Bachelor's Thesis Computing Science

Deep Learning for Video Game Music Generation

Falko Jansen
s1044704

June 4, 2023

First supervisor/assessor:
Professor David van Leeuwen

Second assessor:
Professor Djoerd Hiemstra

Radboud University

Abstract

Advances in the computer-composition of music have enabled algorithms to compose music similar to that composed by humans. However, these algorithms are generally tested only on the more mainstream genres like jazz, pop, and classical, while other, more niche genres are neglected. This leaves some doubt as to how generally these algorithms can perform. In this research paper, we gathered three pre-existing computer-composition algorithms and tested them on the task of generating Video Game Music (VGM). Their generated samples, as well as human-made samples, were presented in an online survey which performed both a Turing test and a Mean Opinion Score test. While one model failed to train, results for the remaining two showed that human-made music performed by far the best, followed by the LakhNES (Transformer) model, and lastly the DeepJ (LSTM) model. For the most part, computer composed music did not compare to human-made. However, results did show the LakhNES model capable of generating music that humans could not accurately identify as computer-generated.

Contents

Chapter 1

Introduction

Music is a form of entertainment that is enjoyed by almost everyone, and it comes in such a variety of styles that there is always something to suit just anyone's tastes. Its use is not limited to only listening to it on its own—it can accompany a (video) game or movie to set a theme and mood—and is also often used as background noise in waiting rooms and shops.

Enjoyable as it is, a well-known problem of music is that it takes a great deal of time, effort, and skill to compose a good piece. Not everyone has those, and although there are plenty of musical pieces in the world already, it is at times difficult to find a piece that matches exactly what one wants. For a designer using it in, say, a game or movie, requiring the music to be royalty-free is another hurdle. An increasingly popular solution to this is computer composed music.

Computer composition is the process of using a computer to algorithmically compose music, either by having it aid a human in composing or by doing it entirely automatically. Approaches to this have varied between, among others, randomization, formal grammar rules, Markov chains, and more recently deep learning Transformers, but in the modern day, the most popular approaches include neural network architectures [15]. These neural network models are capable of being trained on pre-existing music, which adds a great deal of versatility, and means researchers do not have to discover and hardcode a complicated ruleset. Additionally, because music can be approached much like a language, the models can be built similarly to those for Natural Language Processing [11].

Much research has already been done on computer composition, but many music generation algorithms are specialized in a specific style or genre, like jazz, pop, or classical [3, 8–10], leaving other styles with no or unoptimized ways of being computer generated. One genre among many that has lacked attention is Video Game Music (VGM). VGM is, of course, music accompanying video games, but more specifically it is also a retro style of synthesized electronic music made using sound chips. It is something of a

subgenre of Chiptune or 8-bit music, with VGM being 8-bit music originated from or in the style of retro games and consoles, and despite the consoles of old now being long outdated, both VGM and Chiptune still have thriving fanbases even today [7].

Chiptune, and by extension VGM, is a comparatively simple style of music. This is due to the limited number of instruments and possible tones in the music—in present day merely copied over from hardware limitations of retro consoles—and by a score's individual sequences being less vertically dense as a result, this *should* make it easier to train for by a neural network. Despite this—its reasonably large fanbase and comparative simplicity—very little research has been done on the computer composition of Chiptune music. Algorithms have been introduced that are shown to demonstrate good performance on a multitude of styles, but these are only tested on more mainstream genres, usually classical, meaning there is not much evidence suggesting they will perform well on Chiptune music as well. For those wishing to make their own Chiptune music, including some video game designers, the ability to generate new music of this genre using algorithms would be a great help.

In this research paper, we seek to find a pre-existing computer composition model capable of generating quality VGM music. We do this by running three published computer composition algorithms on the task of generating Video Game Music and then performing a Turing test on them to see which algorithm's generated music is most indistinguishable from human-made music. The aim is to get an idea as to which model is best suited for generating this genre of music. It is our expectation that, VGM being comparatively simple compared to other genres, more complex models will not get significantly better results than simpler ones.

In the next section, we give some basic explanation on various model architectures used for computer composition. Section 3 explains the research done for this paper. Previous research into the subject of computer composition is delved into in section 4, and section 5 closes off with the conclusions drawn from our research.

Chapter 2

Preliminaries

Methods of computer composition have varied over the years, but current research focuses on neural networks—specifically, neural networks capable of keeping track of time and structure. This is because music, much like a language, is sequential in nature, meaning that each element, in this case music notes, influences those that come after it. Among the various sequential model architectures used in algorithmic composition, the three that are of importance in this research paper are the following: Recurrent Neural Networks (RNNs); its successor, the Long Short-Term Memory networks (LSTMs) [12]; and Transformers [18]. Below follow explanations on these three architectures.

