COE 476 - Project Report

0. Table of Contribution [This table will determine individual grades for each student]

Clearly lay out the contribution of each team member. Please note that the final grades will be assigned based on the amount and significance of contribution by each team member.

Team member AUS ID	**Team member name**	**Team member's contribution to the project**	**Percent Effort (out of a total of 100%)**
b00087338	Abdelrahman Ahmed	Results, validation. Main implementation of GAN, neural network architecture selection	60%

b00088535	Yousef Mureb	Decription of the problem,data cleaning and pre processing neural network architecture, why do neural networks work	25%
b00071637	Mohamad Mhmndar	Previous work	15%

1. Description of the Problem

Music generation has been an important field in deep learning. With the recent advancement of generative models, the music industry has been affected by this heavily. Numerous applications of music generation apply to piano, guitar, pop culture, and more. With that said music generation on Oud has been kept to a minimum. The Oud instrument is heavily used in Arabic music which makes this research suited for the Middle East. This problem is hard to solve due to the fact that there are no current trained models that generate Oud samples and we intend to solve this by creating a trained model. Moreover, Generative Models for music applications tend to be tricky to evaluate since music is subjective from one person to another. As a result, understanding whether a trained model gives a "correct answer" varies. Evaluating generative NN models is a major challenge that we intend to research.

The Problem is important for the following reasons:

Cultural Preservation: The automatic generation of oud music can contribute to the preservation and dissemination of cultural heritage, making this traditional music more accessible and allowing for novel compositions in authentic styles.

Technical Challenge: Music generation poses unique challenges in terms of capturing the intricacies of rhythm, melody, and harmony that are characteristic of oud music. This includes the subtle nuances within the complex strings.

Innovation in AI and Music: Tackling this problem contributes to advancements in AI, particularly in understanding and creatively applying deep learning to the domain of music, an area where emotional

and artistic expressions are paramount. Moreover, there are no existing generative models that are pre-trained on the Oud dataset.

The project also has the following complexities:

The richness of the Oud's Sound: The oud produces a wide range of tones, which are difficult to replicate and generate authentically through synthetic means.

Improvisational Nature: Much of oud music is improvisational, requiring a system to not only learn structured patterns but also to creatively generate music that maintains the natural flow and spontaneity of live playing.

Data Scarcity: There is a relative scarcity of comprehensive and well-annotated datasets of oud music, which poses a challenge for training deep learning models that typically require large amounts of data.

Musical Complexity: Oud music often involves complex musical structures, intricate playing techniques, and microtonal scales, all of which add layers of complexity to the automatic generation process.

By addressing these challenges, the project aims to not only create a system capable of generating oud music but also contribute to the understanding of how deep learning can intersect with artistic domains to produce creative and culturally significant outcomes.

2. Previous Approaches

GRUV: Algorithmic Music Generation using Recurrent Neural Networks:

[1] Nayebi, Aran, and Matt Vitelli. "Gruv: Algorithmic music generation using recurrent neural networks." *Course CS224D: Deep Learning for Natural Language Processing (Stanford)* (2015): 52.

The GRUV research project utilizes Recurrent Neural Networks (RNNs) to innovate in the realm of algorithmic music generation. Specifically, the project employs two types of RNN gating mechanisms: Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRUs), both adept at handling sequence prediction tasks. In this case, the sequence being predicted is a series of audio waveforms, making the task a regression problem. By framing the challenge as regression, the model gains the flexibility to work with audio waveforms from any instrument. The LSTM model, in particular,

demonstrated promising results, producing musically coherent sequences. However, the GRU variant primarily generated white noise, lacking the musicality of the LSTM outcomes. The study did not venture into crafting a custom model from scratch, mainly due to the limitations posed by the absence of GPU acceleration when dealing with large datasets. Instead, it leveraged the Keras library, which provided pre-built LSTM and GRU structures that could be trained and tested for performance. An interesting observation emerged when the input complexity increased, such as mixing multiple instruments. The model's predictions diverged from the input's musical qualities, suggesting a difficulty in capturing the nuanced interplay of composite audio. After extensive training—around 2000 epochs—the loss metrics for both GRU and LSTM hovered around 0.8 for a dataset featuring music by David Bowie. When experimenting with different sizes for the hidden layers, specifically a dimensionality of 1024, the LSTM reported a development (Dev) loss of 1.165, while the GRU posted a slightly lower Dev loss of 1.15. To better understand the frequency components of the generated music, spectrograms were created. These visualizations revealed that the GRU's frequency output frequently fell outside the 4000-7000 Hz band, which could account for its noisier audio output compared to the LSTM's more harmonious results.

Unsupervised Lead Sheet Generation via Semantic Compression:

[2] Novack, Zachary, et al. "Unsupervised Lead Sheet Generation via Semantic Compression." *arXiv preprint arXiv:2310.10772* (2023).

The research paper titled "Unsupervised Lead Sheet Generation via Semantic Compression" introduces a novel approach to generating lead sheets—a condensed form of music notation consisting of melody, lyrics, and harmony—using an unsupervised machine learning model. The model at the core of this solution is the Multitrack Music Transformer (MMT), which processes inputs that are rich in musical detail; each note is characterized by a multi-dimensional array including attributes such as type, beat, position, pitch, duration, and instrument. The generation process centers on transforming complex multitrack sequences into simplified, information-rich lead sheet representations. The chords in the input are also meticulously labeled. The methodology employs four principal models: An MMT that specializes in converting scores to lead sheets, and learning to select notes and chords that best reconstruct the original music. An Objective Approximation model that ensures the generated lead sheets are as close to optimal as possible. A Lead-to-Score architecture based on an encoder-decoder transformer for the reverse conversion. Pretraining modules utilize the skyline algorithm to optimize model training without necessitating an excessively large training model. The performance of the models was rigorously evaluated through three distinct methods: automated evaluation metrics, a Human Listening Study, and a Human Reading Study. The model underwent pretraining on extensive datasets—POP909 and SOD—resulting in high performance. However, Lead-AE, the model designed for generating lead sheets, exhibits limitations when applied to complex musical compositions featuring a multitude of instruments with subtle leading parts. Such intricacies pose challenges to the model, which is best suited for music with distinct, recurring lead components. The researchers highlight the lack of available datasets that pair full scores with human-annotated lead sheets, noting this as a constraint for broader application across diverse musical genres. For the first evaluation, the model's effectiveness was gauged by its ability to reconstruct the original score from the generated lead sheet. Metrics such as the Music Transformer Evaluation (MuTE) score, which assesses the F1-score over time steps, and the global Jaccard index, were recorded. With a learning rate set at k=0.1, Lead-AE achieved a MuTE score of 64.04 and a Jaccard index of 46.59. In the Human Listening Test, 12 individuals with varying musical expertise were presented with 11 excerpts to assess the generated lead sheets against the baseline skyline lead sheets. A majority of 62.1% of annotators favored the accuracy of the generated sheets, with 60.6% acknowledging their fluency. The Reading Study, focused on the interpretability of the sheets, received a positive accuracy response from 80% of the annotators, while 56.7% concurred on fluency. These findings suggest that

while the model demonstrates proficiency in lead sheet generation within its optimized domain, its application might be limited when confronting a wider spectrum of musical genres and complexities.

Multi-Genre Symbolic Music Generation using Deep Convolutional Generative Adversarial Network [A]:

> [3] C. Chauhan, B. Tanawala, and M. Hasan, "Multi-genre symbolic music generation using deep convolutional generative Adversarial Network," *ITM Web of Conferences*, vol. 53, p. 02002, 2023. doi:10.1051/itmconf/20235302002

This study uses a deep convolutional generative adversarial network (DCGAN) for their architecture in order to produce samples of polyphonic MIDI in different genres. The dataset used consisted of 500 MIDI tracks, 100 for each of the five genres. The genres are Techno, Latin, Trap, Slap house, Trance & Techno, and Rhythm & Blue. In order to present the data to the network, the python library MusPy was used to convert the data into a matrix form with a shape of (128x192) for a piano-roll. The pitch 0-127 range is represented using the 128 rows. While the 192 columns represent the time in beats. For the preprocessing, only 4 pitch octaves were considered useful, giving a total of 48 notes. The five genres were layered and as a result, the data was reshaped into (M, 48, 48, 5) and M is the number of training samples. The input for the generator network is a noise vector that has a (100,) shape which gets passed to the dense layer in which the output gets reshaped into (6, 6, 256). Furthermore, the transpose convolution operations are achieved using 3x3 kernels. In addition, batch normalization with a momentum rate value of 0.9 is used to normalize the weights along with a ReLU activation function to guarantee non linearity. Finally, the output of the generator is a (48, 48, 5) tensor which goes through the Tanh activation function. On the other hand, the discriminator architecture uses the piano-roll roll that was reshaped into (48, 48, 5) earlier. Similar to the generator, it also utilizes 3x3 kernels for its convolution operations. Moreover, to aid with regularization it uses leaky ReLU along with a dropout layer. Finally, the prediction of whether the sample is a real or fake sample is generated through a sigmoid activation function. Moreover, the generator uses the results of the discriminator as feedback in order to help it improve; thus, making the model improve itself through learning. Furthermore, both the generator and discriminator used the adam optimizer for training using a learning rate value of 0.000005 and were trained on two Nvidia Tesla T4 GPUs for 200,000 epochs. In addition, the generator had to be updated five times than that of the discriminator to be able to achieve the Nash equilibrium. Moreover, human evaluation was used as a metric to rate the results of the DCGAN. A survey exclusive to music professionals such as music composers, music directors, mixing engineers, mastering engineers, and music teacher, was used to evaluate the generated samples based on musical aspect as well as the emotional aspect of the samples. Regarding the relation of the samples to its particular genre, the average rating was 3.7 out of 5 for each sample; implying that the samples have captured the elements of their genres. Likewise, the survey results showed that most samples generated by the model were both engaging and expressive. On the other hand, the Latin genre samples had low scores regarding engagement and expressiveness.

