Groovie-Genie: Royalty Free Music Generation

Jay Jhaveri
A59017531
Department of Computer Science and Engineering (CSE)
University of California, San Diego (UCSD)
jjhaveri@ucsd.edu

Andrew Ghafari
A59020215
Department of Computer Science and Engineering (CSE)
University of California, San Diego (UCSD)
aghafari@ucsd.edu

Abstract

Our project aims to address the challenge of using copyrighted music in social media content creation. Creators often face copyright issues when using music in their videos, podcasts, and other social media content, especially for intro/outro beats or mid-video breaks to keep their audience engaged. To avoid copyright infringement, creators have to either purchase a license to use music or create their own. However, these options can be time-consuming and costly and may not necessarily fit the specific requirements of the content. Therefore, we developed Groovie-Genie, which will generate loopable 30 seconds of audio for the influencer to use, which is perfect for intro/outro beats for podcasts, YouTube videos, Instagram videos, and generally all social media posts. We will discuss the three different approaches we took in detail and also the challenges we faced.

1 Introduction

Using copyrighted music in social media content has become something that is very present in today's world. These actions usually result in the content being banned or removed and may even result in legal action by the original creator of the song used. This has a lot of negative repercussions on the creators' ability to reach and engage with their audience because it can get their whole/ a small part of the video removed, denying access to monetary benefits from that specific video or getting shadowbanned by the platform for misconduct. The use of original music can be time-consuming and requires skills and resources that may not be available to every creator. Therefore, a royalty-free music generator can be an efficient and cost-effective solution to this problem. Moreover, this generator can facilitate the creation of more diverse and engaging content, leading to better engagement and, ultimately, higher viewership for creators. It is a way to democratize access to tailored, creative, and captivating music pieces for video content.

With all this being said, we are excited to present our very own multi-instrument music generation model. The model has an inference time of less than a second and can generate an infinite number of unique songs. Because we prioritized only training on Copyright free data sets, all the generated music can be directly used in long-form and short-form social media content without any impending doom.

In the following sections, we will first talk about the various datasets used, then discuss about the three methodologies we implemented, and finally the results for each of the three approaches. We will finally end with highlighting our overall contributions, findings and will discuss the possible future scope for the project.

2 Dataset

While looking for datasets, our aim was to find ones that are totally copyright-free. We mainly utilized two datasets.

2.1 Free Music Archive: FMA-Small

The FMA dataset [1] is a very well-known dataset with over 106,574 mp3 files specially curated for music-related applications and reseach. All the 100k sounds in the dataset are entirely royalty-free and anyone can freely use these sounds as they seem fit. Due to space restriction we take a subset of this dataset called the FMA-small which has around 8000 songs, each 30 seconds long mp3 files.

2.2 Lakh Midi Dataset

The Lakh Midi dataset [2] is again a very well-known dataset of around 176,581 midi files. This dataset is heavily inspired from and is a subset of the Million Song dataset. This dataset again only consists of copyright free music files. Again due to space constraints we limited ourselved to around 5000 midi files to train.

2.2.1 Lakh Pianoroll dataset

This is a derivation of the original Lakh Midi dataset. Instead of midi files, it contains cleaned Pianoroll files for each music set. We will explain Pianoroll in future sections [3].

3 Previous work

Music generation is relatively new, but it has quickly been extensively researched and there are a lot of tools that try to achieve this. There are a lot of examples now that deal with this problem. They all have underlying issues, though. For example, OpenAI's jukebox [4] is a very well-known example, but it has one big issue, it is trained on a copyrighted dataset. That issue is also common with other publicly available solutions. Some of them are copyright-free but are also provided as paid services to people, for example, Mubert [5]. Other services are also very slow, even for a 30-second audio file, for example, P22 [6].

Moreover, the top three top contenders in the AI music generation field are none other than Google, Microsoft, and OpenAI. Google introduced MusicLM [7] last year, a music-generation AI generated using text prompts trained on the publicly available MusicCaps dataset primarily based on the popular MuLan architecture [8]. OpenAI's Jukebox [4] takes a different approach where their model can accept lower-level music details, such as beats per minute, tempo, artist name, voice type, etc., and then uses this to generate audio. Muzic [9], by Microsoft, has taken a different approach where they have divided their model into multiple smaller models such as lyric generation, melody generation, lyric to melody, synthetic voice generation, and more. All of these music generation models have one drawback in common. All of them are trained on some kind of copyrighted material and hence cannot be released to the public for use!

