ARTIFICIAL INTELLIGENT MUSIC CREATION USING RECURRENT NEURAL NETWORKS

Tony Zhang
Department of Computer Science and Engineering
University of Minnesota
zhan6696@umn.edu

ABSTRACT

The abstract paragraph should be indented 1/2 inch (3 picas) on both left and right-hand margins. Use 10 point type, with a vertical spacing of 11 points. The word ABSTRACT must be centered, in small caps, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 INTRODUCTION

Music is a form of art that has been around since prehistoric times and is still prevalent in cultures around the world. This art form has the power to influence our emotions and actions. Even people as far back as the Ancient Greeks were interested in the nature of music and how it works (Roberts, 2018). Given that music is such a powerful medium, it is a great complement to multi-modal pieces of work such as video games. Having a well-written piece of music playing during a video game can enhance the user's overall experience and enjoyment of the game.

Creating a video game takes a lot of time. Among other tasks, the developer has to create a storyline, devise a fun gameplay loop, create visuals and graphics for the user to interact with, and compose music to supplement the atmosphere. With all these tasks, it is valuable to be able to automate any part of the process. Automatic composition of video game music tends to be quite complicated in several aspects which we will mention briefly.

First, the quality of music is subjective. There is no inherent way to measure the quality of music since its value is in the eye of the beholder. However, within a culture like video game players, most members of the culture can agree upon whether a piece is good or bad. This property allows a computer to detect patterns in the music to determine what sounds good to this group. Additionally, the quality of generated music can be checked by whether it's statistical properties match the typical game soundtrack.

Second, video game music is part of a larger context. Each task is part of the overall product and is meant to make sense in the content of the rest of the product. Therefore an automatic machine would need to know the context in which they are creating to generate art like this.

Lastly, music typically has multiple instruments playing notes at different times. This means that a model that wants to produce multi-instrument music must be able to understand the relationship between the notes played by the different instruments. Each instrument has it's own characteristics which should try to work in harmony with each other (Zhu et al., 2020).

An example of multi-instrument music is the soundtracks that accompany games on the Nintendo Entertainment System (NES). Due to its limited technology, the NES only has five channels in which it can produce audio: two synth leads (square waveform), one synth bass (triangle waveform), a noise channel that acts as percussion, and a PCM Sample channel that plays back small samples of recorded audio. Each channel acts like its own instrument being able to play one note at a time.

In this report we focus on generating music for video games in the style of music found on NES titles. to make the project more feasible in the given time frame, we will not attempt to make the music fit into a video game context, but rather be its own standalone piece in the style of NES music. Additionally, we will be ignoring the sample channel as it is not easily represented in music composition files.