For starters, neural networks are a type of machine learning that mimics the way neurons in (human) brains operate and interconnect. They are made up of layers of nodes—an input layer, one or more hidden layers, and an output layer—that are each connected to the layer above it and below it. Each node has a weight and bias, and the sum of all the weighted outputs of the previous layer (which all go into this node), plus the bias, is entered into a nonlinear function. This function's outcome is then weighted with the node's own weight and sent to all nodes of the next layer. This mimics the way human brains function, and by altering each node's weight and bias through a process called training, the neural network can be tuned to recognize certain patterns and give the right output when it spots data matching that pattern. For its pattern recognition, this architecture finds widespread use in a variety of computing tasks.

In *Recurrent* Neural Networks (RNNs), data is processed sequentially. For example, when translating a sentence, the translation is done left to right, and the words that came before are important as context for processing the next word. RNNs achieve sequential processing by having the output of the previous step be part of the input for the current state, thus "remembering" the previous data input while working on the current one, and this function is repeated for each data input (recurrence). For short-term de-

pendencies, this works, but not long ones. This is because as "memories" become stored into memories themselves, older data inputs become increasingly less well-represented in the feedback loop. RNNs suffer from a kind of memory loss where long-term dependencies cannot be captured well.

LSTMs are a way of addressing the long-term dependency issue RNNs have. They do this by using a collection of "gates" to better control what information is remembered and what is not. Forget gates "forget" long-term memory, input gates retain memory, and output gates make it so only relevant information is used for processing the current input. All of this together provides LSTMs the capability of retaining long-term dependencies, and thus capable of producing more globally structured outputs. Even so, the same problems that plague RNNs can still occur with LSTMs if the input is too long. With too much input, the model will not know which previous data inputs are relevant to transforming the current input.

Because they process data sequentially, RNNs are slow, and LSTMs even slower. Additionally, they can forget long-term dependencies if the input sequence is lengthy enough. These are two of the problems faced in neural machine translation (tasks transforming an input sequence to an output sequence) and Transformers are a type of neural network architecture designed to try and solve the latter of these problems. This is achieved through a collection of components in the model, but the most important technique is attention. Attention allows the model to know to "pay attention" to specific dependencies when transforming input to output. In language translation, for example, it will know some words in the sentence are more relevant to a given word than others. It does this by giving each data input values representing its connection to other data inputs. This makes it easier for the model to use the relevant dependencies while transforming and thus improves long-term dependencies.

Chapter 3

Research

Algorithms and Dataset

We train three computer composition algorithms on the training set of the Nintendo Entertainment System Music Database (NES-MDB) as provided by Kaggle [6]. The dataset consists of 5278 songs from the soundtracks of 397 NES games, all in MIDI format, and split into three sets—training, testing, and validation—with 4502, 373, and 403 songs, respectively. Alterations made to the algorithms and their pre-processing methods for the training data are limited to what is absolutely necessary to make the algorithms function with this dataset. However, we do perform fine-tuning on parameters to get the best possible outputs.

The algorithms explored in this research are the following: the VGM-RNN by Nicolas Mauthes [17]; DeepJ by Huanru Henry Mao, Taylor Shin, and Garrison W. Cottrell [16]; and LakhNES by Chris Donahue, Garrison W. Cottrell, Huanru Henry Mao, Julian McAuley, and Yiting Ethan Li [5]. The reason these particular models were chosen and not others is in large part due to availability. We required these models to be accompanied by a scientific paper, be trainable with a MIDI format dataset, and for the official code to be available. These requirements combined resulted in a somewhat sparse collection of options for us. Out of this limited set, the VGM-RNN was chosen because it is a comparatively simple model: a bare-bones LSTM. This made it ideal as a benchmark to compare the more complex models to. DeepJ and LakhNES were selected because they are well-performing and non-genre specific, meaning they should be able to be trained on VGM music.

VGM-RNN

The VGM-RNN is an RNN model consisting of a single LSTM layer with 256 hidden units, and the output layer uses a softmax activation function to build the distribution of note probabilities. For the input, the training

data is first run through several filters to select songs fitting certain criteria. This assures consistency of the dataset. Then, the files are transformed into piano-rolls, which is a 2D-matrix with time in one axis and notes in another, denoting when exactly which notes are playing. These piano-rolls are turned into suitable training sequences by having each sequence in the piano-roll get its subsequent sequence as its label. Fed into the model, this builds a set of weights representing the probability of any sequence given the sequence prior to it.