4 Methodologies

Throughtout the course of the project we used a loosely based CI/CD approach, wherein we kept testing, improving and changing our approaches/datasets as the development needed. We categorize our entire work progress into three main approaches.

4.1 Methodology 1: Facebook EnCodec model + mp3 files

This approach can be decomposed into three main components, each with a distinct role. First, the encoding part that is responsible for taking in input data in the form of a music genre and then pass it through a pretrained BERT model to be able to extract its embedding. The generator part, that gets this embedding along with some noise then passes it through a neural network to generate a 30-second song out of this embedding. Finally a discriminator that just takes in either a fake generated song or a real one, and its job is to guess if its real or not. Essentially it is a binary classifier that distinguishes between real and fake musical compositions. The generator's task is to fool the discriminator and the discriminator's task is to not get fooled by the generator.

We utilized two models, we first started with a very simple GAN, that is just a combination of linear layers and ReLU activation functions.

4.1.1 Architecture 1: Simple vanilla GAN

Although our goal is to generate music, which is innately a time series data we decided to first implement a Conditional GAN architecture using only fully connected layers. We implemented the opensource pretrained EnCodec model from Facebook which helped us convert audio embeddings into mathematical vectors similar to BERT. The EnCodec model [10] is a High Fidelity Neural Audio Compression, encoder-decoder architecture made using transformers. Albeit, the original use case of EnCodec is supposed to be high quality compression we are using it in our architecture as an alternative to CLAP to run in our current low resource machines. The EnCodec model plays a crucial role in converting music into codeblocks based on the Residual Vector Quantization (RVQ) technique. These codeblocks consist of a code vector and a scale vector, which capture essential information about the music. Next, using the true audio encoding, a discriminator network is trained. The discriminator consists of 6 Linear layer followed by LeakyReLU at each step. The last layer is a sigmoid layer to output a binary classification. On the contrary, the Generative model instead accepts the BERT embedding, and uses a 4 linear layer architecture to up-sample and output a vector that can be understood by the EnCodec model. Each layer here is enveloped by a batch normalization and ReLU layer.

In testing, we hooked up our generator model to the decoder part of the EnCodec model which can decode the vectors generated into an audio numpy vector understandable by the torchAudio library. The resulting mp3 files after even 80 epochs are, as expected, just a bunch of discrete random noise.

4.1.2 Architecture 2: C-RNN GAN

As before the music encoding architecture remains the same. Specifically, we are using the Facebook research's pretrained 48 kHz EnCodec model at a bit rate of 6 giving us 4 codebooks per frame. The EnCodec model was originally designed to compress the music, but we believe it has way more applications than just compression. To summarize, EnCodec architecture is made up of three blocks, namely, encoder, quantizer, decoder. The encoder and quantizer together is used to first map the music into a smaller dimension followed by Resilient Vector Quantization (RVQ). This output, called "encoded frames", is made up of a list of tuples of 4 tensors and 1 scalar value we are calling "scale." For now, we are splitting these encoded frames into two tensors, namely: codes and scales, and passing them into the GAN structure.

We have designed the architecture, inspired by the original C-RNN GAN [11], to be a two-header that merges or diverges depending on if it's a generator or discriminator. To elaborate, the generator and discriminator networks are structured as followed:

Generator:

1. Variable Names
    - noise_size: The size of the noise vector input to the generator
    - bert_size: The size of the BERT embedding input to the generator.
    - hidden_units: The number of hidden units used in the LSTM layers.
    - output_size: The size of the output code
    - drop_prob: The dropout probability used in the generator.
2. Layers
    - bi_lstm1: Bidirectional LSTM layer with input_size = noise_size + bert_size, hidden_size = hidden_dim, bidirectional = True

- bi_lstm2: Bidirectional LSTM layer with input_size = hidden_dim x 2, hidden_size = hidden_dim x 2, bidirectional = True
- fc1: Fully connected linear layer with in_features = hidden_dim x 4 and out_features = hidden_dim x 4
- fc_codes: Fully connected linear layer with in_features = hidden_dim x 4 and out_features = output_size
- fc_scales: Fully connected linear layer with in_features = hidden_dim x 4 and out_features = 29

3. In the forward pass, the bert_embedding is concatenated with random noise to form the input to the generator. The input is passed through the LSTM layers, followed by the fully connected layer fc1. Finally, fc_codes and fc_scales linear layers are used to generate the output codes and scales, respectively.