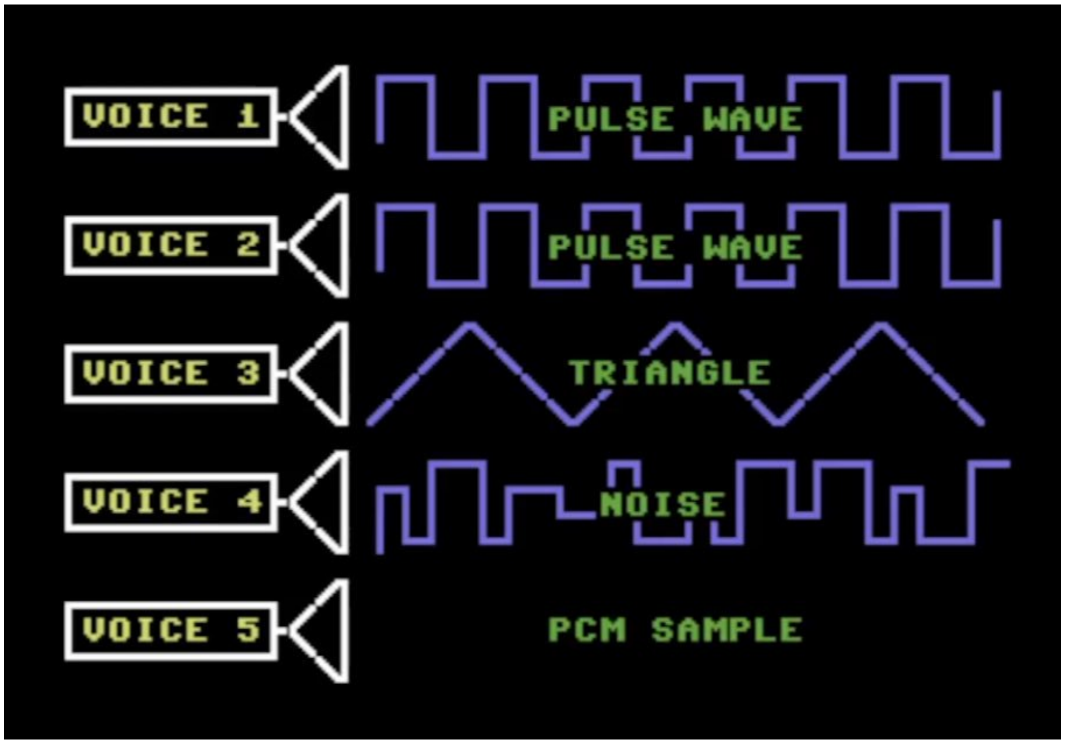


Figure 1: The five sound channels available on the Nintendo Entertainment System (D8A5NR, 2018)

2 BACKGROUND

Music generation is a difficult task so many methods have been devised that attempt to do this task. This section discusses some of the work done in this field to give context to the method we used in this experiment.

2.1 HIDDEN MARKOV MODEL

Markov Models are used to represent sequences of states that change over time. It does this by using a state transition matrix which contains the probability of moving to any state given a fixed number of previous states. They are trained with a sequence of states and the next state, and maximum likelihood estimation is used to find the state transition matrix which maximizes the chance of generating the training sequences.

Hidden Markov Models (HMMs) are different in that the states of the sequence are no longer available, and the model only gets to see the outcome of that state at each point in the sequence. There are two necessary assumptions that the HMM makes. First, given the current hidden state, the current outcome is independent of the previous hidden states/outcomes. In other words, The current output is determined only by the current state. The second assumption is that there is an emission probability matrix that tells the probability of some state producing some output.

With these two assumptions we can then derive the probability that some sequence is generated, and the probability that some sequence of states generated this sequence. This allows us to train for the state transition matrix and the emission probability matrix which has the maximum likelihood given the training sequences. In the context of music generation, hidden states are used to generate notes as the output. SuperWillow tackles this problem by separating melody generation and chord progression into different HMMs (Schulze & van der Merwe, 2011).

Since HMMs use information from the previously generated note to determine the next transition, it tends to think similarly to the ways that humans think about music in short time frames. This makes the music sound coherent in the short term. Some drawbacks of this approach are that long-term structure is difficult to accurately model with this method since the output at each state is independent of all the previous states. Additionally, the solution space tends to be quite large so it takes a long time to train.

2.2 Hierarchical Recurrent Neural Network

A hierarchical recurrent neural network (HRNN) is designed to process sequential data, such as text or time series data. It is called a hierarchical network because it contains multiple levels of memory, with each level processing a different time scale of the data. An HRNN is made up of multiple recurrent neural networks (RNNs), which are neural networks that can process sequential data. The top level of the HRNN processes long-term dependencies in the data, while the lower levels process shorter-term dependencies. The output from the lower levels is fed back into the higher levels, allowing the network to incorporate information from multiple time scales when making predictions.

Song From PI is an HRNN used to generate music. The network has 2 layers at the bottom for generating the melody, 1 level for generating chords, and the top layer for generating drums. This makes sense musically because the bottom layers are meant to encode shorter time dependencies and the melody typically is the fastest-changing part of the music. Chords change less often, and drum patterns tend to be extremely repetitive (Chu et al., 2016).

HRNNs are well suited for processing sequential data which makes them a good fit for dealing with music data. Their hierarchical nature gives them the potential to model the connections between the different levels of music such as how the melody related to the chord being played underneath it. However they are quite complicated, so producing coherent music is difficult. Additionally, they also take a lot of computational power to train.

2.3 Generative Adversarial Network

A generative adversarial network(GAN) is comprised of a generator and a discriminator. The generator creates samples that are meant to be similar to the training data. The discriminator then tries to tell whether the generated sample is real or not. The generator learns from the discriminator, which in turn creates better samples and improves the discriminator. These two parts of the GAN are trained simultaneously and the result is a generator that creates samples that are difficult for the discriminator to distinguish from real data.