A starting sequence is randomly selected from amongst all sequences in the training data. Given as input to the model, this outputs a prediction for the next sequence in the form of a distribution of note probabilities. A sampling threshold determines how likely a note must be to make it into the subsequent sequence, where a low threshold allows more (unlikely) notes, creating more chaotic music, while a higher threshold makes more conservative music and a sparser piano-roll.

Once the second sequence of the song is generated, it too is fed into the model, creating a recurrent loop of self-influenced generation. Once a maximum number of sequences has been reached, the song is finished, and the piano-roll is transformed into a MIDI file. Being an RNN model, this model does struggle with long term dependencies, and this model specifically is not designed to handle multiple instruments.

DeepJ

DeepJ is an improvement on the design of the Bi-axial LSTM [14], which suffered from the inability to compose structured music. DeepJ introduces the option to enforce styles by having a special weight for styles added to the input of each of the four LSTM layers. Some styles have similarities in note sequences, so a combination of those styles should give higher probabilities to such similar sequences.

Data is represented as piano-rolls, which is the common representation, but instead of using a binary system, where a 1 means a note is playing and 0 means it isn't, the Bi-axial LSTM uses values *between* 1 and 0 to represent volume. When generating, the model produces three outputs for every note at every time step: play probability, replay probability, and dynamics. Because there is a difference between *holding* a note and *replaying* a note, replay probability is separate from play probability and shows the likelihood of re-attacking a note immediately after it ends, with no time steps between the two note events. Dynamics was introduced by DeepJ and represents the relative volume of a note.

Notes within each time step are modelled as probabilities conditioned on *all* previous time steps, not only the most recent one, as well as all notes in the current time step that have already been generated. In order to generate sequences, the model samples from its probability distribution, using a coin

flip to determine whether or not to play a note, and replay probability is sampled to determine if a note should be re-attacked. No special method was used to generate the first sequence, and so an adaptive, linear temperature adjustment was applied that increased temperature for each time step of silence produced and reset it upon a non-silent output.

The additions introduced by the researchers allow the DeepJ model to solve the style consistency problems in the Bi-axial LSTM. However, it lacks long term structure—a remaining problem for LSTMs—and does not work for songs with more than one instrument.

LakhNES

The LakhNES model uses an extension of the Transformer model called Transformer-XL [4], which is designed to handle longer sequences. Not much more explanation on the model's architecture is given in the paper, so we surmise it works no differently than a Transformer-XL, which itself does not work much differently than a Transformer. The model bases its predictions of the next musical sequence based on a subset of musical events from the past.

To improve the model's performance beyond what the Transformer-XL itself offers, the LakhNES model applies "standard music data augmentation methods" on the training data to reduce overfitting, as well as some extra methods the researchers developed themselves for the multi-instrumental setting. The extra methods include adjusting the speed of a piece by a random percentage between $\pm 5\%$ and occasionally removing random instruments from the ensemble, but it is not made clear what *standard* methods are meant or how they are applied. Besides the data augmentation, the researchers pre-trained the model on the Lakh MIDI dataset.

When generating music, two hyperparameters influence the model's generation process: *temperature* and *topk*. Temperature is a parameter that affects the "confidence" a model has in a most likely response. At higher temperatures, the model becomes less confident in any one option and more creative, while a lower temperature makes it more confident but also more conservative. Effectively, a temperature of 1 means the outputted probability distribution is not altered, a temperature infinitely high would make the distributions even across the board, and a very tiny temperature means the most likely probability gets close to 100%. Topk, meanwhile, makes it so that only the top so-many probabilities are considered. A topk of 16 would mean that only the 16 most likely outputs could ever be generated while the rest are discarded.

Generation Results

VGM-RNN

Unfortunately, we were unable to generate results that compared to the quality of what the original paper presented. Loss only increased during training, and the unaltered model stopped the training early at only 11 epochs because the difference in loss between subsequent epochs had fallen below a certain threshold. Trained on the original dataset used by the research paper, loss started at 4.7 and only rose from there, and the training stopped before the 20th epoch. Music generated from these trained weights were without structure and completely unusable, and because it was our intention not to change the models we tested, this meant the VGM-RNN was not further used in our research. We tried but could not locate the source of the problem.