Discriminator:

1. Variable Names
   - input_size: The size of the input tensor to the discriminator
   - hidden_units: The number of hidden units used in the LSTM layers.
   - drop_prob: The dropout probability used in the discriminator.
2. Layers
   - bi_lstm_codes: Bidirectional LSTM layer with input_size = input_size, hidden_size = hidden_dim, bidirectional = True
   - bi_lstm_scales: Bidirectional LSTM layer with input_size = 600 x 29, hidden_size = hidden_dim, bidirectional = True
   - fc: Fully connected linear layer with in_features = hidden_dim x 4 and out_features = 1
   - sigmoid: Sigmoid activation function
3. In the forward pass, the input tensor is split into two parts, codes and scales. The codes part is passed through bi_lstm_codes and the scales part is first reshaped and then passed through bi_lstm_scales. The outputs of both bi-LSTM layers are concatenated and passed through the fully connected layer fc, followed by a sigmoid activation to generate the final output (real or fake).

Loss Fucntion used:
We used the torch library's inbuilt Binary Cross-Entropy Loss (BCELoss)

4.1.3 High Level explanation of the code flow

The proposed model for text-to-music generation takes in specified genres, such as "Jazz", and processes them using a pre-trained BERT model. BERT, which stands for Bidirectional Encoder Representations from Transformers, is a language model that is used for natural language processing tasks. By encoding the input genres into a high-dimensional embedding space, the model is able to capture semantic relationships between different genres and extract relevant features that are informative for generating music.

The embedding generated by the BERT model is then passed through a Generative Adversarial Network (GAN) to generate music compositions. GANs are a class of deep learning models that consist of two neural networks: a generator and a discriminator. The generator takes a random noise vector or an input, such as the BERT embedding, and generates a sample, in this case, a music composition. The discriminator takes a sample, which can be either a real or a generated composition and attempts to distinguish between the two.

During training, the generator tries to generate compositions that are realistic enough to fool the discriminator, while the discriminator tries to accurately distinguish between real and fake compositions. After a while from training, the generator should be able to generate real-like compositions that will actually fool the discriminator, which is indeed the goal of a generator.

4.1.4 Mathematical Setup of the Architectures

Vanila GAN Architecture:

We are going to list here the generator and discriminator's loss functions that we are using, and their formulas. Below is the total loss function that incorporates both the generator and the discriminator loss functions 1.

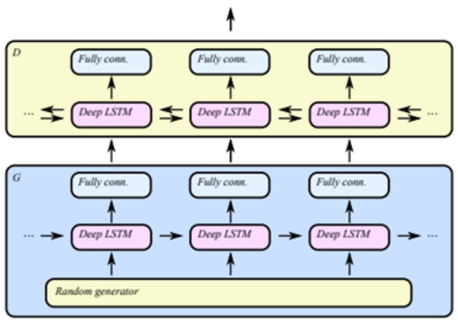
$$\min_G \max_D V(D,G)$$

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

Figure 1: Generator Discriminator BceLoss

C-RNN-GAN Architecture:

The architecture is visualized in 2a, while the loss function is visualized in 2b



(a) C-RNN-GAN Architecture

$$L_G = \frac{1}{m}\sum_{i=1}^{m} \log(1 - D(G(\boldsymbol{z}^{(i)})))$$

$$L_D = \frac{1}{m}\sum_{i=1}^{m} \left[-\log D(\boldsymbol{x}^{(i)}) - (\log(1 - D(G(\boldsymbol{z}^{(i)})))) \right]$$

(b) C-RNN-GAN Loss

Figure 2: C-RNN-GAN Architecture and Loss

4.1.5 Results

The results here were honestly a bit underwhelming. Depending on the hyperparameters, the sound generated was either fully mute with one or two beats scatered, or it was noise. This results were after 48 hrs of training, on about 80 epochs. We believe that more epochs would definitly help it, because learning the relation between codeblocks and scale, and the complex nature of lyrics with beats is hard. Hence, we decided to first start out with a simpler dataset to proove that our architecture works.

4.2 Methodology 2: C-RNN Gan on Piano only midi files

As mentioned before, we were unable to achieve good enough output even after 48 Hrs of training. Hence, we thought about instead of directly starting with mp3 files which are innately complex, let's first tackle simple piano notes. We used the Lakh MIDI dataset's Piano section, having around 2000 midi files.