MidiNet uses a GAN to generate melodies, and the process is briefly described below. A melody can be turned into a discrete 2D piano roll format that restricts the pitches and time frames that a note can fit into. The generator is trained to turn some random noise into this 2D piano roll format trying to emulate the generated piano roll from a real midi file containing a melody. The discriminator then tries to determine whether the piano roll was generated by a real midi file or the random noise. In this way, the generator learns how to create unique realistic melodies from a random input (Yang et al., 2017).

Due to the nature of how GANs are trained, it is possible to create music that sounds natural like a person wrote it. Also, they are versatile for the types of inputs that can be provided. The main challenge of training a GAN is its sensitivity to the quality of training data. If the training data is not diverse or well structured enough, then the results may not be very convincing.

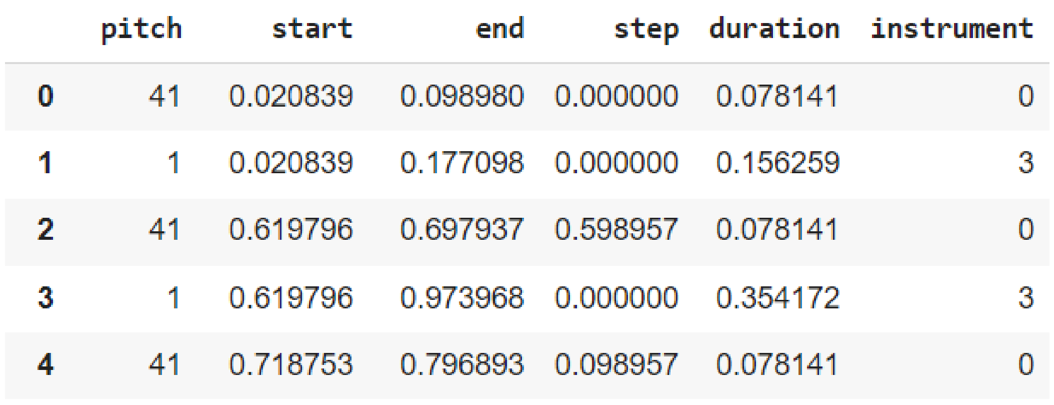
3 Methods

This section goes over the approach that was used to generate music in the style of NES soundtracks. The data used for training is the Nintendo Entertainment System Music Database (NESMDB). This database contains 5278 soundtracks from 397 NES games representing 296 composers. The midi files were created from assembly code within the game's files. This allows researches to study this music without needing knowledge of an archaic audio synthesis chip (Donahue et al., 2018).

3.1 Representation

To compose multi-instrument tracks in the style of NES music, I chose to use a Long Short-Term Memory (LSTM) Recurrent Neural Network (RNN) on a note level to model the patterns of the music. I chose this method because the methods above assume some kind of structure that the music should fit into. This can work well for pop music and folk music which tend to not have a lot of variability from sample to sample. However, there are a lot of different genres of music within video games so I wanted to try a more general approach.

The data for my training music came in the form of midi files. Midi files encode music into different instruments which each contain notes. Notes are discrete events that have properties like pitch, step, and duration. These numbers tell the instrument what frequency to play a note, when to play it, and for how long. To turn the music into trainable data, each midi file was converted to a data frame of note information.



	pitch	start	end	step	duration	instrument
0	41	0.020839	0.098980	0.000000	0.078141	0
1	1	0.020839	0.177098	0.000000	0.156259	3
2	41	0.619796	0.697937	0.598957	0.078141	0
3	1	0.619796	0.973968	0.000000	0.354172	3
4	41	0.718753	0.796893	0.098957	0.078141	0

Figure 2: An example of a data frame of notes information extracted from a midi file.

Additionally, the notes of a song can be visualized by their instrument in a piano roll format.

