# University of Padova


Department of Information Engineering

*Master Thesis in ICT for Internet and Multimedia*


# Restoration Of The Damaged Sound Recordings Using Generative Adversarial Networks (GANs)


*Supervisor*

Prof. Sergio Canazza Targon

University of Padova

*Co-supervisor*

Matteo Spanio

Alessandro Russo

*Master Candidate*

Adnan Kerem AKSOY

*Student ID*

2081392


*Academic Year*

2023-2024

First and foremost, I would like to express my deepest gratitude to my family—my mother Nurçin AKSOY, my father Erkan AKSOY, and my brother Sertaç AKSOY—whose love, encouragement, and unwavering support have been my greatest source of strength throughout this journey.

I would also like to extend my sincere thanks to my professor, Sergio Canazza Targon, and my supervisors, Matteo Spanio and Alessandro Russo, for their invaluable guidance, insights, and patience. Their expertise and support have played a significant role in the completion of this work, and for that, I am truly grateful.

"Victory is for those who can say "Victory is mine". Success is for those who can begin saying "I will succeed" and say "I have succeeded" in the end."
— Mustafa Kemal Ataturk

# Abstract

The recent advancements in Generative Adversarial Networks (GANs) have accomplished incredible success in various fields, including image synthesis, video generation, and natural language processing.

This thesis explores the application of GANs in audio processing, particularly focusing on the reconstruction and enhancement of corrupted audio signals. The main objective of this research is to use GANs to learn the intricate patterns of clean and corrupted audio data, therefore generating reliable harmonic audio reconstructions from corrupted inputs. To achieve this, an old film's audio was used as a dataset that had corrupted parts in it to train and test the result. But to train the GAN in a better way a dataset was used that contains both real and corrupted versions of the same sound samples.

The training process was conducted with this dataset, and the old film's sound file was used to test the GANs results. Preprocess of the audio samples is made both by extracting Mel-Frequency Cepstral Coefficients (MFCCs) and by extracting Short-time Fourier transform (STFT) for different applications to see which works. Both of these representations serve as the input for different GAN applications to find the most suitable one. The test of the reconstructed audio signals we have achieved with different GAN architectures is done by comparing the spectrograms of both original and reconstructed audio signals, and the second way is listening and comparing both original and reconstructed audio signals. With this thesis we aim to make our GAN-based model effectively learn and reconstruct high-quality audio signals from corrupted inputs without losing the originality of it, making it a promising tool for various applications in audio enhancement and restoration.

This research contributes to the growing field of audio processing with GANs, providing insights and methodologies for future explorations in enhancing audio quality using deep learning techniques.

# Contents

# Listing of figures

# Listing of tables

# Listing of acronyms

**STFT** . . . . . Short-time Fourier transform

**GAN** . . . . . Generative Adversarial Networks

**DCGAN** . . . . . Deep Convolutional Generative Adversarial Networks

**iSTFT** . . . . . Inverse short-time Fourier transform

**MFCC** . . . . . The mel frequency cepstral coefficients

# 1 Introduction

## 1.1 Context and Problem Statement

In the realm of audio restoration, maintaining the originality and authenticity of sound recordings is critical. A significant challenge arises when dealing with damaged audio files, especially those from past decades, which often suffer from missing frequency components, distortions, and unwanted noise. The main objective is to restore these damaged sound recordings while preserving their original essence. Traditional denoising techniques focus primarily on removing noise, but this can lead to the loss of important background elements, failing to accurately recreate the original sound.

This issue is compounded in projects involving non-uniform data, like those found in older recordings or films. The dataset in this thesis consists of sound files with a wide variety of characteristics—empty segments, ambient sounds, music with and without lyrics, speech, and background noise typical of older films. This non-uniformity presents a unique challenge because typical machine learning algorithms are designed for uniform datasets where the data samples are similar to each other. However, in this case, the goal is to restore sound recordings with varying content while ensuring that background noise and ambient sounds, which are integral to the authenticity of the recordings, remain intact.

The goal of this thesis is to develop a model capable of restoring damaged sound files while preserving their original qualities and characteristics. By using Generative Adversarial Net-

works (GANs), which have proven effective at learning and generating from non-uniform, high-dimensional data distributions, we aim to reconstruct the damaged sounds of an old film. Unlike conventional audio restoration methods, GANs offer the advantage of generating synthetic data that closely mimics the original data, even when the dataset includes a wide range of audio features. This makes GANs an ideal choice for handling the complexity of non-uniform audio data in this restoration task.

The following sections will introduce the core concepts of GANs and their application to audio reconstruction.

## 1.2 Generative Adversarial Networks (GANs)

Since their initial introduction by Ian Goodfellow et al [4]. in their 2014 study, Generative Adversarial Networks (GANs) have emerged as a revolutionary method in the field of unsupervised learning. The two neural networks that make up a GAN are a Discriminator (D) and a Generator (G). They operate together in a dynamic adversarial process to produce artificial data that closely mimics real-world data.

• **The Generator (G)**: The main job of the generator (G) is to create synthetic data out of random noise in order to trick the discriminator into thinking the generated data is real. The generator improves its capacity to produce realistic, high-quality outputs by recurrent training.

• **The Discriminator (D)**: In contrast, the discriminator acts as a classifier to separate the authentic data from the artificial data generated by the generator. Accurately determining if a sample is real or intentionally manufactured is its goal.

• **Adversarial Training**: A competitive atmosphere is created by the interaction between the discriminator and generator. The discriminator gets better at telling the difference between created and genuine data as the generator keeps improving its outputs to trick it. Both networks are driven toward improved performance by this adversarial process.

GANs have proven to be highly versatile and impactful, with widespread applications in domains such as image synthesis, style transfer, and text-to-image synthesis. By mastering the ability to generate realistic images, GANs have transformed generative modeling and opened the door to novel applications in industries ranging from entertainment to healthcare. In these applications, the discriminator acts as a model designed to differentiate between real and gen-

erated samples, ensuring the generator continuously improves its output.

There are three main parts of a generative adversarial network:

• **Generative**: This describes a model that attempts to learn the data's underlying probability distribution. By generating new instances that closely mirror the original data set, the generator network aims to capture the process of data generation.

• **Adversarial**: The rivalry between the discriminator and the generator is what makes GANs adversarial. The discriminator compares the artificial samples produced by the generator against actual data. Over time, both networks get better because to this competitive structure.

• **Networks**: Deep neural networks serve as the foundation for both the discriminator and generator in GANs. Using artificial intelligence algorithms, these networks are taught in tandem to improve on their individual jobs, producing fresh data and correctly detecting phony samples.

The key to GANs' strength lies in the interaction between the discriminator and generator. Learning from the errors found by the discriminator, the generator gradually improves its capacity to generate realistic data. The adversarial connection between the two networks creates a feedback loop that continuously drives both networks toward higher performance. As a result, synthetic data is produced that is progressively similar to the actual data set.

## 1.3 GAN Loss Functions and Objectives

The Generator (G) and Discriminator (D) in a Generative Adversarial Network (GAN) participate in a kind of tug-of-war as one network tries to outsmart the other. Their individual loss functions, which manage this opposing rivalry and assist in the GAN's progressive improvement, are the center of the training process.

### 1.3.1 The Discriminator's Task

The Discriminator (D) functions like a judge whose job is to tell actual data different from the fake data provided by the Generator. The discriminator attempts to reliably identify each actual data sample $x$ as real. However, the discriminator attempts to avoid being tricked when it examines data $G(z)$, which is produced by the Generator from random noise $z$. Its loss function can be expressed as:

$$\mathcal{L}_D = -\mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] - \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

In simpler terms:

The discriminator's belief in the validity of $x$ (a real data sample) is expressed as $D(x)$.
The degree to which it considers $G(z)$ (a generated sample) to be real is $D(G(z))$.

Aiming to optimize this loss function, $D$ seeks to improve the ability to distinguish between true and fake data. Its score increases with how well it distinguishes between actual and fake data.

### 1.3.2 The Generator's Task

The goal of The Generator (G), however, is entirely different. Its job is to produce data that is so convincing the Discriminator is unable to distinguish it from real data. The generator attempts to trick the Discriminator into believing the created data is real for each batch of noise $z$ it gets. This is how its loss function appears:

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z(z)}[\log D(G(z))]$$

In plain terms: - $G(z)$ is the generated data sample, - $D(G(z))$ is the discriminator's belief that this generated data is real.

The generator wants to minimize this loss function, which means that it wants the Discriminator to make wrong assumptions about the validity of the data more frequently.

### 1.3.3 The Push and Pull of GAN Training

Training a GAN can be viewed as a game between two networks, where the Generator tries to fool the Discriminator with compelling fake data, and the Discriminator works to correctly classify real and fake data. The mathematical description of this back-and-forth competition is as follows:

The basic concept of their relationship can be expressed in the following equation:

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

The Discriminator aims to maximize its ability to distinguish between real and fake data.

The Discriminator's accuracy rate is what the Generator aims to reduce as much as possible.

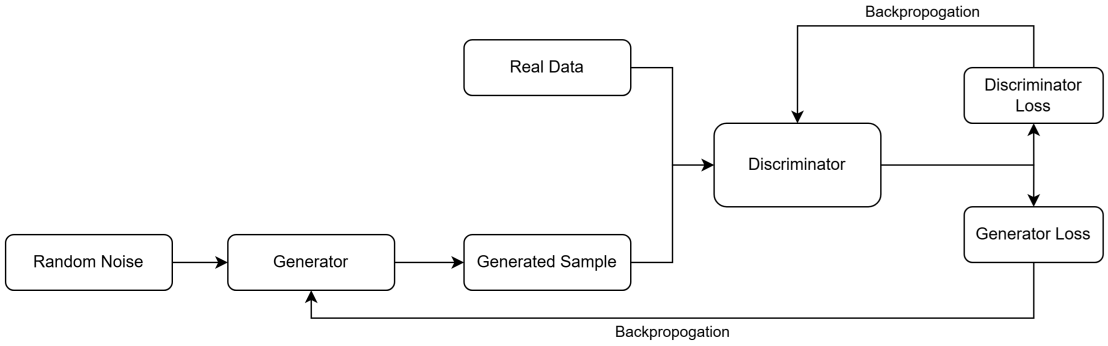



**Figure 1.1:** GAN pipeline

In a sense, the Discriminator and Generator push each other to improve with time. The Discriminator pushes the Generator to generate data that is more and more accurate as it gets better. Similarly, when the Generator gets better, it pushes the Discriminator to get sharper.

### 1.3.4 A Practical Adjustment for Better Training

In reality, though, a small adjustment is frequently done to benefit the generator. It is frequently more efficient to train $G$ to maximize $\log(D(G(z)))$ rather than to minimize $\log(1 - D(G(z)))$, which can result in slow progress early on:

$$\mathcal{L}'_G = \mathbb{E}_{z \sim p_z(z)}[\log(D(G(z)))]$$

This alternative formulation helps the generator learn more quickly and efficiently by ensuring that its gradients don't disappear too soon in the early phases of training.

## 1.4 GAN Application in the Audio Domain

Generative Adversarial Networks, or GANs, are finding a lot of applications in the audio space in addition to their original use in image processing. While GANs were first introduced for visual data by Goodfellow et al.[4] (2014), there is increasing awareness of their potential applications in audio processing, particularly in jobs involving the creation, alteration, or restoration of sound.

Most of the reason for GANs' widespread use in the audio industry is their ability to replicate intricate, high-dimensional data distributions such as sound waveforms and spectrograms. The capacity of GANs to generate audio samples that nearly perfectly mimic the characteristics of real sounds opens up an infinite number of applications across several areas.

### 1.4.1 Audio Enhancement and Denoising

Speech enhancement and noise reduction are two of the most popular uses of GANs in the audio domain. Conventional techniques for improving audio, including spectral subtraction [5] or Wiener filtering[6], frequently fail to strike the right balance between eliminating undesirable noise and keeping the audio's naturalness and clarity. These techniques have the potential to reduce noise, but they frequently introduce artifacts that unbalance the clean signal. Higher fidelity outcomes are possible with GANs since they can learn the underlying patterns in both clean and loud audio, making them more successful in addressing these problems.

Because they can be trained to map noisy audio inputs directly to clean outputs, preserving the basic qualities of the original audio while removing noise, GAN-based models are especially potent. For example, SpecGAN[2] learns the time-frequency structure of spectrograms

to recreate audio that has been degraded by noise. SpecGAN can recover clear spectrograms from noisy inputs and identify intricate correlations in the audio data by utilizing the spectrogram representation. This was successfully illustrated in voice improvement tasks, where GANs have proven to perform noticeably better in noisy situations than standard approaches. [2].

For denoising tasks, other GAN variations such as DCGAN and MelGAN have also been used. By learning complex representations from spectrograms, DCGAN—which first achieved success in the visual domain—was extended to audio processing [1]. Conversely, mel-spectrogram representations are the focus of MelGAN, which has been applied to speech and music denoising. In fields like real-time speech communication applications, telecommunication systems, and hearing aid technology—where voice clarity is essential—these models have demonstrated a great deal of potential. [3].

For instance, it's crucial to preserve voice quality in noisy environments (like busy areas or public transit) when using telecommunication services. Intelligible and natural speech can be preserved while background noise is efficiently reduced by GANs trained on paired datasets of clean and noisy speech. This development creates opportunities for real-time noise reduction in smart assistants and mobile devices, where it's crucial to preserve high-quality audio in a variety of settings.

### 1.4.2 Speech and Music Synthesis

The field of speech and music synthesis has benefited greatly from the development of Generative Adversarial Networks, which enable the production of high-fidelity audio that closely resembles real sounds. Conventional techniques in the field of speech synthesis, including parametric models or concatenative synthesis, frequently produce artificial and robotic-sounding voices. By producing speech that is more fluid and realistic and that captures the subtleties and differences in human voices, GANs have assisted in overcoming this constraint.

MelGAN, a model created to produce realistic speech by learning from mel-spectrograms taken from real speech data, is one well-known example. To generate high-fidelity audio outputs, MelGAN uses a fully convolutional architecture that works directly on the mel-spectrograms. Natural-sounding and expressive speech is crucial in text-to-speech applications, virtual assistants, and voice cloning technologies, where it has been widely employed [3]. For example, GAN-based models provide smoother, more human-like speech in virtual assistants such as Siri and Alexa, improving the user experience.

GANs such as WaveGAN have also held great promise for music synthesis. WaveGAN creates music and sound effects straight from scratch by operating directly on raw audio waveforms [2]. According to conventional methods of music synthesis, which frequently call for hand-crafted features and domain-specific knowledge, WaveGAN uses its time-domain operations to learn how to produce continuous, flowing sounds. This makes it especially appropriate for the production of music, where coherence and smooth audio transitions are crucial.

GANs are also being used in creative domains, including style-transfer-based music generation[7]. For instance, a GAN can be trained to produce new jazz-style compositions after learning the stylistic elements of classical music. This makes it possible to create completely original music while preserving the unique qualities of the intended genre. These applications show showcase GANs' creative potential in the music business and provide new tools for creativity to composers and musicians.

The secret to GANs' effectiveness in speech and music synthesis is their capacity to learn complicated, high-dimensional data distributions. This is necessary in order to produce realistic audio that faithfully reflects the nuances of both musical composition and human speech. These models offer excellent, adaptable, and effective solutions for a variety of applications, marking a substantial advancement in the field of audio synthesis.

### 1.4.3 Audio Super-Resolution

Upsampling low-resolution audio to high-resolution allows audio super-resolution to restore features lost through downsampling or compression. When compared to conventional interpolation techniques, GANs perform better at this task because they are efficient at understanding the intricate mappings between low- and high-resolution data.

In order to preserve the original temporal and spectral features, models such as WaveGAN have been utilized to create high-quality audio from lower-resolution samples [2]. This is especially helpful for restoring music, as compressed audio files tend to lose their richness in tone and timbre. These small details can be efficiently recovered by GANs, bringing the output's quality closer to that of the original.

GANs are used in real-world applications like streaming services and telecoms to improve the quality of transmitted or compressed audio. These algorithms make sure that hearing is more natural and detailed by upsampling voice or audio data.

A major obstacle in audio super-resolution [8] is keeping upsampling artifacts at check. In order to overcome this, GANs create outputs that are perceptually similar to actual high-resolution

audio by learning from genuine high-resolution data. This technique offers a viable way to enhance sound quality in a number of applications, such as real-time communications and digital preservation.

### 1.4.4 Audio Domain Adaptation

The term "audio domain adaptation" describes the process of translating audio from one language, style, or domain to another while preserving its original core[9]. In this area, GANs have demonstrated tremendous promise by learning mappings between several audio domains.

GANs can be used, for example, in voice conversion, which is the process of converting speech from one language or accent to another while maintaining the speaker's individuality. For movies and virtual assistants, this method is very helpful for automatic dubbing or multilingual voice synthesis [2]. Similar to this, musical style transformation—such as translating classical music into jazz—can be used to accomplish music domain adaptation. This creates new avenues for musical creation and composition.

GAN-based models use paired or unpaired data to learn from in order to produce high-quality transformations. They make the changes sound natural by preserving the essential elements of the audio while capturing the original sound's timbre and style and modifying it for the target domain.

The difficulty is in performing transformation without sacrificing accuracy, but as GAN designs progress, models can now perform well on a wide range of audio domain tasks.

### 1.4.5 Sound Source Separation

Isolating distinct sound sources from a mixed audio signal is known as sound source separation[10]. High-quality separation has been achieved through the successful application of GANs to this issue, particularly in complicated audio settings.

For instance, in the process of producing music, GANs can distinguish between vocals and instrumental recordings, or they can be used to isolate specific ambient sounds, like footfall, traffic noise, or speech, from an audio recording. When dealing with such complicated mixtures, traditional techniques frequently falter, but GANs succeed because they can understand the complex connections that exist between mixed signals and their sources. [2].

Researchers have outperformed traditional methods in source separation by teaching GANs to distinguish between several sound sources within an audio sample[11]. The discriminator

makes sure that each divided source stays realistic and faithful to its original shape, while the generator is educated to recreate individual sources.

Although scaling this method for highly overlapping or noisy environments still presents challenges, continuous improvements in GAN models keep pushing the envelope of what is feasible in terms of sound source separation.

### 1.4.6 Music Restoration

Particularly when applied to musical contexts, GANs have demonstrated considerable promise in the repair of degraded audio. Models like SpecGAN and MelGAN are perfect for fixing old or corrupted audio files because they are particularly made to maintain the harmonic characteristics of musical recordings [2],[3].

During the denoising process, traditional audio restoration techniques frequently lose the authenticity of the original sound, which could change the music's distinctive qualities. GANs, on the other hand, successfully restore lost or damaged audio while preserving the original nuance and richness.[12]

For example, GANs may correctly infer the missing pieces based on the surrounding audio context, filling in the gaps in recordings caused by damage or degradation. This capacity to recover music without distorting its unique vibe is essential, particularly for older pieces where maintaining the original sound quality is critical.

As research progresses, using GANs in music restoration not only improves the sound quality of archive recordings but also creates new opportunities for cultural heritage preservation.

### 1.4.7 Challenges in GAN-Based Audio Processing

Although GANs have shown great promise in the audio domain, there are still major issues that need to be resolved. First, because audio data may include many time scales and is sequential, it is by nature more complex than visual data. A time-series data set is what audio signals are, as opposed to images, which are usually organized in 2D grids. More difficult architectural design is required for audio-based GANs since their proper generation necessitates that the model captures both local and global dependencies over time.

A unique challenge in this project arises from the non-uniform nature of the dataset. The sound data used in this work is very diverse, ranging from spoken dialogue to music with lyrics, empty segments, background atmosphere, and music with and without lyrics. This is in contrast to traditional machine learning challenges that rely on homogenous datasets. These dis-

parate components combine to form a complicated dataset with distinct frequency content and sound structure for each segment. Further challenges arise from the requirement for GANs to understand how to manage the unique properties of each segment without overfitting to any specific sort of data, while maintaining coherence across such a wide range of inputs throughout training.[13]

Additionally, audio data frequently has a large number of dimensions, especially when it comes to its raw waveform form. Due to the need to manage massive volumes of data, dimensionality raises the computational cost of producing and processing high-quality audio in models like WaveGAN that work with raw waveforms. Training sessions that require a lot of time and resources may result from this [2].

Lastly, the well-known problem of training instability in GANs is frequently made worse by the requirement of processing non-uniform audio data[13]. In this case, the outputs that are generated have to retain their realism and semantic coherence throughout a broad spectrum of sound kinds. To address these issues, researchers are always improving GAN architectures and loss functions. Sophisticated methods such as Wasserstein loss[14] and spectral normalization have been developed [14] to enhance the output quality and training process stability.

## 1.5 The Goal of this Work

Restoring damaged sound files from an ancient film while maintaining the original quality of the audio is the aim of this project. In particular, the following objectives are searched :

Reconstruction of Damaged Audio Segments: Maintaining accuracy to the original audio is the key goal in recovering audio segments that have worsened or been damaged over time.

Preserving Originality: Background noise is frequently included in the audio of old movies, adding to the period's authenticity and ambience. This background noise needs to be retained in the reconstruction because eliminating it would weaken the sound's originality.

Avoid Away of Noise Reduction approaches: This work stresses keeping noise as a necessary component of the soundscape in order to preserve authenticity, in contrast to traditional sound enhancement approaches that seek to eliminate noise.

Achieving High Fidelity with Minimal Alteration: The objective is to improve the sound

quality without making major adjustments that would damage from the audio's original tone, feel, or ambiance.

Managing Audio with Various Features: The audio in the movie includes a variety of sound effects, such as music, voices, and background noise, all of which need to be handled carefully to avoid disrupting the soundscape's balance.

This effort aims to create a method that carefully restores broken audio recordings while preserving their artistic and historical integrity by concentrating on these factors.

# 2

# State of the Art

Significant advances have been made in a variety of domains, such as picture generation, speech synthesis, and, more recently, audio reconstruction, thanks to the development of Generative Adversarial Networks, or GANs. We will examine the state-of-the-art GAN architectures that are useful to audio processing in this section, emphasizing their contributions, pipelines, and suitability or unsuitability for the goal of reconstructing damaged audio recordings while preserving originality.

## 2.1 Overview of Generative Models in Audio Processing

While GANs were initially designed for image generation, they have since been adapted for audio applications. However, audio data, especially in the form of waveforms, presents additional challenges due to its high dimensionality, temporal coherence, and harmonic complexity. Audio generation tasks, therefore, often involve spectrograms or mel-spectrograms, which simplify the structure by representing sound in a 2D time-frequency domain.

Subsections that follow examine the architectures of DCGAN, SpecGAN, WaveGAN, and MelGAN. They are followed by an examination of other GAN models that were not utilized in this work, including Wasserstein GAN (WGAN), Conditional GAN (cGAN), and Pix2Pix.

## 2.2 GAN Architectures for Audio Processing

### 2.2.1 DCGAN (Deep Convolutional GAN)

Among the GAN architectures, Deep Convolutional GAN (DCGAN) ([1] 2015) is one of the most well-known and reliable. Convolutional layers replaced the conventional fully connected layers in the generator and discriminator thanks to DCGAN. This breakthrough proved crucial in improving the stability of GANs during training and in their ability to process more structured data—like spectrograms or images—more accurately.

**Pipeline:**

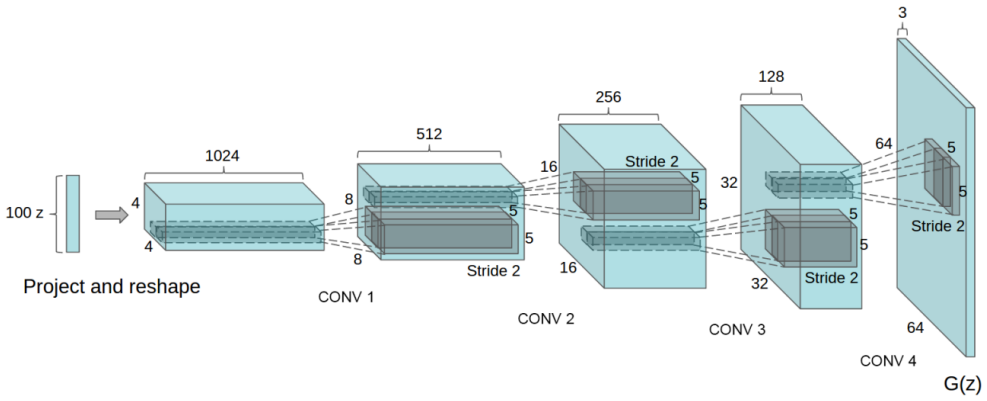



**Figure 2.1:** Generator's pipeline[1].

• **Generator** A latent noise vector, typically taken from a normal distribution, is fed into the DCGAN generator and is then processed through a sequence of convolutional transpose layers. These layers carry out an upsampling procedure, gradually increasing the data's dimensionality until it achieves the required output size, which is usually a spectrogram or a 2D image. To stabilize training and avoid mode collapse, batch normalization and leaky ReLU activation are applied in each layer. The generator can learn spatially coherent patterns thanks to this structure, which is essential for producing realistic spectrograms and pictures.