To preprocess these midi files we took help of the opensource python Music21 Library, and converted the midi files into 2d vectors.

We used the C-RNN-GAN architecture from the previous approach with a slight modification to have one head instead of two. The rest of the architecture was identical to the first one.

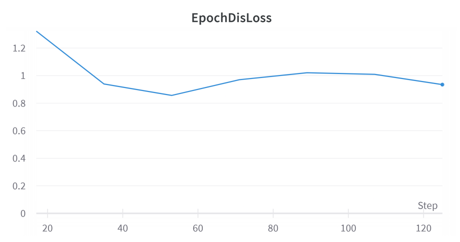
4.2.1 Training

We used multiple hyperparameters, but the ones that worked best and in the most efficient way are the following:

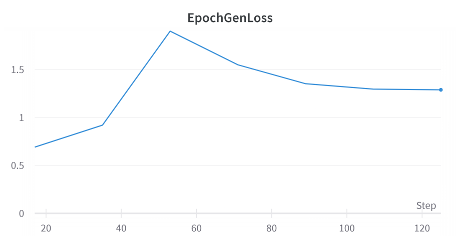
```
SEQ_LENGTH = 100
LATENT_DIM = 1000
BATCH_SIZE = 128
OPTIMIZER_LR = 0.0002
OPTIMIZER_BETAS = (0.5, 0.999)
```

Here, the sequence length represents the length of the midi sequence to be used while training and generating in one pass.

Let us now take a look at the loss plots for both the generator and discriminator.



(a) Approach 2 Discriminator Loss



(b) Approach 2 Generator Loss

Figure 3: Approach 2: Loss Plots

The training loss for the final run is visible in 3

As we can see above, the plots for the discriminator loss and the generator loss are aligned with theory, since we usually expect to have an increasing discriminator loss and a decreasing generator loss and that is somewhat what we are getting. Both losses seem to have converged, which is great. This is also reflected by the music files that we are getting, since we are able to get good piano music output files.

4.3 Methodology 3: C-RNN Gan on Multi-Instrument midi files

In our pursuit of generating more diverse and rich music, we explored a third approach, which aimed at incorporating sounds from multiple instruments into the generated compositions. While the music generated using approach 2 yielded great results in terms of generating a single 2D vector of piano notes, we desired a more comprehensive solution.

Upon conducting thorough research, we discovered a music representation technique known as "PianoRoll." This representation allows for the easy representation of multi-instrument music as vectors. With the incorporation of PianoRoll, we could seamlessly integrate multiple instruments into our music generation process.

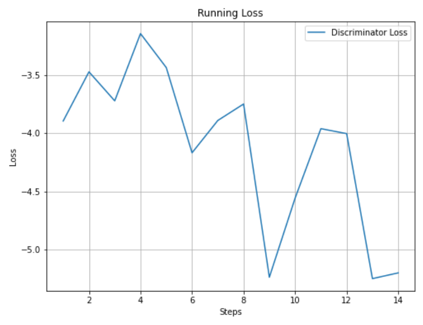
To achieve this, we leveraged the PyPianoRoll library, which provided the necessary functionalities for incorporating five additional instruments into the equation. By utilizing this library, we could manipulate and work with the multi-instrument MIDI files more effectively.

We were able to find a proper documentation that helped us in navigating the PianoRoll representation and going from midi files to PRs and then from PRs back to midi files in the decoding part. We utilized the in-built functions and we changed the DataLoader, Dataset and architecture in order to fit our project, and then we trained the final model.

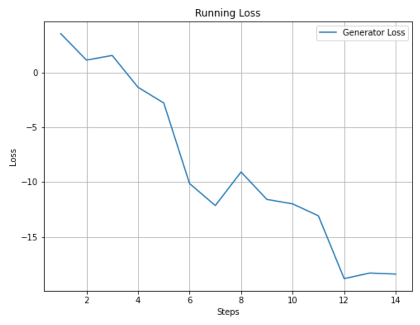
4.3.1 Training

For training our model, we employed the Lakh MIDI Piano roll dataset, which is widely recognized in the music generation research community for its comprehensive collection of multi-instrumental compositions. The dataset served as an excellent resource to train our model and achieve the desired outcomes.

To adapt the original C-RNN-GAN architecture to incorporate the PianoRoll representation, we made several adjustments, yet keeping the original sense of the architecture as is. These modifications allowed us to effectively handle the multi-instrumental data and generate music that seamlessly integrated sounds from multiple instruments.



