

Final Report

PianoPartnerGAN

University of Toronto

Mehul Bhardwaj, Asad Ishaq, Jonathan Sanjay

ESC324 Machine Intelligence, Software & Neural Networks

April 19, 2023

Contents

1	Abstract	2
2	Introduction	3
3	Prior Work	3
3.1	MySong: Automatic Accompaniment Generation for Vocal Melodies	3
3.2	MuseGAN: Multi-Track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment	4
3.3	Accomontage: Accompaniment Arrangement Via Phrase Selection And Style Transfer	5
4	Data	5
4.1	Data Collection	5
4.1.1	Challenges Faced in Data Collection	6
4.2	Supplemental Dataset	6
4.3	Data Pre-Processing	6
4.3.1	Challenges Faced in Data Processing	8
5	Model	8
5.1	Architecture	9
5.2	Changes from the Interim Report	11
5.3	Challenges Faced With Model	11
5.4	Custom Loss	13
5.4.1	Chromatic Difference	13
5.4.2	Polyphonic Rate	14
5.4.3	Number of Pitches Used	14
5.4.4	Hybridization	14
5.4.5	Important Considerations	15
6	Post-Processing	15
7	Results	17
7.1	Comparison of Models	18
7.2	PPGAN Demo	18
7.3	Overall Performance	19
8	Next Steps	19
9	Ethical Implications	20
10	Conclusion	21
11	Appendix	21
12	References	22

1 Abstract

This report details the implementation and evaluation of a Generative Adversarial Network (GAN) for generating piano accompaniments given a melody. We took inspiration from prior work such as MuseGAN to create our model. Custom musical loss functions were implemented in our model to improve the quality and musicality of the generated accompaniments. These loss functions measure chromatic distance, polyphonic rate, and the number of pitches used, which were then used to train several models.

We evaluated the models on two test datasets, one from the Nottingham music database and one scraped from Musescore. Our evaluation criteria included the aforementioned custom loss functions as well as subjective evaluation of the generated accompaniments. Our results show that the model trained with the chromatic distance loss function outperformed the other models in terms of musical quality, and we dubbed it the "Piano Partner GAN" or "PPGAN" for short.

Our PPGAN Model was able to generate accompaniments that were structurally sound and musically interesting. The model is capable of generating music that is mostly musically coherent. The model has effectively learned relationships between notes in music theory as it is able to choose notes that end up making chords from the same key as the input melody and transition between these notes in correct rhythms. It can also add note flourishes outside of a regular chords which add nuance and depth to the music. We believe that our implementation demonstrates the potential of using GANs for generating piano accompaniments, and that our approach to the AI music generation problem can serve as a foundation for further exploration in this area.

2 Introduction

The art of music creation is a highly intricate and time-consuming process that often involves endless experimentation with different chord progressions, rhythms, and styles in search of the perfect accompaniment for a given melody. While recent advances in Artificial Intelligence (AI) have yielded impressive results in music generation, current AI systems still struggle to produce realistic, coherent, harmonious, and creative piano accompaniments that complement a given melody.

The objective of this project is to tackle this problem by developing an advanced AI system that can automatically generate high-quality piano accompaniments that are musically plausible and stylistically consistent with a given melody. To accomplish the project's goal of generating high-quality and musically expressive piano accompaniments, the team drew inspiration from various prior works. The project aims to explore several key questions, such as how to improve the AI system's ability to understand the structure and form of a given melody, how to make the system more versatile and capable of generating accompaniments for a wide range of musical styles and genres, and how to evaluate the quality of the generated accompaniments compared to human-generated ones.

The potential impact of this project is significant, as the ability to generate harmonious piano accompaniments quickly and easily would benefit musicians, composers, and music producers alike. By freeing up more time and resources for these creative professionals to focus on other aspects of their music-making process, this project could spur innovation and experimentation in the music industry, leading to the emergence of new musical ideas and styles. Moreover, exploring the best approaches for training AI to understand the intricate relationship between melody and accompaniment in music could reveal new insights into the capabilities and limitations of AI in music generation, with broad implications for the field of AI and beyond.

3 Prior Work

As far as we are aware, there is not anything that succeeds in answering the questions outlined in our problem statement. Below are summaries of three papers that are similar to the work we are undertaking, each providing important insights into what we want to do.

3.1 MySong: Automatic Accompaniment Generation for Vocal Melodies

MySong is a system that is designed to automatically choose chords to accompany a vocal melody. Vocal melody refers to the tune or series of notes sung by a singer in a song. It is intended to be intuitive for non-musicians. The system uses a Hidden Markov Model to represent a chord sequence and its relationship to a melody. The model is trained using a large database of popular music [1].

In the pre-processing stage of the MySong system, chords in the database are simplified by classifying them into five primary triads that contain the core notes in a chord. The training database is also transposed to a single key (C) to simplify analysis. The model-training process then involves learning the statistics governing transitions among chords in a song, done by counting the number of transitions from each chord type to every other chord type in the training database, resulting in a chord transition matrix with 62 rows and columns. Finally, the system examines the melody during each chord in the database to learn the statistics governing which notes are associated with each chord type, with durations inserted into a table normalized so that each row sums to 1.0, representing the probability of seeing each pitch at one instant in time during a given chord [1]. This

approach is one that we are considering trying out as an alternative starting point, as currently we are struggling to produce outputs that make sense.

The MySong system is validated in its effectiveness in generating acceptable accompaniments and in allowing people with no musical background to quickly create accompaniments of their own. It is a simplified version of what we want to do. We aim for our piano accompaniments to not just understand simple chord progressions, but to be able to add ornamentation in between chords, and be able to utilize more chord voicings.

While the MySong system has given us a starting point for our project, we recognize that our goals may differ from those of the authors and that we may need to take a different approach to achieve them. We may explore different techniques or use additional resources to generate musically coherent and stylistically consistent accompaniments. Nevertheless, we appreciate the insights provided by the MySong system and plan to use them as a foundation for our project.

3.2 MuseGAN: Multi-Track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment

The paper proposes MuseGAN, a GAN-based model for multi-track sequence generation in music, which served as an informative starting point for our project. In music, a track refers to an individual recording of a sound or instrument, while a multitrack refers to a recording that consists of multiple tracks that can be edited separately and combined to create a final mix. The model has the capability to generate music from scratch, as well as follow a human-specified temporal structure, while handling interactions among tracks [2]. The authors employed transposed CNNs to generate music one bar at a time and proposed objective musical metrics for evaluating the results. Inspired by their approach, we also implemented regular CNNs in our GAN model.

MuseGAN is a model for music generation that builds upon previously proposed multi-track and temporal models. It uses inter-track and intra-track time-independent and time-dependent random vectors to generate latent vectors with inter-track and intra-track information. These vectors are then used by the bar generator to produce piano-rolls sequentially. Additionally, an encoder is employed to extract inter-track features from a user-provided track. These features are crucial to our project as we aim to generate not only chord progressions but also creative ornamentations and intelligent note placements [2]. MuseGAN’s approach of utilizing inter-track and intra-track information, time-independent and time-dependent random vectors, and an encoder for inter-track feature extraction are critical components that enable the model to produce high-quality and musically expressive outputs.