• **Discriminator** A convolutional neural network (CNN) serves as the discriminator in the DCGAN. Its job is to classify a created or genuine spectrogram as either real or fake. The design downsamples the input using stacked convolutions, extracting hierarchical features at each layer. Leaky ReLU activation guarantees that the discriminator can tolerate negative input values without creating vanishing gradients, and batch normalization is also used in this instance.

**Advantages**

Convolutional layers enable DCGAN to efficiently extract both local and global features from spectrograms, which makes it a good choice for audio applications involving structured data. DCGAN is one of the most dependable GAN architectures because of the stability that batch normalization and Leaky ReLU provide, particularly when training on 2D data like spectrograms.

**Limitations**:

Although DCGAN is good at producing spectrograms, there may be problems when converting its output back into audio because of spectrogram inversion. Since DCGAN was not intended to handle unprocessed audio waveforms, its direct use is limited to tasks like voice synthesis and music production.

**Relevance to This Work**: DCGAN is a useful tool in the process of reconstructing damaged audio since it can produce spectrograms and maintain the time-frequency structure of the audio. Additionally, the design is stable, which is essential for learning from imperfect and noisy acoustic data.[1]

### 2.2.2 SpecGAN

SpecGAN ([2], 2018) was created with spectrograms in mind, especially for audio generating challenges. It is simpler for the generator to create coherent sound patterns when it acts in the time-frequency domain, where sound is represented as a 2D picture (spectrogram), as opposed to dealing with raw audio waveforms.

**Pipeline**:

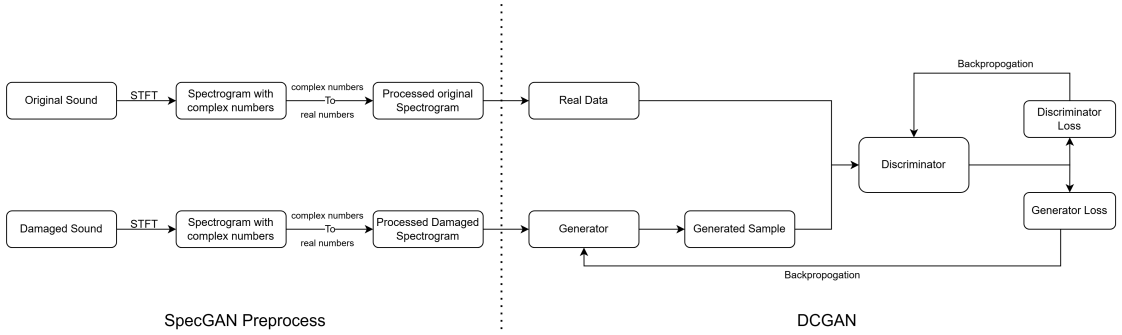



**Figure 2.2:** SpecGAN Pipeline.

**Generator**: The generator in SpecGAN upsamples random noise into a spectrogram using transpose convolutions, same like in DCGAN. As a result, the frequency and temporal patterns necessary for audio reconstruction can be captured by the model. The generator is able to generate high-quality spectrograms that maintain harmonic coherence over time by using batch normalization and Leaky ReLU activation after each layer.

**Discriminator**: The discriminator downsamples the input to a final binary output (real or fake) by processing the produced and real spectrograms through convolutional layers. With batch normalization done at each layer to guarantee training stability, the architecture is almost exactly the same as that of the discriminator in DCGAN.

**Advantages**:

SpecGAN leverages convolutional architectures, which are well-suited to processing 2D data, by concentrating on spectrogram creation. By operating in the time-frequency domain, the model streamlines the creation process and enables the capturing of specific localized sound components.

**Limitations**:

Because SpecGAN relies on spectrogram inversion, which involves turning the created spectrogram back into audio, there is a chance that artifacts can arise, which could lower the output's quality.

**Relevance to This Work**: Because SpecGAN uses spectrograms to learn from the time-frequency structure of the original audio, it is a good fit for the task of rebuilding broken sound recordings. But sustaining high-quality audio fidelity can be difficult during the inversion process, especially when dealing with loud audio. [2]

### 2.2.3 WaveGAN

In contrast to spectrogram-based models such as DCGAN and SpecGAN, WaveGAN (Donahue et al., 2018) generates raw waveforms directly, rather than operating in the time-frequency domain. This eliminates the requirement for post-processing (turning spectrograms back into audio), but it also presents additional difficulties in handling high-dimensional data and maintaining temporal consistency.

**Pipeline**:

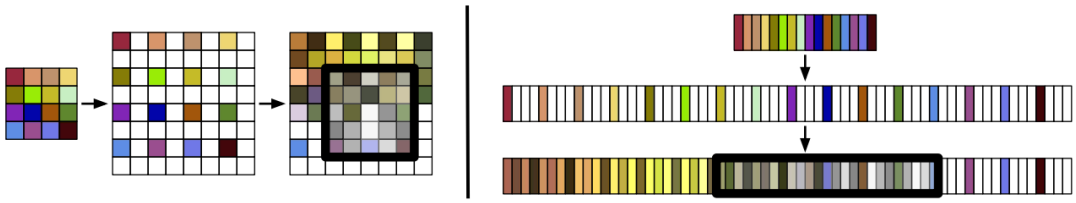


**Figure 2.3:** Generator's pipeline [2].

**Generator**: The WaveGAN generator upsamples a latent noise vector into a waveform by applying a sequence of transpose convolution layers. The model keeps temporal coherence while gradually increasing the waveform's resolution through the use of strided convolutions. WaveGAN's generator operates directly in the time domain, unlike models that generate spectrograms, therefore it may produce continuous waveforms without converting from spectrograms.

You can see the difference between convolutions and the strided convolutions which WaveGAN uses in the figure 2.3, left side is the standard convolutional approach, and right side WaveGAN's strided convolutions.

**Discriminator**: Using strided convolutions, the discriminator processes the generated and real waveforms. Phase shuffle, a regularization strategy that ensures that slight changes in the waveform have no effect on the model's classification abilities and keeps the model from learning trivial solutions, is a crucial component of WaveGAN's discriminator.

**Advantages**:

Because WaveGAN generates waveforms directly, it does not require spectrogram inversion, which lowers the possibility of artifacts being introduced during post-processing. The approach works well for problems involving voice or music generation because it preserves temporal continuity throughout long audio sequences.

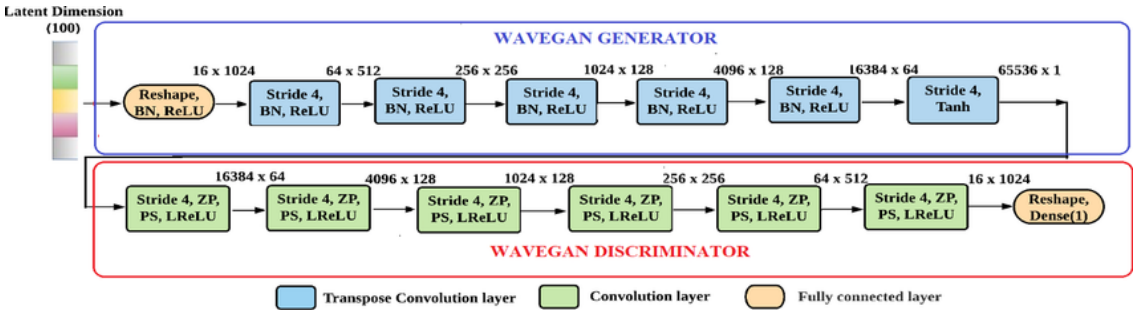



**Figure 2.4:** BN: Batch Normalization, ZP: Zero Padding, PS: Phase Shuffle.[2].

You can see the example architacture of both Generator and Discriminator in Figure 2.4.
**Limitations**: WaveGAN's high dimensionality of waveform data makes it more computationally demanding and less robust during training. Compared to spectrogram-based models, producing realistic waveforms over extended periods of time requires greater computer power and training data.

•**Relevance to This Work**: WaveGAN is not the best tool for this purpose of reconstructing damaged audio, even if it has the benefit of directly producing raw waveforms. Compared to spectrogram-based models like DCGAN or SpecGAN, the model is less practical due to its complexity and difficulties with noisy input data. [2]

### 2.2.4 MelGAN

Mel-spectrograms, which are condensed, perceptually-motivated representations of audio, are the goal of MelGAN ([3], 2019). MelGAN is quite effective, especially when it comes to real-time audio synthesis, since mel-spectrograms distill the essence of audio input with fewer frequency bins than full spectrograms. Since the architecture is non-autoregressive, it can produce audio far more quickly than autoregressive models such as WaveNet[15].

**Pipeline**:

**Generator**: Using a sequence of convolutional layers, the generator in MelGAN creates a mel-spectrogram from an input random noise vector. Upsampling is done by each convolutional layer to improve the resolution of the mel-spectrogram that is produced. MelGAN is suited for real-time applications since it is non-autoregressive and can produce the full mel-spectrogram in a single pass.

**Discriminator**: MelGAN employs a multi-scale discriminator, where many discriminators operate on different scales of the resulting mel-spectrogram. Every discriminator assesses the spectrogram's realism at various levels of detail to guarantee that the high- and low-frequency components are correctly depicted. The model performs better when numerous discriminators are used to evaluate the audio quality produced at various temporal resolutions.

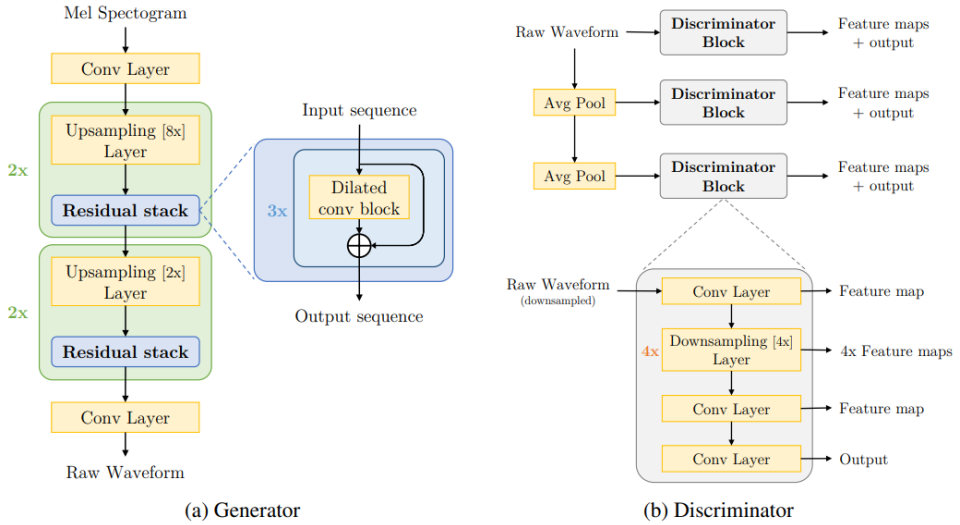



**Figure 2.5:** MelGAN Architecture [3].

**Advantages**:

Real-time audio generation: Because of its non-autoregressive nature, MelGAN is an effective generator that can be used for applications such as speech synthesis. It is ensured that the produced audio retains high fidelity across many frequency ranges by using multi-scale discriminators.

**Limitations**:

MelGAN can be less accurate for longer and more complicated audio sequences; it is best suited for speech and short-duration audio applications.

A loss of detail may result from the compression of audio into mel-spectrograms, which could be troublesome for jobs like audio reconstruction where maintaining minute details is crucial.

Relevance to This Work: Although mel-spectrogram compression is intrinsic, MelGAN's real-time capabilities may not be optimal for recovering the fine details of vintage cinema audio. A full spectrogram-based technique (such as SpecGAN or DCGAN) would yield better outcomes for this task, where maintaining the original background noise and sensitive frequencies is essential.[3]

## 2.3 Additional GAN Architectures in Audio Processing

We examine alternative GAN designs in this section, including Pix2Pix, Conditional GAN (cGAN), and Wasserstein GAN (WGAN), which have been shown to work well in other contexts but were not immediately relevant to this project. These models were not chosen for audio reconstruction for the following reasons, which also acknowledge their contributions to the larger area of GAN research.

### 2.3.1 Wasserstein GAN (WGAN)

The introduction of Wasserstein GAN (WGAN) ([14]., 2017) solved two key issues with conventional GAN training: training instability and mode collapse. The Wasserstein distance, sometimes referred to as the Earth-Mover's Distance, is used by WGAN in place of the conventional GAN loss function to offer a more consistent and continuous measurement of the separation between the generated and real data distributions.

**Pipeline**:

**Generator**: Like other GANs, WGAN's generator creates data (such as waveforms or spectrograms) by transposing convolutions. The Wasserstein distance between the produced and real data distributions is determined by WGAN's loss function, which is where the main distinction between the two approaches is found. The generator seeks to decrease this distance, providing data that is closer to the genuine distribution in a more stable manner.

**Discriminator (Critic)**: With WGAN, the traditional GAN discriminator is swapped out with a critic that does not distinguish between actual and fake data. Rather, the Wasserstein distance between the generated and real data is assessed. Weight clipping is used to enforce the requirement that the critic be 1-Lipschitz continuous in order to guarantee stability.

**Advantages**:

**Stability**: When compared to typical GANs, WGAN provides more steady training, especially when the generator produces low-quality data at first.

**Improved performance**: By using Wasserstein distance, problems such as mode collapse can be avoided and a more accurate indicator of the generator's learning performance can be obtained.

**Limitations**:

The requirement for weight clipping in the critique results in extra computational complexity and occasionally less-than-ideal outcomes. The advantages of WGAN in managing temporal data, such as audio waveforms or spectrograms, have not yet been fully investigated. It has not been extensively used for audio reconstruction tasks.

**Reasons for Not Being Selected**: WGAN's main advantage for this particular task of reconstructing broken audio files is that it stabilizes GAN training, as opposed to enhancing the temporal coherence or frequency fidelity needed for audio reconstruction. There might not be many benefits in this situation due to the extra complication of maintaining Lipschitz continuity and weight clipping.[14]

### 2.3.2 Conditional GAN (cGAN)

In order to enable the model to produce data depending on particular input circumstances (such as class labels), conditional GAN (cGAN) ([16], 2014) adds conditional inputs to both the discriminator and generator. Because of this conditioning, cGANs are beneficial for tasks requiring the development of particular qualities since they provide an extra layer of control over the generation process.

**Pipeline**:

**Generator**: To produce the intended output in cGAN, the generator requires two inputs: a conditioning input (such as a class label or other data) and a latent noise vector. As a result, the model can produce data that is consistent with the specified input condition.

**Discriminator**: In a cGAN, the generator's input also conditions the discriminator. It analyzes whether the generated data satisfies the specified condition (e.g., whether the generated audio matches the given label or prompt) in addition to evaluating whether the generated data is real or fake.

**Advantages**:

Control over generation: cGAN is helpful for applications like voice synthesis and image-to-image translation because it enables targeted generation, in which the output can be affected by the input condition.

**Limitations**: For audio reconstruction tasks, labeled data is necessary for cGAN, but obtaining it can be challenging, particularly when dealing with noisy or historical datasets.

**Reasons for Not Being Selected**: This study does not aim to synthesize audio conditioned on labels or properties; rather, it reconstructs broken audio while preserving its original qualities. As a result, conditioning's additional complexity conflicts with the project's goals.[16]

### 2.3.3 Pix2Pix GAN

A GAN architecture called Pix2Pix GAN ([17]., 2017) was created especially for image-to-image translation, in which the objective is to translate an input image into an equivalent output image. Numerous image-based tasks, such as semantic segmentation, super-resolution, and picture colorization, have been effectively handled by Pix2Pix. Despite being originally designed for visual data, Pix2Pix has been investigated for its possible application in audio-related tasks, especially when mapping one spectrogram to another (e.g., for tasks connected to audio augmentation or denoising).

**Pipeline**:

**Generator**: A U-Net architecture[18], which is appropriate for jobs requiring the preservation of both high-level and low-level properties, powers the generator in Pix2Pix. The input picture (or spectrogram, in the case of audio) is downsampled through a sequence of convolutional layers and then upsampled to its original resolution in a U-Net structure. During the transformation phase, the model is able to capture both local and global information since the downsampled and upsampled features are combined.

**Discriminator**: Pix2Pix uses a PatchGAN discriminator to determine whether individual input-output patches are real or fraudulent, as opposed to the full image. This method guarantees that the model acquires the ability to provide superior local details, while the discriminator verifies realism on a finer scale. The discriminator determines whether the generated output and the original input make sense together by evaluating both of them.

**Advantages**:

Fine-grained control: The generator can do large-scale modifications while maintaining fine details thanks to the U-Net design. Because of this, Pix2Pix is helpful for activities requiring both minor and significant audio alterations, such as audio augmentation or denoising. PatchGAN discriminator preserves high-quality details in the output it generates by concentrating on local features.

**Limitations**:

**Data requirement**: Pix2Pix primarily uses paired data, meaning that in order for the model to train, each input needs to have a corresponding output. Creating such paired data may be challenging in your situation, when the goal is to restore broken audio recordings, particularly when working with historical audio. Pix2Pix's potential for audio applications is still limited, and its effectiveness in image translation tasks may not transfer to audio jobs where temporal coherence is crucial.

**Reasons for Not Being Selected**: Although Pix2Pix's image-to-image translation capabilities might be applied to spectrogram-to-spectrogram mapping in theory (for example, mapping noisy spectrograms to clean ones), its suitability for this project is diminished due to its need for paired data and the difficulty of reconstructing temporal audio features. Rather, more specialist models with an emphasis on audio, such as SpecGAN and DCGAN, are more suited to tackle the difficulties involved in audio reconstruction.[17]

## 2.4 Summary of State of the Art

Applying each of the abovementioned GAN architectures to audio creation and reconstruction has its own set of benefits and drawbacks. Because DCGAN and SpecGAN can handle 2D spectrogram data and have demonstrated stability during training, they are especially well-suited for audio reconstruction. Though useful for real-time synthesis and direct waveform generation, WaveGAN and MelGAN add extra complexity that makes them less suitable for the task of recovering broken audio from vintage movies. Lastly, because of their unique design specifications and application contexts, Wasserstein GAN (WGAN), Conditional GAN (cGAN), and Pix2Pix—all of which provide enhancements in training stability, output control, and image-to-image translation—remain limited in their applicability to audio reconstruction.

The objective of this task is to repair damaged audio files while maintaining their original qualities, including background noises that are inconspicuous yet essential to the authenticity of the movie. Consequently, the best strategy is provided by combining DCGAN with SpecGAN. With the help of these models, realistic time-frequency structured audio that is appropriate for this reconstruction task can be produced.

# 3 Dataset

## 3.1 Old Film's Damaged Files

The principal aim of this thesis was to develop a Neural Network capable of reconstructing the Damaged Files from the old film while maintaining its originality. The corrupted files had a single channel with a sample rate of 48 KHz and a runtime of roughly three minutes each. Three samples have been used. However, the approach demonstrated that these corrupted files cannot be used to train GANs. A separate dataset made up of both good and bad samples has been created in order to move on and assess the likelihood of achieving a positive outcome.

## 3.2 Custom Dataset

The custom dataset is prepared under laboratory conditions with the same specifications as the Original Dataset received from old film. The Custom dataset has one channel and a sample rate of 48 KHz. The custom dataset has a duration of roughly two minutes as well. However, the primary distinction between the original and custom datasets is what the custom datasets include:

• A Clean Original Sound Sample

• A Sound Sample only contains 30 RMS(Root Mean Square) White Noise

• A Original Sound Sample with 30 RMS(Root Mean Square) white noise applied,

• 7 Damaged Original audio files with frequencies above 150, 250, 500, 1000, 1500, 2000, and 3000 Hz cut, respectively

30RMS white noise applied to the original sound sample Used as the original file that the discriminator learns from because if the background noise in the old film's audio reconstruction is cleaned up by a neural network during the reconstruction process, it may lose some of its originality. Consequently, the original sound file used to train the discriminator was replaced with white noise in order to reconstruct it without losing its authenticity. The generator was fed seven damaged original audio files in order to teach it how to recreate the damaged file based on the data it was fed.

## 3.3 Pre-Process and After-Process of the Custom Dataset

The sound files' specifications have been located. The waveform and sampling rate values have been obtained using the "Torchaudio" package. Large-length GANS training may be challenging because of the 48 kHz sample rate. There are 48000 samples with 1 channel every second. The sample rate multiplied by the number of seconds in two minutes can be easily calculated.

The only sound sample that will be used is this data. This limitation aside, the dataset is large. Furthermore, the amount of data needed to build GANs will be large for our hardware and model if we take into account the other examples as well. It would be difficult to train the GAN effectively with the hardware we have because the data is too large. A function was developed to solve this issue, allowing us to divide these samples into segments and provide the dataset that feeds our GANs.

## 3.4 Stft Approach

### 3.4.1 Normal Scaled STFT

#### Preprocess

The function is made to load an audio file, trim it to a specified section, divide it into smaller segments, and convert each segment into a spectrogram for further analysis or processing. Initially, the audio is loaded using 'torch audio', which provides both the waveform and the sample rate.

The start time and duration supplied as inputs are then used to reduce the audio to a certain part. This makes it possible to freely choose any part of the audio to be processed. In order to reduce the size of the dataset, this section is used, for instance, to trim off the empty durations that appear at the beginning and end of the sound files. After the audio has been cropped, it is split up into smaller, assignable parts, each lasting a certain amount of time (two seconds, for example). With this segmentation, the audio is divided into manageable portions for additional GAN training.

After the audio is divided into segments, the Short-Time Fourier Transform (STFT) is used to convert each segment into a spectrogram. During the STFT, a Hann window is used to enhance the transformation's quality and facilitate the transition between segments. Each audio segment's time-frequency representation is provided by the spectrogram, which is essential for activities like audio reconstruction or analysis.

Ultimately, the function returns the sample rate and the spectrograms that were generated, which qualifies it for additional application in GAN-based audio reconstruction. With the flexibility to choose, divide, and modify the audio, this technique guarantees effective preprocessing of the audio.

Real and imaginary components make up the segmented audio files as a result of the Short-Time Fourier Transform (STFT). a function that converts complex input into a real-valued format that the neural network can handle, as neural networks normally work with real-valued data.

It divides the complex spectrogram into real and imaginary components first. It splits into two sections, which it then combines into a single tensor along a new dimension. This effectively eliminates the complex format by enabling the neural network to process the real and imaginary components as real numbers. All the relevant information remains since the function transforms the imaginary part into a real-valued channel rather than discarding it. By returning this merged tensor, the function preserves the data from both parts in a format that can be utilized by the network at later stages. As a result, the data was transformed into a 2D picture for our DCGAN-based model to use.

Following processing, all of the clean and noisy audio data will be kept in different lists. Every noisy sample is coupled with a comparable clean sample to prepare the Data Loader. To keep a matched set for training, the clean samples are cycled through if there are more noisy samples than clean ones.

The training process is made smooth and efficient by this structure, which guarantees that the model always gets well-prepared input (noisy spectrogram) and target (clean spectrogram) pairs to learn from.

### After Process

The resulting information must be transformed back into an audio format after processing. The spectrograms have to be recombined to regain the complex format needed for audio generation because they were first divided into real and imaginary components for processing. The two channels—the real and imaginary portions—that were split apart during preprocessing are blended back together to accomplish this. This makes it possible to reassemble the audio signal by transforming the data back into its original, complicated format. The inverse Short-Time Fourier Transform (iSTFT) is used to transform the 2D spectrogram back into a waveform after the complicated data has been restored. By reconstructing the audio signal from the spectrogram, this approach enables the output to be given back as a regular audio file.