(a) Approach 3 Discriminator Loss



(b) Approach 3 Generator Loss

Figure 4: Approach 3: Loss Plots

The training loss for the final run is visible in 4

The results obtained through this approach were beyond our expectations, as the music generated using this representation exhibited an unprecedented level of realism and richness. The incorporation of multiple instruments added a new dimension to the generated compositions, resulting in an immersive and diverse musical experience.

Overall, Methodology 3 provided a significant advancement in our music generation research by enabling the synthesis of multi-instrumental music through the use of PianoRoll and the adapted C-RNN-GAN architecture. The outcomes surpassed our initial goals and laid a strong foundation for further exploration and development in this field.

The Generator loss here, as we can see in the attached plots above, is converging, and even though the discriminator loss is not exactly what the theory suggests, but we have the audio files to manually check whether the outputs are actually good or not, and they are definitely great.

5 Metrics Used

In this section, we will discuss various metrics that can be used to evaluate the generated music. These metrics provide quantitative measures to assess different aspects of the music generated by our various architectures. Since there is no standardized way to study the efficiency of music generation models, we decided to stick to the ones provided by MusPy library. We will briefly introduce each one of them, and then later compares the results of our 3 approaches.

5.1 Number of Pitches Used (n_pitches_used)

This metric measures the total number of distinct pitches used in the generated music. It provides insight into the melodic richness and diversity of the composition.

5.2 Number of Pitch Classes Used (n_pitch_classes_used)

Pitch classes refer to the set of all pitch values within an octave, regardless of their specific octave. This metric counts the number of different pitch classes utilized in the generated music. It offers information about the harmonic variety and tonal characteristics of the composition.

5.3 Polyphony

Polyphony refers to the presence of multiple simultaneous voices or musical lines in a composition. This metric assesses the degree of polyphony in the generated music, indicating whether the composition tends to have overlapping musical elements or focuses on single melodic lines.

5.4 Polyphony Rate

The polyphony rate is a measure of the proportion of time in the composition where polyphony occurs. It helps in understanding how frequently multiple voices are present and how they are distributed throughout the piece.

5.5 Scale Consistency

This metric evaluates the adherence to a particular musical scale or mode in the generated music. It quantifies how consistent the pitch selection is within a given scale and can provide insights into the tonal coherence of the composition.

5.6 Pitch Entropy

Pitch entropy measures the overall unpredictability or randomness of pitch choices in the generated music. It calculates the amount of uncertainty associated with the distribution of pitches, indicating the level of variation or stability in the composition.

5.7 Pitch Class Entropy

Similar to pitch entropy, pitch class entropy calculates the unpredictability or randomness of pitch classes within the generated music. It captures the diversity of pitch class choices and their distribution.

5.8 Empty Beat Rate

The empty beat rate measures the frequency of beats that do not contain any musical events. It helps assess the presence of silence or gaps within the composition, indicating the rhythmical density or sparsity.

5.9 Drum Pattern Consistency

This metric focuses specifically on drum patterns in the generated music. It evaluates the consistency and repetition of drum patterns, providing insights into the rhythmic structure and stability of the composition.

5.10 Groove Consistency

Groove consistency measures the regularity or consistency of the rhythmic feel or groove in the generated music. It assesses the stability of rhythmic patterns and their adherence to a particular style or genre.

5.11 Empty Measure Rate

Similar to the empty beat rate, the empty measure rate measures the frequency of measures or bars that do not contain any musical events. It provides information about the structural density or sparsity of the composition on a larger time scale.

5.12 Pitch Range

The pitch range calculates the interval between the lowest and highest pitches used in the generated music. It helps assess the melodic span and overall pitch variation in the composition.

6 Results

To evaluate the generated music files, we employed two methods: quantitative evaluation using the above-defined music metrics and qualitative evaluation through human judgment. This section presents the findings from both approaches.

6.1 Quantitative Evaluation

We begin with the quantitative evaluation, where we employ various metrics to assess different aspects of the generated music over time and compare them with other methodologies. Table ?? and Table ?? provide an overview of the quantitative evaluation results.