In their paper, the authors of MuseGAN used a dataset of over 176,000 MIDI files to generate music using their GAN-based model. MIDI files are digital representations of sheet music that contain information about the notes, timing, and other aspects of the music [2]. Taking inspiration from MuseGAN’s approach, we also employed GANs in our project and were encouraged by their ability to perform well on large datasets. They use the python library “pretty midi” to convert the MIDI files to multi-track piano-rolls, these resources may be useful for our project.

The authors of MuseGAN evaluate their model using a variety of musical metrics, including the ratio of empty bars (EB), number of used pitch classes per bar (UPC), ratio of qualified notes (QN), ratio of notes in 8 or 16-beat patterns (DP), and tonal distance (TD) [2]. These metrics are calculated for both the real and generated data, providing an effective means of evaluating the performance of the model. We aim to implement similar metrics to evaluate the performance of our

own model and ensure that our generated accompaniment is musically coherent and stylistically consistent.

3.3 Accomontage: Accompaniment Arrangement Via Phrase Selection And Style Transfer

Accomontage is a cutting-edge system that uses a combination of key and chord analysis, phrase selection, and style transfer techniques to create a musical accompaniment that complements a given melody. It begins by analyzing the melody to identify its key and underlying chord progression, and then searches for suitable musical phrases that match the key and chords [3].

To ensure that the phrases fit the desired musical style, the system leverages the power of style transfer. This process involves adjusting different aspects of the phrases, such as rhythm, harmony, and instrumentation, using a neural network that has been trained on a large dataset of musical examples in the target style. Neural networks are used to identify the specific characteristics of a particular musical genre or style.

By using style transfer, Accomontage can create entirely new pieces of music that blend different styles together in innovative ways. For example, a piece of pop music could be transformed to sound like it was created by a classical composer like Mozart or Beethoven. However, there are still many challenges to implementing style transfer in music, including issues with data quality, neural network architecture, and the subjective nature of musical style [3].

Accomontage builds on the success of previous models like MySong, which showed that choosing the right chords is crucial to creating a good song. However, selecting chords can be challenging because of the vast number of possibilities for each chord progression. Accomontage addresses this challenge by prioritizing the consistency of the musical style while choosing chords that reflect the intended mood and atmosphere of the composition. In summary, Accomontage is an innovative system that has a lot of promise for making it easier for people to create high-quality music even without extensive musical experience.

4 Data

To get started with creating an AI system that is capable of generating piano accompaniments given a melody, copious amounts of music data need to be collected, compiled, handled and processed such that it is ready to be inputted into a neural network for its training.

4.1 Data Collection

To develop our model to create accompaniments for various music pieces, we needed to collect symbolic representations of music pieces with existing accompaniments. The most common ways to represent music is either with sheet music or as a MIDI file. After researching various online libraries that had large amounts of music, we decided to use “Musescore” as it is the largest online library available. Musescore also allows for all their music to be downloaded in many different file formats, including MIDI. For our specific purpose, we needed to gather MIDI files for pieces of music that contain a melody (mainly vocal but can be any instrument) with an accompaniment (mainly piano but can also be guitar). We converged on using the “Piano-Vocal” arrangements from Musescore. We chose specifically these types of arrangements as all of their official scores on Musescore are of the same format: 3 tracks per midi file. The first track is the vocal line (this will

act as the input to the model), the second and third is the treble and bass clefs of the corresponding piano accompaniment (this is what our model will be validated against). There are 52,003 Piano Vocal arrangements on Muscore's website with more being added frequently. While some libraries exist for retrieving data from Muscore, the lack of maintenance has made most of these libraries inadequate for this purpose. Alongside the lack of an official API, we decided to build our own web scraper.

We used Selenium and python to scrape the website. Selenium is a library that allows the automation of chromium-based browsers such as Google Chrome. The scraper works by first logging in to Muscore using our personal account, applying the correct arrangement filter and then begins to download the scores one by one. The downloaded files are then dumped into our google drive where the data is ready for the next part of the pipeline, data processing.

4.1.1 Challenges Faced in Data Collection

Our first challenge we ran into was trying to find consistency in structure between all scores that we were downloading. Since Muscore is a public library that anybody in the world can create and upload to, there is a wide variety of styles in which scores and their corresponding MIDI files are created in. This would be extremely challenging for data processing as there would be no way to generalize across every human's stylistic preferences that they have while creating music. In addition to this, human beings are flawed, and can even create mistakes in what they compose/upload to the public library, this would add noise to our model and result in less effective training. To solve these problems, we decided to take advantage of Muscore's *official* scores, these are scores that are verified to be musically accurate and they are all created in the same structure/style, so there would be no discrepancies in structure between any of the MIDI files in our dataset. To access these however, one needs a premium account, which we then purchased. However, some of these official scores have special copyrights that prevents us from downloading them.

The biggest challenge of all that we ran into during this stage was Muscore's daily download limit. Even after purchasing a premium account, there was still a limit of 20 downloads a day per account, this significantly limits how much music we are able to download.

4.2 Supplemental Dataset

Due to Muscore's limit on downloads, we were working with very little data considering the larger size of our model. After training our model using only the data that we had collected, we were getting poor results and realized we needed significantly more data to train our model on. Therefore, we supplemented our dataset that we collected with another dataset known as the Nottingham Music Database [4]. This dataset consists of over 1000 folk songs that are stored in ABC files. The ABC files were converted into MIDI files and each song's corresponding chords and melodies were separated and added into our dataset. This added over 7000 new inputs for our model to be trained on.

4.3 Data Pre-Processing

Before pre-processing our data, we needed to first research and converge on how we wanted our data to be represented. After looking at various past papers, the two most common ways of representing musical data for machine learning applications are as an image or as a NumPy array. The image would be a picture of the MIDI file visualized on a piano roll and the NumPy array would utilize the metadata of a MIDI file to provide various features of a note. We converged on using the

NumPy array format as the input size would be smaller than the image and would require less computational power. To process the MIDI file and convert it into a NumPy array format that would be suitable for our purposes, we would need to extract all the note information and metadata from the MIDI files. To do this, we used a library called Pypianoroll which allows one to manipulate data within a MIDI file using python in meaningful ways [5]. We switched to this after using Mido for the interim report as Pypianoroll has more built in functions that are easier to work with and simplifies the data processing. After researching different kinds of NumPy array representations of MIDI files, we came across an article by a Data Scientist named Hunagwei Wieniawska [6]. This article implemented a way to do MIDI to array conversion in a format that we found suitable. We adapted parts of his code to get the array in the format and shape for we needed.