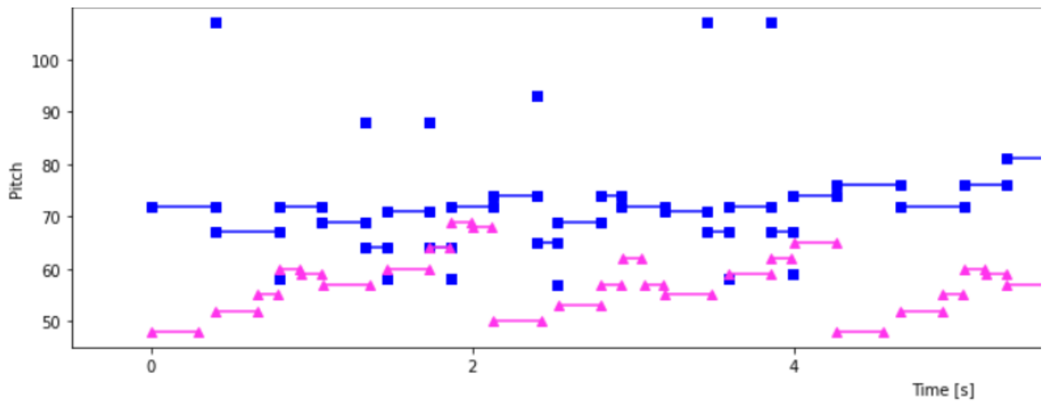


Figure 3: A plot of the first few seconds of the song Clu Clu Land - Bonus Stage by Akita Nakatsuka

All the notes from each song were then aggregated into a single list to prepare for turning the data into the input and expected output. In order for the LSTM to learn from the music, the notes were separated into blocks of 26 notes, where the first 25 act as the training sequence and the last note is the target variable. It then will learn how to predict the next note based on a sequence of previous notes.

3.2 Algorithms

There are 4 features that are being predicted in each note: pitch, step, duration, and instrument. To get the network to learn the pattern of the music, it needs to be able to evaluate how well it's making the prediction. Therefore we will choose a loss function as the heuristic evaluation metric. For discrete variables like pitch and instrument, the discrete values were turned into one-hot encoded vectors for easier training. Then cross-entropy loss was used on the one-hot vectors. For continuous variables, mean squared error was used.

consider removing these equations

Cross-entropy loss: where m is the number of target classes, y is the true value, and $\hat{y}$ is the predicted value

$$L = -\frac{1}{m}\sum_{i=1}^{m} y_i \cdot \log\left(\hat{y}_i\right) \tag{1}$$

For the model to improve its music generation, it needs a way to reduce the loss. This was done with the stochastic adaptive gradient descent algorithm ADAM. This algorithm was chosen because it combines a lot of popular methods used to improve stochastic gradient descent (SGD) performance. By using both an adaptive learning rate and momentum-based gradient, ADAM has been shown to perform better than other SGD algorithms.

4 Design of Experiment

Before a fair experiment could be conducted, an unbiased way to generate audio from the midi files was required. Several methods of rending midi files were evaluated. In particular, NESMDB includes some code to render their midi files into accurate playbacks of what the music would have actually sounded like on the NES. However, I could not get the generated midi files from my model to work with their program. Therefore I chose to use MuseScore3, a free music composition software, to render both the real midi files and the generated ones (Anatoly-os, 2018). The issue with this method is that MuseScore assumes that all midi files come in at 120 beats per minute, which means that it had to use some funky rhythms to approximate the true rhythm of the midi files. This meant that the rhythmic integrity of the real pieces suffered somewhat in the rendered audio files.

To test the quality of the music generated, a survey was conducted that asked participants to rate different aspects of the quality of several pieces of music, and whether they thought the music was from a real video game or if it was generated. Some of the pieces of music were generated and some were picked from the test set of NESMDB.

These are the criteria of the music that the participants were asked to evaluate. Evaluations were on a 1 to 5 scale with 1 being the lowest score and 5 the highest

Rhythm Does the music have coherent tempo and rhythmic patterns?

Melody Does the music show some kind of melodic nature?

Integrity Does the music sound coherent like it was intentionally created?

Enjoyability Would you play a video game that has music that sounds like this?

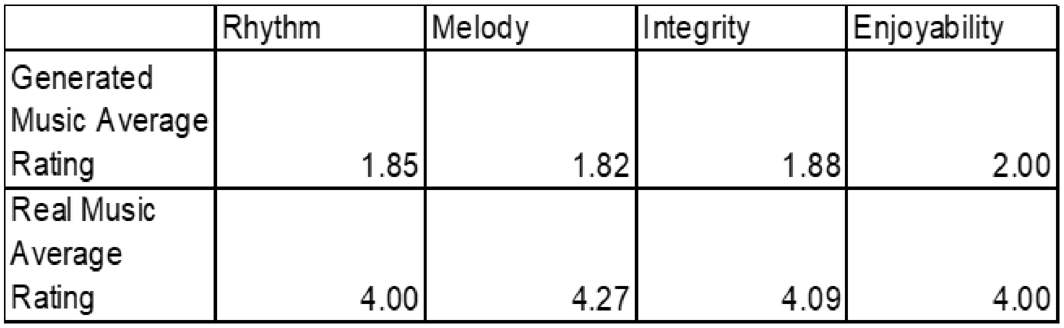
Additionally, participants were asked to say whether they thought the music was real or generated. From this data we will evaluate our generated music against the real music with two questions.