File Name	Method	#pitches	#pitch_classes	polyphony	polyphony_rate	scale_consistency	pitch_entropy
Method1_100.mid	1	2	2	2	0	1	1
Method2_22.mid	2	24	7	1.67	0.21	1	4.235097
Method2_66.mid	2	24	7	1.74	0.24	1	4.093239
Method2_124.mid	2	65	12	3.62	0.83	0.690594	5.504936
Method3_2.mid	3	43	12	4.016393	0.730159	0.65641	5.016256
Method3_3.mid	3	67	12	12.043293	0.976705	0.676417	5.466479
Method3_5.mid	3	57	12	7.609375	0.984375	0.66092	5.287118
Method3_6.mid	3	59	12	12.080398	0.99233	0.623819	5.503118

Table 1: Quantitative Results

File Name	Method	pitch_class_entropy	empty_beat%	drum_used	grooviness	empty_measure%	pitch_range
Method1_100.mid	1	1	0.965517	NaN	0.999966	0.982389	2
Method2_22.mid	2	2.608674	0	NaN	0.998057	0	55
Method2_66.mid	2	2.566275	0	NaN	0.998057	0	52
Method2_124.mid	2	3.518671	0	NaN	0.998057	0	73
Method3_2.mid	3	3.514803	0	1	0.966224	0.029988	59
Method3_3.mid	3	3.536375	0	1	0.96392	0.00227	70
Method3_5.mid	3	3.48582	0	1	0.96392	0	71
Method3_6.mid	3	3.516845	0	1	0.964489	0	70

Table 2: Quantitative Results Continued

From the table ??, it is evident that all sounds generated, except the ones by Method 3, exhibit a drum_used or drum consistency of NaN, indicating a non-existence of drum elements. This is attributed to the fact that Method 1 failed to learn even after 100 epochs, while Method 2 was only trained to generate piano notes.

On the other hand, considering metrics such as pitch entropy and polyphony rate, we observe a clear progression and improvement in the generated music across all methods. Notably, Method 2 demonstrates the most significant enhancement in terms of these metrics. However, it is important to note that the quantitative evaluation may be less accurate for Method 3 due to the mixing of multiple musical instruments.

6.2 Qualitative Evaluation

In addition to the quantitative analysis, we conducted a qualitative evaluation through human judgment. Four individuals, apart from the researchers, were asked to assess the generated music files. The evaluation resulted in unanimous feedback regarding Method 2 and Method 3.

All participants agreed that Method 2 produces more melodic and soothing music, showcasing its ability to generate aesthetically pleasing compositions. On the other hand, Method 3 was appreciated for its incorporation of rhythm and well-defined beats, making it particularly enjoyable for loud music listeners.

Overall, the combination of quantitative and qualitative evaluations provides a comprehensive understanding of the strengths and weaknesses of each methodology in generating music files.

7 Future Scope

In this section, we will discuss the future scope of this project and present three main ideas that can be explored further to enhance its capabilities.

7.1 Conditional GAN

The first idea that we want to talk about is to develop a conditional Generative Adversarial Network (GAN). Building upon the initial idea and architecture we explored, we can create a system that generates songs based on specific genres. Users could provide a genre as a prompt, and the model would generate music tailored to that genre. This approach would allow for more targeted and specialized song generation, catering to diverse musical preferences, which can be really handy for creating music tailored to the content that the user has.

7.2 Expanded Instrumentation

Another way to improve our current project is to include a wider range of instruments in the generation process. By incorporating additional instruments we can provide users with more options for musical expression. Users could select the instruments they want to include in their generated compositions, offering greater flexibility and customization in the post-generation phase.

7.3 Lyrics Generation

To further expand the project's capabilities, we can move beyond the constraints of MIDI-based music generation and incorporate lyrics. By incorporating natural language processing techniques and training the model on a vast corpus of lyrics, we can enable it to generate song lyrics along with the corresponding melodies. This enhancement would open up new possibilities for creating complete songs and give the creator full flexibility over the outcome of the generation.

7.4 Overall Summary

By incorporating these three ideas into the existing project, we can create a more comprehensive and versatile music generation system. Users would have the option to generate songs based on specific genres, choose from a broader range of instruments, and even include lyrics in their compositions. These enhancements would provide users with greater creative control and open up new avenues for musical exploration.

8 Contribution

We have tried different things, some of which worked and some of which didn't work. In this section we will be listing all the new things we tried and their outcome.