DeepJ

Due to the model having been designed to only work with one instrument, the training data had to be altered. Only one instrument's notes could be used. After some review, we discovered that out of the four instruments the dataset uses—two pulse-wave generators (P1 and P2), a triangle-wave generator (TR), and a percussive noise generator (NO)—it was the first pulse-wave generator, P1, that contained the central melody. With this decided, we separated this instrument's notes from the others and saved them as the new training set. Afterwards, a generator was coded to feed the training data into the model, as it could not handle all of it at once. Since all the data belonged to the same style, the DeepJ model's style consistency functions did not play a role. However, the dynamics output for notes was still an improvement over the older Bi-axial LSTM model.

The model successfully trained, taking 57 epochs and 10 hours to go from a loss of 0.0487 to 0.0246, and after having tested that the outputs were coherent, 35 music samples were generated with the default generation parameters. The DeepJ model does not have any generation parameters other than style and length, and since there is only one style and the standard length of 32 bars is more than enough for 15-second samples, these did not require fine-tuning. The length of generated samples does not affect how individual bars are generated: it generates as normal and simply stops when it reaches the last bar. A third-party website was used to synthesize the generated MIDI files into .wav files.

LakhNES

The LakhNES model comes with pre-trained models at various checkpoints, which can save a great deal of time normally spent training, and since our

research uses the same dataset as the LakhNES paper did, we were able to make use of these checkpoints. For our research, we used the NESAug checkpoint, which did not apply pre-training with the Lakh MIDI dataset but did use the NES-MDB training set with data augmentation. We made the choice to not use a pre-trained model because we felt pre-training was not a difference in model, only in training stratagem. Pre-training one model and not the others was treating them unequally, and we wished to keep as many variables equal as possible. Data augmentation, however, *was* a difference in model, and was kept.

By using the checkpoints, we were saved the effort of having to train the model ourselves, which meant we could instead spend our time fine-tuning the generation parameters. After extensive testing it was found that the best results were generated with a temperature of 1.05 and a topk of 32. With these parameters, 35 music samples were generated, and the LakhNES paper provided code that synthesized MIDI files into .wav files.

Evaluation

Once sufficiently fine-tuned, each algorithm is made to generate 35 songs, and from the NES-MDB validation set, an equal number is randomly selected. We then remove any songs that are of poor quality or, in the case of the human-made validation set, are too recognizable in origin. From what remains, the five best samples are selected for each of the four groups and cut down to 10 to 15 seconds. This cut is made such that, where applicable, this cut removes a change in melody. This is done to keep the samples' melodies more or less consistent regardless of the source of the sample. What we are left with is a collection of human-made and computer-generated songs, and we design an online survey to perform the evaluation test.

In the survey, subjects are presented with twenty 10-to-15-second fragments of Video Game Music, distributed as five human-made and fifteen computer-generated. Subjects are not told this distribution, and are instead told not to expect it to be fifty-fifty. This is done to discourage subjects from trying to force some sort of equilibrium in their answers, as this would skew results.

At each fragment, subjects are asked to judge whether the origin is from human-made or computer-generated, and they are asked to rate how much they like the song on a scale of 1 to 10. The former question is a Turing test, judging the model's ability to generate music indistinguishable from a human's, while the second question serves to test the model's ability to generate enjoyable music. We operate by a framework that considers a model successful if the music it produces sounds like or is preferred to human-made [1].

In an attempt to partially account for subjects learning to discern human-

made and computer-generated as they go, as well as to introduce the VGM genre to people new to it, the actual test is preceded by a practice round. In this round, the subjects are presented with two human-made fragments and one computer-generated fragment, and are told which is which at the end of the round. These samples are also selected from the sets of generated data.

Evaluation Results

We shared the link for this online survey with a collection of university students and kindly asked them to fill it in. After 7 days, 102 people had answered. With these responses, we compiled the rate at which music was identified as human-made and the mean opinion scores. This was done both for individual samples and the set of each method.

It became apparent, even without performing significance tests, that in most cases people were in fact capable of distinguishing human-made and computer-generated samples. On average, the human-made samples were identified correctly 86.7% of the time, while the best-performing computer method, LakhNES, was misidentified by only 34.1%. For the ratings, as well, human-made scored 7.1 on average and LakhNES 5.5. Noteworthy, however, is that performance between different samples of the same generation method could vary greatly, and the most "misleading" sample was misidentified as human-made by 55.9% of participants. Even one of the human samples was only identified as human-made 57.8% of the time. Figures displaying all the results are shown in Appendix A.