How our data processing works is as follows:

1. MIDI files from a directory are iterated over and passed to a function to be processed one at a time.
2. A function then converts the MIDI file into two arrays: one for the piano accompaniment and one for the vocals. The arrays are in the form of a binary matrix, where each row represents a unit of time and each column represents a note on the piano or vocals. The value in each cell indicates whether the note is being played or not at that specific unit of time. The unit of time that is being used is called a tick. A tick is the smallest unit of time measurement in Pypianoroll, there are usually 96 ticks for every beat in a song, this is known as the song's resolution.
3. Once the piano and vocal list are computed, all sequences are made to be the same length by padding them with zeros and the lists are then converted to NumPy arrays.
4. To decrease the input size of the song, the resulting array is then shrunk so that the smallest unit of time is no longer a tick, but rather a 1/16 note (24 ticks). This decreases the array to 1/24 of the original size.
5. This final array is then parsed into 4-bar intervals for both the piano and vocal arrays; each 4-bar interval is saved into separate arrays for the vocal and piano parts respectively. This data is now ready to be inputted into our model.
6. An additional step that was added since the interim report and after switching to Pypianoroll is that the key signatures for every song were also extracted from the meta data of each MIDI file and stored in an array, this was to help with the post-processing in section 6.
7. Lastly the output arrays were changed from 1/16 note time steps to 1/4 note time steps. This was done as the real data that the Discriminator is comparing against from the Nottingham dataset is mostly represented as chords in 1/4 notes. We found that having our model output 1/16 note matrices introduced a lot of noise in our outputs. This made it easy for the Discriminator to keep predicting the generated data accurately as it was in a completely different rhythm sequence. So to simplify the output and reduce the noise (number of notes being played overall), this change was made.

The final input to the model is a 64x88 NumPy array. The 64 rows represent each 1/16 note time step in every 4-bar interval. The 88 columns represent the 88 keys on a piano. Therefore, the input the model receives is an array in which each row represents the state of every note at that specific time step in the song. The values found at the cells of the activated notes are their respective note velocities.

4.3.1 Challenges Faced in Data Processing

One of the challenges we ran into when processing our data was trying to decrease the size of our input space. As mentioned earlier, the smallest time step that was being used initially to get the most accurate representation of the MIDI file in the array was a tick, which is $1/96$ of a single beat. We decided we needed to shrink our input space to make our model more computationally efficient, this meant that we needed to increase the size of each time step. We had two feasible options, $1/32$ note time steps and $1/16$ note time steps. Ultimately we decided to go with $1/16$ note time intervals as even though it's not the smallest time step in music theory, $1/32$ notes are extremely rare and not often used when singing or playing piano anyways. This allowed us to shrink our input size to $1/24$ of the original size as there are 120 ticks in a single $1/16$ note.

Another challenge was ensuring that our input space was kept at a constant size. Since we are using multiple MIDI files, each song will be of a different length, which means that an entire song's input size for the model would be different for every song. To ensure our input size remained constant, we parsed the resulting NumPy array of the entire MIDI into 4-bar chunks and processed them separately; any leftover bars would be appended with 0's until it would reach a 4-bar multiple. This also raised another challenge of not being able to parse accurately for time signatures that are not 4/4. We could either limit ourselves to only 4/4 songs to deal with this problem (most songs are in 4/4 anyways), or we could expand our processing to work around other time signatures as well, we chose to limit ourselves to only 4/4 songs.

5 Model

Utilizing the same principle as the MuseGAN researchers we decided to use a Generative Adversarial Network (GAN). The GAN model consists of two separate networks, the generator and the discriminator that are trained simultaneously. The generator creates new data while the discriminator attempts to differentiate between real and generated data points. In the field of generative AI, GANs are very commonly used and have shown great promise. GANs can utilize many different architectures for the generator and discriminator models, some examples include deep neural nets, long-short term memory (LSTM) networks as well as convolution neural networks (CNN).

Our model is inspired by the DCGAN, Deep Convolutional GAN [6]. Our specific model utilizes a data set that contains vocal melodies and a corresponding piano accompaniment. The input to the generator is the vocal melody, while the discriminator is trained using the input's corresponding piano arrangement. Our input space is represented by a 64×88 array, the data looks very similar to a conventional image and so we decided to utilize techniques from image processing. Convolutional Neural Networks have been used quite frequently in the image processing domain with a high degree of efficacy. These CNNs work by applying a convolution to the input with a fixed kernel. This results in the input being abstracted to a feature map. The theory is that the convolution picks up on certain features that are present in the input. After the convolution, it is common to decrease the dimensionality of the output with pooling layers, while this is common in image processing CNNs we did not use such pooling layers. Due to the similarity between our data inputs, we decided to utilize the known effectiveness of a CNN and combine it with the GAN model. This resulted in both the generator and discriminator consisting of a deep CNN that are then joined together in the training routine.

5.1 Architecture

The generator consists of 9 layers, 5 convolution-transpose layers with 4 batch normalization layers in between each convolution layer. We are also utilizing a ReLu activation function between the normalization and next convolution layer. The final layer does not use an activation function. This maps the input 64x88 array to a 16x88 array representing the piano accompaniment. The discriminator similarly consists of 4 layers, 3 convolution and 1 fully connected layer. Instead of a ReLu activation function we are utilizing a leaky ReLu. The final layer uses a sigmoid activation function. This maps the input 16x88 array to a single value representing the confidence in its prediction as 1 which is real data or 0 which is fake data. This model is adapted from "Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks" by Radford et al [8]. In this paper the researchers explain how previously merging CNNs with GAN architecture has led to sub optimal performance. To overcome the challenges of unstable training, and low resolution. The researchers then establish a set of constraints on a model's architecture to ensure the GAN is stable to train even with deeper architectures. The constraints are:

- Replacing pooling layers with strided convolutional layers
- Using batch normalization between layers
- With deeper architectures remove any fully connected layers
- Use the ReLu activation function for the generator except the output which uses a tanh
- Use the Leaky ReLu activation function in the discriminator

These constraints then guided our choices in building our model.

