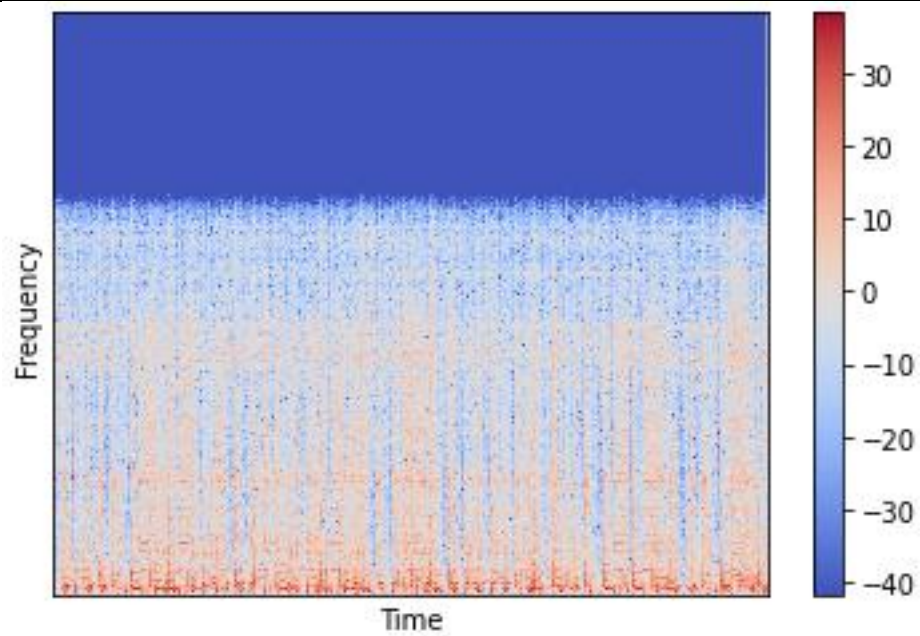
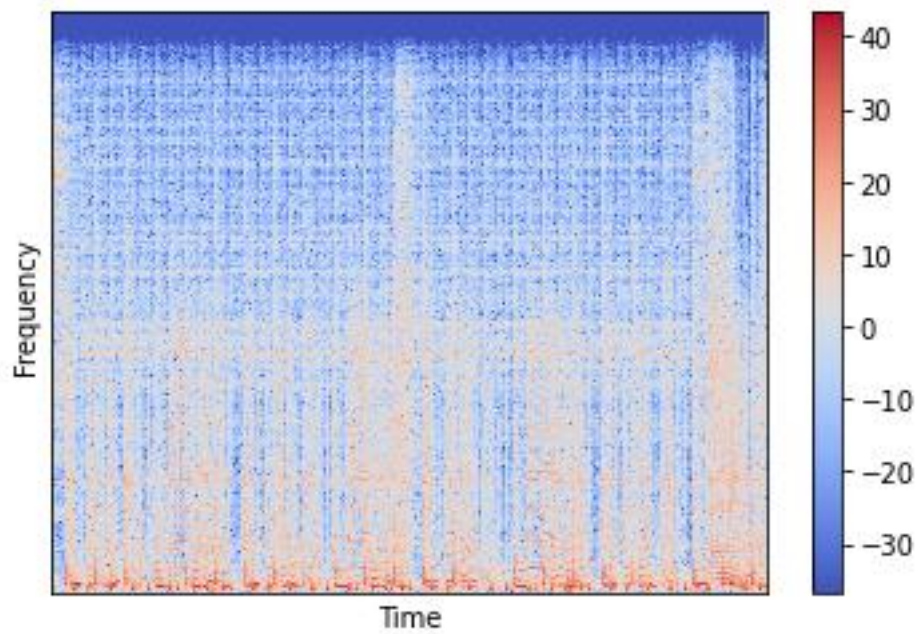
As you can see the newly generated signal is similar to the original but features variations. We can also see that the S1\_U\_2 has lost a bit of dynamic range, but this is not that important as for human evaluation a downsampled version of S1 will be provided.

Frequency Domain-Original Vs Generated (S1\_U vs S1\_U\_2, sample #2 out of 10)