**Algorithm 3.1** Audio Preprocessing with Spectrogram Generation

Load the audio file: $(waveform, sample\_rate) \leftarrow$ torchaudio.load($audio\_path$)
**if** waveform has multiple channels
  Convert waveform to mono by averaging channels
**end if**
Trim the audio based on the start time and duration
Divide the trimmed waveform into smaller segments based on segment_length
**for** each segment in segments
  Apply Short-Time Fourier Transform (STFT) to obtain spectrogram
**end for**
**for** each spectrogram
  Extract real and imaginary parts
  Combine real and imaginary parts into a real-valued tensor
**end for**
Pair noisy and clean spectrograms for training
**if** clean data is shorter than noisy data
  Cycle through clean data
**end if**
Return the processed spectrograms and sample rate

**Algorithm 3.2** After Processing: Reconstructing Audio from Spectrograms

**Input:** Real and Imaginary spectrogram components
**Output:** Reconstructed audio file
Combine real and imaginary components into a complex spectrogram
Apply Inverse Short-Time Fourier Transform (iSTFT) to convert the spectrogram back to waveform
Save the reconstructed waveform as an audio file
**return** Reconstructed audio

### 3.4.2 Logarithmically Scaled STFT

#### Preprocess

The only difference in the preprocessing for the logarithmically scaled version is the logarithmic scale used to modify the spectrogram's magnitude. Following the creation of the spectrogram from the waveform, the magnitude is taken out and scaled logarithmically. The spectrogram's phase is preserved, and it is reconstructed by combining the log-scaled magnitude with the original phase. This procedure makes sure that the data's finer features are more successfully captured while also enhancing the dynamic range.

#### After Process

After the GAN produces its output, the result data must be converted back to its original scale during the post-processing stage. The result data was processed in a format that was logarithmically scaled. This is accomplished by applying the inverse logarithmic function to the magnitude of the resulting spectrogram. The real and imaginary components are joined once the magnitude is reversed, giving the spectrogram its complex format back. Ultimately, the audio can be fully reconstructed by utilizing the inverse Short-Time Fourier Transform (iSTFT) to convert the spectrogram back into the waveform.

By reversing the logarithmic transformation, this strategy assures that the GAN can handle the log-scaled input and produce an accurate reconstruction of the final audio output. The preservation of significant details while preserving the integrity of the audio signal throughout GAN processing makes this method essential to the thesis.

## 3.5 MFCC

### 3.5.1 Normal Scaled MFCC

#### Preprocess

The goal of this approach's preprocessing phase is to extract the audio data's Mel-Frequency Cepstral Coefficients (MFCC). As with other methods, the process starts with segmenting the audio and trimming it to a certain section. The key spectral characteristics of the audio are then compactly captured by converting each segment into an MFCC representation. With an emphasis on lower frequencies, MFCCs are helpful in recognising a signal's frequency characteristics in a way that approximates human hearing.

#### AferProcess

Reconstructing the audio requires turning the MFCCs back into a mel spectrogram once it has been converted into MFCCs and run through the GAN. The spectrogram is then reversed using techniques like the Griffin-Lim algorithm and the Inverse Mel Scale transformation to return it to the original audio format. By taking these precautions, the created data is guaranteed to be converted back into a waveform that closely mimics the source audio signal. By using this technique, the GAN can work with MFCCs, which simplify the data while preserving important audio characteristics. The MFCCs are then processed by the GAN and converted back into a usable audio format.

### 3.5.2 Logarithmically Scaled MFCC

#### Preprocess

With one significant exception, the preprocessing steps for the logarithmically scaled MFCC technique are identical to those for the regular MFCC extraction: once the MFCCs are generated from the audio segments, a logarithmic scaling is applied to the coefficients. This scaling can enhance the GAN's capacity to recreate faint audio characteristics by compressing the MFCCs' dynamic range and highlighting the signal's finer details.

### AferProcess

The opposite procedure is used after the GAN produces the result. By using the inverse logarithmic function on the MFCCs, the logarithmic scaling is reversed. The MFCCs are converted back into a mel spectrogram after being scaled back to their initial value. The audio file is then rebuilt by converting this mel spectrogram into a waveform using the Inverse Mel Scale and the Griffin-Lim technique.

This method makes use of logarithmic scaling to improve the performance of the GAN on MFCC data while guaranteeing that, following the application of inverse transformations, the final output stays faithful to the original audio signal.

# 4

# Experiments

## 4.1 Methodology

The approach used in this thesis was created to handle the difficulties of repairing broken audio files from vintage movies without sacrificing their authenticity. After careful consideration, a number of important GAN models and techniques were chosen for further consideration due to their aptitude for managing the complicated nature of audio data, especially when combined with spectrograms and MFCC representations.

**Reasons for Choosing These Models:**

1. **DCGAN (Deep Convolutional GAN)**: Because of its capacity to manage high-dimensional, structured data such as spectrograms, the DCGAN model was selected as the foundation. Using convolutional layers that are well-suited for 2D data, DCGAN preserves the spectrogram-represented audio's time-frequency structure. Its ability to produce clean, well-organized outputs and its consistency during training made it a sensible place to start when reconstructing audio from spectral data.

**Reasons for Not Picking Other Options**

Although fully connected layers or conventional GANs might have been utilized, they don't have the spatial coherence required for data that resembles images, like spectrograms.

The spatial dependencies in time-frequency representations would be difficult for fully connected networks to handle, which is why DCGAN's convolutional method is so much better.

**2.SpecGAN:**: Because of its emphasis on producing spectrograms of superior quality, SpecGAN was chosen. A compressed but relevant audio representation can be worked on by the model thanks to spectrograms, which depict sound in the time-frequency domain. Because SpecGAN works directly with spectrograms and has been shown to be successful in producing sound using spectrogram synthesis, it was especially useful in our project.

**Reasons for Choosing These Models**

WaveGAN, an option that produces raw waveforms directly, may have been used. However, because raw audio data is highly dimensional, creating waveforms takes significantly larger models and longer training cycles. Conversely, SpecGAN preserves important audio features while operating with a more reduced format—a spectrogram.

**3.Combination of DCGAN and SpecGAN:**These two models were chosen to be combined because of their unique advantages. The combination of SpecGAN's emphasis on spectrograms and DCGAN's architectural stability results in a potent hybrid method for reconstructing damaged audio recordings while keeping noise and other original sound characteristics. The goal was to increase the quality of the reconstruction by inserting missing elements while preserving important time-frequency features by utilizing both models.

**"Reasons for Not Just One Model"**

Although either DCGAN or SpecGAN alone could function effectively, their combination guarantees that we take advantage of DCGAN's stability in output creation while concentrating on high-fidelity sound reconstruction with SpecGAN's spectrogram-based generation approach. The goal of the integrated design is to offer a more reliable system that can manage different noise situations without compromising the underlying audio.

**4.MelGAN:** MelGAN was investigated because of its effective convolutional designs that produce high-quality audio. Mel-spectrograms, which are strongly linked to the perceptual characteristics of audio, are used by MelGAN to function in the time-frequency domain.

MelGAN's ability to work with mel-spectrogram representations proved to be crucial in the following trials when they switched to MFCC (Mel-Frequency Cepstral Coefficients).

**Reasons for Choosing These Models**

WaveGAN was taken into consideration but rejected because of its greater computing cost and lengthier training times. WaveGAN works with unprocessed audio data. MelGAN proved more effective in this specific challenge because of its capacity to generate high-quality audio from compressed representations such as MFCC.

**The Aim of the Chosen Architectures:** The fundamental difficulty of recreating audio while striking a balance between undoing damage and keeping original features was the driving force for the selection and combination of these particular architectures. The models have to be able to replace any missing or damaged sections while maintaining the important elements of the original sound. This was essential to preserving the authenticity of the audio, particularly for historical audio recordings when flaws or background noise are inherent to the original character. When tackling various kinds of noise and audio distortion, the combination of DCGAN and SpecGAN, along with additional research into MFCC preprocessing and the application of MelGAN, allowed for greater flexibility and improved outcomes.

In conclusion, these models were selected and merged from experiment to experiment because of their complimentary qualities, which balanced the requirements for high-fidelity audio production, effective spectrogram processing, and rapid computing.

### 4.1.1 Explanation of the Training Code for DCGAN

The fundamental training protocol for a Deep Convolutional GAN (DCGAN) that processes and produces audio spectrograms is implemented by this training code. We will dissect each portion of the code and explain its purpose as well as why it is crucial to a typical GAN training pipeline below.

#### 1. Model and Data Initialization

The first step in the training process is to initialize the models (generator and discriminator) and load the data:

**Dataloader:** The dataloader is in charge of producing clean, noisy spectrograms in batches. These spectrograms are divided into smaller segments; the generator receives the noisy versions as input, while the discriminator uses the clean versions as the actual data.

**Generator and Discriminator Initialization:** Models for the discriminator and generator are instantiated. Here, the models are populated using default values, although the code has the ability to apply specific weight initialization. The discriminator will determine whether a particular spectrogram is real (from the dataset) or fake (produced by the generator). The generator will be in charge of creating clean spectrograms from noisy input.

**Device Setup:** For training the models, mostly RTX 3090 GPU is used. Using a high-performance GPU like this significantly speeds up training, especially for deep models with large datasets like this one.

Loss Functions and Optimizers:

In a GAN, two loss functions are used: one for the Discriminator and one or both for the Generator model to model.

**BCEWithLogitsLoss:** The raw output from the discriminator, or logits, are used to apply this binary cross-entropy loss. Both the discriminator and the generator are trained with it. The generated and genuine data are compared to their corresponding labels (real = 1, fake = 0) using the discriminator. The generator's objective is to deceive the discriminator into believing that the created data is real.

**MSELoss:** At some models, the generator's training also includes the MSE (Mean Squared Error) loss in addition to the adversarial loss (BCE). This makes it more likely that the spectrograms produced will not only mislead the discriminator, but will also prevent the generator from attempting to make small adjustments to the damaged data it receives before rebuilding it to match the original data.

**Adam Optimizer:** Because of its adjustable learning rate properties, Adam is a well-liked optimizer for GAN training and is used for both the generator and discriminator. It makes use of two parameters, betas (0.5 and 0.999), to regulate the optimizer's momentum and improve its ability to tolerate noisy gradients. The results obtained by training morels have led to adjustments in learning rates. For the purpose of improving the model train

or fine-tuning the model, learning rate experiments have been conducted on the majority of trained models.

3. Training Loop:

The training loop is the main portion of the code and it runs for a predetermined number of epochs. The model processes a batch of spectrograms from the dataloader for every epoch.
**Training the Discriminator**

**BCEWithLogitsLoss:**The discriminator's goal is to correctly distinguish between real and fake spectrograms. The training proceeds in two steps:

**Step 1: Real Data:** There are real spectrograms (clean audio) for the discriminator. In this phase, the target labels are 1, indicating that the data is authentic. Based on how successfully it classifies this actual data as real, the discriminator computes a loss (d loss real).

**Step 2: Fake Data:** The generator creates simulated spectrograms by using noisy spectrograms as input. These bogus spectrograms are then submitted to the discriminator, which tries to classify them as fake (target label = 0). The discriminator's ability to recognize created data as fake can be seen in the step's loss, or d loss fake.

**Discriminator Loss:** The total discriminator loss is the sum of the real and fake losses (d loss = d loss real + d loss fake). This loss is used to update the weights of the discriminator. The discriminator needs to develop the ability to distinguish between legitimate and fraudulent data with accuracy.

**Training the Generator:** The generator wants to produce spectrograms that are convincing enough to trick the discriminator into thinking they are real.

**Adversarial Loss (BCE):**The level to which the generator can trick the discriminator determines how much of a loss it has. It creates spectrograms from noisy input, which are then classified as fake by the discriminator. In essence, the generator wants to reduce the binary cross-entropy loss such that the discriminator will mistakenly identify fake spectrograms as real.

**Content Loss (MSE):** An MSE loss is introduced to guarantee that the generated spectrograms are near in structure to real spectrograms and good enough to trick the discriminator. This loss reduces the pixel-wise difference between the generated and real spectrograms from the dataset.

**Combined Loss:** The adversarial loss and the content loss add up to the overall generator loss. The generator is guaranteed to concentrate on creating spectrograms that closely resemble real audio, rather than just tricking the discriminator, because to the high weighting (100x) for the MSE.

**Multiple Updates for Generator:** Each batch, the generator receives many training sessions. In situations where the discriminator is already effective in distinguishing between real and fake data, this aids in the generator's improvement more quickly.

4. Logging and Saving:

**Logging:** The generator and discriminator losses are recorded at the end of each epoch. Gaining insight into the learning capacities of both networks is made possible by this knowledge. In order to prevent mode collapse or weak convergence, it is important to keep an eye on the performance of both the discriminator and the generator.

**Model Saving:** The generator and discriminator's current weights are periodically recorded to disk. Model checkpoints and the option to resume training from a saved state if needed are made possible by this. Torch.save is used to save the models and holds the network weights as of right now.

In conclusion, the DCGAN's training procedure is similar to the standard GAN paradigm in that the discriminator and generator engage in competition with one another[19] [20]. The generator is trained to create realistic-looking spectrograms from damaged sound files in order to trick the discriminator, while the discriminator is trained to discriminate between actual and reconstructed spectrograms. The generator generates high-quality outputs by combining binary cross-entropy loss for adversarial training and occasionally mean squared error loss for spectrogram quality. The Adam optimizer, which offers consistent and effective updates to the model parameters, is used to optimize the training sequence, which is created to balance the learning of both networks.

## 4.2 Experiments

In the first section, we have used all the damaged datasets which are 150, 250, 500,1000,1500, 2000, and 3000Hz cut versions of the white noise applied dataset for the generator. Lastly, in all sections, the Discriminator used only white noise and applied uncut sound.

### 4.2.1 Experiment 1

**Used Data** At the first experiments, a complete model that can understand all of the 7-cut versions of the sound aimed, therefore as the dataset all of the 7-cut versions has been used.

**Pre-Process of the Dataset:**As the first experiment, the data was segmented into 2-second intervals. The Short-Time Fourier Transform (STFT) was applied to the segmented sound files, creating spectrograms for each segment. The resulting spectrograms, with dimensions of (2, 1025, 188), were processed to be suitable for feeding into the DCGAN. Specifically, the real and imaginary parts of the spectrogram were separated, and the imaginary numbers were transformed into real numbers. These two channels were then combined into 2D images with dimensions of (2, 1025, 188), effectively creating 2-channel images that could be fed into the DCGAN. From a 126-second sound sample, each divided into 2-second segments, a total of 63 segments were created per sound file.

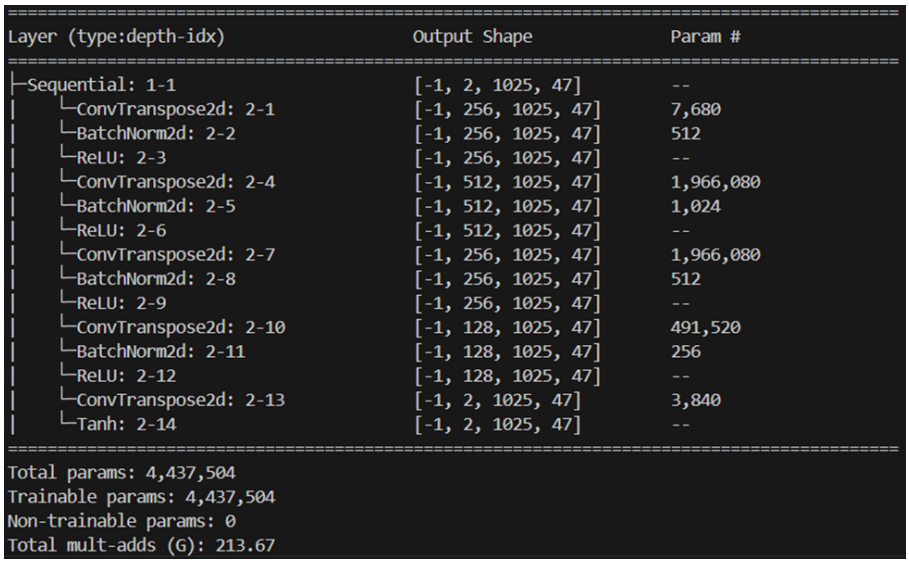
To properly train the model, the clean and noisy data needed to be paired consistently. The clean dataset contained one version of the original audio file, while the noisy dataset included seven distinct variations of the same file, each altered by different noise conditions. This setup was crucial to ensure that the generator could learn to reconstruct the original sound from various types of noise.

Since the generator needed to handle multiple noisy versions of the same clean audio file, the pairing mechanism was designed so that the same clean audio sample was repeated across different noise conditions. Each noisy sample was paired with the original clean version, allowing the generator to learn from multiple noisy examples while always comparing its generated output to the same clean target. As a result, with 63 segments per audio sample and seven noisy variations for each, a total of 441 paired examples were created in the dataset, significantly enhancing the learning capacity of the model.

**Structure:** The generator focused on rebuilding missing or damaged frequencies while maintaining the undistorted portions of the input audio through the application of the ReLU activation function. ReLU encourages the generator to regenerate the cut-out or noisy frequencies while preventing it from affecting the audio's clear portions. It does this by outputting the input directly if it is positive and zero otherwise. In the convolutional layers, higher feature

counts have been used to improve the generator's understanding.