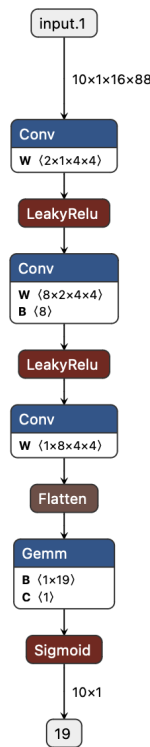


Figure 1: The Discriminator Model

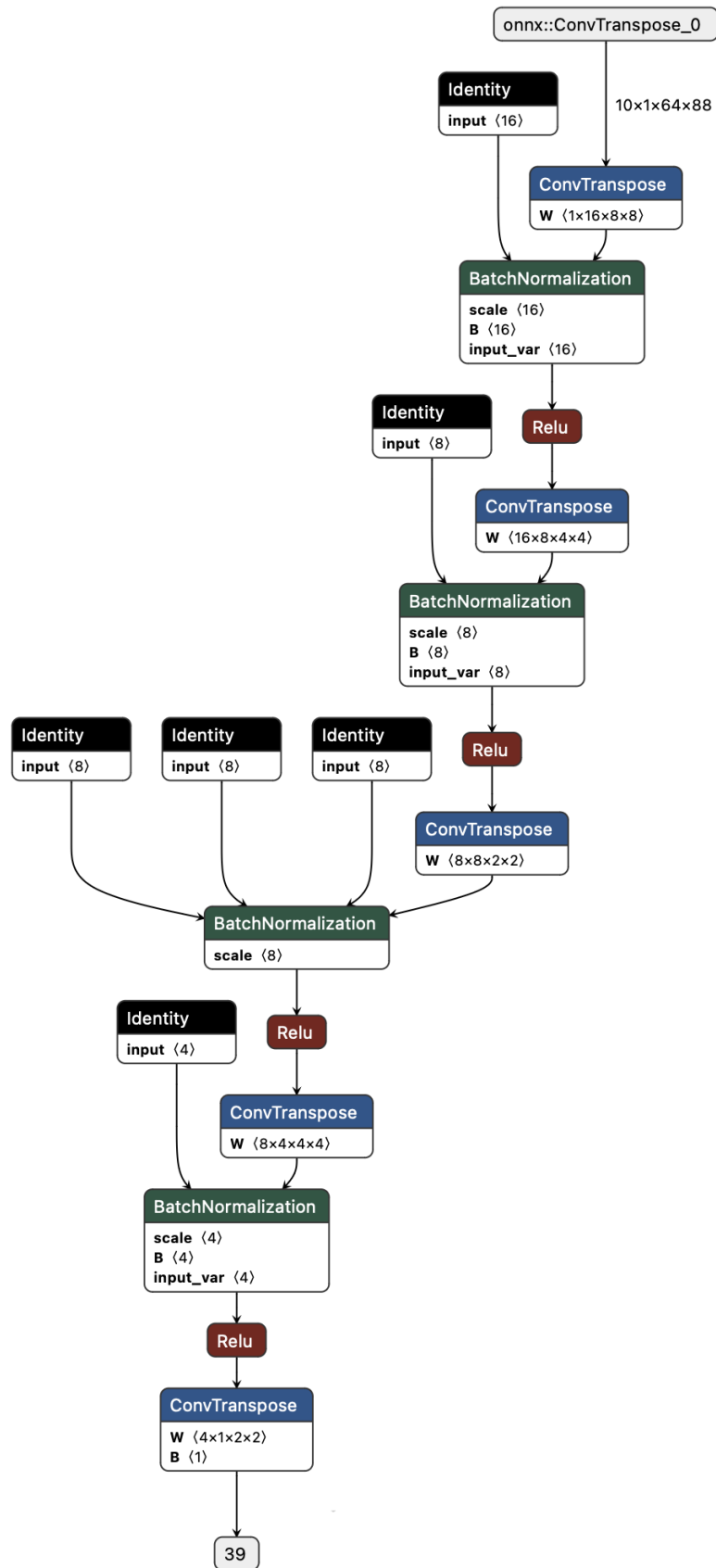


Figure 2: The Generator Model

Training the model consisted of a routine to train the discriminator one step and then a second routine to train the generator one step. The discriminator was first fed real human generated piano accompaniments and the loss was calculated. We are using a Binary Cross Entropy loss. After the real data was fed, fake data generated by the generator was fed to the discriminator, again loss was calculated. Using the sum of these losses, the discriminator is updated one step. Since the generator has already generated a set of fake values, the generator loss is calculated with those generated data points. This is then used to update the generator one step. We are utilizing the ADAM optimizer with mini-batch stochastic training.

5.2 Changes from the Interim Report

When writing the interim report our model faced a few issues. Firstly the model could only handle square inputs, this resulted in us feeding 64x64 arrays instead of 64x88, our first major step was ensuring our model worked with the correct inputs and produced the correct outputs. This was done by slightly modifying the architecture of the GAN to ensure it correctly padded the input array. The next issue we faced was converting the models array output back to a MIDI file. Our initial implementation had logic errors and as a result produced incorrect MIDI files. These issues mainly resided in our implementation of the convert to piano roll function. After some research we were able to find a library that handles MIDI files and converts to and from a piano roll representation. We utilized the Pypianoroll library and aquired our first set of reasonable outputs. With these issues resolved we were now able to generate correct MIDI's in the correct tempo and we decided that it was worth pursuing the development of this model further.

5.3 Challenges Faced With Model

As we began training our initial model we quickly realized that the Discriminator was performing exceptionally well. It would quickly learn to differentiate between the generated and real samples which resulted in the Generator stagnating whilst learning. To overcome these issues we began by training the model for more epochs. This unfortunately only further improved the Discriminator and did not improve our Generator. We then separated the learning rates of the Discriminator and Generator, lowering the learning rate of the Discriminator. This improved the loss disparity but still led to the Discriminator overtaking the Generator after a few epochs. We then changed the architecture of our model, reducing the Discriminator complexity and improving the Generators. This finally stabilized the loss curves of the Generator and Discriminator and no longer was one dominating the other.

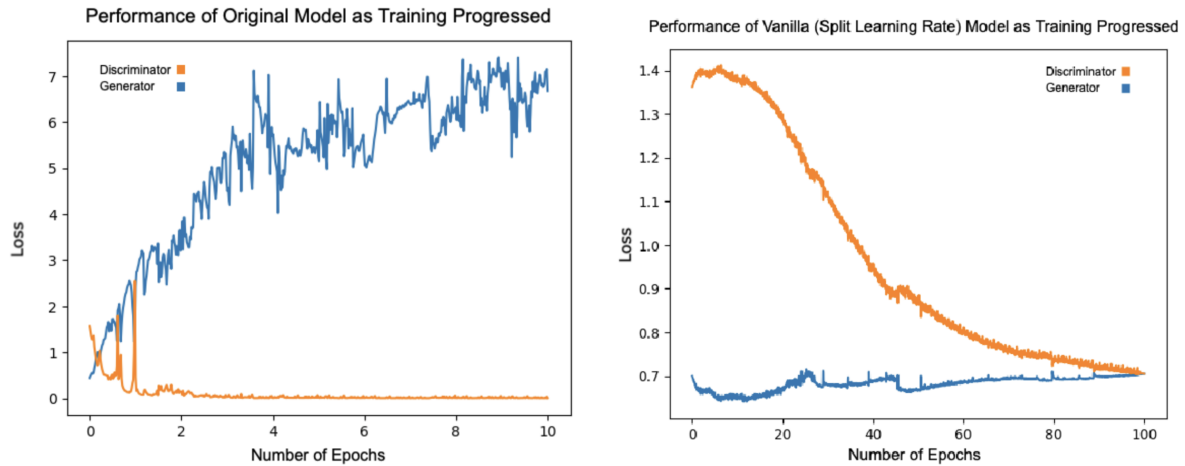


Figure 3: On the left we can see the discriminator quickly overtaking the generator, and the generator progressively getting worse. The right shows good convergence of both the generator and discriminator to a common loss