Music generation and human voice conversion based on LSTM:

> [4] G. Li, S. Ding, Y. Li, and K. Zhang, "Music Generation and Human Voice Conversion based on LSTM," *MATEC Web of Conferences*, vol. 336, p. 06015, 2021. doi:10.1051/matecconf/202133606015

This study utilizes a long short-term memory generation model to generate piano music while combining it with an audio cutting technique to convert the generated piano samples into human audio. The LSTM model had four gates which are, input gate, output gate, forget gate, and update gate. The model was trained with a dataset containing a huge number of piano music MIDI. Furthermore, the model also

implements a gradient checkpoint method to reduce memory consumption and consequently performing more computations. Furthermore, the piano sequence is filtered to isolate single-key notes. Then, the filtered data is segmented to align with specific frequencies of human voice audio. Moreover, these filtered segments are then aligned with the different frequencies of the human voice yielding the relationship between the human voice and the piano music in the form of a matrix. By mapping and sequentially numbering the location of each note on the real piano keys and within each column of the resultant piano music matrix, direct correlation is established. As a result, for each piano key pressed, its corresponding tone produces a specific human voice audio. After defining a corresponding relationship, the LSTM model generates a piano music sequence. This sequence is used to gather human voice information by connecting related audio segments to form complete voice audio. Finally, these audio pieces are spliced together according to the piano music sequence to create the desired vocal audio representation of the piano music. Further, two distinct methods for processing human vocal information were used, which are Individual Recording (IR) and Overall Recording (OR). The vocal fragments corresponding to each note are obtained using segmentation and slicing and aligned with the human audio segments. The IR method resulted in an audio segment in which the integrity of it is preserved; however, each segment contains only one note, hence causing losing the fluidity post-splicing. On the other hand, the results of the OR method had smoother waveforms. One thing to note is that this approach required exact recording times to guarantee that even after slicing, each audio segment contains one vocal note. Consequently, this results in achieving finer refinement of the quality of the human voice.

Chinese style pop music generation based on recurrent neural network:

> [5] J. Wang and C. Li, "Chinese style pop music generation based on recurrent neural network," *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2022. doi:10.1109/imcec55388.2022.10019796

The aim of this study is to generate Chinese style pop music, which is underrepresented in the field of automatic music generation compared to Western classical music. The study utilizes a recurrent neural network (RNN) to learn the characteristics of Chinese style pop music using a structured dataset. The purpose of the model is to generate samples that correspond to the rhythm and harmony of Chinese pop music. The study implements polyphony piano transcription technology to convert the music from the MP3 to the MIDI format, and then it represents and preprocesses this data to create a dataset suitable for training the RNN model. Moreover, the model used is a convolutional recurrent neural network capable of high-resolution piano transcription. The study utilizes a model which uses four convolutional layers, and their channel numbers are 48, 64, 96 and 128 along with a bidirectional gated recurrent unit (GRU) which has 256 layers. Furthermore, the purpose of the model is to convert MP3 audio samples into a logarithmic Mel spectrum. Through this process, a total of 148 MIDI samples of Chinese style piano music was created. In order to represent the music properly and quantify its information such duration, pitch, tempo, velocity and position in MIDI, representation of media (REMI) was used. Then, this information is converted into a python list in order to be fed to the generating neural network. Moreover, the data is also one-hot encoded resulting in a two-dimensional one-hot matrix which represents the length of the music and the vocabulary size. The model responsible for generating the Chinese pop-style music is based on a long short-term memory architecture consisting of two layers. This model is fed the one-hot matrix as its input, which then is outputted to the dropout layer and finally reaching the dense layer which utilizes a

SoftMax activation function resulting in a probability matrix of two dimensions. Moreover, this model was trained on a machine running on an Intel Xeon 4210 CPU, 32GB of RAM and an Nvidia RTX 3090 GPU for 50 epochs in batches of 4. The resultant training loss came out to 1.085. Finally, the generated samples were evaluated based on the pentatonic scale and the linear feature of the melody. The verdict of the study was that the model was able to generate some Chinese style pop piano music, albeit experiencing random and uncontrollable generation.

PopMNet: Generating structured pop music melodies using neural networks

[6] J. Wu, X. Liu, X. Hu, and J. Zhu, "PopMNet: Generating structured pop music melodies using Neural Networks," *Artificial Intelligence*, vol. 286, p. 103303, 2020. doi:10.1016/j.artint.2020.103303

This study uses a Structure Generation Net (SGN) based on a Convolutional Neural Network (CNN) coupled with a Melody Generation Net (MGN) based on a Recurrent Neural Network (RNN). The purpose of the SGN is to generate melody structures while the MGN creates melodies that are based on these generated structures and chord progressions. The model utilizes the General MIDI standard for melody representation. In addition, it uses a one-hot vector format to represent notes and pauses, with each note and rest quantized into 16th note segments for accurate timing. This method results in the creation of a token sequence that corresponds to each measure of the melody. Moreover, PopMNet focuses the connections between various bars in a melody, and classifies them into either repetition or sequential patterns. The MGN utilizes these patterns, adapting the rhythm of the original bars in order to create new ones. It includes components for extracting chord progression features and source melody features, as well as a melody generator, all of which are based on bidirectional Gated Recurrent Units (GRU) layers. Furthermore, the dataset used was obtained from Wikifonia and contains 3859 MusicXML format lead sheets. Moreover, the dataset was divided as follows: 90% of the data was for training, and 10% was for validation. Finally, the quantitative measurements showed that both PopMNet and Music Transformer closely resembled actual data regarding repetition and rhythm sequence percentages. To evaluate the subjective quality of the music they generated, experiments involving human participants were carried out. These experiments evaluated the melodies based on their pleasure, realism, smoothness, and integrity. The tests concluded that PopMNet performs better than four other existing models by generating melodies that are both better quality and clear structures.

[7] H.-W. Dong and Y.-H. Yang, "Convolutional Generative Adversarial Networks with Binary Neurons for Polyphonic Music Generation," *arXiv (Cornell University)*, Apr. 2018, doi: 10.48550/arxiv.1804.09399.

The authors of this paper state that since music is temporal, a temporal model is needed to generate it. They are trying to generate musical pieces that include multiple instruments, and each instrument can be time independent of each other. The idea that each instrument can be generated independently of one

another is taken from ‘jamming’- a practice where different instruments improvise. Thus, they created a jamming model. The jamming model creates different instrumental pieces concurrently but independently, using a different latent vector for each one. Essentially, there is a different generator for each instrument

A composer model is also made where all the instrumental pieces are composed from the same latent vector. The generator generates multi-track music (multiple instruments concurrently from the same generator), which is then fed into one discriminator (uses multiple in jamming). They also have a hybrid model where a constant latent vector z is added to the jamming vectors, generating different tracks concurrently (with the latent vector acting as the composer’s will), then feeding these tracks to a single discriminator.

Not only did they generate music from a latent vector but allowed for collaboration between human/user and the model by allowing them to input a bar sequence into the model. The model then tries to learn that track and tries to generate the rest of the song based on it.

Their models seem to have an understanding of the aspect of time. They also conducted a user survey where the hybrid model seemed to always outperform the others. The performance was above average in the user’s opinion based on: H: harmonious, R: rhythmic, MS: musically structured, C: coherent, OR: overall rating

[8]“Shibboleth Authentication Request.” https://www-sciencedirect-com.aus.idm.oclc.org/science/article/pii/B9780128197141000245#s0110

The paper talks about the different approaches taken to generate music. The authors start by describing non-adaptive AI implementations such as L-systems: generate string sequences of symbols belonging to a given alphabet, based on substitution rules over an initial sequence. The paper covers explicit and implicit AI modelling. For example, probabilistic generative models can capture probabilities of certain elements occurring in a dataset. One such example can be the hidden markov models. It also goes in depth regarding implicit models such as early RNN adaptations that were able to learn scales and generate melodies but struggled to capture long term structure. LSTMs selectively forget information from the past which allows them to learn and generate longer term structures. Finally, the paper details evolutionary techniques- inspired by Charales Darwin.

Previous work	Features (eg architecture)	Strengths	Weaknesses	Results
GRUV: Algorithmic Music Generation using Recurrent	The GRUV project focuses on algorithmic music generation using	The LSTM model produced musically coherent sequences,	The GRU model primarily generated white noise, indicating a	After training the model for 2000 epochs, the loss for the GRU was

Neural Networks	Recurrent Neural Networks (RNNs). Utilizes Long Short-Term Memory (LSTM) units and Gated Recurrent Units (GRUs) for sequence prediction of audio waveforms. Approaches music generation as a regression problem, allowing for flexibility across different instruments. Employs Keras library implementations of LSTM and GRU for model training.	showcasing its potential for music generation. The study demonstrated the RNNs' capabilities in learning and generating complex audio waveform sequences. Avoided the need for custom model development by using established Keras library structures.	lack of musical coherence. The model struggled with increased input complexity, like the interplay of multiple instruments. Limited by the absence of GPU acceleration for handling large datasets. The model's loss metrics did not significantly improve even after a large number of epochs (2000). .	near 0.8, and for the LSTM it was near 0.8 for the David Bowie dataset. For different sized hidden layer dimensions such as 1024, the LSTM model had a Dev loss of 1.165 and the GRU Dev loss was 1.15 after 2000 epochs. A spectrogram was created to visualize the range of frequencies that were generated. For the GRU, frequency generation was outside the band range of 4000-7000 which explains why the GRU method was noisier than the LSTM
UNSUPERVISED LEAD SHEET GENERATION VIA SEMANTIC COMPRESSION	Utilizes a Multitrack Music Transformer (MMT) for generating lead sheets. Process notes as multidimensional inputs, including attributes like type, beat, position, pitch, duration, and instrument. Employs four models: MMT for score-to-lead conversion, Objective	Capable of generating musically plausible lead sheets by learning from a variety of music notes and chords. High performance due to pretraining on large datasets (POP909 and SOD). Evaluation includes comprehensive methods: automated metrics and human-centric	The Lead-AE model may not perform well on complex music with multiple instruments and subtle leads. Limited applicability across diverse musical genres due to dataset constraints. Current datasets lack paired scores and human-annotated lead sheets for extensive training.	MuTE score: 64.04 Jaccard index = 46.59 Human listening = 62.1%