Normally ReLU activation function is not used in the DCGAN's structure like this structure, But to preserve the feeded frequencies, and just to make the Generator reconstructs the missing frequencies addition to feeded frequencies, ReLU activation function has been used. Because while training, it have a big possibility to overwrite the feeded frequencies, and try to learn those frequencies by itself with training.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 2, 1025, 47]         --
|    └─ConvTranspose2d: 2-1              [-1, 256, 1025, 47]       7,680
|    └─BatchNorm2d: 2-2                  [-1, 256, 1025, 47]       512
|    └─ReLU: 2-3                         [-1, 256, 1025, 47]       --
|    └─ConvTranspose2d: 2-4              [-1, 512, 1025, 47]       1,966,080
|    └─BatchNorm2d: 2-5                  [-1, 512, 1025, 47]       1,024
|    └─ReLU: 2-6                         [-1, 512, 1025, 47]       --
|    └─ConvTranspose2d: 2-7              [-1, 256, 1025, 47]       1,966,080
|    └─BatchNorm2d: 2-8                  [-1, 256, 1025, 47]       512
|    └─ReLU: 2-9                         [-1, 256, 1025, 47]       --
|    └─ConvTranspose2d: 2-10             [-1, 128, 1025, 47]       491,520
|    └─BatchNorm2d: 2-11                 [-1, 128, 1025, 47]       256
|    └─ReLU: 2-12                        [-1, 128, 1025, 47]       --
|    └─ConvTranspose2d: 2-13             [-1, 2, 1025, 47]         3,840
|    └─Tanh: 2-14                        [-1, 2, 1025, 47]         --
==========================================================================================
Total params: 4,437,504
Trainable params: 4,437,504
Non-trainable params: 0
Total mult-adds (G): 213.67
```

**Figure 4.1:** Structure of Genererator

• The LeakyReLU activation function, which has a standard negative slope, is utilized by the Discriminator. This option allows the Generator to receive more feedback from the Discriminator by permitting modest negative values in the gradient. As opposed to the conventional ReLU, which completely eliminates any negative input, LeakyReLU allows the Discriminator to identify even minute errors made by the Generator. This feedback system corrects faults that might otherwise go unnoticed if only positive gradients were taken into account, assisting the Generator in learning more efficiently.

• ReLU activation function has been not used in the Discriminator because the model still should be able to learn the negative resulting changes in the generated spectrograms.

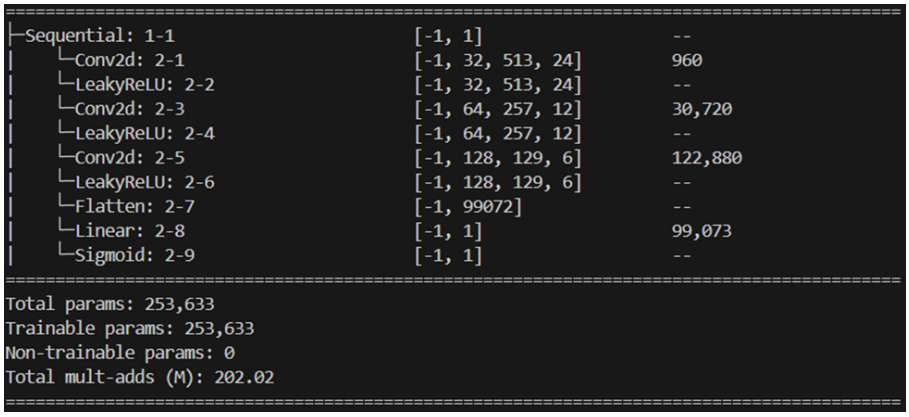
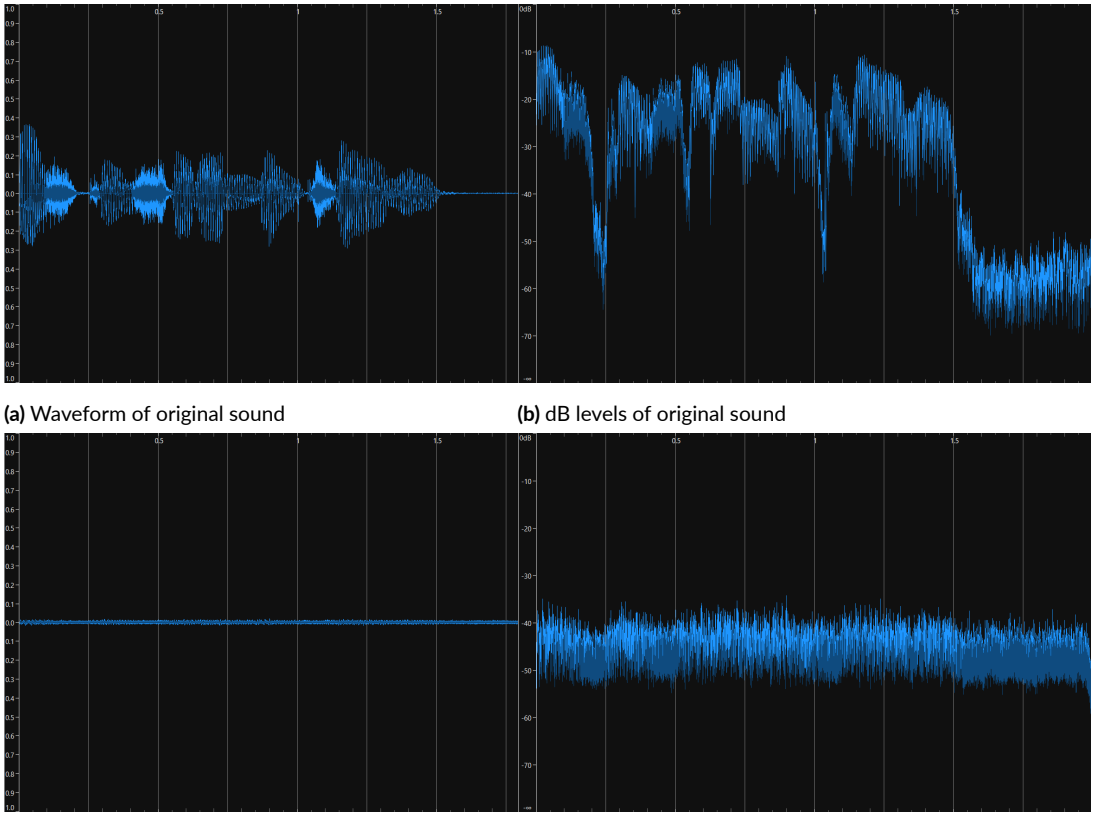



**Figure 4.2:** Structure of Genererator

• Adam Optimizer has been used for both Generator and Discriminator, with a learning rate of 0,002 at the start.
• Both Discriminator and Generator used Binary Cross Entropy with Logits loss (BCEWithLogitsLoss).
• Each of the Generator and Discriminator have been trained once in every iteration
• The DCGAN has been trained for 50 Epochs
• No dropouts were used in this first experiment to see the balance of the network.
• **Training**: During the training phase, the model checkpoints were saved every 10 epochs to not lose any pieces of training. But in this experiment, the Convergence did not appear and the Discriminator totally overpowered the Generator in every aspect. Also, the balance may not achieved because of the layer counts of the Generator and Discriminator.
• Also in the results, it can be seen that the Generator generates the output while maxing out or mining out almost every output. This model creates 2 channel spectrograms, which it max out all the values in one channel, and min out all the values in the other channe.. And lastly, a clear decrese in dB levels are visible in the results.

By looking at the waveform and dB level results, it was clear that we could do different approaches first to make the Discriminator not overpower the generator and create the balance between.



**(a)** Waveform of original sound

**(b)** dB levels of original sound

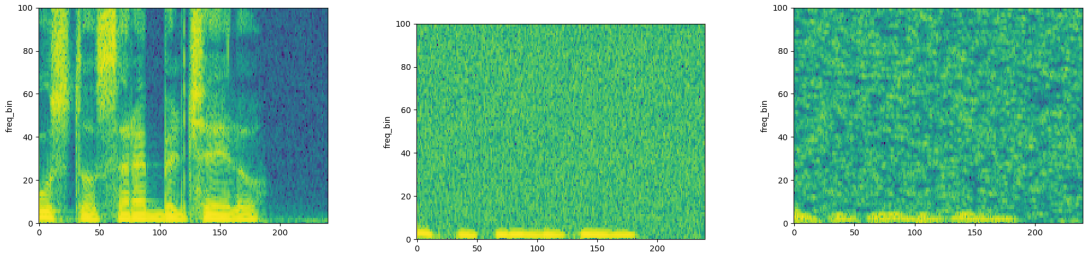
**(c)** 50 epoch result:Waveform

**(d)** 50 epoch result:dB levels

**Figure 4.3:** Original and Cut sound's Waveforms and dB levels

From the look of the results in figure 4.3, it can be said that training completely minimized the waveform values, and db levels almost flattened out.

Also from the spectrograms of the results in Figure 4.4, It can be seen that the network still tries to understand the fed data, but it does not understand the fed data fully yet.




**(a)** 50 epoch: Original Sound Spectrogram

**(b)** 50 epoch:Cut Sound Spectrogram

**(c)** 50 epoch: Reconstructed Sound File Spectrogram

**Figure 4.4:** 50 epoch model Spectrograms

**Negative Results:**

- Generated output channels either maximized or minimized
- Training was unbalanced
- Each Epoch took too much time to train, and the modes is not feasible
- A clear decrease in dB levels of the results can be seen.

### 4.2.2 Experiment 2

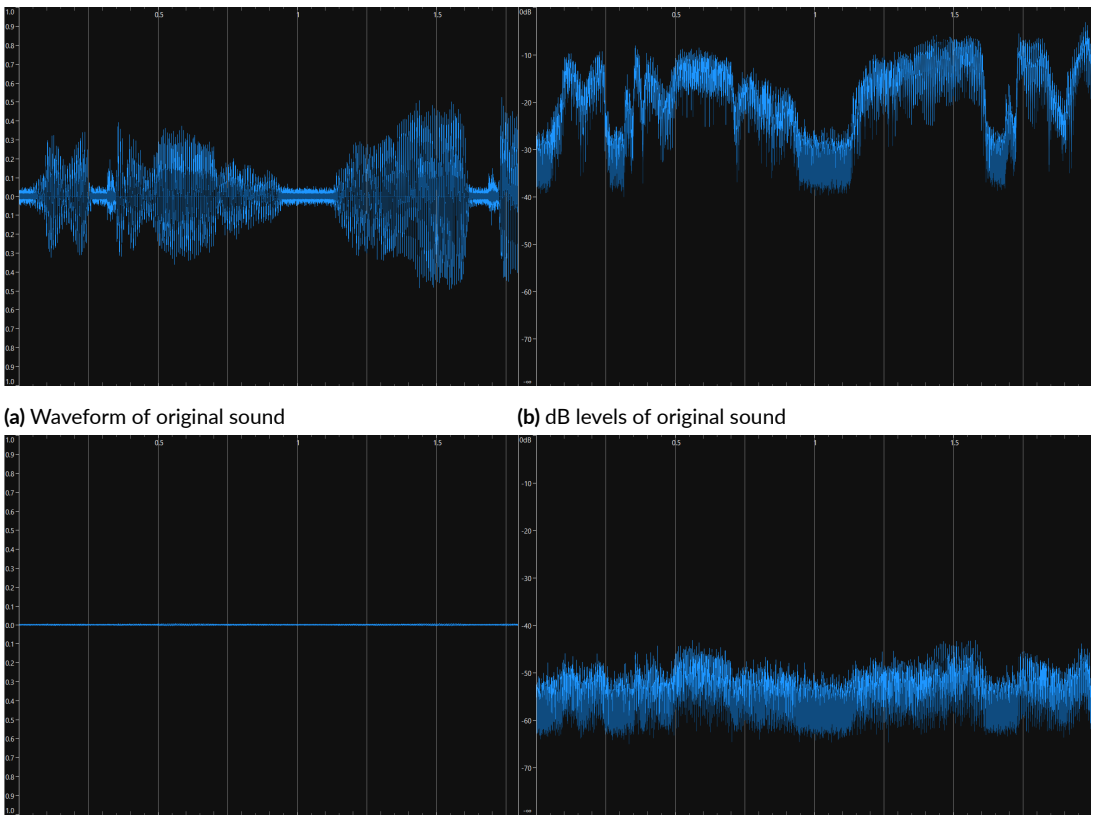
The same structure for both the Generator and Discriminator, the same pre-process, and the same segment lengths were used in the first experiment. This experiment aimed to balance the network training in the first place.

After some investigation of the dataset, it could be clearly seen that we have pre-processed each 7 of the cut sound files which were processed to feed the Generator and segment it. Still, it also feeds to train the network by matching segments with the segments created from original data corresponding to its sections. This means that we are feeding each of the 7 cut sound files to the Generator, and feeding the original sound to the Discriminator 7 times.

To overcome this, in the second experiment, the learning rates have been arranged differently than each other. The learning rate of the Discriminator has been set to 0.00007 and Generator's learning rate has been set to 0.00021. As a result, there is a 3 times difference between the learning rates. This has been made to prevent the Discriminator from overpowering the

generator. However, this change in the learning rates will probably cause some problems to occur in future training at the fine-tuning step.

With this change in the Learning rates, it was possible for us to train the network for 500 epochs but around the 500 epoch line, the Discriminator again started to overpower the Generator and cause mode collapse. Therefore Discriminator overpower the generator, and the results do not improve with training. The results were close to the first experiment. The Generator maxes out the results again. The waveform is not even similar to the original result, but it gets smaller. Also, we can say the same with the dB levels too. DB of the generated sounds getting smaller, smaller than the first experiment. It can be thought from this, that the dB levels decrease with each training.



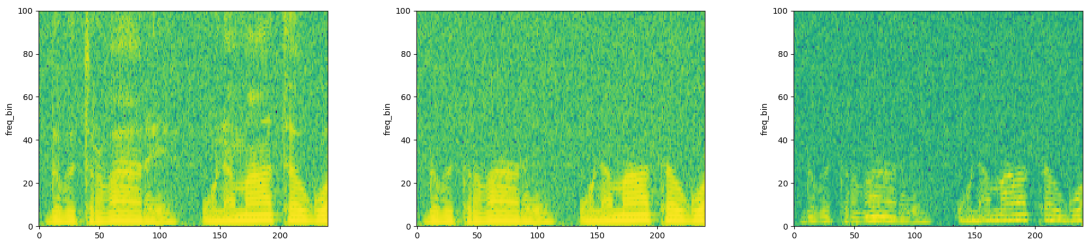
**(a)** Waveform of original sound

**(b)** dB levels of original sound

**(c)** 500 epoch result: Waveform

**(d)** 500 epoch result: dB levels

**Figure 4.5:** Original and Cut sound's Waveforms and dB levels




**(a)** 500 epoch result:Original Sound Spectrogram

**(b)** 500 epoch result:Cut Sound Spectrogram

**(c)** 500 epoch result:Reconstructed Sound File Spectrogram

**Figure 4.6:** 500 epoch result:Spectrograms

• In the figure 4.5, it is visible that the waveform is flattened again like experiment 1. But this time dB levels are not flattened as the experiment 1. In results dB levels, it can be said that there are some improvements. But from looking at the levels of dB, it is clear with more training, the dB levels are decreasing.

• According to the figure 4.6 (c) which is a bit vanished version of the fed data which is figure 4.6(b), It can be thought that generator still tries to understand the fed data before it can start to generate the missing frequencies as the original version figure 4.6(a).

**Adopted Strategies:**

• According to the prepared dataset, the learning rates of the both discriminator and generator have changed

**Negative Results:**

• generated outputs channels either maximized or minimized again

• Training was unbalanced after around epoch 450

• Using a higher learning rate fixed the balance problem for only 450 epoch

• More training decreases the dB levels

**Positive Results:**

• dB levels are not flattened like first experiment, the dB is still showing similar features as the original sound's dB visually.

### 4.2.3 Experiment 3

At this point it was obvious the Balance between the Generator and Discriminator was not the cause of the learning rates or the datasets. Making the Learning Rate of the Generator 7 times higher than the Discriminator's learning rate could still be made because the Discriminator has been learning the original file 7 times, while generator learns the all uncut files 1 time. It could be logical to the ear but this big difference between the learning rates will make the training unbalanced. After some point, when it comes to fine-tuning the results, as result of of Discriminator's learning rate is much smaller than the Generator's learning rate, the Discriminator will learn the data with more details. But the Generator understands it is not as detailed as the Discriminator. This means that the Fine tuning of the generations will be unbalanced for sure.

In the training phase, label smoothing [21],[22] has been utilized to address the imbalance issue. Label Smoothing is a method that modifies the target labels only a little bit in order to stabilize GAN training. It assigns 0.9 for actual data and 0.1 for fake data, rather than explicit labels like 1 for real data and 0 for fake data. This keeps the discriminator from over-confidence, resulting in more stable training and smoother gradients.

Also the Max out and Min out problems have not been solved. However, these problems are not the result of the balance between the Generator and Discriminator training. It can be thought that way because some results that are not minimized or maximized should be seen after the 450 epochs as done in the previous experiment. As a result, the Structure of the Discriminator and Generator was checked again. Changes in the activation functions of the generator structure have been decided. The Relu activation function has been changed with the Leaky ReLU activation function
. And lastly because the data we are using is big, for example, each segment's dimensions are (2,1025,188), and our Generator and Discriminator structures contain 5 layers, decrease of the layer amount and the features of the layers have been decided. To make the training more feasible, and to see a result much faster layer amount and the feature amount in the layers have been decreased.

The generator's 5 layers decreased to 4 layers. Also, the features on the layers decreased too. As a result, 4 convolutional layers have been used on the generator, with a change of Activation functions from ReLU to LeaykReLU with a standard negative slope. This change has

been made to make the training more feasible and to prevent the generator results from getting min out or max out.

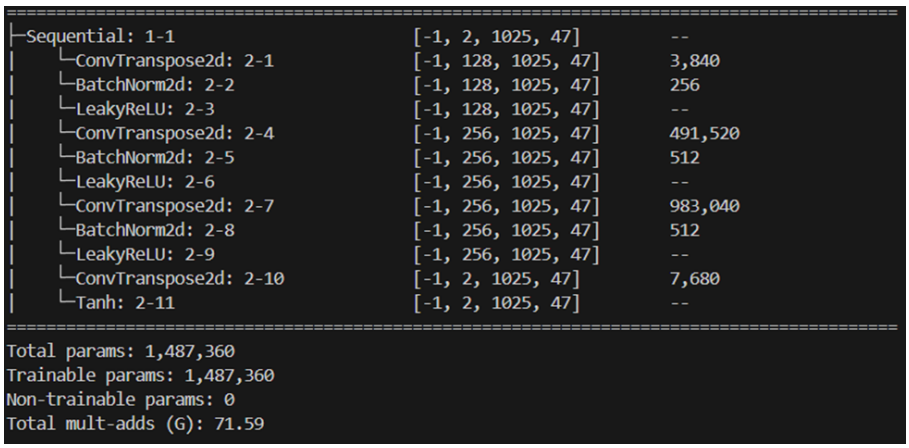



**Figure 4.7:** New Structure of Genererator

The discriminator's layer count has not been changed. But to prevent overpowering the generator, a dropout rate of 0.25 was added for each layer of the Discriminator.