After achieving this goal we realized that all of our outputs were remarkably similar and upon closer inspection we realized that our Generator was repeatedly generating the same artifact across all inputs. Our network was experiencing mode collapse. To resolve these issues we utilized some tuning tricks from "Improved Techniques for Training GANs" by Salimans et al [9]. The first decision we made was to buy Colab PRO, and utilize the increased compute power to run significantly larger batches. This increased the diversity of the batches which helped solve our problem. Now our model was generating varied outputs but again there was an artifact at the beginning and at the extremes of the generated input that was present across the entire input space. Despite this partial mode collapse we were able to see that the model was beginning to learn rhythm and basic temporal structure of the music. We then decided to reduce the resolution of our output. Instead of generating in 1/16 notes, we switched to 1/4 notes. Now for our given input of 4 bars, instead of producing a 64x88 array we were producing a 16x88 array. The change in resolution does not affect an accompaniment as much since accompaniments usually do not utilize smaller notes like 1/8 and 1/16 and so minimal information was lost with this simplification. We also implemented a penalty for notes played in the very extremes of the piano as it is quite rare in western music to play incredibly high or low notes. With this new loss implementation we were finally able to generate our first valid musical accompaniments with no strange artifacts or mode collapse. Our success with implementing a rudimentary musical penalty motivated us to adapt our model to utilize known metrics to evaluate music.

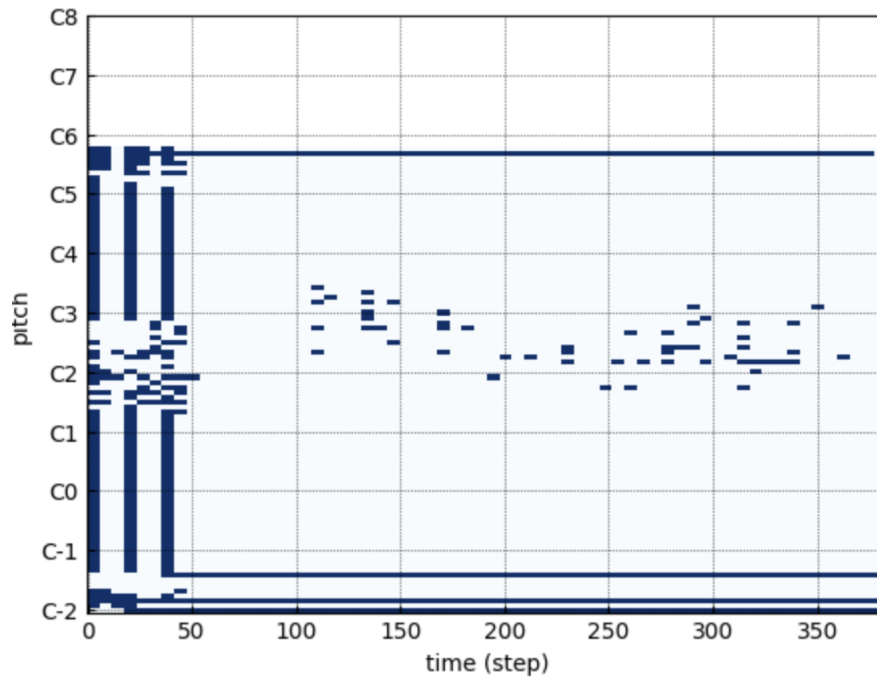


Figure 4: Partial Mode Collapse, the artifact on the far left of the model was consistently found across the input space. The section in the middle however was rhythmic and the beginnings of chord progressions emerged

Lastly, to clean the output of our data further, we introduced a static threshold value so that notes in the array that had a value greater than a certain probability would remain "turned on" and all other notes would be zeroed or "turned off". This meant that only values predicted with high probability were outputted and notes that the model was not as certain about were eliminated. At this point we also experimented with taking only the top 3-5 probable notes from the array and only outputting those instead of using a threshold value. This was done in cases where the differences in probability between the notes in the 80th percentile were relatively similar to the probability of the notes in 50th percentile. This is because in these cases, the threshold value would no longer be an effective filter as the probability distribution between all notes are significantly more evenly distributed than not.

5.4 Custom Loss

The MuseGAN authors use a series of metrics to evaluate the music that they generated [2]. These metrics include but are not limited to tonal difference, polyphonic rate, number of pitches used. We decided to implement custom version of these metrics as loss functions for our Generator to penalize outputs that did not meet these metrics.

5.4.1 Chromatic Difference

The first metric we implemented is chromatic difference. This is a simplification of the tonal difference metric utilized by MuseGAN. In western music the 12 semitones that form an octave are essentially indistinguishable from one another. While a C0 and C4 may sound different, having one or the other does not affect the discordance of a musical piece, only the fact that the note

is a C affects this. We can essentially reduce the 88 keys of a piano to these 12 semitones, also known as a chroma. For instance in a single bar, if C4, D4, C5, E4 are all played once; the chromal representation of this bar would be **C:2, D:1, E:1**. By representing the input and output of the model as a chroma, we can then calculate the difference between the two. Reducing the chromatic distance will lead to music that is more harmonious as it promotes choosing notes in the same key as the input. Our second model optimized with regards to this metric in addition to the original binary cross entropy loss.

5.4.2 Polyphonic Rate

Another metric to measure the musicality of a given piece of music is polyphonic rate. A polyphony is when multiple notes are played at the same time, it is commonly considered that there needs to be at least 3 separate notes played for a polyphony to occur. Music that only consists of a few notes played at a time lacks significant depth. This is because there is a much smaller subset of playable intervals with 2 notes whereas with more notes there is a greater set of playable intervals. Due to this increase, increasing the occurrence of a polyphony may result in more musical music. Polyphonic rate is simply the ratio of polyphonies occurring to total number of time steps in a given bar of music. You would want this rate to approach 1, so that at each time step a polyphony is occurring. We implemented this metric as an additional loss for our third model.

5.4.3 Number of Pitches Used

The final metric we utilized while training the models is the number of pitches used. This metric simply calculates the number of unique notes played in a certain time interval. In our case we measured across a single bar of music. If music only consists of a small subset of the 88 keys on the piano, it would be dull and boring. Thus Increasing the number of unique notes played would help the musicality of the generated piece. It is important to remember though that having too many different notes may lead to keyboard mashing where all the notes are just played constantly. With this in mind, we implemented this metric in our fourth model with a tunable hyper parameter so that we could adjust just how important it is that the number of pitches used is increased. By tuning this parameter we were able to create our fourth model.

5.4.4 Hybridization

While individually these metrics provide useful information about the generated music, we believed that good music would excel in these metrics simultaneously. This led us to training 2 other models that utilized a mix of the two loss functions. We decided to implement mixing of loss functions in two ways. The first which we call the Hybrid model works by implementing the loss function as a sum of both chromatic distance and polyphonic rate. We chose these two metrics as we believed these two metrics do not directly oppose each other. As long as the additional notes chosen to improve polyphony match the input, or outputs chroma it would not affect the chromatic difference. Similarly choosing notes that match with the input, or outputs chroma would not affect the polyphonic rate. With this in mind we trained the first of our hybrid loss models. Our second joint loss model which we called the Hybrid-Step model, implements the joint loss in a different fashion. First the model optimizes with regards to the polyphonic rate metric, after training for some amount of epochs the model is then optimized with regards to the chromatic difference metric, i.e. the model is trained in steps. This model required us to tune the number of epochs for each step of training.