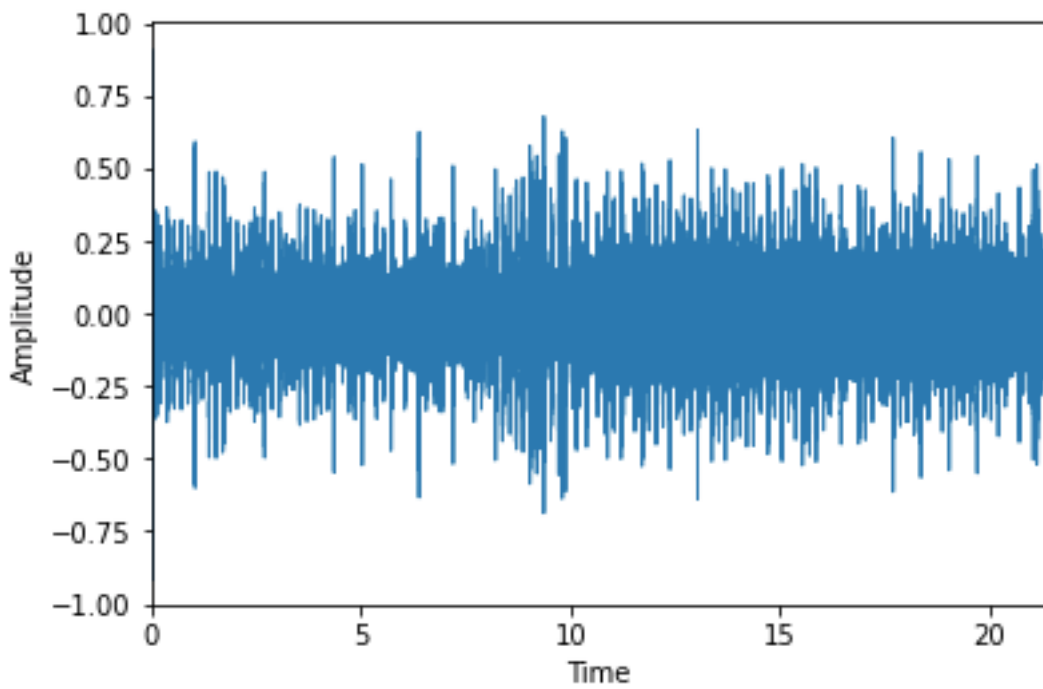
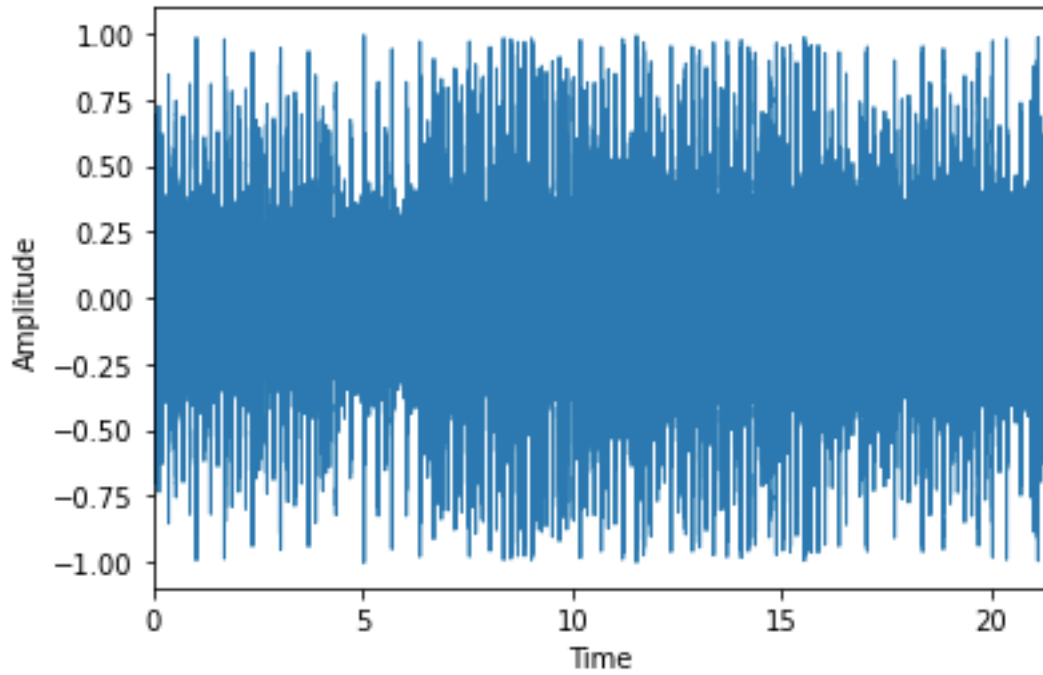
The figure shows that sonic translation is well achieved, it can also be seen that the model generates output at 16Khz (So at about 8k range) but again this will not be an issue as we have already factored in and the real S1 sample for human evaluation will be downsampled.