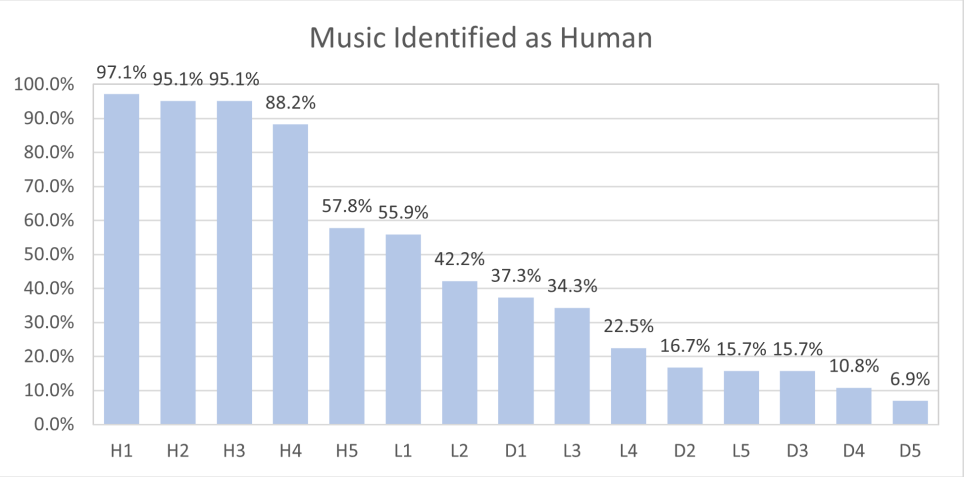

Figure 3.1: Human estimates on whether music was human-made for all samples. "H" stands for a human-made sample, "D" for DeepJ-made, and "L" for LakhNES-made.

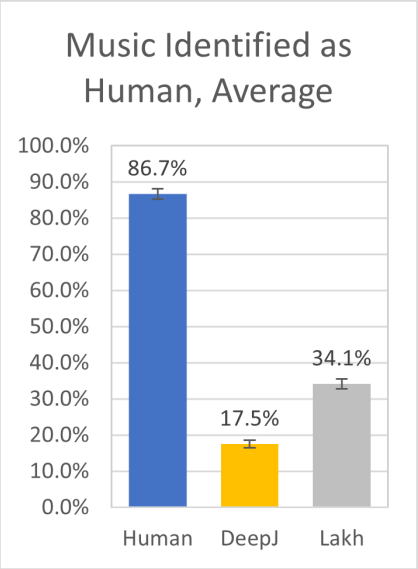

Figure 3.2: Human estimates on whether music was human-made, averaged over methods. Error bars are standard error.

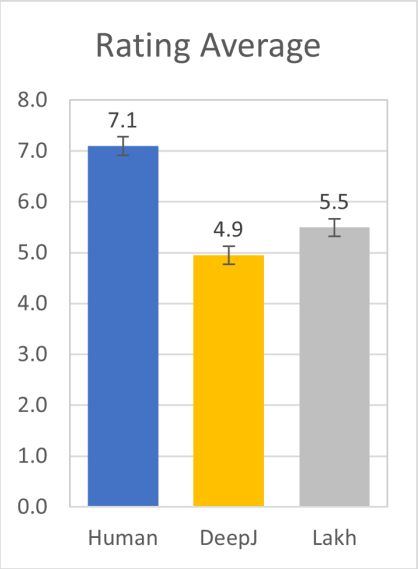

Figure 3.3: Average human ratings of music samples. Error bars are standard error.

LakhNES music "deceived" listeners almost twice as often as DeepJ music did, which shows it generates the most human-like VGM music of the two methods tested, but it was not even half as convincing as genuine human-made samples. For the sake of certainty, we performed a proportion test between the averages of DeepJ music and LakhNES music, then LakhNES and human, to test statistical significance. The results of the two-sample proportion tests indicated that there is a significant difference between DeepJ's percentage (17.5%) and LakhNES's percentage (34.1%) with $Z = 2.71$ and $p = .007$, *and* a significant difference between LakhNES's percentage and the human's percentage (86.7%) with $Z = 7.68$ and $p < .001$. Additionally, a two-tailed one sample proportion test between the best-performing computer sample (LakhNES's at 55.9%) and random guessing (50%) showed with $Z = 1.19$ and $p = .2334$ that the null-hypothesis—here the probability of the computer sample being correctly identified by subjects being equivalent to that of random guessing—could not be rejected. This means that this sample of computer-generated music was misidentified with a frequency that may not be distinguished statistically from random guessing.