5.4.5 Important Considerations

It is to be noted that with all of our models that utilized a custom loss metric we did not need to deal with mode collapse and had significant variation within the input space. It is also important to say that we could not implement these metrics exactly as the MuseGAN authors did as they did not vectorize the operations. Implementing the metrics without vectorization would have greatly increased the compute power needed and would have severely decreased our training capability.

6 Post-Processing

Once we receive an output from the model in the form of the NumPy matrix, we post-process the data to clean it and prepare it for conversion back into a MIDI file. Firstly, the first 20 columns of the matrix (the lowest 20 notes on the piano) are all turned off as the model tends to produce more noise in that note range. Next, in order to help the music sound better, notes that the model predicts that are not from the song's key signature are moved to the index corresponding to the closest note in the correct key signature. This is done by iterating through every note that is turned on in the array and checking whether it is part of an array that contains all valid indices of notes in the scale. If the note in the matrix is in a column that is not from the scale array, the nearest index from the scale array is found, and the incorrect note is moved to the new valid index after the note in its previous position is turned off.

Once this is done, all notes in the array should be part of the correct key signature. Next, we deal with minimizing the number of dissonant intervals that occur in the array. The dissonant intervals in music are minor seconds (1 semitone apart), major seconds (2 semitones apart), augmented fourths (6 semitones apart), minor sevenths (10 semitones apart) and major sevenths (11 semitones apart). Since the array has already been processed to move all notes into the correct key signature, the only dissonant intervals that are left to minimize are the minor seconds, major seconds and major sevenths. To eliminate these intervals, our script loops through each row of the matrix and identifies invalid notes that create dissonant intervals. Specifically, if a note appears in the same time step as a note that is either 1, 2 or 11 semitones away from it in any direction, it is considered invalid. The function creates a list of invalid notes for each row and stores it in a "banned" list that contains the invalid notes for each time step. The function then loops through each row of the matrix argument again and checks if each note is in the list of invalid notes for that time step. If it is, it sets the note to 0 and finds the closest valid note to replace it with. If it is not, it adds the note to the list of invalid notes for that row, along with the notes that cause dissonant intervals from it in any direction.



Figure 5: The Dissonances of Western Music Theory [10]

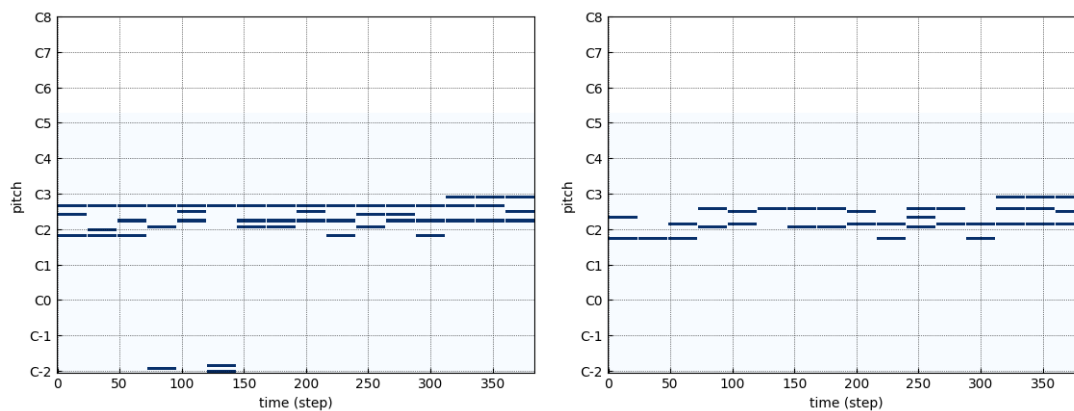


Figure 6: Piano roll visualization of generated output from model before (left) and after (right) post-processing

As seen in the piano rolls above, the noise in the low note range is removed, and all the dissonant intervals are removed. The piano roll without post-processing has many minor seconds and major seconds within the same time steps as seen by the thicker lines (two consecutive notes stacked on top of each other). These are dealt with successfully after post-processing as seen by the piano roll on the right that has no notes stacked on top of each other and has notes spaced in non-dissonant musical intervals.

It is important to note that this final stage of processing is only done on our demo generated samples to help improve the listening experience. This final step on average only shifts a small subset of notes preserving most of what was generated.