**Mel-Original Vs Generated (S1\_U vs S1\_U\_2, sample #2 out of 10)**



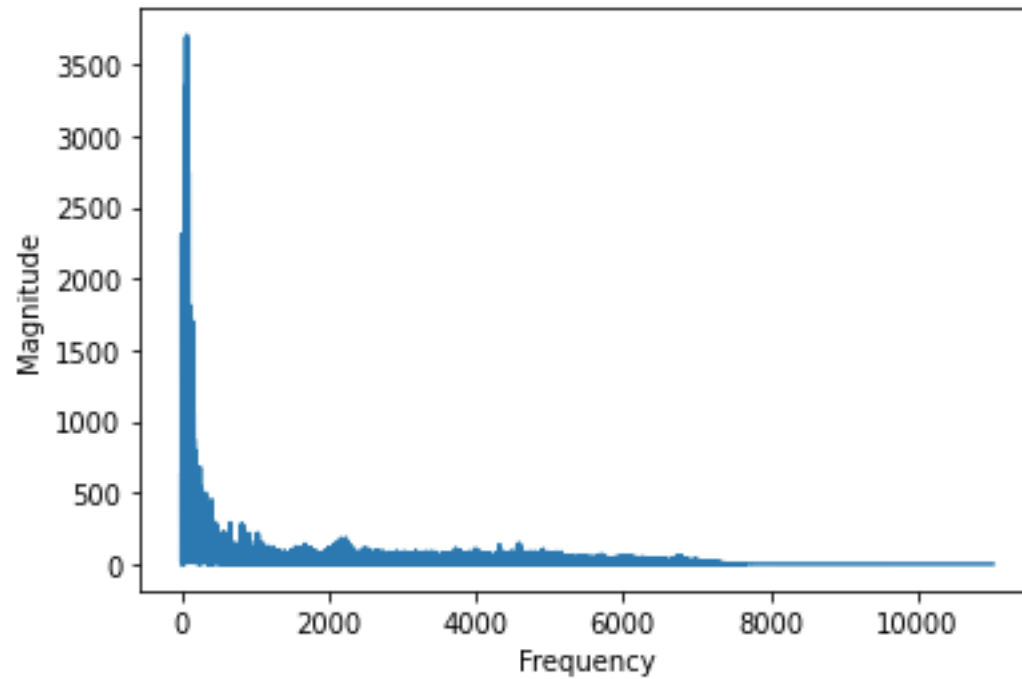
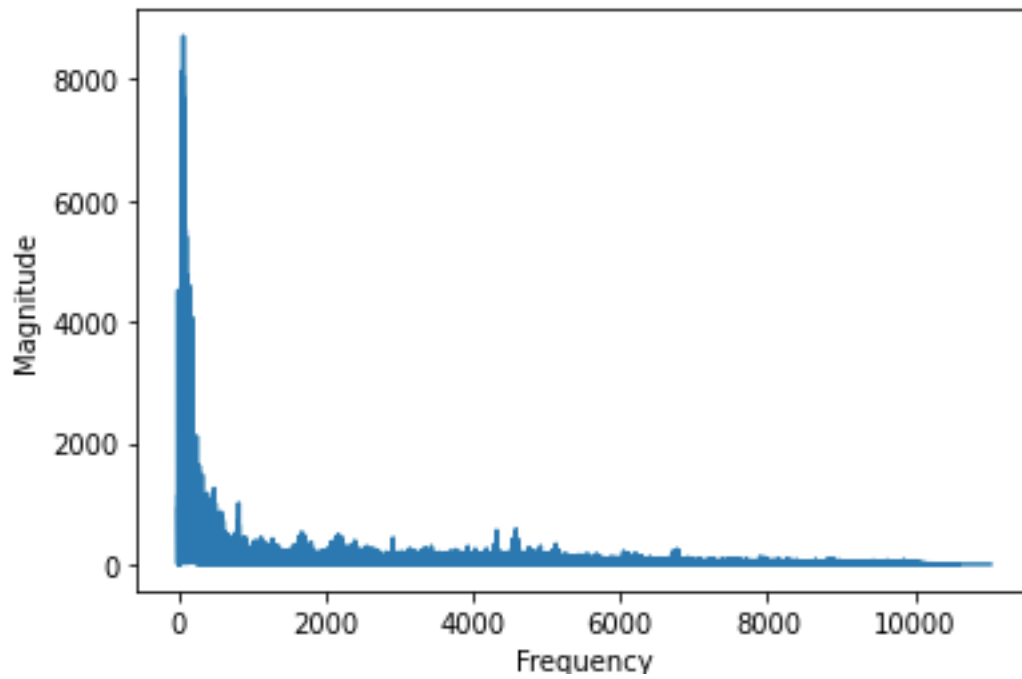
**Mel Spectrograms**

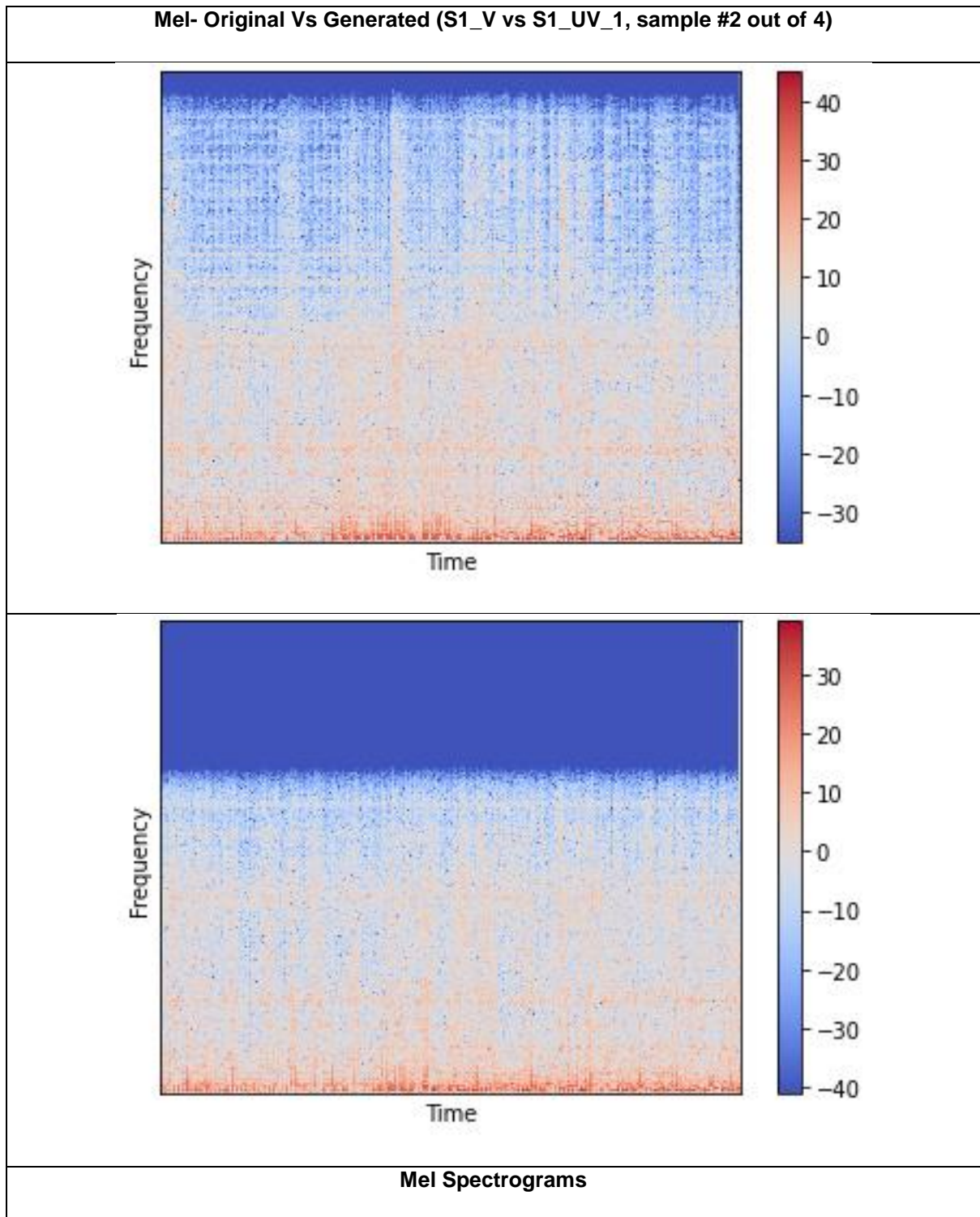
Wav-Original Vs Generated (S1\_V vs S1\_UV\_1, sample #2 out of 4)