For statistical significance testing of the ratings, neither proportion testing nor t-testing is appropriate. MOS ratings are not numeric, they are *ordinal*. It is not possible to show that the difference between a 6 and a 7 is the same as between 1 and 2. Also, we cannot guarantee a normal distribution needed for a t-test. Therefore, we utilized the Mann-Whitney U test

instead. We performed this between the averages of DeepJ and LakhNES and the averages of LakhNES and human. The results indicated that there is a significant difference between DeepJ's rating and LakhNES's rating with $Z = -5.04$ and $p < .001$, *and* a significant difference between LakhNES's rating and the human's rating with $Z = -13.37$ and $p < .001$. From this, as well as the earlier proportion tests, it is clear that in general, human-made music performs the best, followed by the LakhNES model, and lastly the DeepJ model.

Chapter 4

Related Work

As mentioned in the introduction, much research has been done on computer composition, but the models presented are rarely tested on anything other than mainstream genres. That said, research on generating Chiptune or VGM is not entirely unheard of. The VGM-RNN introduced in [17] applied an LSTM network to probabilistically generating VGM music. The authors succeeded in the task of training the model, and then ran a Turing test on the resulting songs, as well as asking subjects to rate the songs' quality. The result was that subjects could correctly discriminate little over half the time, not much better than chance, and the computer-generated song got the highest rating. However, only six people answered the survey, and the model was trained on a small dataset of 181 MIDI files. With such a small sampling size, the results are inconclusive.

Another attempt was made by Chris Donahue et al in the form of the LakhNES model, which utilized an improved Transformer architecture to generate multi-instrumental music based on the NES-MDB dataset [5]. They proposed several methods to adapt the Transformer architecture to multi-instrumental music and also a pre-training technique, then compared their generated chiptune clips to human-made music and several baselines from other generation methods through use of Turing tests and preference tests. The results were compelling, and they found that their methods improved performance over older approaches like the Transformer-XL. However, in the Turing tests, clips of only 5 seconds long were used, meaning long-term dependencies played less of a role in the generated pieces. This may have been done to give the other methods a fighting chance, as most of those were not suited for such dependencies, but it does fail to present the Transformer architecture's strengths. Additionally, for the Turing tests, computer-generated clips were always compared to human-made in pairs, which should make it easier to identify which is human-made and which is not. It does not represent how the algorithms perform "in a vacuum".

For non-Chiptune music, a much earlier attempt at making *monophonic*

music—music with a maximum of one note at each time step—utilized an LSTM to overcome some of the limitations RNNs faced [9]. The results were preliminary, but they paved the way to further study. Early work in *polyphonic* composition—multiple notes possible in one time step—attempted to model musical sequences using a combination of RNNs and Restricted Boltzman Machines (RBM) [2]. One key aspect of their design was that the model not only looked at the *previous* notes when attempting to model the music's sequences, but also at current ones, working under the understanding that while technically any combination of notes is *possible*, in reality only a few are ever used. This work demonstrated excellent results.

Taking notes from this architecture, [14] created the Bi-axial LSTM, which used probability distributions similar to those of the RNN-RBM. The probabilities of notes in the current time step were conditioned on all previous time steps and the notes already generated for the current time step. It was also translation invariant, meaning that, in the context of music, a structure of notes could have its pitch be "translated" up or down and still be considered the same. The Bi-axial LSTM was successful on prediction tasks but lacked the ability to generate consistent music itself.

Improving upon the design of the Bi-axial LSTM, DeepJ was introduced by the authors of [16]. The main addition here was the ability to enforce a music style.

More recently, a multi-modal AI system called AudioGPT was introduced for the generation, processing, and understanding of sound, including speech and music [13]. For this purpose, it complements ChatGPT, a Large Language Model (LLM), with foundation models and input/output interfaces, allowing the chat system to handle complex audio tasks. This ties developments in chat systems with Generative Pre-trained Transformers (GPTs) into those of audio systems by having a collection of audio processing models be accessible through ChatGPT, and experimental results of AudioGPT have shown potential.

Chapter 5

Conclusions

In this paper we tasked three computer composition algorithms on the task of generating Video Game Music (VGM) to see how they compare. We did so by training them on the Nintendo Entertainment System Music Database (NES-MDB), fine-tuning their generation parameters, and offering up the best generated samples in an online survey along with human-made samples. Though one model failed to train, results on the remaining two show that human-made music performed by far the best, followed by the LakhNES (Transformer) model, and lastly the DeepJ (LSTM) model. Additionally, though the survey subjects were not often fooled by the computer-composed music, we did encounter an instance where subjects estimated a LakhNES sample to be human-made with a frequency not statistically distinguishable from random guessing. This shows it is possible for a computer to compose VGM music that humans cannot accurately identify as human or not human, and that the LakhNES model is better suited for this than the DeepJ model.