```
==========================================================================================
├─Sequential: 1-1                        [-1, 1]                   --
|    └─Conv2d: 2-1                       [-1, 64, 513, 24]         1,920
|    └─LeakyReLU: 2-2                    [-1, 64, 513, 24]         --
|    └─Dropout: 2-3                      [-1, 64, 513, 24]         --
|    └─Conv2d: 2-4                       [-1, 128, 257, 12]        122,880
|    └─LeakyReLU: 2-5                    [-1, 128, 257, 12]        --
|    └─Dropout: 2-6                      [-1, 128, 257, 12]        --
|    └─Conv2d: 2-7                       [-1, 128, 129, 6]         245,760
|    └─LeakyReLU: 2-8                    [-1, 128, 129, 6]         --
|    └─Dropout: 2-9                      [-1, 128, 129, 6]         --
|    └─Flatten: 2-10                     [-1, 99072]               --
|    └─Linear: 2-11                      [-1, 1]                   99,073
==========================================================================================
Total params: 469,633
Trainable params: 469,633
Non-trainable params: 0
Total mult-adds (M): 593.39
```

**Figure 4.8:** New Structure of Discriminator

• 220 epochs of training have been done with this structure. The training was balanced. Also from the loss plot below, it can be seen the convergence. It can also be seen that the Generator tries new strategies to deceive the discriminator. And as result, It can be thought that the label smoothing worked to balance the training.

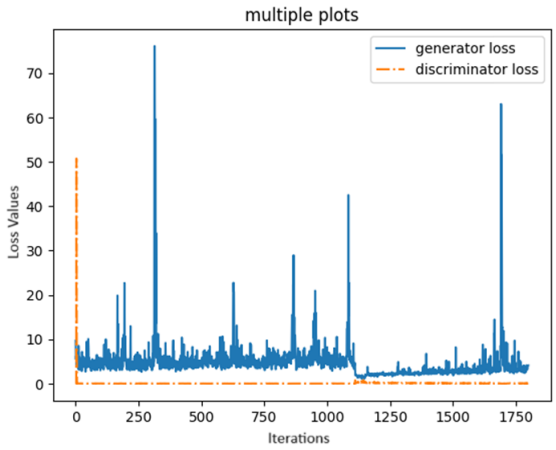
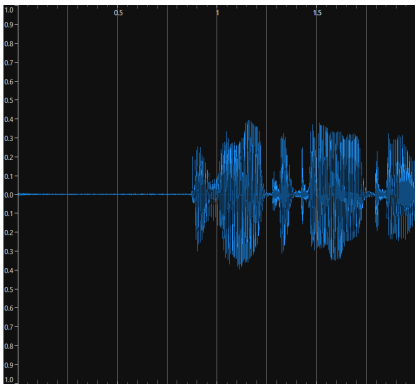



**Figure 4.9:** Loss Values of Generator and Discriminator

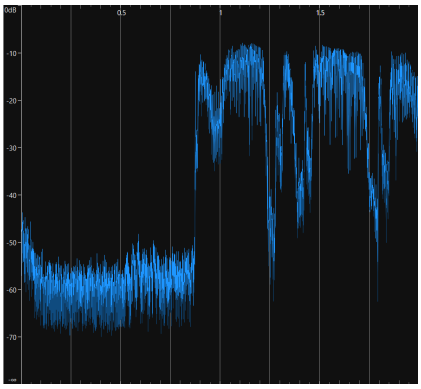
• Generated results have been checked at the 70 epoch, and seen that the results are not min out or max out anymore.

• 70 epoch, 120 epoch, 145 epoch, and 220 epoch results have been saved to see the results. The dB levels decrease with each trained epoch.

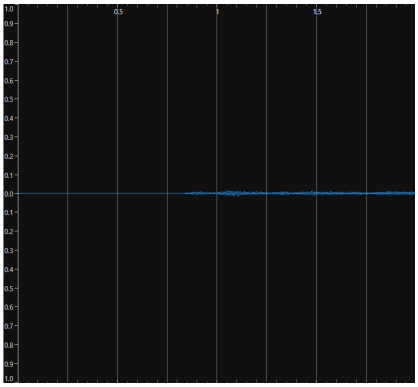
• The most clear result was created at 70 epochs.



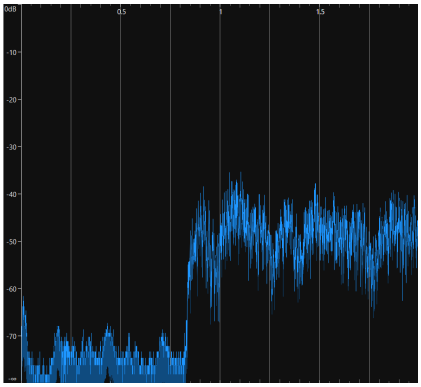
**(a)** Waveform of original sound



**(b)** dB levels of original sound

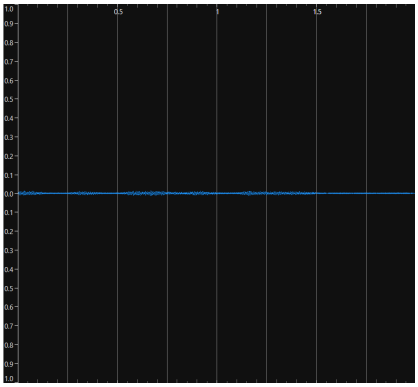


**(c)** 70 epoch results: Waveform of reconstructed sound

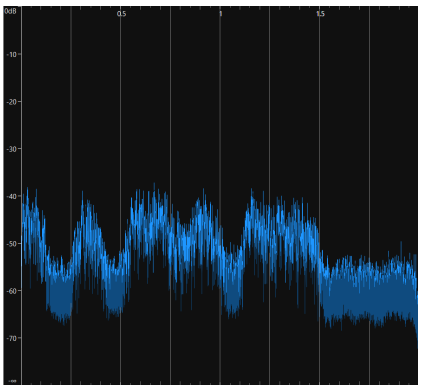


**(d)** 70 epoch results:dB levels of reconstructed sound

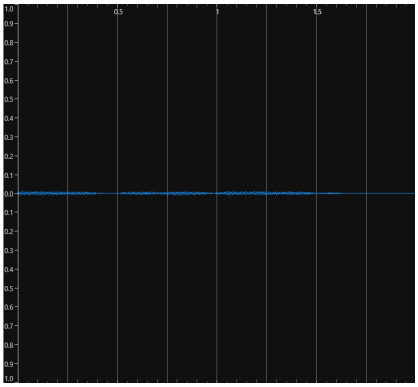
**Figure 4.10:** Original and 70 epoch result's Waveforms and dB levels



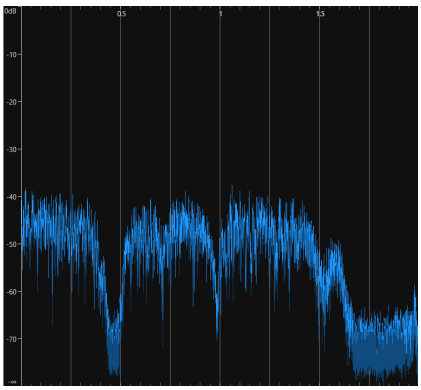
**(a)** 120 epoch results: Waveform of reconstructed sound



**(b)** 120 epoch results:dB levels of reconstructed sound



**(c)** 220 epoch results: Waveform of reconstructed sound

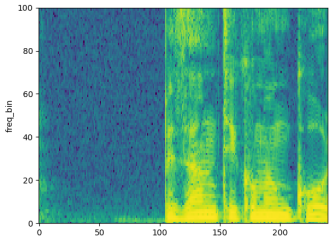


**(d)** 220 epoch results:dB levels of reconstructed sound

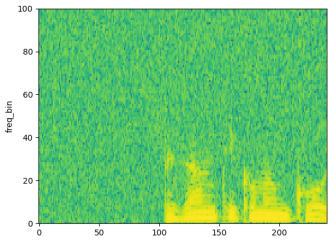
**Figure 4.11:** 120 and 220 epoch results: reconstructed sound's Waveforms and dB levels

• From the dB levels(figure 4.10 and 4.11) and waveforms, it can be seen the results looks better if it compared with the previous experiments. The dB levels are still lower, but the features are not flattened out this time, visually it can be seen. But in the waveform side, it is still looking like flattened at all of them.

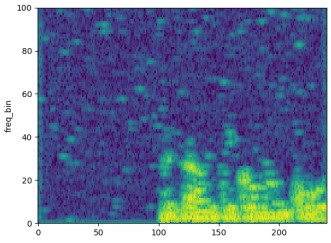
Sadly, from the results of this experiment, we can say with more training, the cut version of the sounds that we feed to the generator not learning. The generator should understand that the fed data already is true at some point, it should just generate the missing parts of the fed data. But from the results we can say the fed data is disappearing with more trains. 70 epoch trained result should not be the best result. However according to the spectrograms in Figures 4.12,4.13 and 4.14, the best looking, and the most visible frequencies appear in 70 epoch training in Figure 4.12 (c). More training did not make the base frequencies clearer as it seems in Figure 4.13 (c) and 4.14 (c).




**(a)** 70 epoch: Original Sound Spectrogram

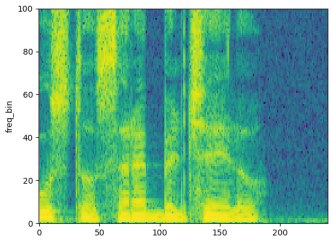



**(b)** 70 epoch:Cut Sound Spectrogram

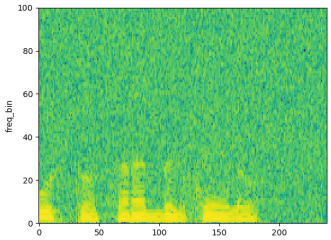



**(c)** 70 epoch:Reconstructed Sound Spectrogram

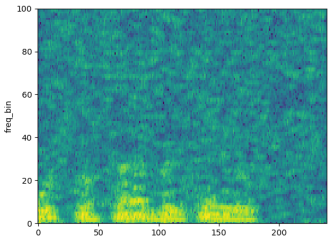
**Figure 4.12:** 70 epoch Spectrograms




**(a)** 120 epoch: Original Sound Spectrogram

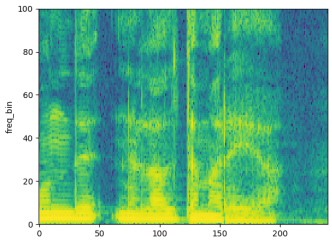



**(b)** 120 epoch: Cut Sound Spectrogram

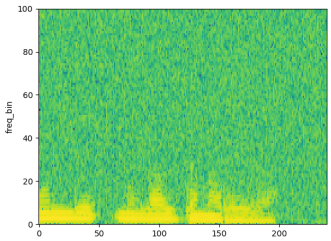



**(c)** 120 epoch: Reconstructed Sound Spectrogram

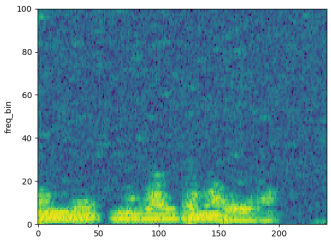
**Figure 4.13:** 120 epoch Spectrograms




**(a)** 220 epoch: Original Sound Spectrogram




**(b)** 220 epoch: Cut Sound Spectrogram




**(c)** 220 epoch: Reconstructed Sound Spectrogram

**Figure 4.14:** 220 epoch Spectrograms

**Adopted Strategies:**

- Label smoothing used to balance the train
- Learning Rates kept the same as Experiment 2
- For feasibility, for time and hardware, the layer count of the Generator, and those layer's feature (neuron) counts decreased
- To prevent minimizing or maximizing the values, Generator's Activation Functions changed from ReLU to LeakyReLU with standard negative slope

**Positive Results:**

- From the spectrograms, It can be seen the generator starts to learn at least the fed data to itself, but not the way we wanted.
- The training balanced
- More features are visible in dB levels.

**Negative Results:**

- With each training the waveform gets smaller, and dB levels decrease like previous experiments
- No new frequencies generated to acquire the original result, the base frequencies which we fed to Generator disappears with more training.
- Network training in balance but after some point in the training, it cannot be learned further. It could be seen if the results of epoch 120, and the 220 compared(figure 4.13(c) and 4.14(c)).

### 4.2.4 Experiment 4

The same pre-process is used, and the same dataset to feed the Generator and Discriminator have been used on this experiment. As seen, after the 70th epoch, the generator could not learn better and as a result, generated results started to break down. This result can be divided into 2 possibilities:

- The first possibility is that after some point the label smoothing prevented the Discriminator learn the original data properly. As a result of this, after the 70th epoch, discriminators did not have any improvements for the generator. Therefore, because the Discriminator loss is not changing anymore, the Generator tries to keep learning more and still get the same results from the discriminator. This causes the Generator to start breaking down the learned ways of generating data and disturb the results.

- And the second possibility was the Learning rate difference caused this problem.(the generator's learning rate was 3 times of the Discriminators) Because the learning rate of the Generator is 3 times higher than the Discriminator's, this big learning rate difference results in the

Discriminator's ability to capture more features than the Generator's. This causes that after some point of the training, it is getting harder or impossible for the Generator to find and use new features like the Discriminator,

As a result of these possibilities, the difference between the learning rates of the Generator and Discriminator equalized. And both of the Learning Rates have been set to 0.0002. Also to see if the Label Smoothing was the problem, the label smoothing has been canceled. These two changes in the Training step will show us if the result of Experiment 3 was caused by them.

As a result of these changes, some strategies should be used at the Training step. Because canceling the Learning Rate difference between the Generator and Discriminator and also canceling Label Smoothing will cause the balance to be disturbed again. The mod collapse should not happen, therefore a new strategy was added to the Training step to keep balancing the Training.

As an alternative training strategy for the Generator to be trained multiple times for each training step of the Discriminator has been implemented. This approach was employed to maintain the balance between the Generator and the Discriminator, especially as the Discriminator tended to learn faster and overpower the Generator. By training the Generator more frequently within each iteration, the Generator is given more opportunities to adjust and improve its outputs, thus preventing the Discriminator from dominating the training process. This strategy aims to ensure that the Generator remains competitive, enabling it to produce better reconstructions and gradually close the gap in performance between the two networks.

The Training step has been changing according to this strategy. For each iteration, while the Discriminator trained once, the Generator Trained twice. In the end, the difference between the learning rates of the Generator and Discriminator occurred again but in a different way which will not affect the results, or there will be no difference in how much detail they will learn at each iteration.

Lastly, as a result of the new Training Step Result, the Dropout rates of the Discriminator have been canceled. To see how the new Training Strategy affects the Training balance this has been made.

- The Training has been made for 1100 Epoch

- 320 epoch, 700 epoch, and 1100 epoch results have been saved to see the results. The Db levels decrease with each trained epoch again.

- The Training plot looks pretty solid and balanced, but the actual values show after some point, the discriminator will start to overpower the Generator again.

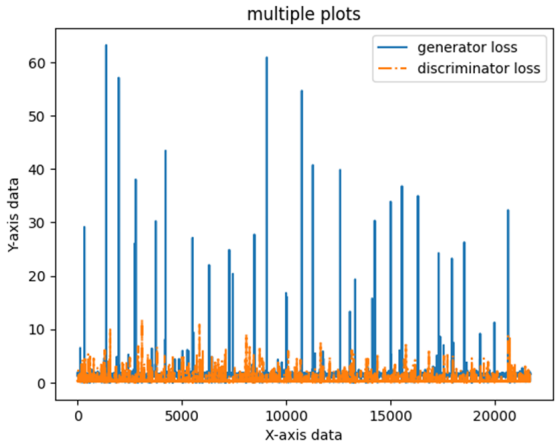
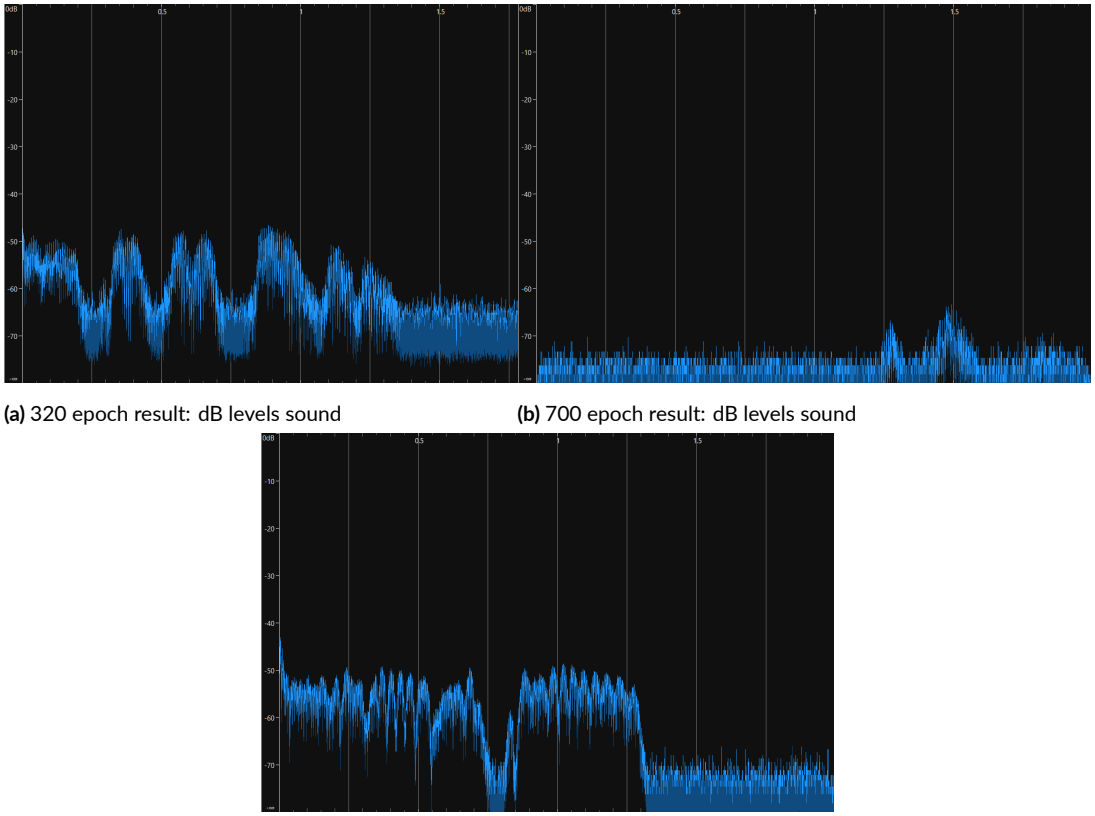



**Figure 4.15:** Exp4 Balanced Training

The results were not much different from Experiment 3's results, especially the reconstructed waveforms are flattened and look almost the same as Experiment 3. From Figure 4.16, it can be seen that 320 epoch results have higher dB levels than 700, but 1100 epoch result is almost the same as 320 epoch results in dB levels. from this, it can be said that the model is learning the base for some epochs, and until the point it learns, the results decrease in quality, but after it increases the quality again. An example of this can be seen in Figure 4.16 (a) to (c).

On the other hand, if we look at the spectrograms in Figures 4.17, 4.18, and 4.19, we can see that our analysis of the dB levels is accurate. The 320 epoch training result has a good baseline figure 4.17 (c) because it did not have much time to modify the baseline. The result of 700 epoch training in Figure 4.18 (c), shows visible the model is trying to modify the baseline because the baseline is half vanished. And the figure 4.19(c) it is visible that the model is more confident about the baseline there.



**(a)** 320 epoch result: dB levels sound

**(b)** 700 epoch result: dB levels sound

**(c)** 1100 epoch result: dB levels sound

**Figure 4.16:** Original and Cut sound's Waveforms and dB levels

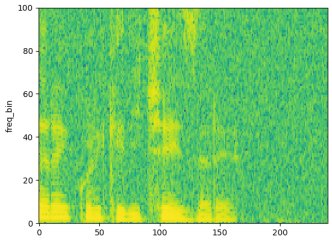
### Adopted Strategies:

- Label Smoothing Cancelled.

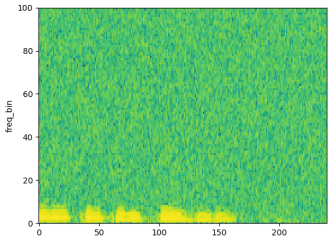
- Learning rate of the Both Generator and Discriminator have been set to 0.0002

- To avoid disturbing the training balance, an alternative training strategy was implemented. In each iteration, the Generator will trained multiple times, but the Discriminator will train once.

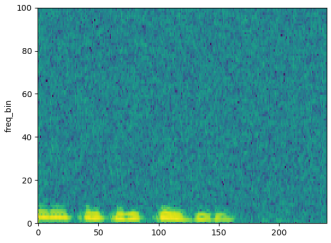
- Dropout rates in the Discriminator Structure have been cancelled to see the effects of new




**(a)** 320 epoch result: Original Sound Spectrogram

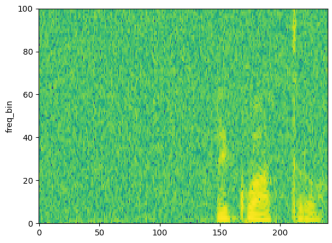



**(b)** 320 epoch result: Cut Sound Spectrogram

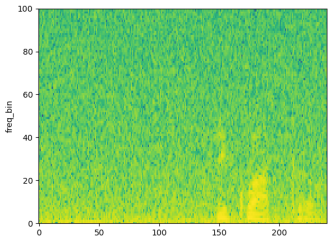



**(c)** 320 epoch result: Reconstructed Sound Spectrogram

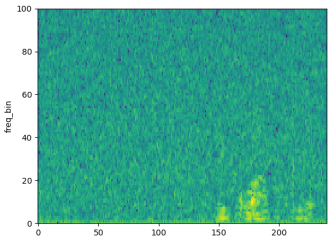
**Figure 4.17:** 320 epoch Spectrograms




**(a)** 700 epoch result: Original Sound Spectrogram

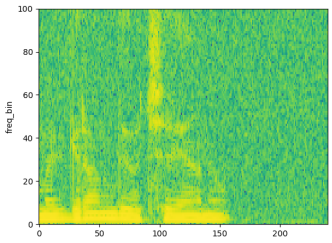



**(b)** 700 epoch result: Cut Sound Spectrogram

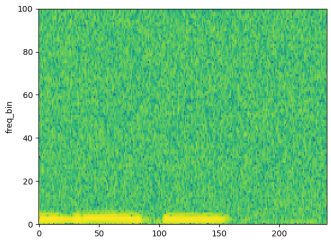



**(c)** 700 epoch result: Reconstructed Sound Spectrogram

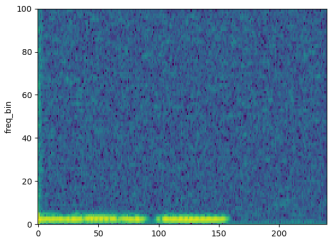
**Figure 4.18:** 700 epoch Spectrograms




**(a)** 1100 epoch result: Original Sound Spectrogram




**(b)** 1100 epoch result: Cut Sound Spectrogram




**(c)** 1100 epoch result: Reconstructed Sound Spectrogram

**Figure 4.19:** 1100 epoch Spectrograms

strategy for training step

**Positive Results:**

- Generator's ability to learn slightly improved, Generator learned the baseline.

- Frequencies look slightly more clear.

- canceling label smoothing has affected the results positively, but it also affects the balance. Therefore, it should be used to acquire training balance. The train-step strategy was not as effective as label smoothing to maintain balance completely, but it did a good job.

- The training was almost balanced, It was balanced at the end of the 1100 epoch.

**Negative Results:**

- With each training the waveform gets smaller, and dB levels decrease as Experiments 2 and 3

- No new frequencies were generated to acquire the original result as Experiment 2

- Balance of Training will be disturbed after some point. The discriminator overpowers the generator slowly but surely. It was not overpowering around the 1100 epoch, but in the losses, the linear increase with more training is visible.

### 4.2.5 Experiment 5

In this experiment, the methodology and pre-processing were modified in order to achieve the goal of seeing the working structure with the working train step as feasible as possible. Since the

dataset's two-second segments were used in the first four experiments, the DCGAN Structure was forced to work with data that had dimensions of (2,1025,188), meaning that each segment was as large as 360 to 360 RGB images. This may have caused the improvement to stall or required overly sensitive hyperparameter value adjustments. As a result, the segment lengths were changed from 2 seconds to 0.5 seconds each.

Additionally, this experiment focuses on a more focused training technique than the previous ones, which paired the clean audio sample with seven distinct noisy versions, each of which corresponded to the same part of the original sound. The clean audio had to be repeated seven times in the prior way, one for each noisy variant. This produced a dataset in which each of the seven different noisy cuts was associated with the clean sound. This experiment, on the other hand, pairs the clean audio with a single noisy counterpart because it only employs one particular noisy version. The generator may concentrate more intently on accurately recreating the original sound from this particular noise state thanks to this simplicity. The noisy variation uses the 2000Hz chopped version.

The generator structure was altered when the preprocess and dataset techniques were changed. The original architecture of the generator uses Leaky ReLU in its second and third layers, which allows a small gradient for negative inputs due to its constant negative slope. Leaky ReLU's linearity in the negative area makes it difficult to always extract finer information from complex data, such audio signals.

To address the problem, it was replaced by the Exponential Linear Unit (ELU), which offers a more non-linear and adaptive response. Because ELU smoothes the transitions for negative inputs, the generator might capture more subtle variations in the audio. This change should improve the generator's ability to reconstruct finer details from the audio inputs, resulting in outputs that are more accurate and of higher quality.

Tiny dropout layers have been added in between a few of the network's layers to improve the generator's results even further. By randomly deactivating a subset of neurons during training, dropout forces the network to acquire more robust characteristics and helps prevent overfitting. A small dropout in the generator encourages the model to generalize more effectively, which can result in more stable and high-quality output. By making this modification, the generator should be less likely to become unduly dependent on particular patterns, which should enable it to generate reconstructions of the audio data that are more accurate.

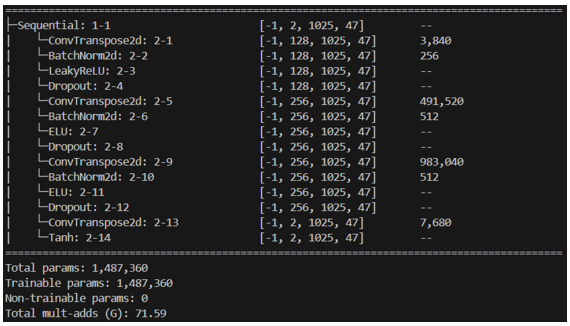



**Figure 4.20:** New Structure of the Generator

Finally, for discriminators, batch normalization has been added. For the discriminator, batch normalization is typically not desired. Because the logic of batch normalization aligns the results with the feed batches, it makes sense. The reconstruction work done for this thesis does not contain many data points with similar frequencies or a great deal of similarities. There are neither similarities nor parallels among any of the input samples because every segment we feed to the generator or discriminator is a separate section of a complete sound file.

We expect our network to be able to reconstruct the missing frequencies based on the feed frequencies. As such, Batch Normalization has never been applied to the Discriminator before. Considering that the discriminator may not learn in the manner that it is intended to due to batch normalization. But in this instance, it's crucial to note how using Batch Normalization modifies the results.

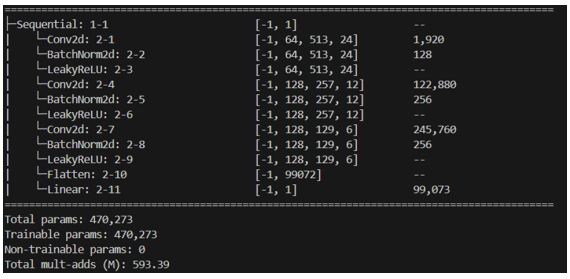
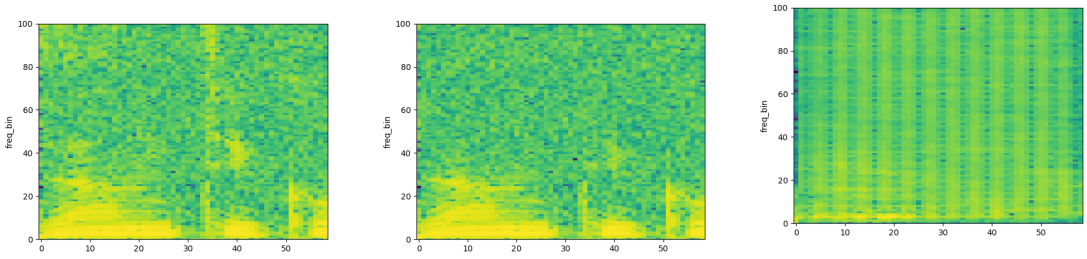



**Figure 4.21:** New structure of the discriminator

Learning rates 0.02, 0,0002, and 0,0004 have been used in this experiment's training.

2 models have been trained until 1000 epoch, the first model with 0.02 learning rates, and the second model with 0.0002 to see the difference in the results to be able to say which learning

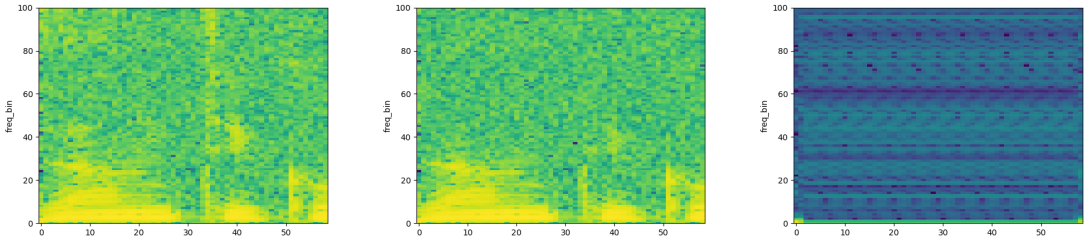



**(a)** 1000 epoch with 0.0002 learning rate: Original Sound Spectrogram

**(b)** 1000 epoch with 0.0002 learning rate: Cut Sound Spectrogram

**(c)** 1000 epoch with 0.0002 learning rate: Reconstructed Sound File Spectrogram

**Figure 4.22:** 1000 epoch with 0.0002 learning rate Spectrograms




**(a)** 1000 epoch with 0.02 learning rate: Original Sound Spectrogram

**(b)** 1000 epoch with 0.02 learning rate: Cut Sound Spectrogram

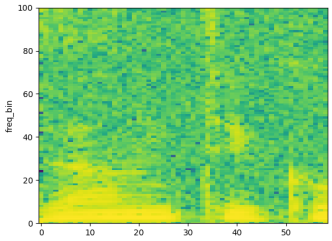
**(c)** 1000 epoch with 0.02 learning rate: Reconstructed Sound Spectrogram

**Figure 4.23:** 1000 epoch with 0.02 learning rate Spectrograms

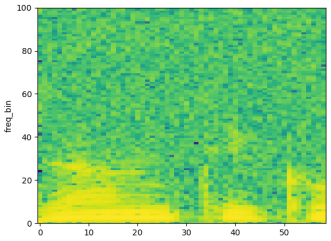
rate could be better. In old experiments, a high learning rate like 0.02 was never used. Therefore to be able to compare the results between the learning rate 0.0002 we used in the previous experiments and this 0.02, this has been done.

According to the results you can see in figure 4.22 and 4.23, the learning rate of 0.0002 works way better than the learning rate of 0.02. Both of the results does not looks good, but if we look at the baseline, it can be easly said that learning rate 0.0002 does look better(figure 4.22 (c)). Because it did not get rid of all of the baseline frequencies like the model which trained with learning rate 0.02 (figure 4.23 (c)).

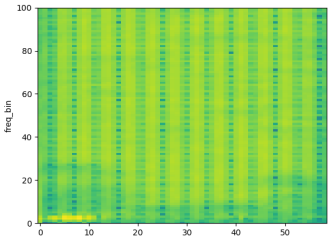
2 experiments with different learning rates started to train at the same time which both models have the same structures, and both have used the 1000 epoch pre-trained version of the model with 0.0002 learning rate which results have shown in figure 4.22. Both models have




**(a)** 4400 epoch with 0.0002 learning rate: Original Sound Spectrogram

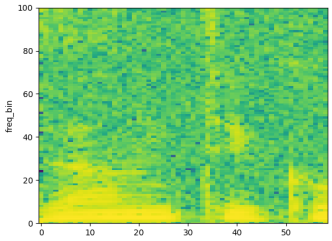



**(b)** 4400 epoch with 0.0002 learning rate: Cut Sound Spectrogram

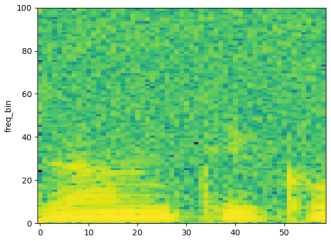



**(c)** 4400 epoch with 0.0002 learning rate: Reconstructed Sound Spectrogram

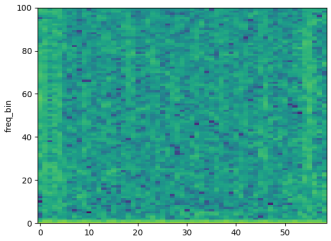
**Figure 4.24:** 4400 epoch with 0.0002 learning rate: Spectrograms




**(a)** 4400 epoch with 0.00004 learning rate: Original Sound Spectrogram




**(b)** 4400 epoch with 0.00004 learning rate: Cut Sound Spectrogram




**(c)** 4400 epoch with 0.00004 learning rate: Reconstructed Sound Spectrogram

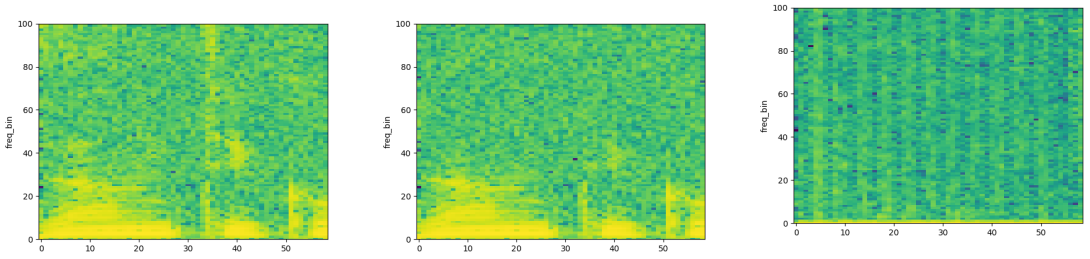