Frequency Domain -Original Vs Generated (S1\_V vs S1\_UV\_1, sample #2 out of 4)





The same analysis can be applied to the variations sample.

## 4.5 Other Models (Jukebox, SampleRNN)

As stated at the beginning, autoregressive models are also successful at generating new music from raw audio. We have tried to generate new music with the following models: Jukebox<sup>38</sup> and PRISM SampleRNN<sup>39</sup>. For Jukebox, we have been able to generate some music using some of our own samples but for PRISM we found too many problems and we decided to scrape the training process as a whole. Although a fairly new model it has not been maintained and it has not aged well. The number of problems we encountered made the training inviable, which is a shame as we had spent some time going through the model and preparing a dataset<sup>40</sup> to feed it.

## 4.6 Comparison

Jukebox and CAW are quite similar in their approach as they both can be trained with a single sample. PRISM needs a dataset to produce some results, the more data the better. It is only limited to 10-second mono samples. Jukebox is a little bit faster in terms of training and produces music at three levels of quality after 6 hours of training. We tried CAW at 100, 1500, 2000 and 3000 epochs and it requires some 10+ hours of training to produce a sample, at 3000 epochs. This can be considered the threshold for good auditory quality. Jukebox samples maintain a certain coherence with the original sample as they add to it after a number of seconds; but after that amount, it's chaotic and very cacophonous. The vocals are unintelligible, and it is a very disturbing sound. Some vocal structures are created but have very little resemblance with L'anima; after a few seconds of listening to those vocals, one feels the urge to press the stop button to end that cacophonous chaos. In that aspect CAW's samples, are more coherent and cohesive. They are definitively more fit to present to our songwriter as they serve better for our purpose. Therefore, we can conclude that choosing CAW has been a good option against other models. Below you can see some spectrograms for both generated signals for the same input sample S4.

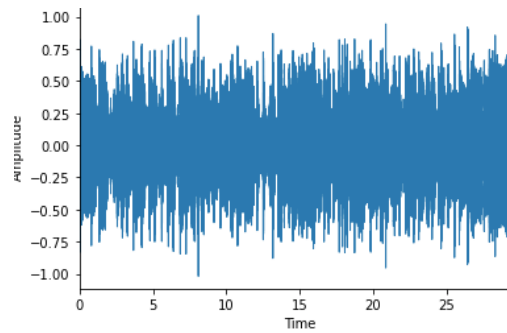
---

38 <https://openai.com/blog/jukebox/>

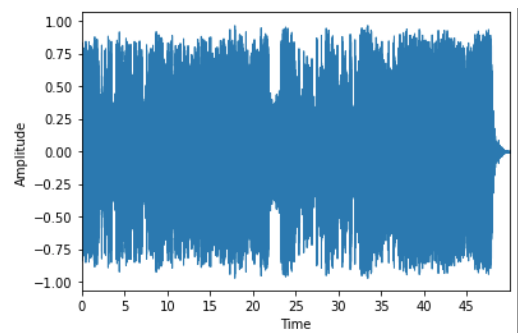
39 <https://github.com/mcm-prism/prism-samplernn>