Bibliography

[1] Christopher Ariza. The interrogator as critic: The turing test and the evaluation of generative music systems. *Computer Music Journal*, 33:48–70, 06 2009.

[2] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. ICML'12, page 1881–1888, Madison, WI, USA, 2012. Omnipress.

[3] Hang Chu, Raquel Urtasun, and Sanja Fidler. Song from PI: A musically plausible network for pop music generation, 2016.

[4] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context, 2019.

[5] Chris Donahue, Huanru Henry Mao, Yiting Ethan Li, Garrison W. Cottrell, and Julian McAuley. LakhNES: Improving multi-instrumental music generation with cross-domain pre-training, 2019.

[6] Chris Donahue, Huanru Henry Mao, and Julian McAuley. The NES Music Database: A multi-instrumental dataset with expressive performance attributes. In *ISMIR*, 2018.

[7] Kevin Driscoll and Joshua Diaz. Endless loop: A brief history of chiptunes. *Transformative Works and Cultures*, 2, 02 2009.

[8] D. Eck and J. Schmidhuber. Finding temporal structure in music: blues improvisation with LSTM recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pages 747–756, 2002.

[9] Douglas Eck and Juergen Schmidhuber. A first look at music composition using LSTM recurrent neural networks. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 2002.

[10] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. DeepBach: a steerable model for bach chorales generation. 2016.

[11] Carlos Hernandez-Olivan and Jose R. Beltran. Music composition with deep learning: A review, 2021.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

[13] Rongjie Huang, Mingze Li, Dongchao Yang, Jiatong Shi, Xuankai Chang, Zhenhui Ye, Yuning Wu, Zhiqing Hong, Jiawei Huang, Jinglin Liu, Yi Ren, Zhou Zhao, and Shinji Watanabe. AudioGPT: Understanding and generating speech, music, sound, and talking head, 2023.

[14] Daniel Johnson. Generating polyphonic music using tied parallel networks. pages 128–143, 03 2017.

[15] Madiha Mansoori and Rajiv Murali. A systematic survey on music composition using artificial intelligence. In *2022 International Conference for Advancement in Technology (ICONAT)*, pages 1–8, 2022.

[16] Huanru Henry Mao, Taylor Shin, and Garrison Cottrell. DeepJ: Style-specific music generation. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*. IEEE, jan 2018.

[17] Nicolas Mauthes. VGM-RNN: Recurrent neural networks for video game music generation. Master's thesis, San José State University, 2018. `https://scholarworks.sjsu.edu/etd_projects/595`.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Appendix A

Appendix

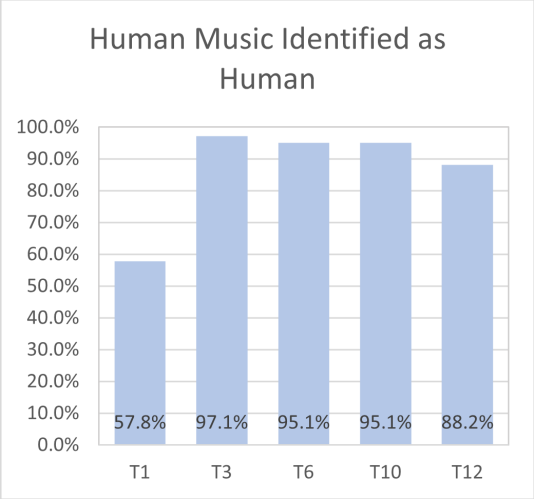

Figure A.1: Human accuracy on identifying human-made music. They are labeled in order of occurrence in the test.

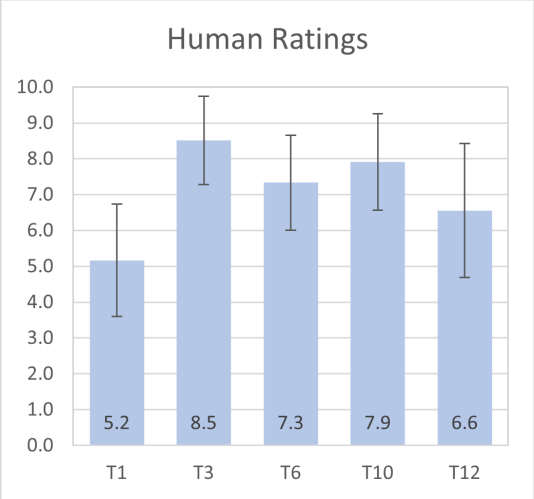

Figure A.2: Average human ratings of human-made music samples. Error bars are standard deviation. They are labeled in order of occurrence in the test.