**Figure 4.25:** 4400 epoch with 0.00004 learning rate:Spectrograms

trained for a 3400 epoch more, which results in a 4400 epoch. one model trained with 0.0002 learning rate, and the other one trained with 0.00004 learning rate.

The results of the experiment with a 0.0002 learning rate look more promising. but as you can see in Figures 4.24 and 4.25, both of the results are bad. The model trained with a 0.00004 learning rate completely gets rid of all baseline as it seems in Figure 4.25 (c) but this may be the result of the low learning rate. So, this model trained for 1600 epochs more.

It seems that the results above have not changed much from the 4400 epoch version. But we can say that the model started to learn the baseline again from the result Figure 4.26 (c).

### Adopted Strategies:




**(a)** 4400 epoch with 0.00004 learning rate:Original Sound Spectrogram

**(b)** 4400 epoch with 0.00004 learning rate:Cut Sound Spectrogram

**(c)** 4400 epoch with 0.00004 learning rate:Reconstructed Sound File Spectrogram

**Figure 4.26:** 4400 epoch with 0.00004 learning rate Spectrograms

• With pre-processing of the data, and changing segment lengths, lighter and faster networks have been made. Therefore many feasible pieces of training have been done.

• Different ranges of learning rates have been used to see which rate is more compatible to use

• Strategy, which trainin generator multiple times for each training of the discriminator have been used.

• Batch Normalization has been added to the Discriminator's structure

• Label smoothing not used

**Positive Resulst:**

• More balanced networks have been made, but still, the Generator loss increased and Discriminator loss decreased with more epochs. From this results, it can be said, that label smoothing should be used to create a completely balanced model

**Negative Results:**

• With each training the waveform gets smaller, and dB levels decrease as the other experiments

• No new frequencies were generated to acquire the original result.

• Balance of Training may be disturbed after some point. Generator loss increases slowly.

• The result spectrograms show the batch normalization did not work well. But figure 4.26 (c) shows some signs of rebuilding base frequencies.

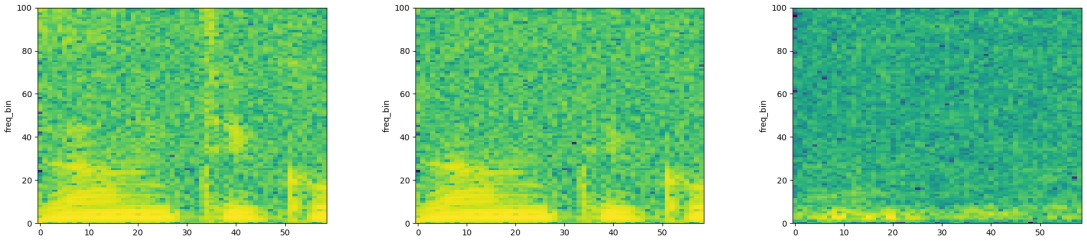
### 4.2.6 Experiment 6:

In experiment 6, the same preprocess and same data as experiment 5 were used. 2000Hz cut sound has been used like experiment 5

In experiment 5 we have seen that to be able to train our network without thinking about the balance, label smoothing is a must. This result of label smoothing has been achieved according to the experiments 4 and 5. But this time the label smoothing will not made with values 0.9 for the original and 0.1 for fake data. Because with these values, the discriminator's values will be too softened, therefore discriminator will not learn the data properly. This will be resulting the generator not learning the data properly as well as the Discriminator because of the Binary Cross Entropy (BCE) loss function. This means the Generator maximum learns the data as well as the Discriminator, cannot learn more than the Discriminator.

Discriminator's dropout rates have been increased in each layer. All of the dropout rates have been made over 0.7. The main logic behind increasing the dropout rates this much

The result of this model looks more promising than the others. According to the results of this model, the fine-tuning attempt has started.The training is completely balanced.Multiple pieces of training have been made, in each 1500 epoch results have been checked.

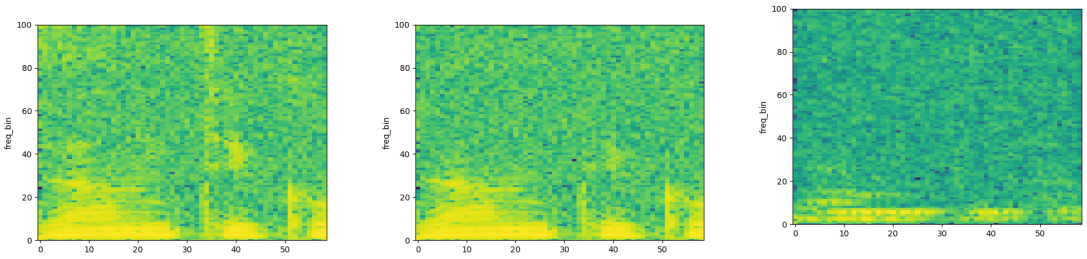



**(a)** epoch 17500:Original Sound Spectrogram

**(b)** epoch 17500:Cut Sound Spectrogram

**(c)** epoch 17500:Reconstructed Sound File Spectrogram

**Figure 4.27:** epoch 17500 Spectrograms




**(a)** epoch 19000 with learning rate 0.00004:Original Sound Spectrogram

**(b)** epoch 19000 with learning rate 0.00004:Cut Sound Spectrogram

**(c)** epoch 19000 with learning rate 0.00004:Reconstructed Sound Spectrogram

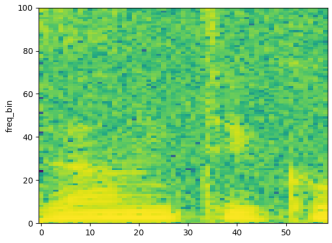
**Figure 4.28:** epoch 19000 with learning rate 0.00004:Spectrograms

According to the checked results, when it was obvious that the model was not learning anything anymore, the label smoothing values and learning rate values changed. The values of label smoothing softened from 0.9 for real,0.1 for take to 0.99 for real, and 0.01 for fake, from this spot it softened to 0.995 for real, 0.005 for fake, and the last level of label smoothing values was 0.9995 for real and 0.0005 for fake.

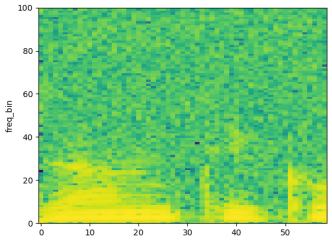
By the end of this experiment, we have obtained 32000 models that have been trained for a specified number of epochs.

The best results are shown in the figures for the training amount and the learning rates.

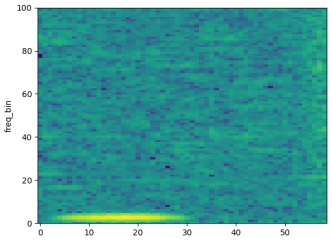
The result of the 17500 epoch can be seen in the figure 4.27. It is visible that the model




**(a)** epoch 19000 with learning rate 0.0002:Original Sound Spectrogram

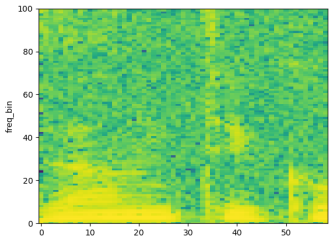



**(b)** epoch 19000 with learning rate 0.0002:Cut Sound Spectrogram

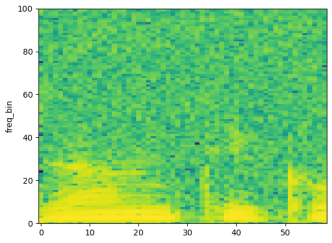



**(c)** epoch 19000 with learning rate 0.0002:Reconstructed Sound File Spectrogram

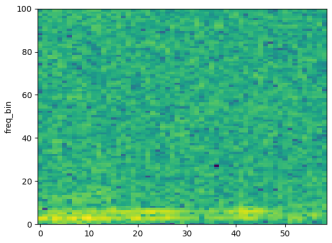
**Figure 4.29:** epoch 19000 with learning rate 0.0002:Spectrograms




**(a)** epoch 28000 with learning rate 0.000008:Original Sound Spectrogram

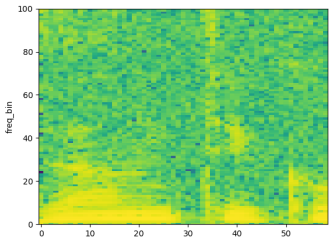



**(b)** epoch 28000 with learning rate 0.000008:Cut Sound Spectrogram

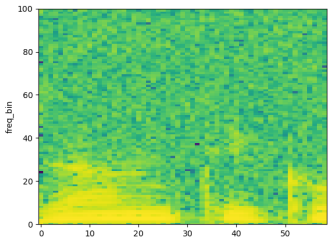



**(c)** epoch 28000 with learning rate 0.000008:Reconstructed Sound Spectrogram

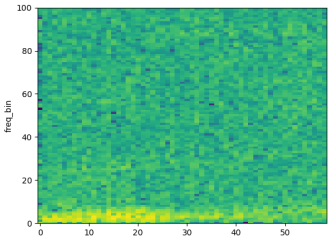
**Figure 4.30:** epoch 28000 with learning rate 0.000008 Spectrograms




**(a)** epoch 28000 with learning rate 0.00002:Original Sound Spectrogram

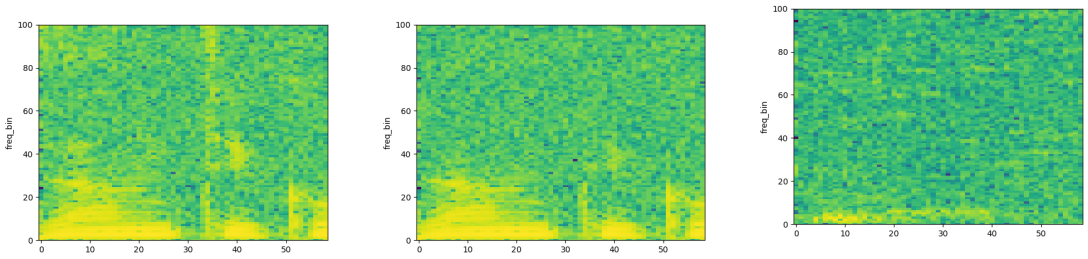



**(b)** epoch 28000 with learning rate 0.00002:Cut Sound Spectrogram




**(c)** epoch 28000 with learning rate 0.00002:Reconstructed Sound Spectrogram

**Figure 4.31:** epoch 28000 with learning rate 0.00002 Spectrograms

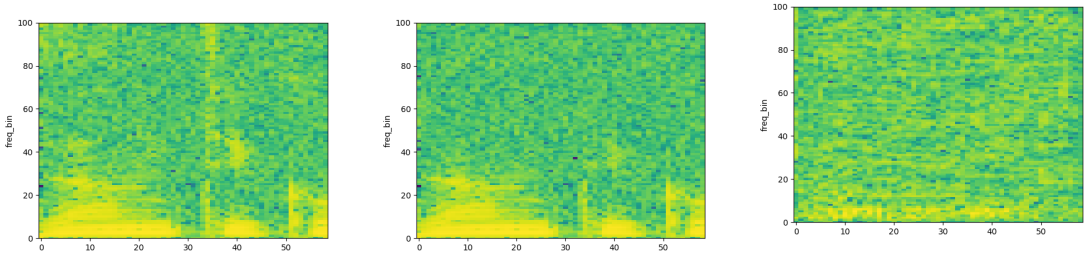



**(a)** epoch 32000 with learning rate 0.000008:Original Sound Spectrogram

**(b)** epoch 32000 with learning rate 0.000008:Cut Sound Spectrogram

**(c)** epoch 32000 with learning rate 0.000008:Reconstructed Sound Spectrogram

**Figure 4.32:** epoch 32000 with learning rate 0.000008 Spectrograms




**(a)** epoch 32000 with learning rate 0.00002:Original Sound Spectrogram

**(b)** epoch 32000 with learning rate 0.00002:Cut Sound Spectrogram

**(c)** epoch 32000 with learning rate 0.00002:Reconstructed Sound File Spectrogram

**Figure 4.33:** epoch 32000 with learning rate 0.00002:Spectrograms

tries to generate the baseline from Figure 4.27 (c) and it looks promising. But because the results look like this for at least 4500 epochs, the label smoothing values have changed to 0.99 for real, and 0.01 for fake to make the Discriminator learn better, therefore the generator will learn more from a better Discriminator. Also from this point on, two models with the same label smoothing but different learning rates have been trained to be able to do fine-tuning.

In Figures 4.28 and 4.29, we can see the results after training the previous model 1500 epochs more with learning rates 0.00004 and 0.0002. Making this train with different learning rates which can be seen in Figures 4.28 (c) and 4.29(c) shows us the smaller learning rates better. As you can see in both results, we can say our generator has much more confidence about the base frequencies. But the main difference between the two is the details. The 0.00004 learning rate has worked better and created more detailed results than the model with a learning rate of 0.0002.

As the next step, we have loaded the 19000 epoch model with a learning rate of 0.00004 as a pre-trained model, decreasing the label smoothing even more from 0.99 for real and 0.01 for fake to 0.999 for real and 0.001 for fake and trained 9000 epochs more. But as in the previous training sequence, we have trained 2 models again with learning rates of 0.00002 and 0.000008. In Figure 4.30 (c) and 4.31 (c), you can see the results. The results are not as good as the result in Figure 4.28 (c), but we can still say that both models are confident about the base frequencies.

On the last try, the label smoothing decreased even further from 0.999 for real and 0.001 for fake to 0.9995 for real and 0.0005 for fake. And the 2 models we had trained for 4000 more epochs with the same rate of learning rates. In Figure 4.32 (c), we can see the learning rate of 0.000008 has started to affect the model badly because it is visible that the model is not as confident as 4000 epochs before the version of itself. On the other hand, in Figure 4.33, it is visible that the model is still confident about the baseline and it started to create some frequencies. But the result is still not the way we wanted.

**Adopted Strategies:**

• Label smoothing got softened according to the results which shows the model is not learning properly anymore. The label smoothing effect, which was explained as a possibility in experiment 4 has been confirmed.

• Different ranges of learning rates have been used to see which rate is more compatible to use

• According to the results, more training has been done

• Strategy, which in each iteration, trains the Generator multiple times, but the Discriminator once has not changed.

• Batch Normalization used

**Positive Results:**

• The model is completely balanced.

• The First positive result in the spectrograms has been achieved with batch normalization.

• at 32000 epoch, one of our models has generated some new frequencies.

**Negative Results:**

• The waveform and the dB levels did not change much.

• Generated new frequencies at 32000 epoch is not accurate.

• this experiment took too much time to train.

### 4.2.7 Experiment 7

This experiment followed the identical preprocessing and dataset preparation procedures as the previous two (Experiments 5 and 6). The audio samples were split into intervals of 0.5 seconds, the same as in the previous studies. A major difference in this experiment, though, is that the spectrograms that were produced were logarithmically scaled prior to being ready for training after the audio segments were subjected to the Short-Time Fourier Transform (STFT). This is the main change made to the model in this experiment. Instead of training it on linear data, it will be trained on logarithmically scaled spectrograms, which should improve the model's

capacity to catch and recreate frequency patterns.

The dynamic changes in the discriminator's training frequency based on the generator's performance are an important addition to this training procedure. By using an adaptive technique, it is ensured that during training, the discriminator won't overrun the generator, a common problem that can result in mode collapse or generator failure.

The discriminator's training in this implementation is dependent on the value of the discriminator's fake loss (d loss fake). In particular, only when the d loss fake surpasses a threshold value of 0.1 is the discriminator taught. The discriminator's training is stopped if the false loss falls below this threshold, signaling that the discriminator is growing too strong and preventing the generator from catching up and producing better results.

Additionally, the technique includes a counter (discriminator train counter) to monitor consecutive occasions where the fake loss remains low. The discriminator training is stopped temporarily if the counter rises above a predetermined threshold. This makes sure an excessively precise discriminator doesn't continuously penalize the generator, which could hinder its ability to learn.

In essence, this dynamic approach introduces the following steps:

- Discriminator Skipping: If the discriminator's fake loss is below 0.1, it is considered to be performing too well. In such cases, the training of the discriminator is skipped for several iterations, allowing the generator to improve without excessive pressure.
- Discriminator Reset: Once the "discriminator train counter" reaches a threshold value, the discriminator is retrained, resetting the balance between the generator and discriminator.

By introducing a more balanced adversarial training procedure, this technique makes sure that the performance of the discriminator and generator does not significantly diverge. The approach sustains beneficial competition between the two networks, enabling better convergence and raising the standard of the audio outputs that are produced, by dynamically regulating the

discriminator's updates.

The Exponential Linear Unit (ELU) performs better overall in deep learning tasks, it was initially chosen as the activation function in the generator's second and third layers in this experiment. ELU is renowned for producing smooth output, which helps improve learning dynamics. But more importantly, when inputs are much below zero, ELU tends to converge to a constant negative value, adding to the computational complexity. Although helpful in certain situations, this behavior made it difficult to use the generator for the current purpose of reconstructing audio data because it made it harder to learn from specific mistakes.

Because of this, the activation function in the second and third layers of the generator has changed from ELU to the Leaky ReLU activation function. Leaky ReLU, in contrast to ELU, provides negative values proportionate to the mistake rather than convergent to a constant by applying a constant negative slope to negative inputs. This guarantees that the gradient stays active in the event that erroneous predictions are made, enabling the generator to more dynamically modify its output. Leaky ReLU fits this model better because of its adaptability to negative values, which allows it to capture the mistake size without being limited by a preset response.

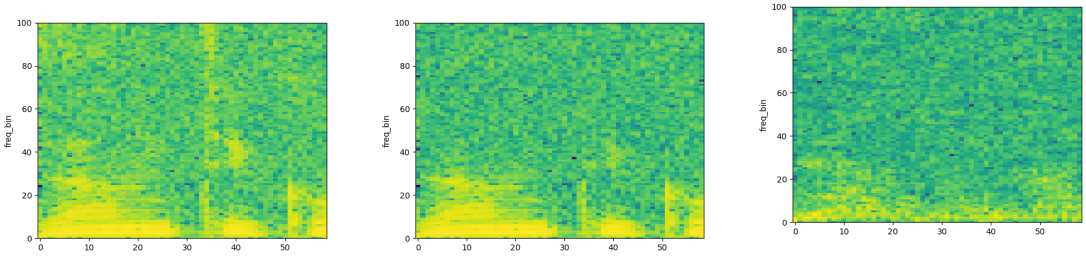
3 models have been trained in this experiment.

As a result of feeding the logarithmically scaled data to the both generator and discriminator, the most suitable learning rate could be changed. Therefore 3 models with the same structure, but different learning rates have been made. Learning rates of 0.00008, 0,000008, and 0,0000008 have been used. All of the 3 models have been trained for 6000 epochs. Result of the high epoch count of the training, and to see the effects of low values of learning rates, these learning rate values have been used

The model with a learning rate of 0.00008(4.36 (c)) produced the best spectrogram, as evidenced by the comparison in Figure 4.34 (c), 4.35(c), and 4.36(c).

The model achieved the second-best result with a learning rate of 0.0000008 (4.34 (c)).

Result of the model with a learning rate of 0.000008 was not good as we can see in Figure 4.35 (c). However, it was illogical because this learning rate is within the range of the

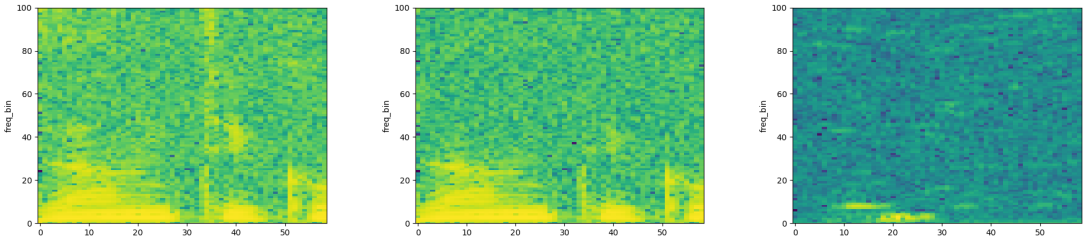



**(a)** 6000 epoch with 0,0000008 learning rate:Original Sound Spectrogram

**(b)** 6000 epoch with 0,0000008 learning rate:Cut Sound Spectrogram

**(c)** 6000 epoch with 0,0000008 learning rate:Reconstructed Sound File Spectrogram

**Figure 4.34:** 6000 epoch with 0,0000008 learning rate Spectrograms

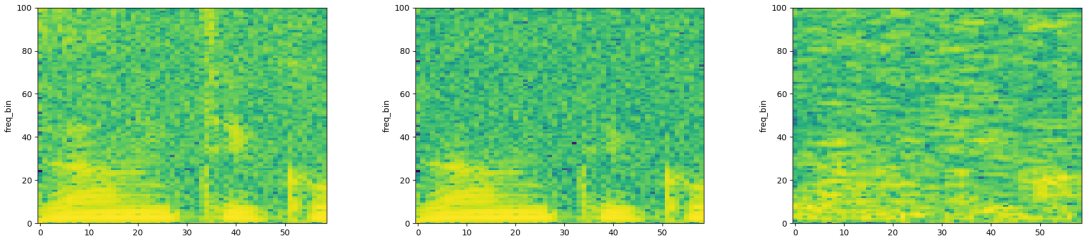



**(a)** 6000 epoch with 0,000008 learning rate:Original Sound Spectrogram

**(b)** 6000 epoch with 0,000008 learning rate:Cut Sound Spectrogram

**(c)** 6000 epoch with 0,000008 learning rate:Reconstructed Sound Spectrogram

**Figure 4.35:** 6000 epoch with 0,000008 learning rate Spectrograms

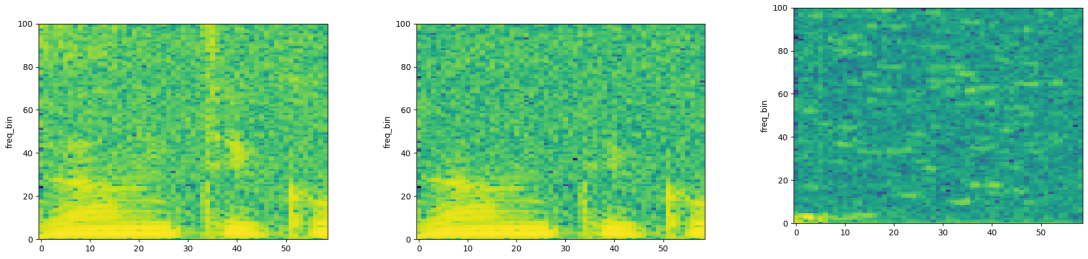



**(a)** 6000 epoch with 0,00008 learning rate:Original Sound Spectrogram

**(b)** 6000 epoch with 0,00008 learning rate:Cut Sound Spectrogram

**(c)** 6000 epoch with 0,00008 learning rate:Reconstructed Sound Spectrogram

**Figure 4.36:** 6000 epoch with 0,00008 learning rate Spectrograms




**(a)** 9000 epoch with 0,00008 learning rate:Original Sound Spectrogram

**(b)** 9000 epoch with 0,00008 learning rate:Cut Sound Spectrogram

**(c)** 9000 epoch with 0,00008 learning rate:Reconstructed Sound File Spectrogram

**Figure 4.37:** 9000 epoch with 0,00008 learning rate Spectrograms

lowest and highest rates we have used in this experiment. The stochastic nature of neural networks could be the cause of this. Therefore to get a better result and to see if this is because of the stochastic nature of the neural networks, this model is trained for 3000 epochs more. The results worsened, as shown in Figure 4.37 (c). The model is not confident about any kind of frequency.

**Adopted Strategies:**

- At the pre-processing step, the data logarithmically scaled.
- Label smoothing was used with 0.9995 for real, and 0.0005 for fake labels like the experiment 6
- Different ranges of learning rates have been used to see which rate is more compatible with logarithmically scaled data
- Strategy: in each iteration, the Generator is trained multiple times, while the Discriminator is utilized.
- Batch Normalization has been used
- Generator's first two layers' activation functions have been changed from ELU to LeakyReLU.

**Positive Results:**

- The training was balanced
- Better results with less training have been acquired

- Batch Normalization looks like still working
- Changing the activation functions in the Generator increased the speed of the model's learning, making it more feasible.

**Negative Results:**

- created frequencies still not good
- with learning rates, the results change too much. Hyperparameters are too sensitive.
- Balance of Training may be disturbed after some point.

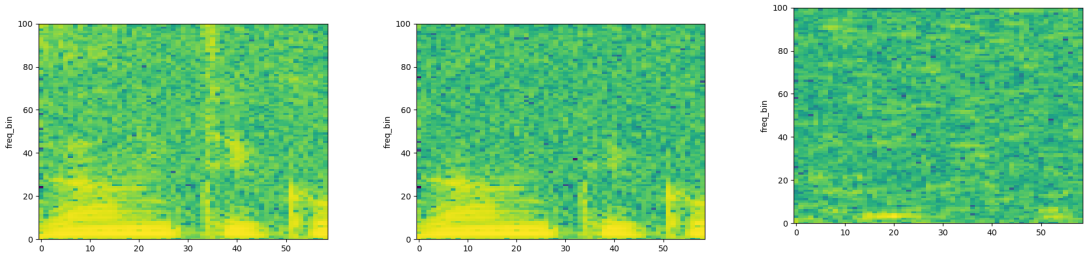
### 4.2.8 Experiment 8

**Effect of Negative Slopes in Leaky ReLU:**

This set of tests examined the effects of changing the Leaky ReLU activation function's negative slope. In order to avoid the "dead neuron" issue, leaky ReLU is preferred over normal ReLU. This is because it permits a little, non-zero gradient when the input is negative. This is necessary to ensure stable and effective training in deep neural networks by preserving gradient flow even in areas where the input is less than zero.

Initially, as is typical of GAN architectures, the generator and discriminator employed Leaky ReLU with a standard negative slope of 0.01. The outcomes were examined following 6,000 epochs of training this model. Another model was trained for 6,000 epochs with the negative slope increased to 0.3 in both the discriminator and generator in order to further examine the impact of negative slopes. This experiment aimed to examine the effects of various negative slope values on learning and output quality.

**model with a Negative Slope of 0.01 (Baseline):** Although the learning process in the model trained with a slope of 0.01 was comparatively slow, the results were reasonably consistent. Although the spectrograms produced by the model were stable, they could not perfectly recreate the original sound. This is probably because the network was unable to significantly alter the generated data because to the tiny negative slope, which produced weaker gradient signals during backpropagation. The waveforms were visible but not very distinct, and as training went on, the total decibel level dropped a little.




**(a)** 6000 epoch with standard negative slope:Original Sound Spectrogram

**(b)** 6000 epoch with standard negative slope:Cut Sound Spectrogram

**(c)** 6000 epoch with standard negative slope:Reconstructed Sound File Spectrogram

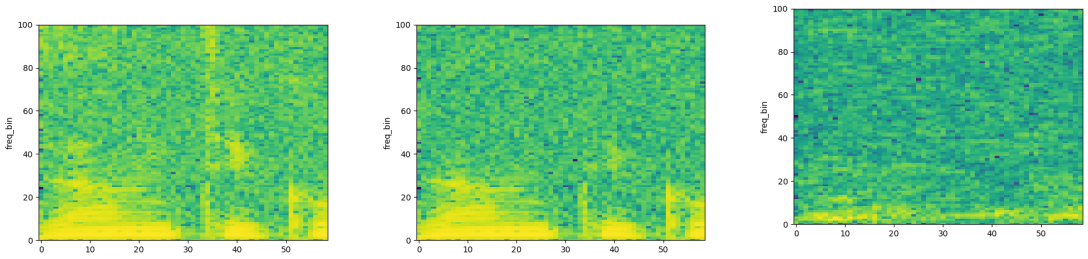
**Figure 4.38:** 6000 epoch with standard negative slope Spectrograms

### Model With a Negative Slope of 0.3 (Experiment):

When the negative slope in the second model was raised to 0.3, the learning process considerably improved. When compared to the baseline model, the generated audio shown more consistent decibel levels and cleaner waveforms after 6,000 epochs. Larger gradient signals in the negative input region were made possible by the increased negative slope, which helped the network make more significant training corrections. This showed that the model could better capture the underlying structure of the target sound as it led to faster learning and more accurate reconstruction of the audio.