40 Please see attached link:

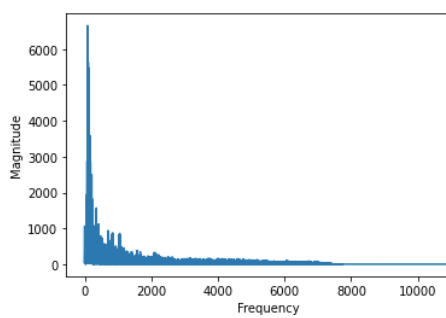
CAW



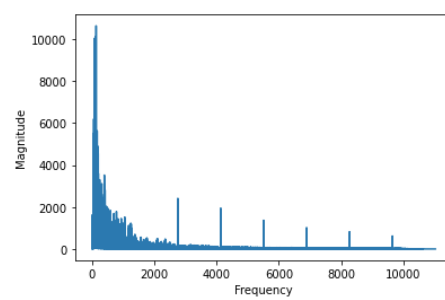
JUKEBOX



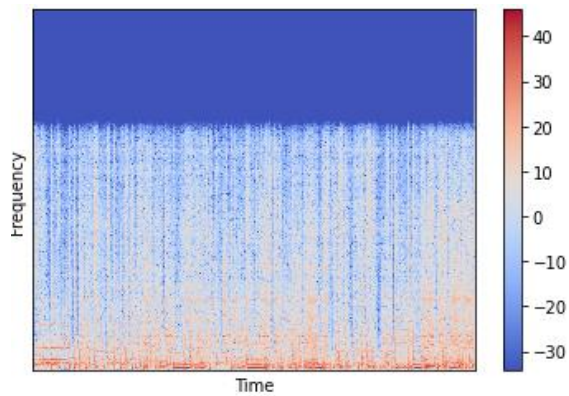
WAV



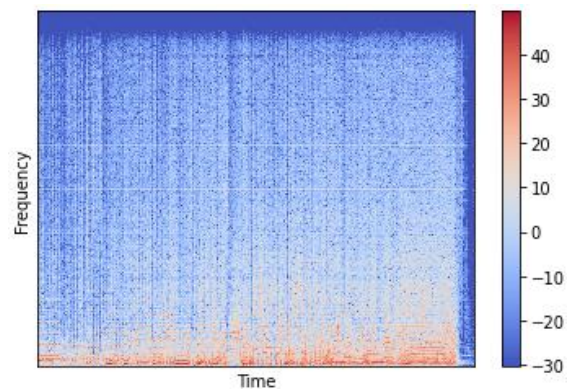
WAV



Frequency Domain



Frequency Domain



MEL

MEL

Jukebox samples cover more range of Frequencies and are more powerful sonically but as seen on the Mel they have a poorer translation of frequencies and therefore poor timbre. Jukebox also feature peaks every 2KHz that look like overtones. That will cause a lot of auditory disruption to the listener. CAW signal is much more rounded overall.

## 5 User Study

This is the second phase of analysis of the newly generated samples and the most important as we will test our hypothesis with the main songwriter of our band.

### 5.1 Methodology

For this part, we have curated two datasets, one with samples that address issue 1 and the second addressing issue 2. The audios have been carefully selected and do contain original material. The UMG dataset is formed by ten samples from songs S1, S3, S4, S5, S7, S9 and S11 where 8 are newly generated and 2 originals. The original samples have been downsampled to 12Khz to match the output quality and provide a similar auditory experience. The VAR dataset is formed by ten samples from songs S1, S3, S4, S5 and S7. It also features two variations generated with Jukebox: these variations were generated using S4 samples as the original input. The composition is 6 CAW samples, 2 Jukebox and 2 originals. Again, the originals have been downsampled. Jukebox output's quality is 16Khz.



Two questionnaires have been created in order to conduct a survey on what the views of the main songwriter of our band will be with regard to the new samples. Then we will analyse his responses to ascertain whether our hypothesis is valid as a solution for our problem or not.

The protocol is as follows:

- Send both datasets and questionnaire with instructions
- Iterative process: for  $i$  in range (20):
  - Listen to one sample
  - Answer a range of questions regarding the sample. The questions are varied in length and type in order to capture the maximum information.
- Analysis of the data and comparison with my conclusions.
- Hypothesis validation

Find below a sample questionnaire for one single sample:

# Lanima Questionnaire


ibantxodrums@gmail.com (not shared)
[Switch account](#)


## Sample 1

What's the name of the sample you have just listened? (please write exact title)

Your answer

Do you think the sample is real or a fake?

☐ Real
 ☐ Fake

If you answered fake, was it easy to spot is was fake?

☐ Easy
 ☐ Don't know
 ☐ Difficult
 ☐ Other:

Can you rate the quality of the sample? Is is musical at all?

0 1 2 3 4 5 6 7 8 9 10
   
 Terrible ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ Quite good

Do you find it inspiring? does it give you ideas to generate new music?

☐ Yes
 ☐ No
 ☐ Maybe
 ☐ Other:

Overall feedback? Short answer please

Your answer

[Back](#)
[Next](#)

Clear form

Figure 37 Example questionnaire

(source: author)

## 5.2 Results

### 5.2.1 Unconditional Music Generation (addressing issue 1)

Below we can see a transcript of the results of the human evaluation performed by the main songwriter of our band (MS). The first two bullet points for each section feature everything he says on the questionnaire, after that it is me doing some remarks providing extra insight.

SAMPLE	COMMENTS (Main Songwriter + author adding remarks if needed)
S1_U_3	Easy to spot it was fake, 4/10 musicality. Not inspiring (too mechanical sounding)
S1_Ud	MSS thought it was fake, it is a real sample. 8/10 musicality. Maybe inspiring (good sense of groove, weird sense of harmony) Interesting comments indicate how subjective composing is.
S3_2000_U	Easy to spot it was fake, 2/10 musicality It could be inspiring (chaos) It is quite clear that less than 3000 epochs make the recordings unmusical
S4_1_U	Easy to spot it was fake, 1/10 musicality Uninspiring and no comment This is one of the first samples we got from the training, it is indeed terrible, just put it in there as check. It is quite clear the MS is paying attention to the recording