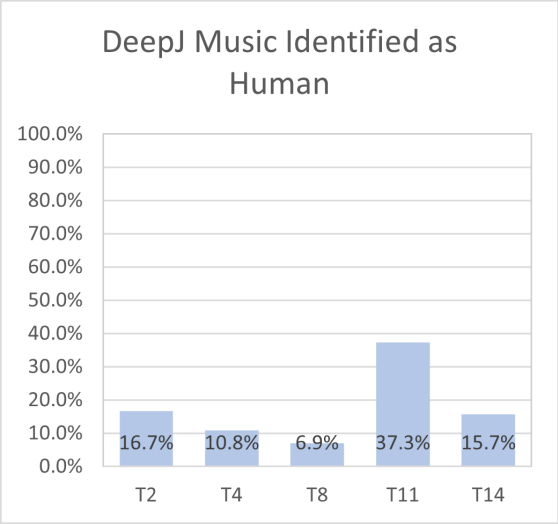

Figure A.3: Human estimates on whether DeepJ music was human-made. They are labeled in order of occurrence in the test.

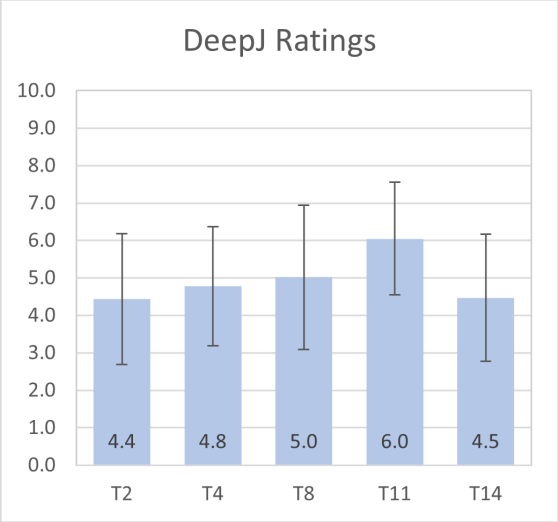

Figure A.4: Average human ratings of DeepJ music samples. Error bars are standard deviation. They are labeled in order of occurrence in the test.

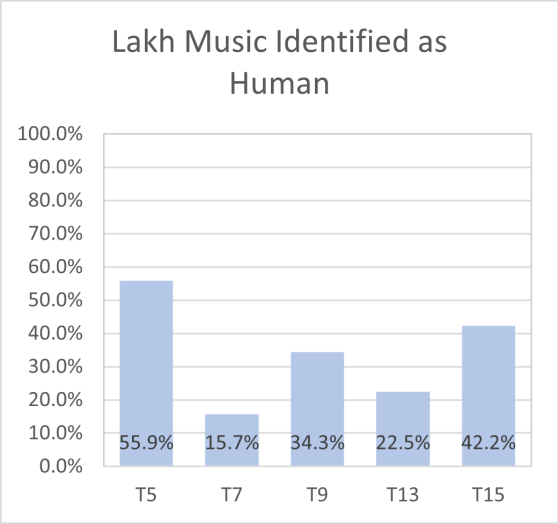

Figure A.5: Human estimates on whether LakhNES music was human-made. They are labeled in order of occurrence in the test.

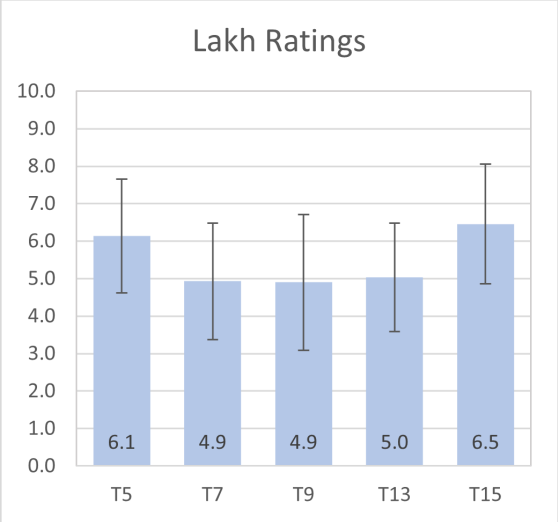

Figure A.6: Average human ratings of LakhNES music samples. Error bars are standard deviation. They are labeled in order of occurrence in the test.