	Approximation model, Lead-to-score architecture, and a module pretraining using the skyline algorithm	studies. l.		
Multi-Genre Symbolic Music Generation using Deep Convolutional Generative Adversarial Network	This study uses a deep convolutional generative adversarial network (DCGAN) in order to produce samples of polyphonic MIDI in five different genres. The dataset used consisted of 500 MIDI tracks, 100 for each of the five genres. The architecture implemented was a generator discriminator architecture in which the generator generates samples while the discriminator classifies them as real or fake helping the DCGAN improve through learning	The capability of the system to create music across various genres represents a notable strength, broadening the range of MIDI music production possibilities. In addition to that, the generator discriminator architecture allows the model to improve through learning. The study also used a robust evaluation method. Surveying professional music experts to evaluate the samples and getting good feedback regarding the model's effectiveness in making the music expressive of its genre	Although the model is decently capable in four different genres, it still somewhat struggles in Latin music. Not to mention it is not capable of generating samples for other genres of music out there. In addition, The study concluded the need for further improvement in some areas; namely, arbitrary-length music generation and the harmony and flow of the music.	The average rating for the model in terms of relation of the samples to the genres was 3.7 out of 5 for each sample meaning that the samples have captured the elements of their genres. In addition, most samples generated by the model were both engaging and expressive. On the contrary, the Latin genre samples had low scores regarding engagement and expressiveness.
Music generation and human voice conversion based on LSTM	This study uses an LSTM neural network in combination with audio cutting	The ability of processing information on different scales	One big issue when it comes to this model is that the longer the sequence in	The model was able to successfully convert piano music into human

	techniques in order to generate piano samples and convert these samples into human voice audio samples. The LSTM used four main gates; namely, input gate, output gate, forget gate and update gate.	and being able to deal temporal details as well as historical information is a strength of the LSTM model used. Moreover, the simpler structure of the neural network allows it to perform better than other neural networks in terms of generating audio samples. Not to mention that general neural networks struggle when it comes to complex operations and processing speed.	training, the model's memory consumption increases along with it, which had to be dealt with using gradient checkpoint. Another limitation of the model is that it can only deal with single-key piano samples, meaning it will struggle with complex piano pieces.	voice. Moreover, The IR method was able to preserve the integrity of the audio segment but only containing a single note. OR on the other hand had smoother waveforms. The results show that the LSTM model used in the study can represent samples of piano music as human audio.
Chinese style pop music generation based on recurrent neural network	The aim of this study is to generate Chinese style pop music. The study utilizes a recurrent neural network (RNN) to learn the characteristics of Chinese style pop music using a structured dataset. The study uses Polyphony Piano Transcription Technology to convert MP3	The idea of the study is novel as chinese style pop music which has not been done much in the field of neural networks. Moreover, The piano transcription and REMI representation successfully capture elements of Chinese style pop music, allowing the RNN to learn and produce music that maintains the distinctive features of the style. In addition, the use	The model had issues with experiencing random and uncontrollable generation which can hinder the model's generated samples' quality. The model is also specific to this genre of Chinese pop music, making it limited to the many other genres of music.	The LSTM network was capable of generating 16 bars of music saved in MIDI format. Upon analysis, these generated samples aligned with the pentatonic scale and exhibited features typical of Chinese style pop music. The samples were also manually evaluated and the verdict was that the model's samples capture the identity of chinese style pop music.

	samples to MIDI format. This allowed the researchers to generate a dataset and convert it to python lists to feed it to the generating neural network.	of one-hot encoding improved the efficiency of the music generation.		
Musegan	Multi-track GANs; either jamming (multiple generators to multiple discriminators) or composer(one composer to one discriminator), or a hybrid between the two.	Can produce multi-track pieces using any of the three models with satisfactory results	conveying/ detecting vibes in music is still a challenge	

3. Data Selection

Oud datasets have yet to be compiled due to the low research this field has. To solve this, we used a two-hour-long video on YouTube of oud music and segmented the video into three-second samples. This problem could have been solved using the MusicCaps dataset. However, this dataset only includes 28 samples for oud music out of the 5,521 examples ranging from piano and pop, etc. Due to the unavailability of a downloadable dataset online, we intend to create the dataset using YouTube videos. Moreover, the availability of oud music on YouTube is vast making it possible to decide on the style of oud music and having access to a large number of samples. To be specific, we used the following YouTube video link to sample the dataset.
https://youtu.be/CcZw2yeKZho?si=qsfagXOlfeo0sZvN

https://youtu.be/_RVT5wlYPsU?si=IVJipbpwTWD-Rdk3

It is important to note that there are many YouTube compilation videos of oud music. For the project, we will use two one-hour-long videos and sample them as mentioned below. The chosen videos contain high-quality oud tunes with very low no-frequency sounds. This makes our training set more reliable. The first YouTube link also displays the work of a popular oud artist (Naseer Shamma), where the quality of the samples is high. There is also a richness in tone, rhythm, and harmony with these samples making it very suitable for feeding it into a generative model. If we see that our model constantly overfits even after fixing the architecture such as adding pooling layers and regularization, we can factor in the fact that only two videos were made with a total of around 2300 samples.

4. Data Cleaning and Feature Engineering

As mentioned above, we will make use of two youtube videos as our dataset. The following python script demonstrates how we can get three second samples from the youtube video.

```
import os
```

```
import librosa

import numpy as np
```

```
import soundfile as sf


def chop_audio(filenames, segment_length=3, output_folder="segments"):
    """
    Chops multiple audio files into segments and stores them in a single folder.

    Parameters:
    - filenames: List of paths to audio files.
    - segment_length: Length of each segment in seconds.
    - output_folder: Directory to store all segments.
    """
    # Ensure the output folder exists
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)


    for filename in filenames:
        # Load the audio file
        y, sr = librosa.load(filename, sr=None)


        # Convert segment length to samples (frames)
        segment_samples = segment_length * sr
        num_segments = len(y) // segment_samples
```

```
        # File-specific prefix for segment filenames
        base_filename = os.path.splitext(os.path.basename(filename))[0]

        for i in range(num_segments):
            start_sample = i * segment_samples
            end_sample = (i + 1) * segment_samples
            segment = y[start_sample:end_sample]

            # Export each segment with a unique filename
            segment_filename = os.path.join(output_folder, f"{base_filename}_segment_{i}.wav")
            sf.write(segment_filename, segment, sr)

        print(f"Created {num_segments} segments for {filename}.")

def remove_silent_segments(segment_folder="segments", threshold=0.007):
    """
    Removes silent or near-silent audio segments based on RMS energy.

    Parameters:
    - segment_folder: Path to the directory containing the audio segments.
    - threshold: RMS energy threshold below which a segment is considered silent.
```

```
    """
    # List all files in the segment folder
    segment_files = [f for f in os.listdir(segment_folder) if
os.path.isfile(os.path.join(segment_folder, f))]

    for segment_file in segment_files:
        filepath = os.path.join(segment_folder, segment_file)

        # Load the segment
        y, sr = librosa.load(filepath, sr=None)

        # Compute the RMS energy
        rms_energy = np.sqrt(np.mean(y**2))

        # If RMS energy is below the threshold, delete the segment
        if rms_energy < threshold:
            os.remove(filepath)
            print(f"Removed silent segment: {segment_file}")

    print("Silent segment removal complete.")

# Example usage
if __name__ == "__main__":
    filenames = [
```

```
"C:\\Users\\User\\Desktop\\ProjectNeuralNetworks\\DataSample1.mp3",


"C:\\Users\\User\\Desktop\\ProjectNeuralNetworks\\DataSample2.mp3.mp3"

        # Add more file paths as needed

    ]

    chop_audio(filenames)

    remove_silent_segments()
```

Important to note is that an online tool was used to convert YouTube videos into a . Wav file. https://ytconverter.app/en1/youtube-to-wav/
This is the link for the tool used. After passing the link, we downloaded the youtube video as a .wav file.

Key points with the above script:

- The user enters the path of the video and a directory is made that creates the segments. Each segment is 3 seconds.
- Bad-quality samples with low frequency are removed.
- We also use an nfft of 1300 since the minimum value for the samples required was 1200

```
if rms_energy < threshold:
```

```
            os.remove(filepath)
            print(f"Removed silent segment: {segment_file}")

    print("Silent segment removal complete.")
```

- We compare the RMS values with a specific threshold to see if the samples match it.