- First thing is using Facebook's Encoded model, that was previously designed to compress and decompress music, to make it generate music by generating a compressed version of an audio file and using the in-built decoder to retrieve an audible music file. We faced three main issues with this:
  - The open source model is too big for datahub, especially after downloading the dataset, we didn't have enough memory to run it freely and do experimentations
  - Training seemed promising, but nonetheless, very slow. With the computational power that we had, this seemed like a very long process
  - The dataset that we were using was very diverse, which probably extended the overall expected training time needed for the model to converge.
  - One last thing we tried here, was implementing a new architecture called c-RNN-GAN (c standing for continuous), to generate music file, this was more promising than the vanilla architecture that were initially trying but still fell short because of the same reasons mentioned above.

- For our second approach, we wanted to check if the c-RNN-GAN is actually good, so we decided to reduce our dataset to only piano files and running it again. This gave much better results after a lot of hyperparamter tuning. One thing we noticed is how volatile the model is, ie a bit of change in the hyperparameters can make the model produce either noise or music. So, after a lot of trial and error we were able to make it converge to good audio outputs.
- We discovered a new library called PyPianoRoll that helps with generating a new type of music representation called piano rolls, and using it we were able to fulfill our initial promise of making multi-instrument music. Using the same architecture as above, and trying both bi-directional and uni-directional LSTM in the generation process, we were able to make good multi-instruments music outputs. As we have mentioned above, we were able to find a proper documentation that helped us in navigating the PianoRoll representation and going from midi files to PRs and then from PRs back to midi files in the decoding part. We utilized the in-built functions and we changed the DataLoader, Dataset and architecture in order to fit our project, and then we trained the final model.
- One last find is that there is another library called MusPY library, that has different metrics that are used to evaluate the ingenuity of the music produced and we utilized it to compare our outputs from the three described approaches above.

9 Conclusion

In this report and project, we were trying to generate music files for content creation. As we have previously mentioned, we have tried three different methods, and each one of them was a huge lesson for us moving forward. We tried using FaceBook's EncoDec model to generate music, then we moved on to using a novel c-RNN-GAN structure on only piano files, which served as a proof of concept that the architecture is actually a good one. Then we discovered a library called PyPianoRoll that generates a new way of representation for music files called piano rolls, which helped us extend the scope of our music outputs to multi-instruments instead of only piano. Finally we were able to find another library called MusPY, using which we were able to quantify how good the results that we got for the three iterations actually were. This has been a fun project to work on, we hope you'll enjoy listening to the audio outputs.

10 Audio Outputs

Please navigate to this link to be able to hear some audio outputs from the three different methodologies.

Inside the link below, there are three folders called getMetrics_Iteration1, getMetrics_Iteration2 and getMetrics_Iteration3. Each folder contains sample audio file from the specified iteration number. Enjoy!

Link to audio outputs in midi form

Link to audio outputs in mp3 form

References

[1] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson. FMA: A dataset for music analysis. In *18th International Society for Music Information Retrieval Conference (ISMIR)*, 2017.

[2] C Raffel. *Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching. 331 Ph. D.* PhD thesis, thesis, Columbia University, 2016.

[3] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[4] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music, arxiv. *arXiv preprint arXiv:2005.00341*, 2020.

[5] Mubert - Thousands of Staff-Picked Royalty-Free Music Tracks for Streaming, Videos, Podcasts, Commercial Use and Online Content — mubert.com. https://mubert.com/. [Accessed 13-Jun-2023].

[6] P22 Music Text Composition Generator (A free online music utility) — p22.com. https://p22.com/musicfont/. [Accessed 13-Jun-2023].

[7] Andrea Agostinelli, Timo I Denk, Zalán Borsos, Jesse Engel, Mauro Verzetti, Antoine Caillon, Qingqing Huang, Aren Jansen, Adam Roberts, Marco Tagliasacchi, et al. Musiclm: Generating music from text. *arXiv preprint arXiv:2301.11325*, 2023.

[8] Qingqing Huang, Aren Jansen, Joonseok Lee, Ravi Ganti, Judith Yue Li, and Daniel PW Ellis. Mulan: A joint embedding of music audio and natural language. *arXiv preprint arXiv:2208.12415*, 2022.

[9] AI Music - Microsoft Research — microsoft.com. https://www.microsoft.com/en-us/research/project/ai-music/. [Accessed 13-Jun-2023].

[10] Alexandre Défossez, Jade Copet, Gabriel Synnaeve, and Yossi Adi. High fidelity neural audio compression. *arXiv preprint arXiv:2210.13438*, 2022.

[11] Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.