1. Do people identify the generated music as fake more often than real music?

2. Is there a difference in the rating of musical criteria between the real and generated music?

5 Results

In total, 11 participants were gathered for the experiment. Each participant evaluated 3 real songs and 3 fake songs without knowing if they were real or not.



	Rhythm	Melody	Integrity	Enjoyability
Generated Music Average Rating	1.85	1.82	1.88	2.00
Real Music Average Rating	4.00	4.27	4.09	4.00

Figure 4: Average Ratings of both music types by criterion

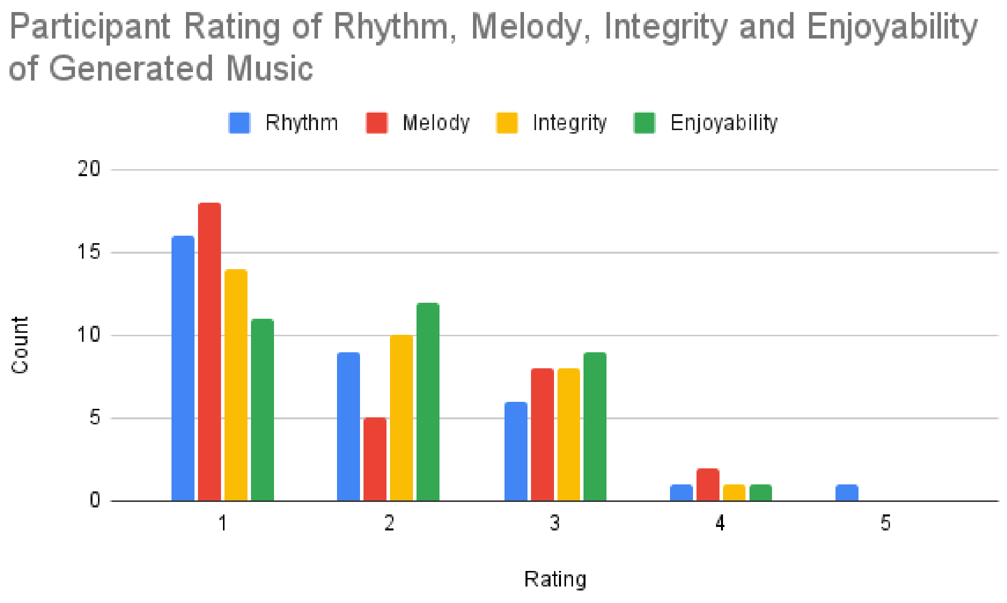


Figure 5: Counts of ratings for musical criteria of generated music

6 Discussion

The data above shows that the participants rated the generated music far less favorably than the real music. Enjoyability was the best metric for the generated music, and melody was the best metric for the real music. Additionally, melody was also the metric with the largest difference in average ratings.

With an accuracy of 79%, it is clear that participants were quite good at determining if the music was real or not.

Therefore, the results from the experiment indicate that that there is a significant difference in nearly all the evaluation criterion between real and the model's generated music.

7 Conclusion

It can be concluded that people prefer the music of real composers over the generated music of this project. It seems to preform less well that the results shown in the models referenced earlier in the Background section. This could be due to the model the model I chose not have enough structure to it. Since the model did not have built in levels of separation between melody and harmony, the model