S4_U_2	<p>Easy to spot it was fake, 1/10 musicality.</p> <p>Uninspiring (no sense of music)</p> <p>Again, sample taken from the first batch, it is indeed very noisy and chaotic, little epochs. The model needs 3000 + epoch to produce something meaningful</p>
S5_U_4	<p>Easy to spot it was fake 5/10 musicality</p> <p>It could be inspiring (weird sense of groove)</p> <p>The model is trying to do naïve improvisations and generates new rhythmical patterns but no sense to time signature there (that's something that should be learned). Still might be inspiration</p>
S5_Ud	<p>Easy to spot it was really sample</p>
S7_U_8	<p>Easy to spot it was fake, 7/10 musicality</p> <p>Could be inspiring (I like the weirdness here, especially the sound of the sections with no drums)</p> <p>I can understand why MS finds it inspiring, there a 'progressive' feel to this sample, I though the same when I heard it. Again, this is more to the model 'random' approach that proper 'learning' which might work in certain situations</p>
S9_2000_U	<p>Easy to spot it was fake, 4/10 musicality</p> <p>Could be inspiring (within the nonsense I could hear some possible melody)</p> <p>Again, this is quite similar to the previous sample, the 'random' approach of the model could be inspiring because it provides options that the 'trained' brain of MS rarely could create by himself. Therefore, it can tap into unexplored territory</p>
S11_U_3500_1	<p>Easy to spot it was fake, 1/10 musicality</p>



	<p>uninspiring (nothing special here)</p> <p>Poor quality sample where do vocals really mess the whole thing. It is quite clear why MS is put off by it.</p>
--	--

This questionnaire reveals that my hypothesis of the 25 seconds needs to be revised. MS finds that needs a little longer to decide if the sample is going to be inspiring. There is room, computationally speaking to create 60-second samples. That could a developmental point for the future.

As I mentioned in my analysis, the model lacks information to 'learn' long dependencies in a more 'musical' way, but that 'randomness' can be inspiring, as the results show. The model does not understand the concept of time signature or melody, but it has the capability of generating material that a 'trained' musician would not be able to generate, as that training is a limiting factor in itself (hence the constant search for inspiration) and shapes the way you would try to explore those 'random' and 'chaotic' avenues. This is quite a positive outcome in this case, as this favours the type of creative mind like MS as it contributes to expanding his creative vision past the boundaries of his own training and experience.

We made a good decision removing the vocals and it puts MS off. Overall, the samples do produce a reaction in the listener and could trigger responses close to a Eureka effect but mostly due to its 'random' approach and naïve way of reconstructing musical structure rather than a process of learning what are those 'L'Aanima' patterns that the model ideally should pick up.

### 5.2.2 Variation (addressing issue 2)

Below we can see a transcript of the results of the human evaluation performed by the main songwriter of our band (MS). The first two bullet points for each section feature everything he says on the questionnaire, after that it is me doing some remarks providing extra insight.

SAMPLE	COMMENTS (Main Songwriter + author adding remarks if needed)
J1	<p>Easy to spot it was fake, 8/10 musicality.</p> <p>Maybe (I like what I hear)</p>

	<p>It is quite obvious why likes it as it is the original sample with random stuff added. That's how Jukebox works, again this is tapping on the random approach that a trained musician finds difficult to compose/ recreate</p>
J2	<p>Easy to spot it was fake. No musical rating.</p> <p>Maybe inspiring (Not coherent but interesting)</p> <p>Similar to J1, MS attracted by the weirdness that he cannot purposely create</p>
S3_2000_V	<p>Easy to spot it was fake, 1/10 musicality</p> <p>uninspiring (I think I have already played this one, not musical intention)</p> <p>It is quite clear that less than 3000 epochs make the recordings unmusical for the listener</p> <p>MS not picking the extra section in added by the model</p>
S1_V_0	<p>Easy to spot it was fake, 7/10 musicality</p> <p>Could be inspiring (not defined groove but nice atmosphere)</p> <p>MS has made a mistake labelling the sample. Again, the model not generating naïve time signature that are appealing to MS due to their weirdness</p>
S4_3_V	<p>Easy to spot it was fake, No rate.</p> <p>Uninspiring (nothing special here)</p> <p>Again, sample taken from the first batch, it is indeed very noisy and chaotic, little epochs. The model needs 3000 + epoch to produce something meaningful</p>
S5_V_6	<p>Easy to spot it was fake 4/10 musicality</p> <p>It could be inspiring (does not communicate much)</p> <p>This is great because I put this on purpose to check on the timbre. It is quite clear that the guitar tone is translated, and MS has made no remarks at all.</p>

S5_Vd	<p>Easy to spot it was a real sample (just to short to inspire but I like the suspense).</p> <p>Again, interesting that he does not pick up on the tone of the guitar, He only focuses on the composition</p>
S7_1500_U	<p>Easy to spot it was fake, 0/10 musicality</p> <p>Could be inspiring (Not much to say really)</p> <p>Another catch sample, MS clearly listening with intention. Also clearly proves that 1500 epochs is not enough</p>
S7_V_2	<p>MS got it wrong thought it was real, difficult to spot 8/10</p> <p>Could be inspiring (difficult to tell if it's real due to all the added digital distortion although the groove is there)</p> <p>The model is quite successful at translating the sonic qualities although it adds too much distortion to the guitar, again the 'random' approach of the model could be inspiring because it provides options that the 'trained' brain of MS rarely could create by himself. Therefore, it can tap into unexplored territory. In this case by adding an extra section.</p>
S7_Vd	<p>Easy to spot it was real, 1/10 musicality</p> <p>uninspiring (this is the real take of the previous sample with no distortion)</p>

The model does a good job adding extra sections, the naïve copy-paste approach works and adds interesting elements but the most important takeaway here is the sonic translation. MS has not picked up any sonic 'faults' and that is quite promising for future reference.