This eventually gives us a directory with all the samples. After getting the samples, we convert the .wav files into a suitable format to be fed into a neural network. For this, we follow the notes done in class for

audio processing. We convert the .wav files into a mel spectrogram to be used in a neural network. Lastly, an appropriate input and output pair samples are created. The code for this is the same as the ones in class which is the following script:

```
import tensorflow

from tensorflow.keras.utils import to_categorical

from tqdm import tqdm

from python_speech_features import mfcc, logfbank

import librosa



# summing all the lengths and dividing by 0.1 gives

# us the total number of 0.1 second segmentws

n_samples = 2 * int(df['length'].sum()/0.1)



# make fname the index again



# Building and normalizing data

def build_random_feat():



    desired_number_of_frames = 30

    X = []

    y = []

    _min, _max = float('inf'), -float('inf')

    for _ in tqdm(range(n_samples)):
```

```
        # randomly generate a label based on distribution
        rand_class= np.random.choice(class_dist.index,p=prob_dist)
        # randomly select one record of the class selected
        df_rand = df[df.label==rand_class]
        file = np.random.choice(df_rand.index)
        # read the wav file
        rate,wav                                                    =
wavfile.read("C:\\Users\\User\\Desktop\\ProjectNeuralNetworks\\segments\\"
+file)
        # find the corresponding label
        label = df.at[file,'label']
        # get a random sample of 0.1 seconds from the sample
        # subtract 0.1 second to make sure we always get
        # at least one second
        random_index = np.random.randint(0,wav.shape[0]-config.step)
        sample = wav[int(random_index):int(random_index+config.step)]
        # Now create the feature set and transpose it because we
        # want time to be the x-axis and mfcc to be y


        # Ensure each sample is of fixed length
        if len(sample) < config.step:
            sample  =  np.pad(sample,  (0,  config.step  -  len(sample)),
'constant')
        elif len(sample) > config.step:
```

```
            sample = sample[:config.step]



        X_sample = mfcc(sample,
                        rate,
                        numcep = config.nfeat,
                        nfilt = config.nfilt,
                        nfft = config.nfft)
        # Update min and max



        fixed_size = [desired_number_of_frames, config.nfeat]



        if X_sample.shape[0] < fixed_size[0]:
            pad_width = fixed_size[0] - X_sample.shape[0]
            X_sample = np.pad(X_sample, ((0, pad_width), (0, 0)),
'constant')
        elif X_sample.shape[0] > fixed_size[0]:
            X_sample = X_sample[:fixed_size[0], :]


        __min = np.amin(X_sample)
        __max = np.amax(X_sample)
```

```
        if(__min<_min):
            _min = __min
        if(__max>_max):
            _max = __max

        # add X and y to correponding lists
        X.append(X_sample)
        # add the integer label correponding to the class
        y.append(classes.index(label))

    # convert into numpy arrays

    X, y = np.array(X), np.array(y)

    # normalize X
    X = (X - _min)/ (_max-_min)

    # Reshape for the NN
    X = X.reshape(X.shape[0], X.shape[1], X.shape[2], 1)
    y = to_categorical(y,num_classes=10)

    return X, y
```

```
# actually build it



X, y = build_random_feat()
```

5. Neural Network Architecture Selection

For this project, we will make use of a Generative Adversarial Network (GAN) specifically, a deep (unconditional) convolutional GAN. From the research we did, GANS perform very well for generative tasks and provide promising results. GANs are particularly well-suited for this task due to their two-component structure:

Generator: Learns to create data that mimics the real samples from the training set.

Discriminator: Learns to differentiate between real and generated samples.

This adversarial setup should enable the network to produce new music that is indistinguishable from genuine oud recordings.

Characteristics Influencing Architecture Choice:

Temporal Complexity: if we were to train and attempt to generate oud samples longer than three seconds, we would have had to make use of LSTMs or another form of attention. This way, the model would be able to "remember" the older notes/frequencies played and generate new ones based on them. However, since we are focusing on shorter samples, the need for such architecture is not present

Data Diversity: The discriminator's convolutional layers are effective in identifying subtle nuances across diverse training examples, ensuring that generated music is varied yet authentic.

We also could have used variational autoencoders (VAEs) to generate new music. This would be done by training a VAE using samples and then generating a new 'sample' by choosing a random point in the VAE's latent space. Although this may allow us to generate samples that are very close to our original samples (given that we chose a reasonable latent space), it may not be as high-quality as a GAN. The continuous latent space of a VAE makes it attractive to discover the diverse possibilities, but it would likely be outperformed by a GAN that has been trained on enough data in terms of quality of the produced audio.

The following provides the three main neural network architectures that we will use in order to generate m

Architecture	Components	Characteristics	Reason for Selection
Generative Adversarial Network (GAN)	Generator and Discriminator (Convolutional)	Generator captures temporal dynamics, Discriminator assesses authenticity	Proven ability in generating complex data such as images and audio
Convolutional Neural Networks (CNN)	Convolutional layers for feature extraction	Spatial hierarchy of features, efficient pattern recognition	Effective in classifying and differentiating complex data patterns

Since our task is to generate samples, GANS provide the most efficient method for generating oud music.

6. Why do chosen NN Architectures Work

The selected neural network architectures are expected to perform effectively for oud music generation based on their inherent strengths and proven capabilities in learning and modeling complex patterns. Here's a detailed explanation for each architecture:

Generative Adversarial Network (GAN)

Basic Idea: GANs consist of a generator and discriminator that are trained simultaneously through adversarial processes. The generator creates outputs that are as close as possible to the real data, while the discriminator becomes better at distinguishing real data from the fake data produced by the generator.

Expected Efficacy: The adversarial training setup allows GANs to produce high-quality outputs that are often indistinguishable from the real data. For oud music, this means generating new audio samples that mirror the authentic sound and style of traditional oud compositions.

Convolutional Neural Networks (CNN)

Basic Idea: CNNs are adept at extracting hierarchical spatial features from data, such as images and, by extension, spectrograms. They employ filters to capture local dependencies and can learn increasingly abstract representations.

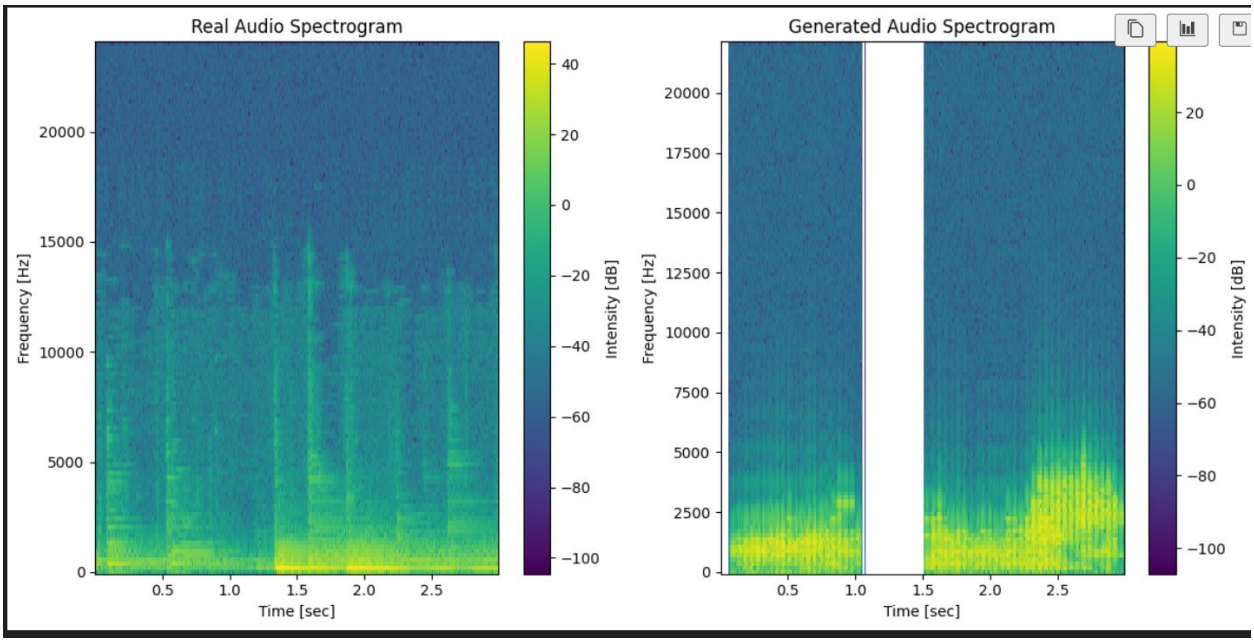
Expected Efficacy: For oud music generation, CNNs can efficiently process and differentiate the spectral features of audio signals, making them valuable for the discriminator in identifying the authenticity of generated samples.

NN Architecture	Basic Idea	Why It Works for Oud Music Generation
GAN	Two-part adversarial network with one part generating data and the other evaluating it.	Enables the production of new music samples that closely mimic authentic oud recordings.
CNN	Uses filters to capture local dependencies and learn abstract data representations.	Efficient at processing and learning from the spectral content of oud music for authenticity assessment.

7. Validation Methodology

If we were not using GANs, we may have used ROC curves, f1-scores, etc. If we were generating images, we could have used FID, inception-score and so on. Since we are generating audio, it is harder to compare objectively. That being said, we can use spectrograms to compare the generated audio to the original sample. We will do this by:

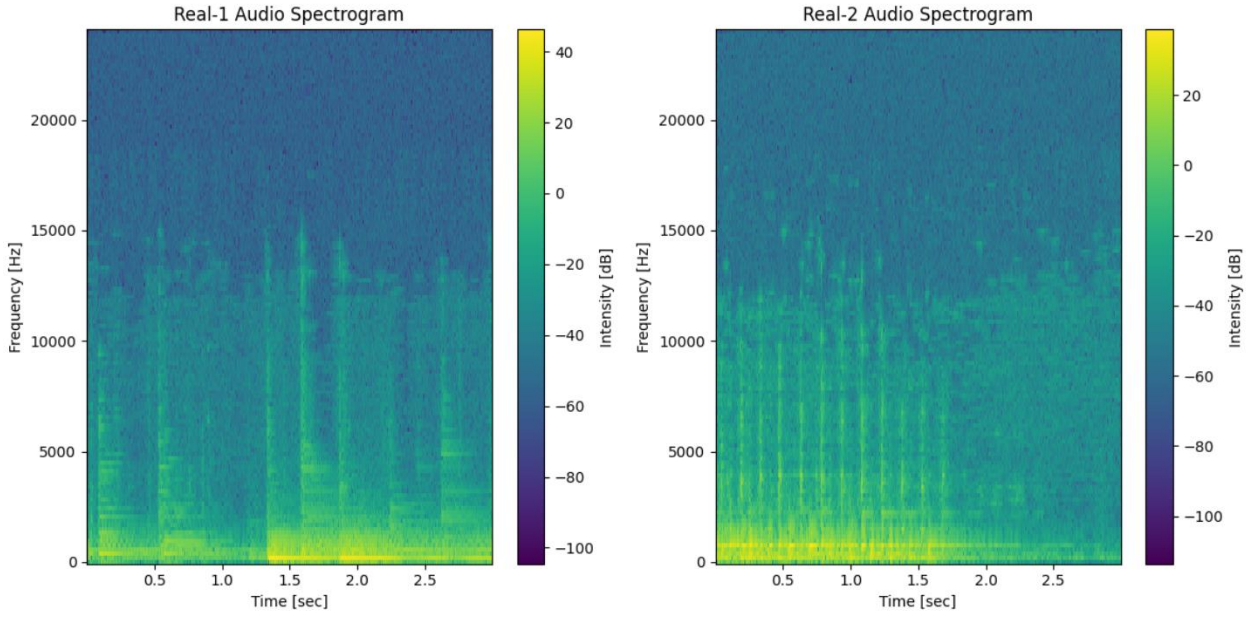
1) Drawing the spectrogram of a sample and of a generated audio clip:



We can then use this in multiple ways, one of which is to calculate the euclidean distance: Euclidean distance between the spectrograms which in our case was 173495.61.