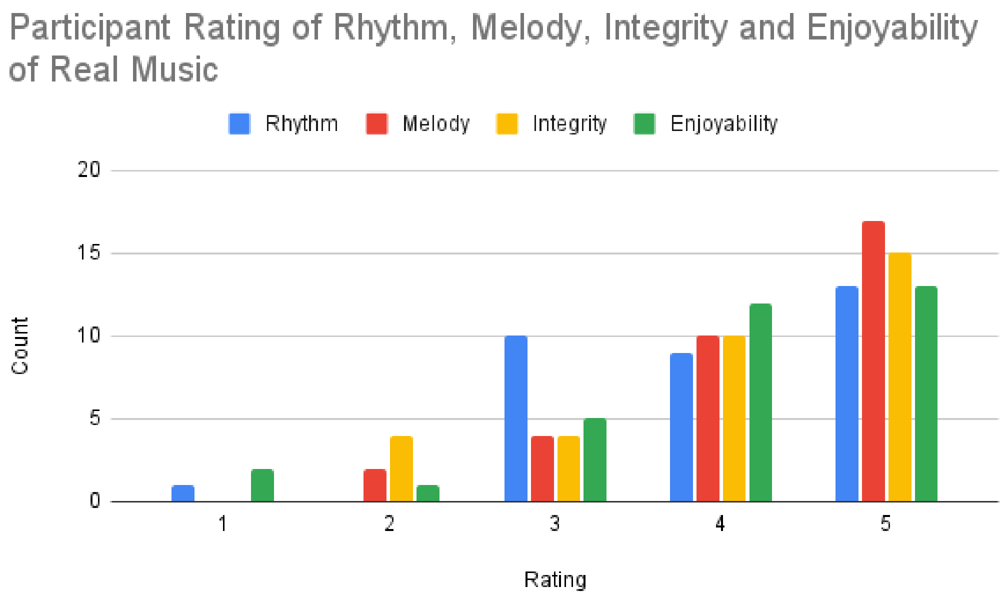
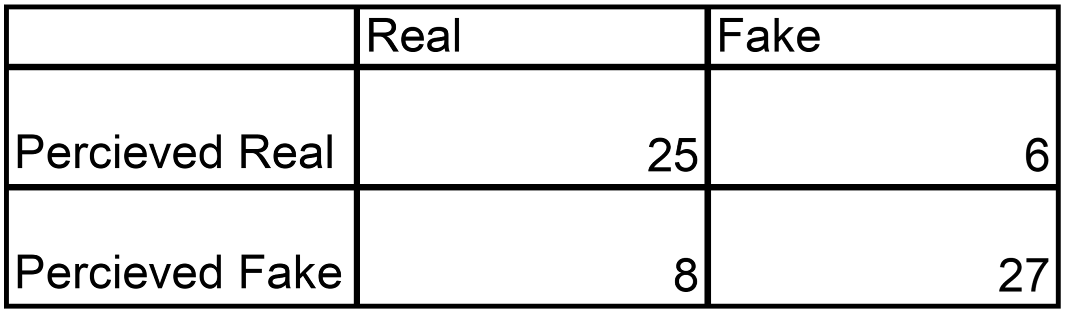


Figure 6: Counts of ratings for musical criteria of real music



	Real	Fake
Percieved Real	25	6
Percieved Fake	8	27

Figure 7: Confusion matrix of perceived realism of generated music

was not able to train well to these elements. This is likely the reason why the model's generated music failed to obtain good ratings in the experiment. Future work in this project could include building more structure into the LSTM in terms of separating melody and harmony generation into separate levels and having them connect through a similar architecture as described by (Chu et al., 2016).

REFERENCES

Anatoly-os. Musescore 3 released, Dec 2018. URL https://musescore.org/en/3.0.

Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from pi: A musically plausible network for pop music generation, 2016. URL https://arxiv.org/abs/1611.03477.

D8A5NR. The 5 voice nintendo nes and apple ii oscillators shown below had non-variable waveforms per voice. in contrast, commodore64 had only 4 voices (no pcm) but all four voices could playback any waveform available. retrogaming sounddesign 8bit d8a5nr pic.twitter.com/llavfvjetg, Jan 2018. URL https://twitter.com/datasinner/status/955152846931992576.