It was clear from comparing the two models that the generator and discriminator performed better when the negative slope in Leaky ReLU was increased from 0.01 to 0.3. Slower convergence was the result of the model's inability to perform large updates during training due to the lesser slope. The greater slope, on the other hand, made learning faster and more efficient, leading to outputs that were clearer and more accurate. This experiment emphasizes how crucial it is to adjust Leaky ReLU's negative slope since it has a direct impact on the model's ability to learn and get better over training.

A comparison of the spectrograms in Figures 4.39 (c) and 4.40(c) shows the higher negative slope has given better results. Figure 4.39 (c) shows that the model with an increased negative slope has more confidence in the base frequencies. Therefore the model with an increased negative slope value of 0.3 has trained 3000 epochs more to see the result.

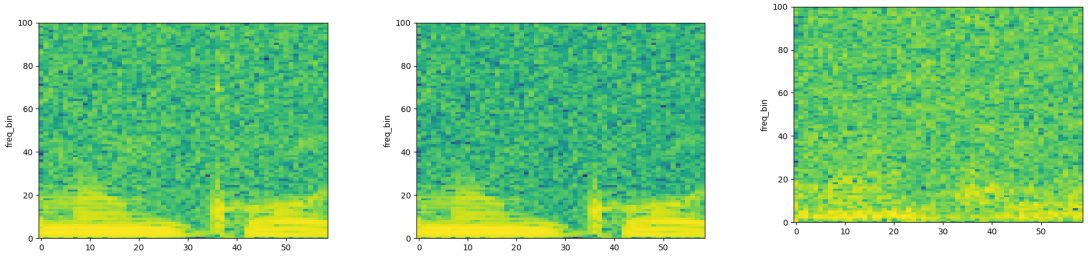



**(a)** 6000 epoch with negative slope value 0.3:Original Sound Spectrogram

**(b)** 6000 epoch with negative slope value 0.3:Cut Sound Spectrogram

**(c)** 6000 epoch with negative slope value 0.3:Reconstructed Sound File Spectrogram

**Figure 4.39:** 6000 epoch with negative slope value 0.3:Spectrograms




**(a)** 9000 epoch with negative slope value 0.3:Original Sound Spectrogram

**(b)** 9000 epoch with negative slope value 0.3:Cut Sound Spectrogram

**(c)** 9000 epoch with negative slope value 0.3:Reconstructed Sound File Spectrogram

**Figure 4.40:** 9000 epoch with negative slope value 0.3:Spectrograms

The Figure 4.40(c) which is trained 9000 epoch with an increased negative slope of 0.3 shows the best result. This result is the first result that deletes the base frequency but re-creates it after more training. The model is confident about the base frequencies, and with more training, it can be more confident.

**Adopted Strategies:**

- Only strategy difference with experiment 7 is leaky relu negative slope change.The negative slope value 0.3 is used.

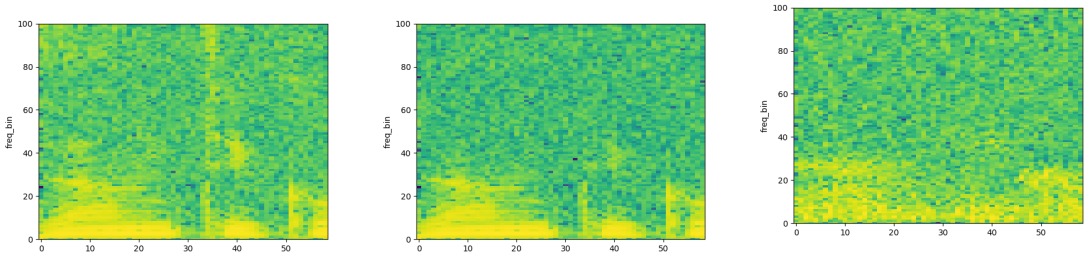
**Positive results:**

- The increasing the negative slope value is effected results positively.
- Training for 9000 epochs initially removes the base frequency but re-creates it after further training. This is the first model which re-creates the base frequency after deleting it.

### 4.2.9 Experiment 9

The generator's loss now includes Mean Squared Error (MSE). The main goal of including the Mean Squared Error (MSE) in the generator's training phase is to keep important frequency components that we feed to the Generator as base frequencies from vanishing during the feed-forward procedure. The generator's first objective was to enhance the incomplete frequency information in the baseline data with the missing frequencies. However, after a few rounds, the baseline itself started to disappear, which resulted in a wrong reconstruction of the original audio. In order to mitigate this, the MSE loss was added to make sure that the output that is produced stays very similar to the original data, which helps in maintaining the baseline frequencies.

Incorporating MSE encourages the model to reconstruct the missing sections while preserving the integrity of the input data, preventing the generator from changing the fundamental frequencies to deceive the discriminator. When the missing frequencies are being reconstructed, the adversarial loss (BCE) and content loss (MSE) combination helps stabilize the training process and guarantees that the recreated sound stays true to the original. In essence, the obtained results are progressively corrected and refined over time by the MSE loss.

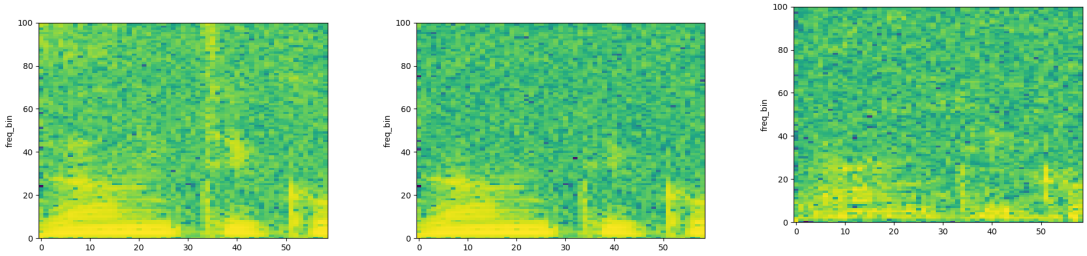



**(a)** 4000 epoch with Learning Rate 0.000008:Original Sound Spectrogram

**(b)** 4000 epoch with Learning Rate 0.000008:Cut Sound Spectrogram

**(c)** 4000 epoch with Learning Rate 0.000008:Reconstructed Sound File Spectrogram

**Figure 4.41:** 4000 epoch with Learning Rate 0.000008 Spectrograms




**(a)** 4000 epoch with Learning Rate 0.00008:Original Sound Spectrogram

**(b)** 4000 epoch with Learning Rate 0.00008:Cut Sound Spectrogram

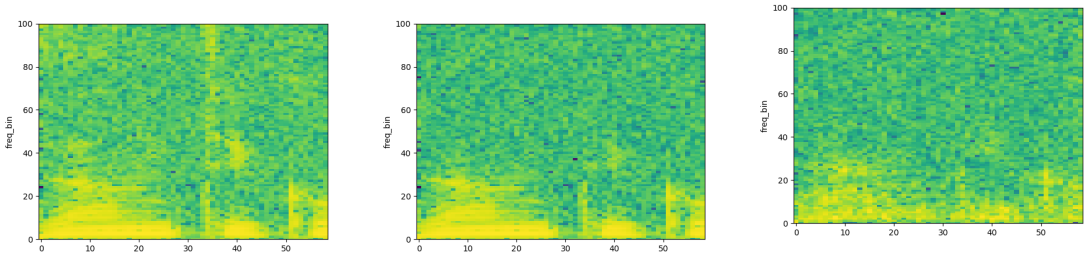
**(c)** 4000 epoch with Learning Rate 0.00008:Reconstructed Sound File Spectrogram

**Figure 4.42:** 4000 epoch with Learning Rate 0.00008:Spectrograms

We made the decision to discard the label smoothing method that had been applied in the past to stabilize the discriminator's training due to the corrective mechanism that MSE had introduced. The generator was able to learn without the label smoothing since the MSE made sure it was doing so.

We trained three models with learning rates of 0.0008, 0.00008, and 0.000008 for a total of 4000 epochs to assess the effects of the MSE and various learning rates in more detail. This made it possible for us to compare the results at various learning rates while keeping the same general structure, which allowed for a comprehensive assessment of the ideal training setup.

MSE was effective which we can see in the figure 4.41(c), 4.42(c) and 4.43(c). If we compare the reconstructed versions with their original and noisy versions in the figures, we can see that we have some good results.




**(a)** 4000 epoch with Learning Rate 0.0008:Original Sound Spectrogram

**(b)** 4000 epoch with Learning Rate 0.0008:Cut Sound Spectrogram

**(c)** 4000 epoch with Learning Rate 0.0008:Reconstructed Sound File Spectrogram

**Figure 4.43:** 4000 epoch with Learning Rate 0.0008:Spectrograms

The best result between them achieved with the learning rate 0.000008. It is not much different from the other ones but if the figure 4.41 (c) compared with the figure 4.42 (c) and 4.43 (c), it can be said that the 4.41 (c) is more confident about the base frequencies.

**Adopted Strategies:**

Mean Square Error has added to the Generator's loss at the train step.

Leaky Relu negative slope value is 0.3

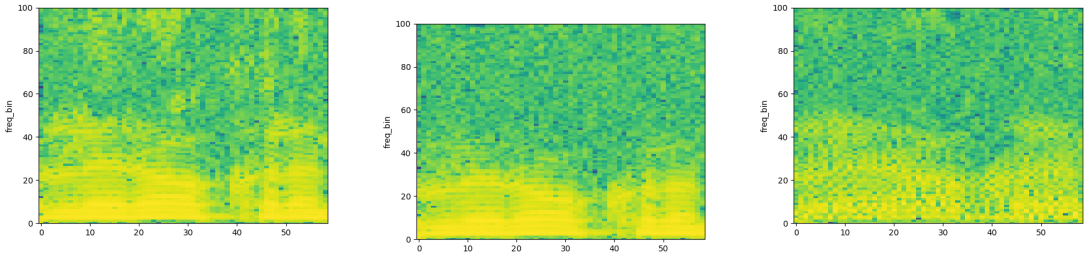
Other strategies and structures same as the experiment 8

**Positive Results:**

• The model is more feasible this way. Better base frequencies generated with less training

**Negative Results:**

• The Mean Square Error may cause the model to overfit after some point. So, it should be used dynamically.




**(a)** 1500 epoch :Original Sound Spectrogram

**(b)** 1500 epoch: Cut Sound Spectrogram

**(c)** 1500 epoch: Reconstructed Sound File Spectrogram

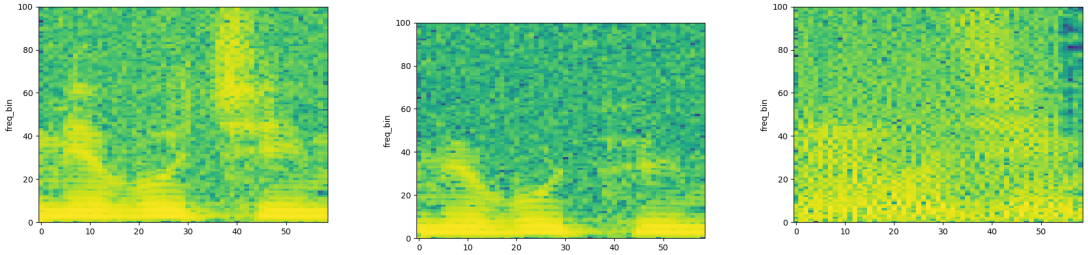
**Figure 4.44:** 1500 epoch: Spectrograms

### 4.2.10 Experiment 10

The primary objective of this experiment was to preprocess the audio files using Mel-Frequency Cepstral Coefficients (MFCC) as opposed to the previously employed Short-Time Fourier Transform (STFT). The goal of this method is to more accurately represent the frequency characteristics of audio by using a mel-scaled representation that is more consistent with human perception.

Initially, the audio material was split up into smaller segments, each lasting 0.5 seconds. For similarity across all files, the original waveform was loaded and, if needed, converted to mono. The MFCC was calculated for each of these segments once the waveform had been divided into them. The MFCC data were subjected to logarithmic scaling in order to improve the training procedure even more. This modification is essential because it ensures that the model concentrates on minute variations in the low-energy portions of the signal by compressing the audio data's enormous dynamic range.

The dataset was then prepared using each of these logarithmically scaled MFCC segments, containing both clean and noisy versions of the audio files. To enable the model to learn how to rebuild the clean audio from the noisy input, the clean and noisy MFCC segments were linked together. This technique made it possible to create a dataset that effectively taught the model how to rebuild clean audio from noisy inputs by capturing both the crucial frequency features and the time-domain fluctuations of the audio. With the earlier tests, the structure remained the same.

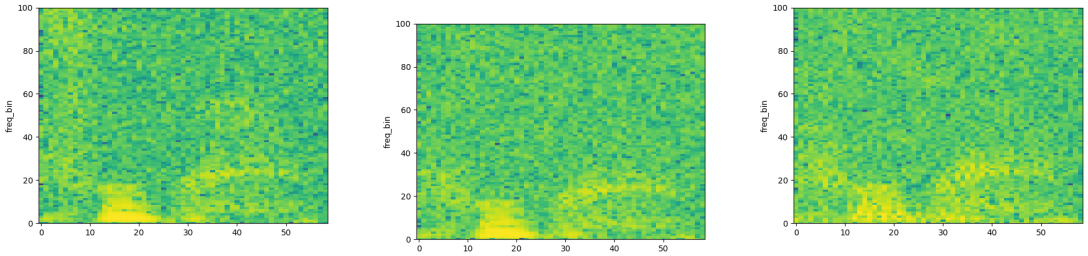



**(a)** 3000 epoch: Original Sound Spectrogram

**(b)** 3000 epoch: Cut Sound Spectrogram

**(c)** 3000 epoch: Reconstructed Sound File Spectrogram

**Figure 4.45:** 3000 epoch: Spectrograms




**(a)** 3500 epoch: Original Sound Spectrogram

**(b)** 3500 epoch: Cut Sound Spectrogram

**(c)** 3500 epoch: Reconstructed Sound File Spectrogram

**Figure 4.46:** 3500 epochSpectrograms

Figure 4.44 (c) shows we have trained a confident model using both BCE and MSE at the Generator loss. Figures 4.45 and 4.46 show us the models even with more confidence.
The best result ever gotten in all of the experiments is the figure 4.45 (c). It can be seen that the model did not delete the base frequencies, and it generated the new frequencies accurately. We can say that if we compare the figure 4.45 (a), (b) and (c). But this may cause the overfitting.

**Adopted Strategies:**

The dataset has been updated to use logarithmically scaled MFCCs instead of STFTs.

The structure is the same as the previous experiment, MSE was used in the Generator loss with BCE as the previous experiment.

**Positive results:**

The first results achieved with the newly generated frequencies are accurate.

the model looks confident.

**Negative Resuls**

• MSE might make the model Overfit.

### 4.2.11 Experiment 11

Transition to Layer Normalization [23]

In the last experiment, layer normalization was used in place of batch normalization in both the discriminator and the generator. This change was necessary due to the structure of the dataset, where each data point represented a different segment of a single full sound recording. Unlike datasets where samples share similar characteristics, the segments in this dataset were highly varied. Some segments contained more frequencies, some fewer, and others had parts with almost no sound. The goal of the model was to learn how to reconstruct missing frequencies based on the unique characteristics of each segment.

Batch normalization, which normalizes based on the statistics of entire batches by calculating the mean and variance across the batch, was problematic in this context. Each segment in the dataset represented a different frequency profile, leading to inconsistency in the batch statistics. Batch normalization assumes some degree of similarity between batch members, but

because the segments were so varied, this resulted in unstable gradient updates and inconsistent learning. The model would struggle to generalize the frequencies to reconstruct since no two segments were alike, which led to a fluctuating performance in the discriminator and difficulty for the generator to converge.

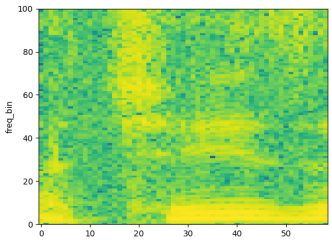
To address this, layer normalization was used, which normalizes across the features of each individual data point rather than across the entire batch. Since layer normalization is independent of batch size and focuses on the characteristics within each data point, it provided a more stable learning dynamic for this particular dataset. By switching to layer normalization, the model was better able to handle the varied frequency content of each segment, avoiding the instability and oscillations that occurred when using batch normalization. This change resulted in smoother training dynamics and more reliable frequency generation across the entire range of segments.

The results of the training made the significance of this change especially clear. Lower dB values in the reconstructed waveforms and noisier results were probably caused by the use of batch normalizing. The majority of the input's frequency values are low or very close to zero, therefore it's possible that batch normalization further decreased these values during training, lowering the decibel levels even further. The waveforms that became flatter and had a decreasing amplitude after each epoch showed that batch normalization was attenuating significant frequency components.

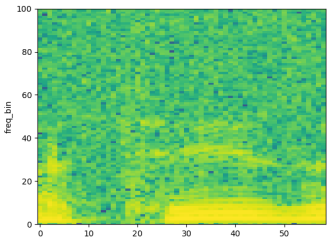
The model avoided this problem by using layer normalization, which led to more consistent training and clearer reconstructions with waveforms that better preserved their amplitude and frequency information. To see the effect of the layer normalization in the results, the MSE loss have cancelled in this experiment.

Best results have been achieved without using MSE in the Generator. In Figure 4.47, it is clearly seen the Reconstructed version (4.47 (c)) contains the generated missing frequencies. And it generated those missing frequencies pretty much accurately if we compare it with the 4.47 (a). The first result with using just Adversarial Loss(Binary Cross entropy (BCE)).

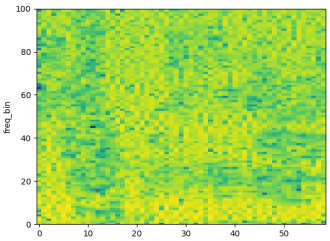
To see the results in other models with the completely same structure, 5 models have been trained which results can be seen in figures 4.48, 4.49, 4.50, 4.51, and 4.52. 2 models were trained until 6000 epoch, and both the models were able to re-create some missing frequencies,




**(a)** 5000 epoch Best Result:Original Sound Spectrogram

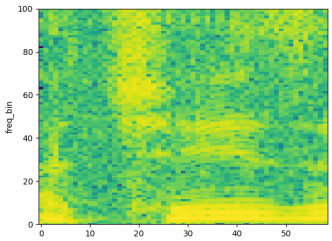



**(b)** 5000 epoch Best Result: Cut Sound Spectrogram

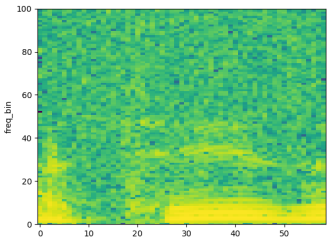



**(c)** 5000 epoch Best Result: Reconstructed Sound File Spectrogram

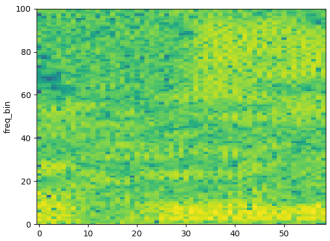
**Figure 4.47:** 5000 epoch Best Result: Spectrograms




**(a)** 6000 epoch first model: Original Sound Spectrogram

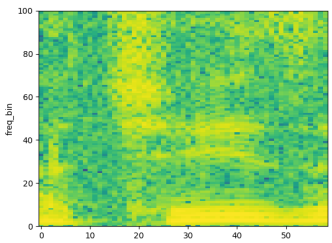



**(b)** 6000 epoch first model:Cut Sound Spectrogram

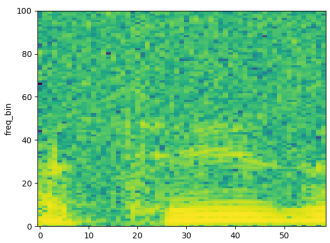



**(c)** 6000 epoch first model:Reconstructed Sound File Spectrogram

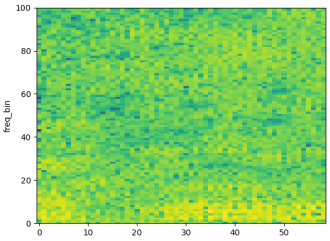
**Figure 4.48:** 6000 epoch first model:Spectrograms




**(a)** 6000 epoch Second model:Original Sound Spectrogram

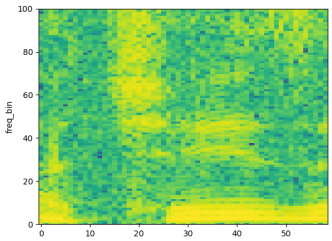



**(b)** 6000 epoch Second model:Cut Sound Spectrogram

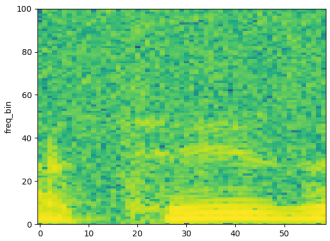



**(c)** 6000 epoch Second model:Reconstructed Sound File Spectrogram

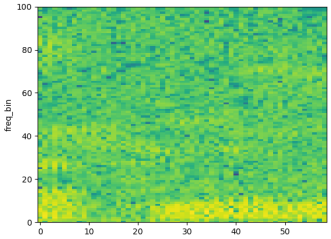
**Figure 4.49:** 6000 epoch Second model Spectrograms




**(a)** 8000 epoch Second model:Original Sound Spectrogram

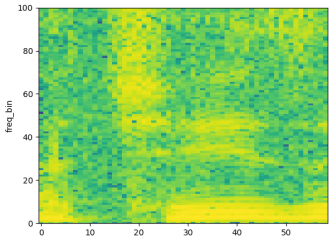



**(b)** 8000 epoch Second model:Cut Sound Spectrogram

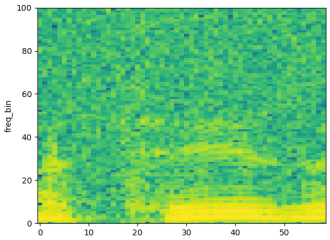



**(c)** 8000 epoch Second model:Reconstructed Sound Spectrogram

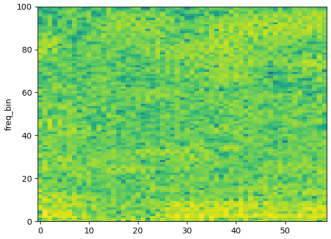
**Figure 4.50:** 8000 epoch Second model Spectrograms




**(a)** 10000 epoch Second model:Original Sound Spectrogram

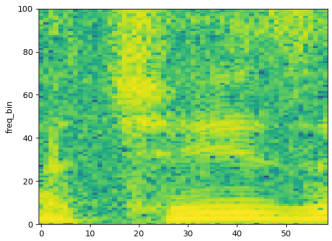



**(b)** 10000 epoch Second model:Cut Sound Spectrogram

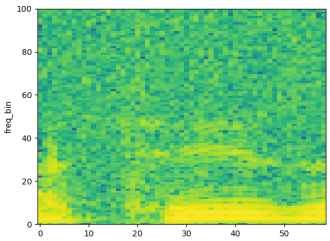



**(c)** 10000 epoch Second model:Reconstructed Sound Spectrogram

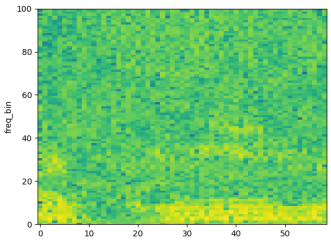
**Figure 4.51:** 10000 epoch Second model Spectrograms




**(a)** 21000 epoch model:original Sound Spectrogram




**(b)** 21000 epoch model:Cut Sound Spectrogram




**(c)** 21000 epoch model:Reconstructed Sound File Spectrogram

**Figure 4.52:** 21000 epoch model Spectrograms

not as accurately as 5000 epoch version of themselves which can be seen in figure 4.47. But still, the layer normalization results can be easily seen from the results we are getting. All of the results show that each of them is confident about the base frequencies and trying to generate the missing frequencies.

we can see that the models are confident about the base frequencies and try to generate the missing frequencies in Figures 4.48 (c), 4.49 (c), 4.50 (c), 4.51 (c), and 4.52 (c).

**Adopted Strategies:**

Started using layer normalization instead of batch normalization.

to see the effect of the Layer normalization, MSE loss for the generator has been canceled.

**Positive results:**

- First results with generated accurate missing frequencies using Adversarial Loss (BCE) have been acquired.
- The model works stable, and trains with balance,
- Model does not need the label smoothing for balance.

**Negative Results:**

- The stochastic nature of GANs can result in delayed generation of missing frequencies.
- In order to achieve better results, this structure still requires some further improvement.

# 5 Results

In this thesis, we explored multiple strategies to improve the generation and reconstruction of damaged audio files using a combination of SpecGAN and DCGAN architectures. Throughout the experiments, various preprocessing techniques, model architectures, and training strategies were implemented to evaluate their impact on the quality of audio reconstructions. Below, we summarize each of the key strategies and the rationale behind their use.

## 5.1 Spectrogram-Based Audio Representation

The first step in our process involved converting raw audio data into a format that could be effectively used by the generator and discriminator. Initially, Short-Time Fourier Transform (STFT) was applied to 2-second segments of the input audio to generate spectrograms, which represent the frequency content over time. The real and imaginary components of the spectrograms were separated, and the imaginary parts were transformed into real numbers, resulting in two-channel images that could be processed by the DCGAN. This approach was selected because DCGANs excel at generating two-dimensional data, making spectrograms a natural fit for training.

As we progressed, we experimented with logarithmically scaled Mel-Frequency Cepstral Coefficients (MFCCs) instead of using the STFT-based spectrograms. The rationale was that

MFCCs, being a more perceptually meaningful representation of sound, would allow the model to focus on the most important frequencies for human hearing. Additionally, the logarithmic scaling of the MFCCs was expected to help the model better handle the large dynamic range of audio signals, potentially improving reconstruction quality.

## 5.2 Generator and Discriminator Architectures

The generator and discriminator architectures were based on Deep Convolutional GAN (DCGAN), which employs convolutional layers for spatial feature extraction. Convolutional layers are well-suited for learning the time-frequency patterns found in spectrograms, and by using transposed convolutions in the generator, the model was able to upsample low-resolution inputs into higher-resolution outputs.

**Over time, several modifications were made to the architectures**:
**Leaky ReLU in the Discriminator:** Throughout the thesis, the Leaky ReLU activation function was consistently used in the discriminator. This choice was crucial, as Leaky ReLU allows small negative values to pass through, enabling the model to provide feedback to the generator even when it makes poor predictions. If ReLU had been used instead, negative values would be zeroed out, depriving the generator of important information. By adjusting the negative slope of Leaky ReLU (from the standard 0.01 to higher values like 0.3), we observed that the discriminator was able to provide more nuanced feedback, resulting in better learning dynamics.

**Generator Activation Functions:** Initially, ELU activation functions were used in the generator's second and third layers, as ELU is known to handle negative inputs more gracefully and accelerate convergence. However, it was later replaced by Leaky ReLU due to the constant negative output produced by ELU. By using Leaky ReLU, we allowed the generator to generate negative outputs based on the magnitude of error rather than forcing constant negative values, which improved the generation process by providing more adaptive feedback to the network.

## 5.3 Training Strategies

To stabilize training and improve the convergence of the model, we implemented several training strategies:

**Multiple Generator Updates Per Discriminator Update:** One of the major challenges in training GANs is ensuring that neither the generator nor the discriminator overpowers the other. To address this, we introduced a strategy where the generator was updated multiple times for each discriminator update. This allowed the generator to ”catch up” to the discriminator, ensuring a more balanced training process. The logic here was simple: by training the generator more frequently, it could quickly adapt to the feedback provided by the discriminator, preventing the discriminator from becoming too dominant.

**Dynamic Discriminator Training:** Another key strategy was dynamically adjusting the training frequency of the discriminator based on its fake loss. If the discriminator's fake loss dropped below a certain threshold (indicating that it was becoming too good at distinguishing fake data from real data), we limited its training to prevent it from becoming overly dominant. This adaptive training strategy ensured that both networks continued to improve without either one overwhelming the other, promoting more effective learning.

## 5.4 Incorporating Mean Squared Error (MSE) Loss

While the Binary Cross-Entropy (BCE) loss function was used for the adversarial aspect of training (i.e., fooling the discriminator), it was not sufficient for ensuring that the generated audio closely matched the target audio. To address this, we added a Mean Squared Error (MSE) loss term to the generator's objective. The MSE loss encourages the generator to minimize the difference between the generated and real spectrograms on a pixel-by-pixel basis. This addition was crucial for preserving the frequency content of the input audio, particularly for ensuring that the baseline frequencies (those present in the noisy data) were maintained.

The logic behind adding MSE was that while the adversarial loss helps the generator learn to fool the discriminator, it doesn't directly optimize for the accuracy of the generated spectrogram. By adding MSE, we could ensure that the generator focused not only on fooling

the discriminator but also on creating outputs that closely resembled the ground truth. This combined loss function resulted in more accurate reconstructions and helped to prevent the generator from discarding important frequency information during training.

## 5.5 Experiments with Leaky ReLU Slope

We experimented with different negative slopes for the Leaky ReLU activation function, specifically testing values of 0.02 and 0.3. The results demonstrated that the model with a slope of 0.3 performed significantly better, especially in terms of waveform clarity and decibel levels. A smaller slope of 0.02 slowed down the learning process, causing the network to make smaller adjustments during training, resulting in lower-quality audio reconstructions. In contrast, the higher slope of 0.3 allowed the model to make larger, more meaningful updates during each training step, leading to improved performance and faster convergence.

This experiment highlights the importance of tuning activation function parameters to ensure that the model can learn effectively. In this case, the higher slope of 0.3 provided a good balance between learning speed and stability, improving the overall performance of the GAN.

## 5.6 Normalization Strategies

Initially, batch normalization was employed in both the generator and discriminator. However, it became evident that this led to training instability, especially with smaller batch sizes. Batch normalization, which normalizes inputs across the entire batch, introduced noise into the training process, resulting in inconsistent gradient updates. To mitigate this, we switched to layer normalization, which normalizes across features within each individual input. This change improved the stability of the training process and led to clearer reconstructions, as the network no longer overfit to specific batch statistics. Additionally, batch normalization seemed to contribute to lower decibel levels in the reconstructed waveforms, likely due to its dampening effect on the low-amplitude frequency components.

## 5.7 Logarithmic Scaling of MFCCs

In one of the later experiments, we switched from using STFT-based spectrograms to logarithmically scaled MFCCs. MFCCs provide a more perceptually meaningful representation of sound, and the logarithmic scaling allowed the model to better handle the wide dynamic range of audio signals. This preprocessing step significantly improved the generator's ability to reconstruct missing frequencies in noisy audio, as the logarithmic scaling helped amplify low-amplitude components that are often critical for human perception.

## 5.8 Final Remarks

Through these various strategies and experiments, we were able to incrementally improve the performance of our GAN model for reconstructing damaged audio files. The combination of dynamic training strategies, activation function adjustments, and preprocessing techniques played a crucial role in enhancing the model's ability to generate clear, high-fidelity audio. By carefully balancing the adversarial loss with the content loss (MSE), we ensured that the generator could both fool the discriminator and accurately reproduce the original audio's frequency content. As a result, the final model, trained with layer normalization, logarithmically scaled MFCCs, and a carefully tuned Leaky ReLU slope, produced the best overall reconstructions. Future work could explore further refinements, such as incorporating Conditional GANs or using advanced loss functions like the Wasserstein loss, to push the boundaries of audio reconstruction even further.

## 5.9 Result comparison:

The comparisons of the sounds have been made.

**MSE (Mean Square Error)**: Measures the average squared difference between the original and generated waveforms. The lower the MSE, the closer the generated waveform is to the original, indicating better reconstruction quality.

**RMSE (Root Mean Square Error)**: The square root of the MSE, representing the error in the same units as the original signal. Like MSE, a lower RMSE indicates that the generated audio is closer to the original. It is often more interpretable since it maintains the same units as the original signal.

**SNR (Signal-to-Noise Ratio)**: Measures the ratio of the original signal's power to the noise's power in the generated signal. A higher SNR value indicates that the generated waveform contains less noise relative to the original signal, implying a more accurate reconstruction.

**LSD (Log-Spectral Distance)**: A measure of the spectral distortion between the original and generated audio in the frequency domain. It is calculated by comparing the logarithmic power spectra of the original and generated signals. Lower LSD values indicate better frequency preservation, as this metric evaluates how well the generated audio maintains the frequency characteristics of the original.

**STOI (Short-Time Objective Intelligibility)**: Evaluates the intelligibility of the generated audio by assessing how well the reconstructed signal matches the original in terms of speech intelligibility. STOI is commonly used in tasks like speech enhancement and audio reconstruction. A higher STOI score (closer to 1) suggests that the generated audio is more intelligible and closer to the original in terms of auditory clarity.

**Table 5.1:** Comparison Results for Each Experiment

| Experiment | MSE | RMSE | SNR | LSD | STOI |
|---|---|---|---|---|---|
| Experiment 1 | 0.0047 | 0.0685 | 0.1554 | 17.0925 | 0.3640 |
| Experiment 2 | 0.0202 | 0.1421 | 0.0580 | 26.1517 | 0.7303 |
| Experiment 3 | 0.0075 | 0.0868 | 0.1677 | 21.3527 | 0.5529 |
| Experiment 4 | 0.0111 | 0.1052 | 0.0670 | 45.7435 | 0.3455 |
| Experiment 5 | 0.0199 | 0.1410 | 0.0033 | 20.2756 | 0.0961 |
| Experiment 6_1 | 0.0198 | 0.1409 | 0.0093 | 23.0172 | 0.4385 |
| Experiment 6_2 | 0.0198 | 0.1408 | 0.0161 | 38.7754 | 0.2548 |
| Experiment 6_3 | 0.0201 | 0.1417 | -0.0414 | 21.8296 | 0.4998 |
| Experiment 6_4 | 0.0199 | 0.1410 | 0.0036 | 20.9818 | 0.4293 |
| Experiment 6_5 | 0.0198 | 0.1409 | 0.0106 | 24.3463 | 0.4207 |
| Experiment 6_6 | 0.0199 | 0.1412 | -0.0118 | 12.3863 | 0.4108 |
| Experiment 6_7 | 0.0199 | 0.1412 | -0.0102 | 18.0141 | 0.2418 |
| Experiment 7 | 0.0197 | 0.1405 | 0.0334 | 14.0605 | 0.5530 |
| Experiment 8 | 0.0181 | 0.1345 | 0.2097 | 9.6679 | 0.5550 |
| Experiment 9 | 0.0190 | 0.1379 | 0.1950 | 8.8990 | 0.5159 |
| Experiment 10_1 | 0.0811 | 0.2848 | 0.0013 | 12.6809 | 0.5886 |
| Experiment 10_2 | 0.0483 | 0.2197 | -0.0028 | 7.6526 | 0.6098 |
| Experiment 10_3 | 0.0017 | 0.0414 | -0.1185 | 11.4884 | 0.7542 |
| Experiment 11_1 | 0.0172 | 0.1311 | -0.0701 | 6.5866 | 0.6517 |
| Experiment 11_2 | 0.0170 | 0.1305 | -0.0117 | 14.5981 | 0.3509 |
| Experiment 11_3 | 0.0170 | 0.1302 | -0.0120 | 13.4141 | 0.3848 |
| Experiment 11_4 | 0.0170 | 0.1305 | -0.0260 | 15.8169 | 0.5218 |
| Experiment 11_5 | 0.0168 | 0.1298 | -0.0113 | 14.8529 | 0.4082 |
| Experiment 11_6 | 0.0171 | 0.1308 | -0.0442 | 15.8698 | 0.2933 |

• **MSE (Mean Squared Error):**

**Best**: Experiment 10-3 (0.0017)

**Worst**: Experiment 10-1 (0.0811)

• **RMSE**: (Root Mean Squared Error):

**Best**: Experiment 10-3 (0.0414)

**Worst**: Experiment 10-1 (0.2848)

• **SNR (Signal-to-Noise Ratio)**:

**Best**: Experiment 8 (0.2097)

**Worst**: Experiment 11-1 (-0.0701)

• **LSD (Log-Spectral Distance)**:

**Best**: Experiment 11-1 (6.5866)

**Worst**: Experiment 4 (45.7435)

• **STOI (Short-Time Objective Intelligibility)**:

**Best**: Experiment 10-3 (0.7542)

**Worst**: Experiment 5 (0.0961)

### Analysis of the Results

#### Balanced Performers

Experiment 10-3 has the best overall performance in MSE, RMSE, and STOI. Although its SNR is negative, it performs well in LSD, showing a balance between different measures.

Experiment 11-1 is also balanced, with competitive results in MSE, RMSE, and LSD, making it a top contender, though the SNR is negative.

Experiment 8 shows a strong balance between SNR, LSD, and STOI, making it a reliable choice for balancing overall intelligibility and fidelity.

#### Analysis of the Best Balanced Performer

**MSE**: Experiment 11-1 has a relatively low value of 0.0172, which is competitive with other well-performing experiments like Experiment 10-3 (0.0017) and Experiment 8 (0.0181).
**RMSE**: Similarly, Experiment 11-1 shows a low RMSE of 0.1311, indicating it performs well in minimizing overall error.
**SNR**: Though it has a slightly negative SNR of -0.0701, this value is still reasonably close to zero, showing that the signal-to-noise ratio is not heavily skewed.
**LSD**: With 6.5866, Experiment 11-1 demonstrates strong performance in maintaining spectral fidelity, second only to Experiment 10-2.
**STOI**: Experiment 11-1 has a STOI score of 0.6517, placing it among the higher scores, meaning that the intelligibility of the reconstructed audio is quite good.

#### Balanced Performer

Given that Experiment 11-1 maintains consistent and competitive results across MSE, RMSE, LSD, and STOI, while only having a slightly negative SNR, we can conclude that Experiment 11-1 is the most balanced in terms of overall performance across all metrics.

#### The Most Balanced Experiment: Experiment 11-1

**Low Errors**: The MSE and RMSE values are consistently low, indicating that the generated

audio is close to the original. Good Spectral Preservation: The LSD is one of the best scores across all experiments, which means it maintains the integrity of the spectral content.

**Intelligibility**: With an STOI score of 0.6517, the generated audio is sufficiently intelligible, even in the presence of noise.

**SNR Trade-off**: While the SNR is slightly negative, this trade-off is acceptable given the balance it strikes with other metrics, particularly in maintaining good intelligibility and spectral content.

In summary, Experiment 11-1 shows the best balance across the board, and based on the results, it is the most well-rounded experiment. This experiment strikes an optimal trade-off between intelligibility, spectral preservation, and low error.

# 6 Conclusion

In this thesis, we investigated the use of Generative Adversarial Networks (GANs) to reconstruct damaged audio files, focusing on old film restoration, where maintaining the authenticity of the sound is essential. We executed and reviewed several GAN designs, such as SpecGAN, Deep Convolutional GAN (DCGAN), and a blend of these with MelGAN. We experimented with different performance-enhancing adjustments, like modifying activation functions, normalization methods, and training approaches.

Our approach aims to find an equilibrium between restoring the lost audio elements and maintaining the original sound's authenticity, which includes the ambient noise typical of old film recordings. We attempted to give the GAN models useful input material while reducing the loss of spectral information by breaking up the audio into consumable segments and turning them into spectrograms or MFCCs for processing.

We conducted a thorough investigation into various GAN models and training approaches, assessing the outcomes with quantitative metrics including MSE, RMSE, SNR, LSD, and STOI. Experiment 11 consistently showed the most balanced results across all metrics, even though some models did particularly well on certain metrics. This highlights how crucial it is to approach audio reconstruction holistically, since concentrating only on one metric—for example, minimizing mean square error—can frequently result in poor outcomes in other domains, including spectral fidelity or intelligibility.

Overall, this study's findings highlight the promise of GAN-based architectures for audio reconstruction in situations where maintaining the original sound—with all of its flaws—is important. Even though there is still room for improvement, especially in terms of fine-tuning the model's capacity to manage extremely diverse and noisy datasets, the experiments offer a solid basis for additional research in this area. To further improve the caliber of audio reconstructions, future studies should investigate more GAN variations, different loss functions, and more reliable assessment techniques.

## 6.1 Future Improvements

In the future, I want to further enhance the results of Experiment 11, which was the first model in this study to effectively use adversarial loss to create the missing frequencies with a high degree of accuracy. Nevertheless, neural networks' stochastic nature frequently poses difficulties, especially when it comes to reliably reconstructing missing frequencies during training. In order to address this, I suggest changing the training phase in a way that would lessen the unpredictable nature of neural network results and increase the dependability of missing frequency production.

In addition to the adversarial loss, the suggested improvement includes dynamically incorporating the Mean Square Error (MSE) between the fed cut audio data and the created output into the generator's loss function. This strategy is innovative in that it conditions the MSE contribution, either on a predetermined number of epochs or at the moment the MSE between the input and the generated output approaches a minimal threshold. The generator would then only depend on the adversarial loss for additional training after this requirement was satisfied, canceling the MSE loss in the process. This dynamic addition and subtraction of the MSE loss should help the model understand the missing frequencies' structure more accurately in the beginning while avoiding an over-reliance on MSE as training goes on.

Several discriminators, each entrusted with acquiring and enforcing unique audio features,

could be another enhancement for upcoming experiments. One discriminator could, for example, be made to specialize in frequency reconstruction, making sure that the frequencies that are produced closely resemble the original ones. In order to make sure that the reconstructed audio has the correct frequencies and volume levels, a second discriminator would assess the recreated audio's decibel (dB) levels. The generator would receive more focused feedback from many angles by merging these two specialized discriminators, which might greatly enhance the learning process.

Thus, the generator could balance its efforts between precisely replicating frequencies and sustaining suitable volume levels. The generator loss would thus be a sum of the adversarial losses from both discriminators. By employing several discriminators, the reconstructed audio could have better aural quality and fidelity due to a more sophisticated reconstruction of its features. The efficacy of this method in yielding more precise and well-rounded audio reconstruction outcomes will be examined in subsequent studies.

# References


[1] S. C. Alec Radford, Luke Metz, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[2] M. P. Chris Donahue, Julian McAuley, "Adversarial audio synthesis," *arXiv preprint arXiv:1802.04208*, 2018.

[3] T. d. B. L. G. W. Z. T. J. S. A. d. B. Y. B. A. C. Kundan Kumar, Rithesh Kumar, "Melgan: Generative adversarial networks for conditional waveform synthesis," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 14 910–14 921.

[4] M. M. B. X. D. W.-F. S. O. A. C. Y. B. Ian J. Goodfellow, Jean Pouget-Abadie□, *Generative Adversarial Nets*, 2014.

[5] N. Upadhyay and A. Karmakar, "Speech enhancement using spectral subtraction-type algorithms: A comparison and simulation study," *Procedia Computer Science*, vol. 54, pp. 574–584, 2015, eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050915013903

[6] N. Upadhyay and R. K. Jaiswal, "Single channel speech enhancement: Using wiener filtering with recursive noise estimation," *Procedia Computer Science*, vol. 84, pp. 22–30, 2016, proceeding of the Seventh International Conference on Intelligent Human Computer Interaction (IHCI 2015). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050916300758

[7] G. Brunner, Y. Wang, R. Wattenhofer, and S. Zhao, "Symbolic music genre transfer with cyclegan," 2018. [Online]. Available: https://arxiv.org/abs/1809.07575

[8] V. Kuleshov, S. Z. Enam, and S. Ermon, "Audio super resolution using neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1708.00853

[9] S. Deshmukh, R. Singh, and B. Raj, "Domain adaptation for contrastive audio-language models," 2024. [Online]. Available: https://arxiv.org/abs/2402.09585

[10] J. Pons, X. Liu, S. Pascual, and J. Serrà, "Gass: Generalizing audio source separation with large-scale data," 2023. [Online]. Available: https://arxiv.org/abs/2310.00140

[11] E. Karamatlı, A. T. Cemgil, and S. Kırbız, "Source separation and classification using generative adversarial networks and weak class supervision," *Digital Signal Processing*, vol. 154, p. 104694, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1051200424003191

[12] T. Ince, S. Kiranyaz, O. C. Devecioglu, M. S. Khan, M. Chowdhury, and M. Gabbouj, "Blind restoration of real-world audio by 1d operational gans," 2023. [Online]. Available: https://arxiv.org/abs/2212.14618

[13] Y. Gong, Z. Xie, G. Duan, Z. Ma, and M. Xie, "Distribution fitting for combating mode collapse in generative adversarial networks," 2024. [Online]. Available: https://arxiv.org/abs/2212.01521

[14] L. B. Martin Arjovsky, Soumith Chintala, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[15] H. Z. K. S. O. V.-A. G. N. K. A. S. K. K. Aaron van den Oord, Sander Dieleman, *WaveNet: A Generative Model for Raw Audio*, 2016.

[16] S. O. Mehdi Mirza, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[17] T. Z. A. A. E. Phillip Isola, Jun-Yan Zhu, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5967–5976.

[18] T. B. Olaf Ronneberger, Philipp Fischer, "U-net: Convolutional networks for biomedical image segmentation," *arXiv:1505.04597*, 2015.

[19] L. B. Martin Arjovsky, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017.

[20] W. Z. V. C. A. R. X. C. Tim Salimans, Ian Goodfellow, "Improved techniques for training gans," in *Advances in neural information processing systems*, 2016, pp. 2234–2242.

[21] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[22] G. H. Rafael Müller, Simon Kornblith, "When does label smoothing help?" *arXiv:1906.02629v3*, 2015.

[23] G. E. H. Jimmy Lei Ba, Jamie Ryan Kiros, "Layer normalization," *arXiv:1607.06450*, 2016.