Finally, we can see below MS answering the final questionnaire to validate if our hypothesis is correct or not. Please see below:

Responses cannot be edited

## Final feedback model

Summing up the contents of the previous questionnaires

Do you think the model can help you to compose and finish some of the songs you composed for album 2?

it can offer a different texture in terms of sounds and moods but I don't believe it to be helpful to compose songs.

Do you think that some of the samples have tapped into musical areas that you would not go as composer?

Maybe, but again, they don't procure enough solid ground to base a composition on.

Have you got any suggestions to improve the model? what would you need from it to produce so you get more inspired?

I believe that as a proposal, this model of AI can contribute to textures and a variety of rhythmic patterns but to be more effective towards songwriting it needs to work more along the guidance of contemporary music harmony structures overall.

*Figure 38 Final questionnaire submission*

(source: author)

Takeaway 1: The model is still not ready to produce fully-fledged musical prompts.

Takeaway 2: So far it can help by providing some extra textures and sounds.

Takeaway 3: Training on one single sample is a limiting factor.

Takeaway 4: More work is needed to provide a solution in addressing issues 1 and 2.

## 6 Conclusion and Future Directions

We can conclude that the model although viable for music generation and capable of producing a good level of music translation (especially by keeping the timbre) is still a bit short if it is to provide chunks of music that can work as effective prompts to trigger a Eureka effect. Having said that, it can offer help and a certain level of inspiration as the model is able to tap into areas that a trained musician might find difficult, offering some textures and sounds that can definitively help. This has been corroborated by both the author and the main songwriter of L'anima. Therefore, our hypothesis is not valid yet and more work needs to be done in order to generate prompts that will do better.

In this case, we have identified the fact of training only with one single sample as the main limitation to generating better prompts. The next step would be to move from naïve 'improvisations' to fully-fledged 'musical' prompts. But how can we move from one to the other? So far, we have a model that can produce variations of a single sample as input that it's fairly good at sonic translation. Maybe the way forward is to combine this model with another model that can learn from our GAN dataset. We could look at it with a Bayesian approach and try to maximize the probability of getting the best-generated sample based on another one.

$$P(1|2) \sim P(2|1) * P(1)$$

Where  $P(2|1)$  would represent what CAW already does: finding a good distribution of the initial data with a good sonic translation of the input; then we combine it with what it lacks: more prior knowledge of those more 'musical' elements and long dependencies that would be 'learned' after training on the GAN dataset. We already have the dataset (which can be improved and made even bigger) so the challenge would be finding the model for  $P(1)$  and a training procedure to maximize  $P(1|2)$ . Maybe combining CAW with a variational autoencoder?

Thank you very much.

Ivan Sanz

# List of References

- [1] Vanderplas, Jacob T. Python Data Science Handbook: Essential Tools for Working with Data. Beijing Etc., O'Reilly, Cop, 2017.
- [2] Summers, B. (2005). *How Watermelon Man was composed*. [Masterclass at Drumtech, London UK] He explained the 'Eureka' moment and how Herbie Hancock heard his improvisation and made a song out of it. He was not given any credit for it and makes no money out of this song in royalties.
- [3] Huzaifah bin Md Shahrin, M. (2020). *Deep Generative Models for Musical Audio Synthesis*. <https://arXiv:2006.06426v1>
- [4] Foster, D. (2019). *Generative deep learning: teaching machines to paint, write, compose, and play*. Beijing; Boston; Farnham; Sebastopol; Tokyo O'Reilly.
- [5] Goodfellow, Ian J, et al. "Generative Adversarial Networks." *ArXiv.org*, 2014, <https://arxiv.org/abs/1406.2661>.
- [6] Donahue, C., McAuley, J. and Puckette, M. (2019). Adversarial Audio Synthesis. *arXiv:1802.04208 [cs]*. [online] Available at: <https://arxiv.org/abs/1802.04208>.
- [7] Engel, J., Agrawal, K.K., Chen, S., Gulrajani, I., Donahue, C. and Roberts, A. (2019). GANSynth: Adversarial Neural Audio Synthesis. *arXiv:1902.08710 [cs, less, stat]*. [online] Available at: <https://arxiv.org/abs/1902.08710>
- [8] Mehri, Soroush, et al. *SAMPLERNN: AN UNCONDITIONAL END-TO-END NEURAL AUDIOGENERATION MODEL*. 6 June 2016. Available at: <https://openreview.net/pdf?id=SkxKPDv5xl>
- [9] Carr, C.J. and Zukowski, Z. (2018). Generating Albums with SampleRNN to Imitate Metal, Rock, and Punk Bands. *arXiv:1811.06633 [cs, eess]*. [online] Available at: <https://arxiv.org/abs/1811.06633>
- [10] Aaron, et al. "WaveNet: A Generative Model for Raw Audio." *ArXiv.org*, 2016, <https://arxiv.org/abs/1609.03499>.
- [11] Caillon, Antoine, and Philippe Esling. "RAVE: A Variational Autoencoder for Fast and High-Quality Neural Audio Synthesis." *Arxiv.org*, vol. 2, 9 Nov. 2021, <https://arxiv.org/abs/2111.05011>.