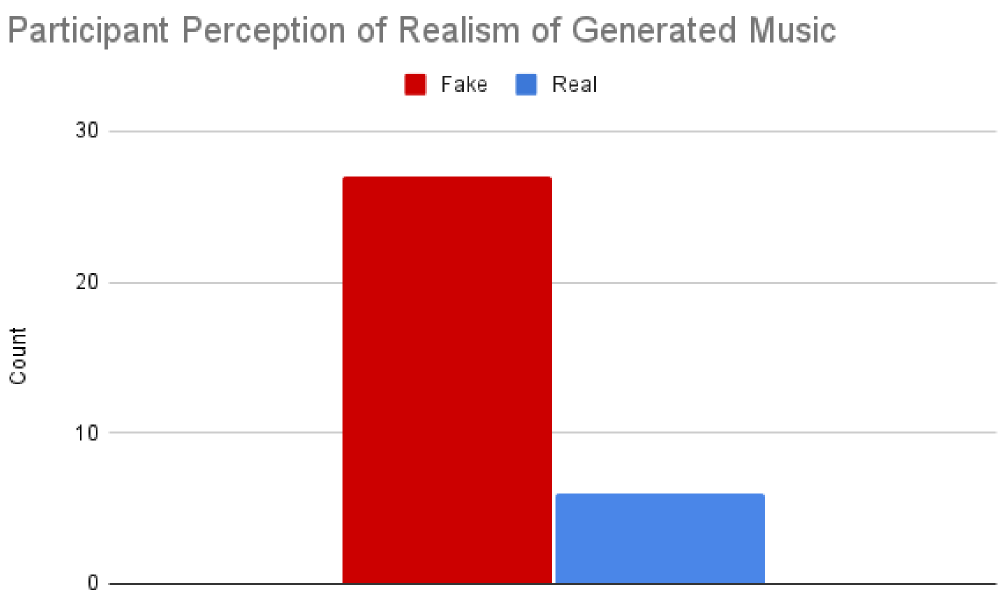


Figure 8: Counts of perceived realism of generated music

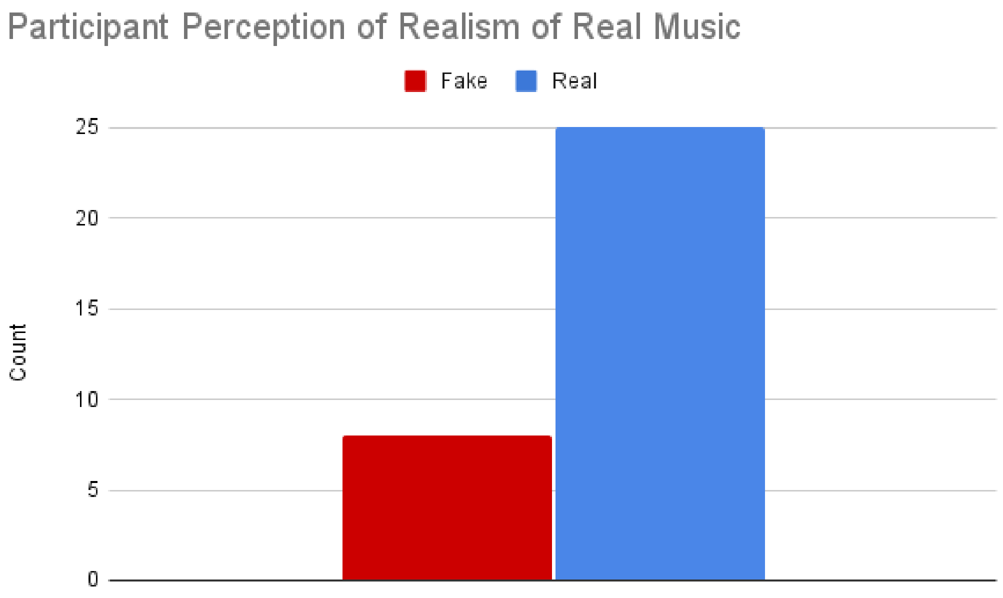


Figure 9: Counts of perceived realism of real music

Chris Donahue, Huanru Henry Mao, and Julian McAuley. The nes music database: A multi-instrumental dataset with expressive performance attributes. In *ISMIR*, 2018.

Maddy Shaw Roberts. How did music notation actually begin?, Mar 2018. URL `https://www.classicfm.com/discover-music/how-music-notation-began/`.

Walter Schulze and Brink van der Merwe. Music generation with markov models. *IEEE multimedia*, 18(3):78–85, 2011. ISSN 1070-986X.

Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation, 2017. URL `https://arxiv.org/abs/1703.10847`.

Hongyuan Zhu, Qi Liu, Nicholas Yuan, Kun Zhang, Guang Zhou, and Enhong Chen. Pop music generation: From melody to multi-style arrangement. *ACM transactions on knowledge discovery from data*, 14(5):1–31, 2020. ISSN 1556-4681.