Although it is large, we thought we should also compare two real audio samples to each other:

2) Comparing two real samples' spectrograms:

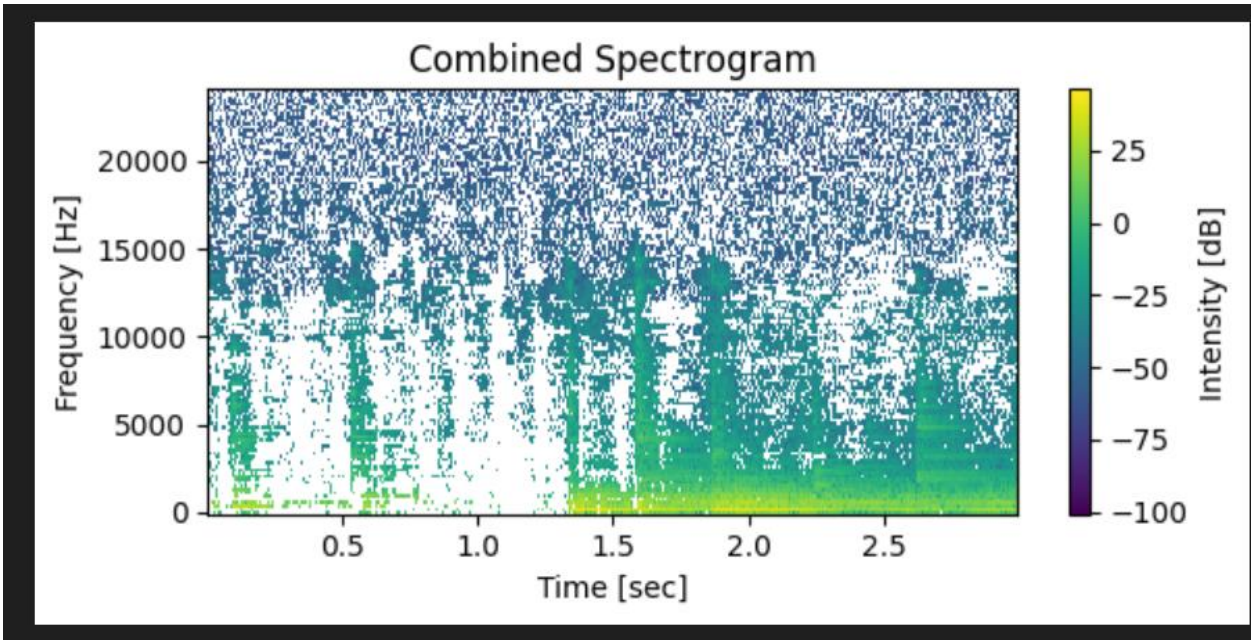


The Euclidean distance value between the spectrograms of two real samples was 168411.89.

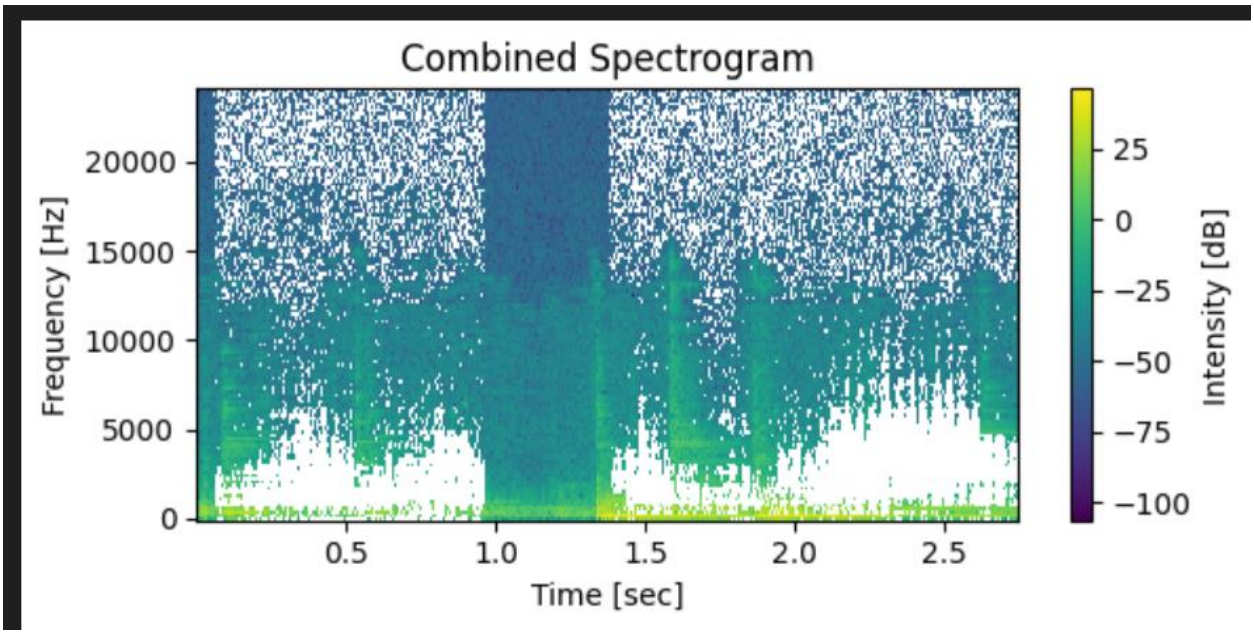
Although it is smaller than the euclidean distance of the generated spectrogram, they are both large, suggesting that perhaps the distance between the real and fake samples is not that large. However, one look at the spectrograms and two main differences can be observed: the generated spectrogram is not as intense, and perhaps more importantly it is not continuous.

As a final means of comparison, we thought that subtracting the spectrograms from each other would show how much they have in common. We did this by inverting one of the spectrograms, then added them together.

This is the resultant spectrogram produced by subtracting the two real spectrograms:

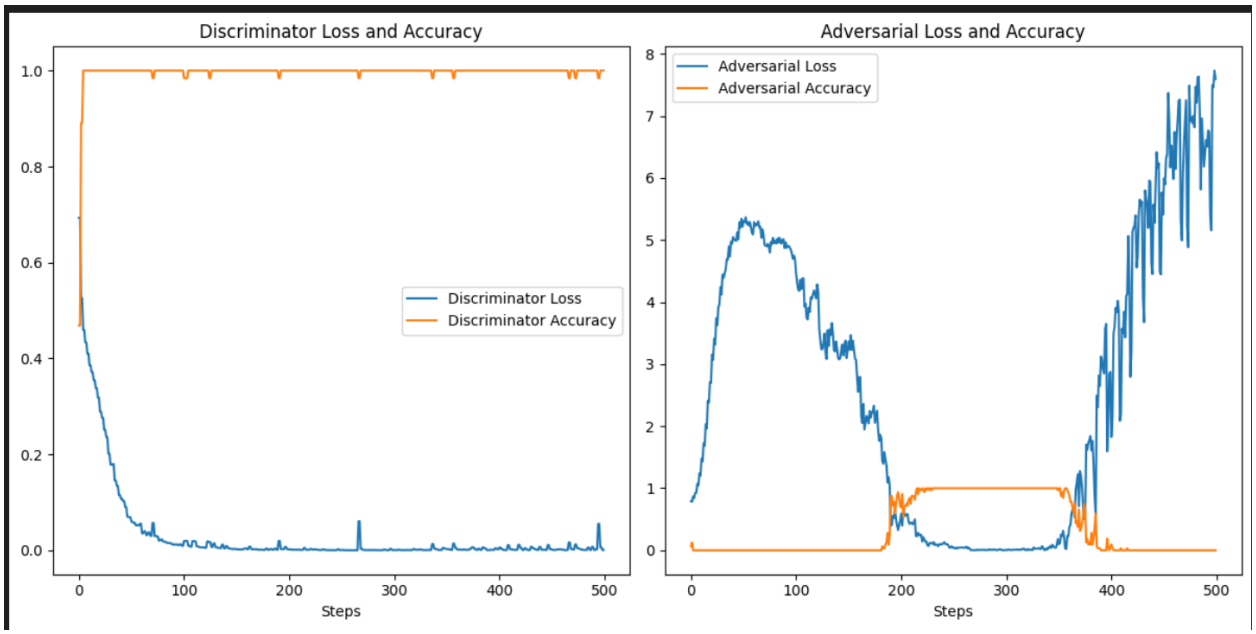


And this is the resultant spectrogram produced by subtracting the generated and the sample spectrograms:



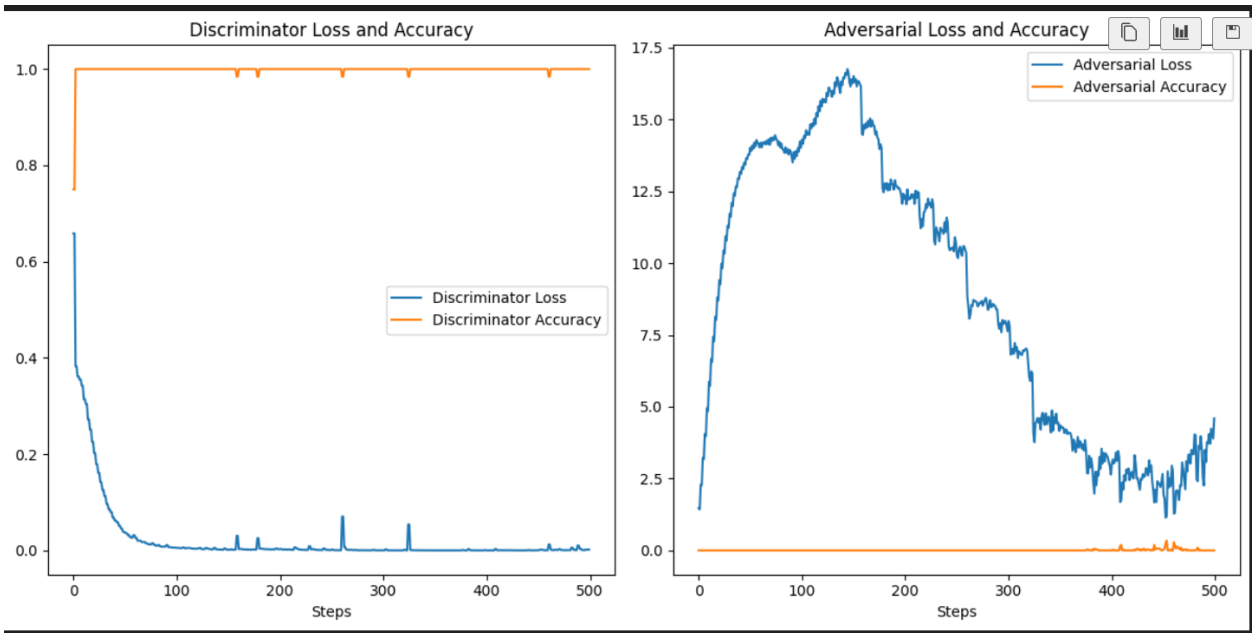
8. Results

Clearly show the results of running the various models. Show confusion matrices, ROC and Precision/Recall curves and other metrics showing under or over fitting. Explain the results and summarize key takeaways in the form of a table. The table should show the architecture parameters and conditions and key comparative results. Feel free to include bar charts as well. Which model worked best and why?

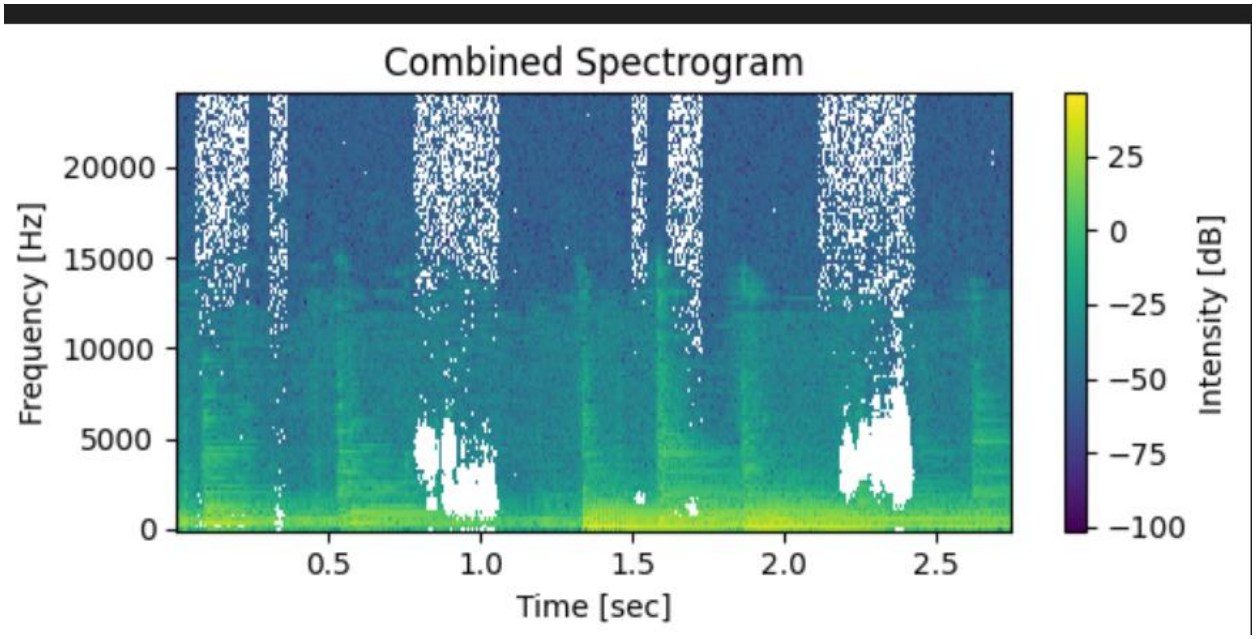


These are the loss and accuracy plots for our model over time. As can be seen, adversarial loss was decreasing until ~380 epochs. Although it seems like the discriminator is overpowering, so we made it even weaker than it already was. At this point, the discriminator consisted of 2 convolutional layers with strides = 2. The first one had 8 filters while the second had 16. Additionally, the discriminator was not trained every epoch, rather every other epoch.

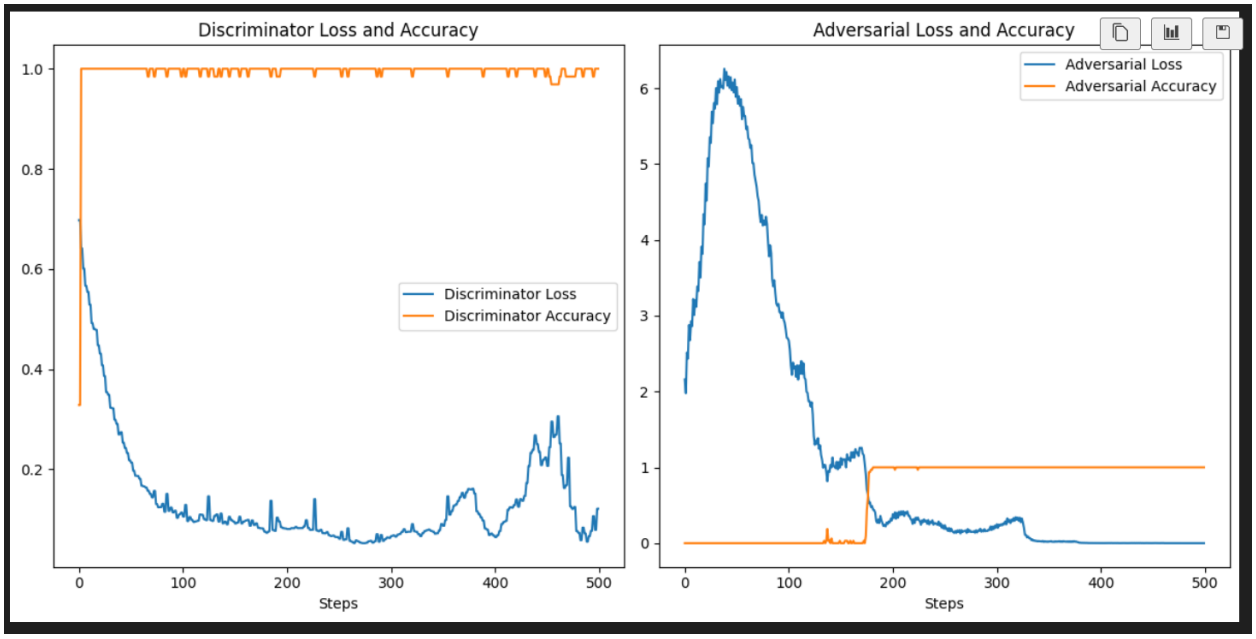
With the change in the discriminator, we achieved the following:



This shows that the discriminator is still overpowering. The adversarial loss may be decreasing, but the accuracy is not increasing. We are not sure why this is the case. The generated audio does not resemble an oud, and the spectrogram obtained by subtracting the real from the generated audio reflects that:

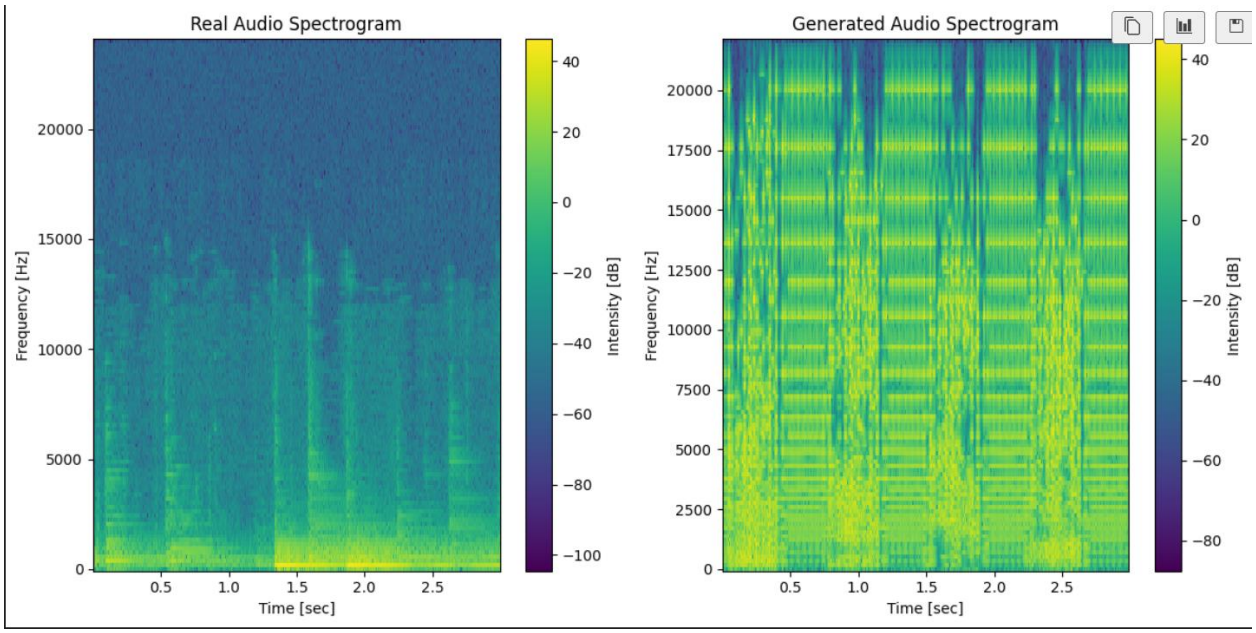


Although, seeing as the discriminator seemed to still be overpowering, we decided to make it even simpler: a single convolutional layer with 4 filters.