- [12] Donahue, C., McAuley, J. and Puckette, M. (2019). Adversarial Audio Synthesis. *arXiv:1802.04208 [cs]*. [online] Available at: <https://arxiv.org/abs/1802.04208>.
- [13] Engel, J., Agrawal, K.K., Chen, S., Gulrajani, I., Donahue, C. and Roberts, A. (2019). GANSynth: Adversarial Neural Audio Synthesis. *arXiv:1902.08710 [cs, less, stat]*. [online] Available at: <https://arxiv.org/abs/1902.08710>
- [14] Foster, D. (2019). *Generative deep learning: teaching machines to paint, write, compose, and play*. Beijing; Boston; Farnham; Sebastopol; Tokyo O'Reilly.
- [15] Huzaifah bin Md Shahrin, M. (2020). *Deep Generative Models for Musical Audio Synthesis*. *arXiv:2006.06426v1*
- [16] Foster, D. (2019). *Generative deep learning: teaching machines to paint, write, compose, and play*. Beijing; Boston; Farnham; Sebastopol; Tokyo O'Reilly. Chapter 1-4
- [17] Zukowski, Z. and Carr, C.J. (2018). Generating Black Metal and Math Rock: Beyond Bach, Beethoven, and Beatles. *arXiv:1811.06639 [cs, eess]*. [online] Available at: <https://arxiv.org/abs/1811.06639>
- [18] Foster, D. (2019). *Generative deep learning: teaching machines to paint, write, compose, and play*. Beijing; Boston; Farnham; Sebastopol; Tokyo O'Reilly page 31.
- [19] Lecun, Yann, et al. "Gradient-Based Learning Applied to Document Recognition." *PROC. OF the IEEE*, vol. 1, 2006, vision.stanford.edu/cs598\_spring07/papers/Lecun98.pdf.
- [20] Radford, Alec, et al. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks." *ArXiv:1511.06434 [Cs]*, 7 Jan. 2016, <https://arxiv.org/abs/1511.06434v2>.
- [21] Griffin, D., and Jae Lim. "Signal Estimation from Modified Short-Time Fourier Transform." *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, 1 Apr. 1984, pp. 236–243, [ieeexplore.ieee.org/document/1164317](http://ieeexplore.ieee.org/document/1164317), 10.1109/TASSP.1984.1164317.
- [22] Engel, J., Agrawal, K.K., Chen, S., Gulrajani, I., Donahue, C. and Roberts, A. (2019). GANSynth: Adversarial Neural Audio Synthesis. *arXiv:1902.08710 [cs, less, stat]*. [online] Available at: <https://arxiv.org/abs/1902.08710>
- [23] Wang, Yuxuan, et al. "Tacotron: Towards End-To-End Speech Synthesis." *ArXiv.org*, 2017, <https://arxiv.org/abs/1703.10135>.
- [24] Vasquez, Sean, and Mike Lewis. "MelNet: A Generative Model for Audio in the Frequency Domain." *ArXiv:1906.01083 [Cs, Less, Stat]*, 4 June 2019, <https://arxiv.org/abs/1906.01083>.

- [25] Greshler, Gal, et al. "Catch-A-Waveform: Learning to Generate Audio from a Single Short Example." *ArXiv:2106.06426 [Cs, Eess]*, 26 Oct. 2021, <https://arxiv.org/abs/2106.06426>.
- [26] Greshler, Gal, et al. "Catch-A-Waveform: Learning to Generate Audio from a Single Short Example." *ArXiv:2106.06426 [Cs, Eess]*, 26 Oct. 2021, <https://arxiv.org/abs/2106.06426>.
- [27] Arjovsky, Martin, et al. "Wasserstein GAN." *ArXiv.org*, 2017, <https://arxiv.org/abs/1701.07875>.
- [28] Gulrajani, Ishaan, et al. "Improved Training of Wasserstein GANs." *ArXiv:1704.00028 [Cs, Stat]*, 25 Dec. 2017, <https://arxiv.org/abs/1704.00028>.
- [29] Shaham, Tamar, et al. *SinGAN: Learning a Generative Model from a Single Natural Image*. <https://arxiv.org/abs/1905.01164>



# Appendix

## Link to GitHub:

<https://github.com/Birkbeck/msc-data-science-project-2021-22-ibantxodrumz>

Note: GitHub features only a part of the data for this project, please see below for links to all the data that has been created in the different iterations during the project.

## Links to datasets in Google Drive:

GAN dataset:

<https://drive.google.com/drive/folders/1PBv7MZQbmeFQnaxsYM1VceF51vHiX7KI?usp=sharing>

CAW training dataset:

[https://drive.google.com/drive/folders/1TYpCQSoj\\_ec6J0e0LNhU-TQHUABCuu08?usp=sharing](https://drive.google.com/drive/folders/1TYpCQSoj_ec6J0e0LNhU-TQHUABCuu08?usp=sharing)

CAW (originals/ generated/ human evaluation):

<https://drive.google.com/drive/folders/1PGVhRqTiAYltXdo4zAMT7SL6GI0o2wxS?usp=sharing>

CAW for human evaluation:

[https://drive.google.com/drive/folders/1vSUXe6AEGljq\\_YTJs4kdNq\\_WfMOik3GV?usp=sharing](https://drive.google.com/drive/folders/1vSUXe6AEGljq_YTJs4kdNq_WfMOik3GV?usp=sharing)

Prism SampleRNN

<https://drive.google.com/drive/folders/1Qf3N1cemrxukMgOc8Bs0Eh3afTUzkoRS?usp=sharing>