7 Results

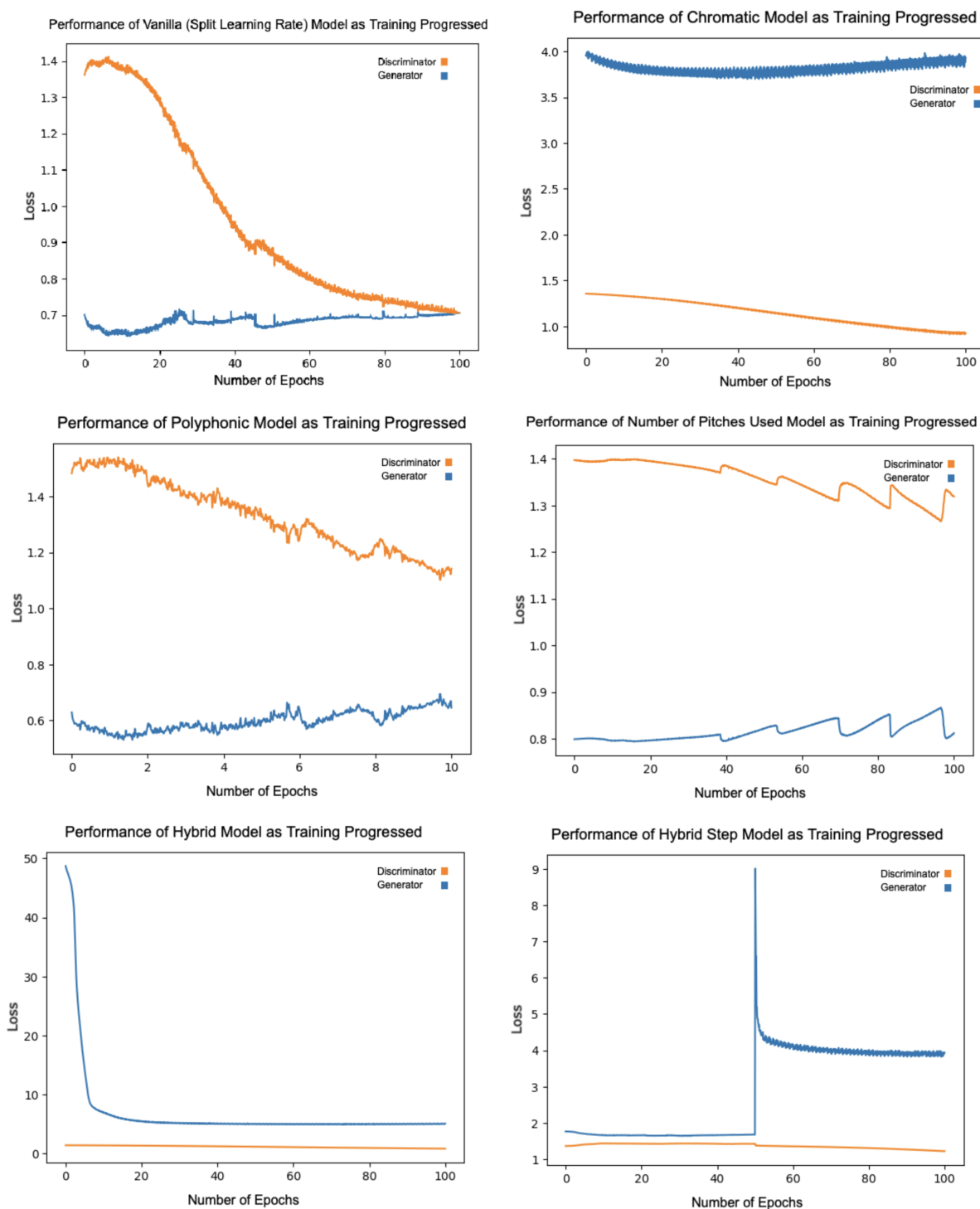


Figure 7: The graphs show the loss with respect to number of epochs for all our models. From left to right we have the Vanilla Model, the Chromatic Model, the Polyphonic Model, the Number of Pitches Used Model, the Hybrid Model, and the Hybrid Step Model

Since the loss calculated for these various models is different and utilizes different metrics we cannot use these graphs to compare the models with one another. However we can use these graphs to visualize how training is progressing and if any GAN training failures are occurring. Looking at the Vanilla model we can see that after tuning several hyper parameters and splitting the learning rates for the Generator and Discriminator we were finally able to achieve stable training of the model. The two losses converging over time shows that our model is stable and that the networks have stabilized. We can also see in the Hybrid Step Model's graph exactly where the step from one loss function to the other occurred.

7.1 Comparison of Models

To compare and evaluate our various models, we implemented the above mentioned metrics on two test data sets. One test set was a subset of the Nottingham data set and was very similar in genre and structure to the music that the model was trained on. Our second test batch consisted of music that we scraped from Musescore. This was a more varied test batch as the Musescore data set spans many different genres and even incorporates melody lines that are not vocal, for instance violin. The overall performance of the 6 models are shown in the graph below.

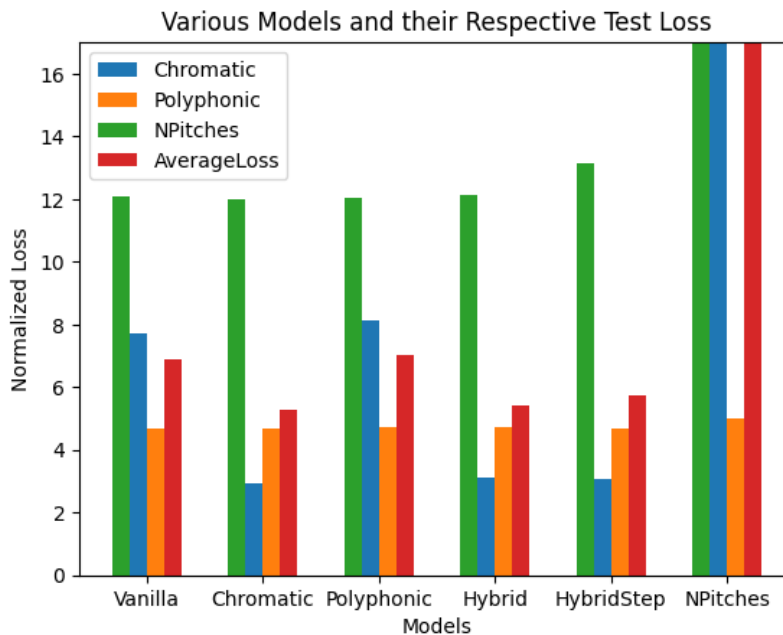
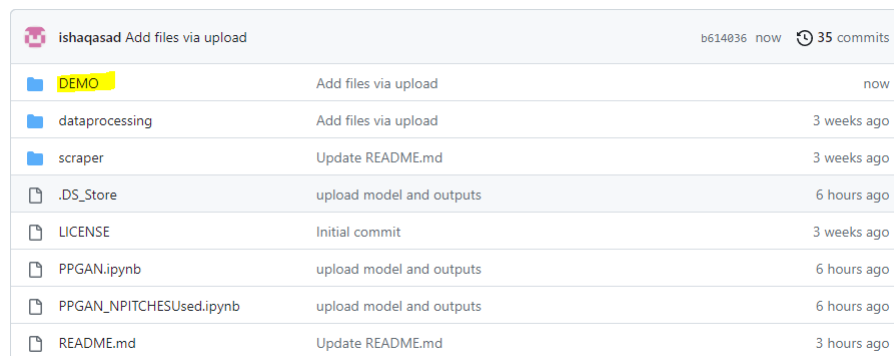


Figure 8: Various models and the average loss computed across the two test sets. The average loss was used to select the Champion Model.

7.2 PPGAN Demo

To choose our final champion model that would be dubbed "PPGAN" We chose the model that had the lowest normalized mean loss across all 3 metrics. This was found to be the model trained with the chromatic distance loss metric. Though not far behind was our hybrid model. This model gives us the smallest loss while outputting arrays that have higher probabilities in its prediction, this gives us confidence that our model is able to output something that sounds like music. Therefore to officially select our champion model, moving forward the chromatic distance model will be called

PPGAN. However, as music is something that needs to be listened to, these losses and numerical metrics are hard to use to gauge whether the generated music is coherent. We then used PPGAN to produce accompaniments for various popular pieces of music. We began with simpler music, that is, music with fewer changes in the melody, basic rhythms and increased repetition, such as "Mary Had a Little Lamb" and "As the Deer". We then tested it on slightly more complex music, so we chose some pop music such as "All of Me" by John Lennon and "Someone Like You" by Adele; these songs have more complex melodies as they span a longer range and more diversity in rhythm. Finally we stepped up to music with even greater complexity, that is, classical music; it has more than the basic four pop chords, and again even more complex melodies; we chose "Canon in D" by Johann Pachelbel. The 4 bar outputs were then stitched together by us into short segments that can be heard on our GitHub page. [here](#)



ishaqasad Add files via upload		b614036 now 35 commits
DEMO	Add files via upload	now
dataprocessing	Add files via upload	3 weeks ago
scraper	Update README.md	3 weeks ago
.DS_Store	upload model and outputs	6 hours ago
LICENSE	Initial commit	3 weeks ago
PPGAN.ipynb	upload model and outputs	6 hours ago
PPGAN_NPITCHESUsed.ipynb	upload model and outputs	6 hours ago
README.md	Update README.md	3 hours ago