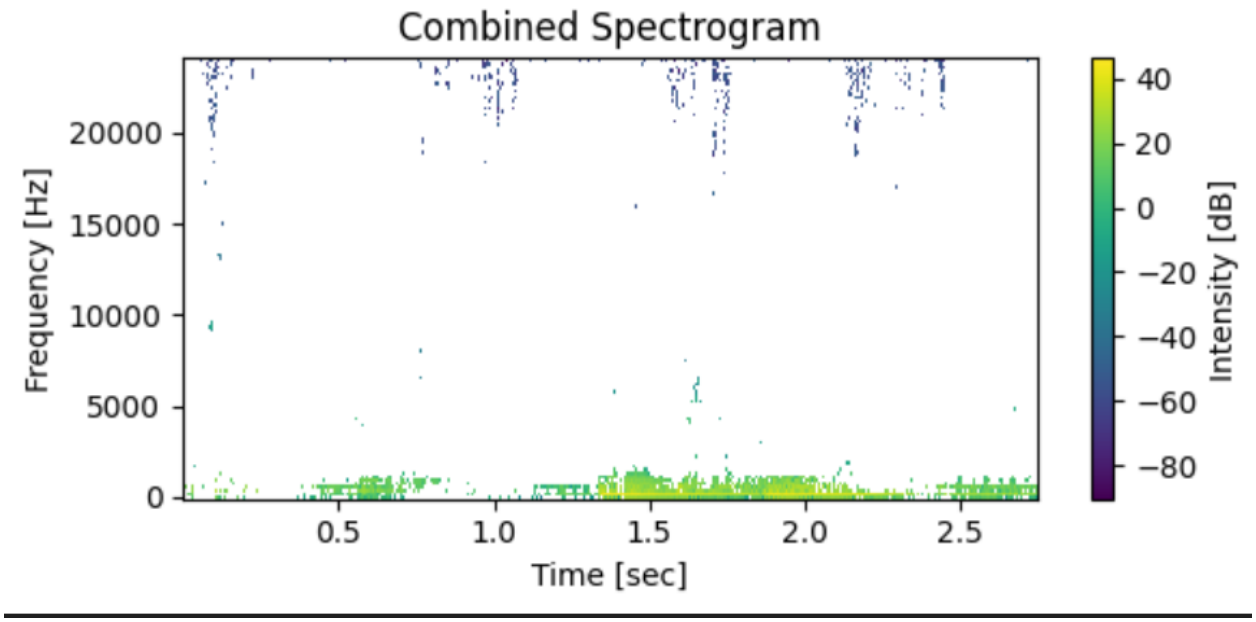


It was still overpowering, although not as much. However, going any lower with the discriminator does not make sense.

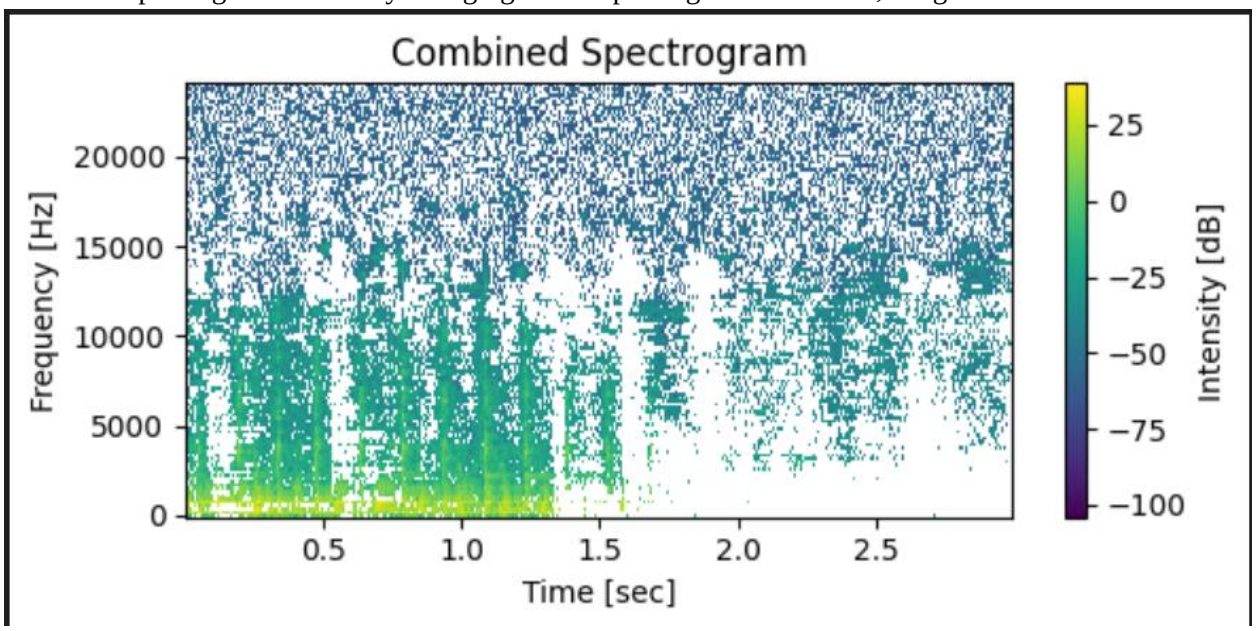
We also looked at the spectrograms, and noticed that the generated image was much noisier than usual:



This ended up producing a combined spectrogram that looked almost empty:



Which, usually, would be good, except we know that the reason behind this was due to the way our combined spectrogram works. By changing which spectrogram is inverted, we get this:



Which we believe is more representative of how bad the generated audio is (again, closer to a blank space is better).

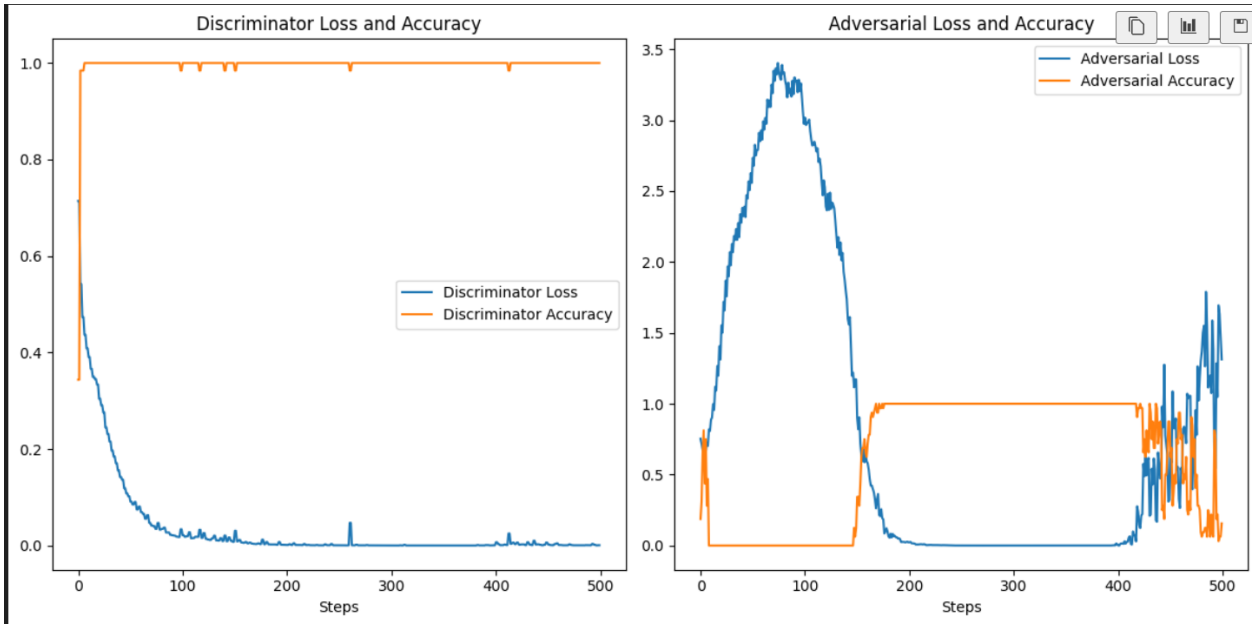
This has all been using a generator with four Conv2DTranspose layers, with increasing kernel size between each layer:

```
def build_generator():

    model = Sequential()



    # Start with a fully connected layer and reshape

    model.add(Dense(32 * 8 * 8, input_dim=latent_dimensions))

    model.add(Reshape((8, 8, 32)))



    # Upsample to the target size

    model.add(Conv2DTranspose(32, kernel_size=(2,2), strides=(2, 4),
padding='same'))
```

```
    model.add(BatchNormalization())

    model.add(LeakyReLU(alpha=0.01))



    model.add(Conv2DTranspose(16, kernel_size=(4,4), strides=(2, 4),
padding='same'))

    model.add(BatchNormalization())

    model.add(LeakyReLU(alpha=0.01))



    model.add(Conv2DTranspose(8, kernel_size=(5, 5), strides=(2, 2),
padding='same'))

    model.add(BatchNormalization())

    model.add(LeakyReLU(alpha=0.01))



    model.add(Conv2DTranspose(1, kernel_size=(5, 5), strides=(2, 2),
padding='same', activation='tanh'))



    return model
```

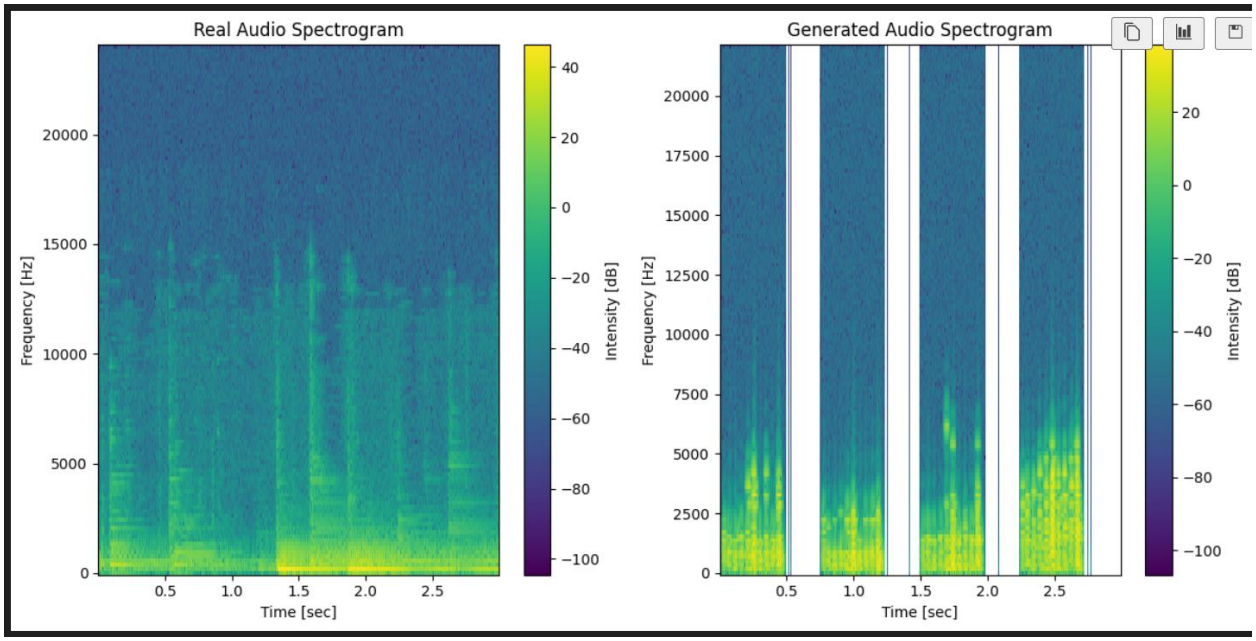
Next, we decided that perhaps increasing the number of neurons/Conv2DTransposes would make a difference.

Therefore, we went with a Dense(64*8*8) followed by a conv2dtranspose with 64 filters, then 32 then 16.

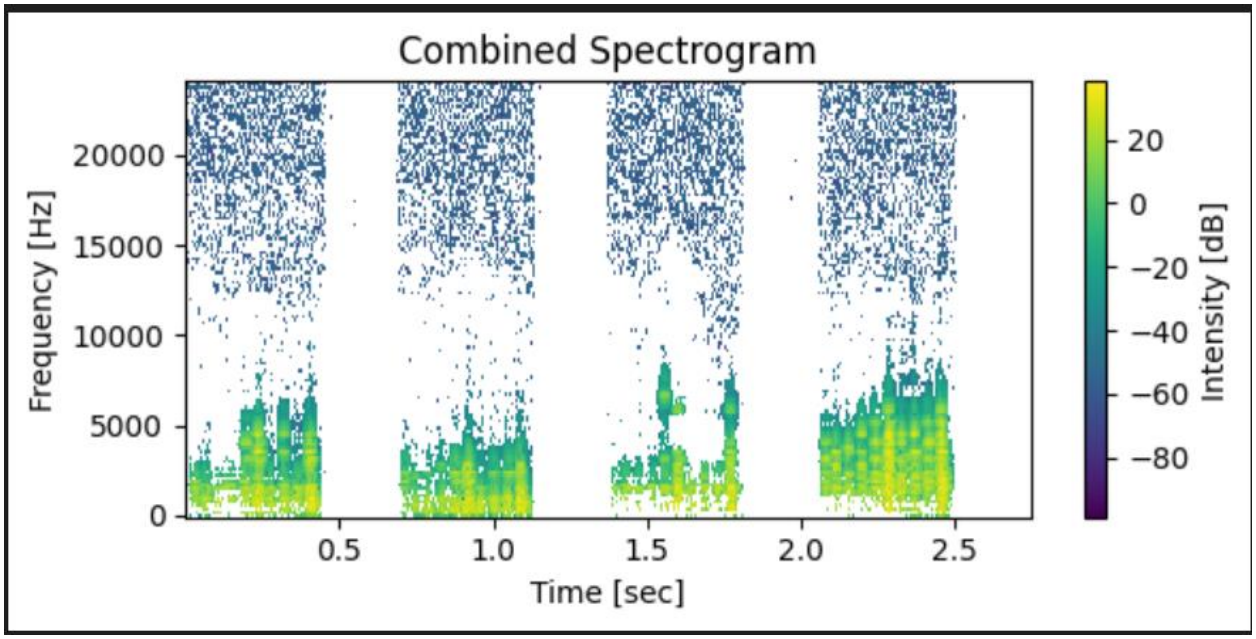


Our discriminator was back to dominating, but our adversarial loss and gain seemed to fluctuate a lot at the end, this is likely due to exploding gradients.

Again, the spectrogram is not that good as it is not continuous:



And this is the combined spectrogram:



Finally, we though of making our generator deeper rather than wider (more filters), so we added another layer before the final one, and also changed the kernel sizes:

```
model.add(Dense(32 * 8 * 8, input_dim=latent_dimensions))

    model.add(Reshape((8, 8, 32)))



    # Upsample to the target size

    model.add(Conv2DTranspose(32, kernel_size=(2,2), strides=(2, 4),
padding='same'))

    model.add(BatchNormalization())

    model.add(LeakyReLU(alpha=0.01))



    model.add(Conv2DTranspose(16, kernel_size=(3,3), strides=(2, 4),
padding='same'))

    model.add(BatchNormalization())

    model.add(LeakyReLU(alpha=0.01))
```

```
    model.add(Conv2DTranspose(8, kernel_size=(4,4), strides=(2, 2), padding='same'))

    model.add(BatchNormalization())

    model.add(LeakyReLU(alpha=0.01))



    model.add(Conv2DTranspose(4, kernel_size=(5, 5), strides=(2, 2), padding='same'))

    model.add(BatchNormalization())

    model.add(LeakyReLU(alpha=0.01))



    model.add(Conv2DTranspose(1, kernel_size=(5, 5), strides=(2, 2), padding='same', activation='tanh'))
```

9. Discussion and Future Work

Explain why the models worked well or not. Based on this discussion propose very specific steps and a plan that could be followed to improve upon the results you got. The future work should be based on what your learnt (new information you gained) while running YOUR models and not on general information that was available before you started running your models.

Firstly, we may want to think about using a different way of normalizing our data. For some reason, even when tested on the samples, the denormalization process tends to produce a distorted audio. Additionally, we may want to look into different representations of sounds. Although the Mel spectrum is compact, perhaps we may be losing too much information, especially in 3-second audio clips. There have been

Moreover, using a larger dataset could be beneficial to avoid overfitting and will add variety to our data.

10. Link to a YouTube Video **[-0% / -30% if link not provided]**

A YouTube video demonstrating the following three items:

a) **Your models being trained and showing the history of training.**

b) **Your models running and predicting results.**

c) **Showing different results like ROC curves, etc. being created on the fly.**

https://youtu.be/Aof2yXDl1_I

References

[1] Nayebi, Aran, and Matt Vitelli. "Gruv: Algorithmic music generation using recurrent neural networks." *Course CS224D: Deep Learning for Natural Language Processing (Stanford)* (2015): 52.

[2] Novack, Zachary, et al. "Unsupervised Lead Sheet Generation via Semantic Compression." *arXiv preprint arXiv:2310.10772* (2023).

[3] C. Chauhan, B. Tanawala, and M. Hasan, "Multi-genre symbolic music generation using deep convolutional generative Adversarial Network," *ITM Web of Conferences*, vol. 53, p. 02002, 2023. doi:10.1051/itmconf/20235302002

[4] G. Li, S. Ding, Y. Li, and K. Zhang, "Music Generation and Human Voice Conversion based on LSTM," *MATEC Web of Conferences*, vol. 336, p. 06015, 2021. sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss

[5] J. Wang and C. Li, "Chinese style pop music generation based on recurrent neural network," *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2022. doi:10.1109/imcec55388.2022.10019796

[7] H.-W. Dong and Y.-H. Yang, "Convolutional Generative Adversarial Networks with Binary Neurons for Polyphonic Music Generation," arXiv (Cornell University), Apr. 2018, doi: 10.48550/arxiv.1804.09399.

[8] "Shibboleth Authentication Request." https://www-sciencedirect-com.aus.idm.oclc.org/science/article/pii/B9780128197141000245#s0110