Figure 9: Where to find the generated sample files

7.3 Overall Performance

Overall, as seen from the final outputs of PPGAN, it is capable of generating music that is mostly musically coherent. PPGAN has effectively learned relationships between notes in music theory as it chooses notes that end up making chords from the same key. It has also learned how to generate walk ups (a progression of chords from a low root note to a higher note in the scale), when transitioning between chords/time steps. Alongside this, PPGAN would also choose interesting note flourishes outside of a regular chord which added nuance and depth to the music. PPGAN is also able to understand basic rhythm as it changes notes only at the start of a beat. Due to the input of the GAN being an actual melodic line when we stitched the various outputs that we generated together they came together well. We hypothesize that this is due to the model learning the underlying rhythm from the input which is mostly consistent across a bar line. PPGAN has learned to generate very basic piano accompaniments on its own given a melody which we deem a success. However, the outputs lack creativity; there is a lot of repetition in the notes themselves and also in the rhythm. This makes sense as the chromatic difference loss makes it such that it is actually more beneficial to play notes that the model knows are correct repeatedly so as to minimize this loss.

8 Next Steps

Currently the only human verification that we have implemented is listening to select generated outputs and commenting on their quality based on our own subjective musical preferences (based

on our knowledge of objective musical metrics of course). A key portion of generative artificial intelligence is generating plausible outputs that would be able to fool a human agent. We need to create a listening survey where individuals of varying musical training listen to and rate the models output.

Another way to validate how well our model is performing in practice is to test whether an individual could easily identify the song that one of our outputs belongs to. We propose another survey where individuals would be asked to select from a given set of melodies what the input to our generator was. This would really test the effectiveness of our model as a tool to aid in music generation.

Currently our model is trained on a very small subset of music available, we need to expand the current data set and train our model on significantly more data, hopefully allowing us to generalize to other genres. Currently our generator model consists of 3,872 parameters, and our data set consists of 7,000 datum points. This is a smaller and possibly insufficient training set for the advanced results we want to achieve and we hope to increase the size by at least one order of magnitude to ensure a more generalizable model.

There are also many other metrics that we can use to measure the quality of the music generated. In any given key of western music theory there are several known dissonances, specifically the intervals minor second, major second, augmented fourth or tritone, minor seventh and lastly the major seventh. These are summarized in figure 5. Measuring the quantities and rate of these dissonances can be implemented as an additional custom loss function, we theorize that this loss in addition to the chromatic difference loss, has potential to perform even better than the current PPGAN.

9 Ethical Implications

As we develop machine learning models to generate music, it is essential to consider their broader impact and ethical implications. One of the most significant potential benefits of this project is the ability to generate high-quality music with fewer human resources, which could make music creation more accessible and affordable to a broader range of people.

However, we must also consider the potential ethical implications of our work. One potential concern is that the proliferation of AI-generated music could lead to a devaluation of human creativity and originality. If AI-generated music becomes widely available and highly realistic, it may be challenging to distinguish between compositions created by humans and those generated by machines. This could lead to a devaluation of the work of human musicians, composers, and producers, potentially causing economic harm and eroding the cultural value placed on human creativity.

Another concern is the potential misuse of AI-generated music, such as in deepfakes, propaganda, or other types of misinformation. It is essential to ensure that our models are not used to generate music that is used to manipulate or deceive people. These deepfakes can also offend the music producers as their work is compared with machine music, significantly reducing their work's value.

Overall, as we continue to develop our machine learning models, it is critical to consider their potential broader impact and ethical implications. By doing so, we can ensure that our work contributes positively to society while minimizing the potential risks and negative consequences.

10 Conclusion

In conclusion, PPGAN has effectively learned the fundamentals of music theory as it is able to choose notes that end up making chords from the same key as the input melody. It is also able to transition between these notes in correct rhythm patterns. PPGAN utilizes a deep convolutional GAN (DCGAN) architecture with a custom chromatic difference loss function resulting in a system that can generate musically plausible and stylistically consistent accompaniments. Although challenges such as mode collapse and an overpowered discriminator were faced during the development process, these were resolved through architecture modifications and tuning of learning rates. The development of various models that utilized different musical loss functions like chromatic difference, polyphonic rate and number of pitches used proved to be useful in understanding which losses had the most impact on improving musicality. These metrics also aided in comparing the various models' performance. In the end, the chromatic difference model was chosen as the champion model and dubbed PPGAN.

The potential impact of this project is significant, as it could allow musicians, composers, and music producers to focus more on other aspects of their music-making process while reducing the need for skilled accompanists. The ability to generate harmonious piano accompaniments quickly and easily would benefit both professionals and amateurs alike, leading to the emergence of new musical ideas and styles. Moreover, exploring the best approaches for training AI to understand the intricate relationship between melody and accompaniment in music could reveal new insights into the capabilities and limitations of AI in music generation, with broad implications for the field of AI and beyond.

11 Appendix

Link to the Github repository

<https://github.com/ishaqasad/PPGAN>

12 References

1. I. Simon, D. Morris, and S. Basu, “MySong,” *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Apr. 2008, doi: 10.1145/1357054.1357169.
2. H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, “MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment.” [Online]. Available: <https://arxiv.org/pdf/1709.06298.p>
3. J. Zhao and G. Xia, “ACCOMONTAGE: ACCOMPANIMENT ARRANGEMENT VIA PHRASE SELECTION AND STYLE TRANSFER.” Accessed: Feb. 02, 2023. [Online]. Available: <https://arxiv.org/pdf/2108.11213.pdf>
4. James Allwright, B. V. V. (n.d.). ABC version of the Nottingham Music Database. The ABC Music project - The Nottingham Music Database. Retrieved April 18, 2023, from <https://abc.sourceforge.net/NMD/>
5. “Pypianoroll,” *Pypianoroll - Pypianoroll documentation*. [Online]. Available: <https://salu133445.github.io/pypianoroll/>. [Accessed: 18-Apr-2023].
6. H. Wieniawska, “Convert MIDI file to numpy array in Python,” Medium, 08-Aug-2020. [Online]. Available: <https://medium.com/analytics-vidhya/convert-midi-file-to-numpy-array-in-python-7d00531890c>. [Accessed: 27-Mar-2023].
7. “MidiTok: A python package for Midi File Tokenization.” [Online]. Available: <https://hal-sorbonne-universite-fr.ezproxy.u-pec.fr/hal-03418930/document>. [Accessed: 02-Feb-2023].
8. A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative Adversarial Networks,” arXiv.org, 07-Jan-2016. [Online]. Available: <https://arxiv.org/abs/1511.06434>. [Accessed: 27-Mar-2023].
9. T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” arXiv.org, 10-Jun-2016. [Online]. Available: <https://arxiv.org/abs/1606.03498>. [Accessed: 18-Apr-2023].
10. “Consonance and Dissonance ,” 5.3 Consonance and Dissonance. [Online]. Available: <https://www.earmaster.com/music-theory-online/ch05/chapter-5-3.html>. [Accessed: 18-Apr-2023].