Can GAN originate novel rock music?—Generating novel rhythm patterns using GAN with Genre Ambiguity Loss

*Re-implementation of the paper by Tokui N. [1]

1st Léon Dankert
University of Osnabrück
Osnabrück, Germany
ldankert@uni-osnabrueck.de

2nd Vera Proiss
University of Osnabrück
Osnabrück, Germany
vproiss@uni-osnabrueck.de

3rd Inga Wohlert
University of Osnabrück
Osnabrück, Germany
iwohlert@uni-osnabrueck.de

***Abstract*—The generation of music using deep learning is a constantly evolving field. Throughout history, many algorithm-based approaches have been tailored to create original music. In this documentation, we summarised our work on novel rhythm pattern generation within the genre of Rock using an extended GAN model. This study was motivated and re-created after the previous work of Tokui N. [1], where the author implemented a generator and two discriminator models. While the generator created new rhythm patterns from the random noise, the first discriminator decided if those pieces originated in real or fake data distribution. In the second discriminator, we changed the functionality to specialise in the genre of rock while training, so to encourage the generator to create new rhythm patterns within the specified style. Furthermore, we used grid computing at the University of Osnabrück in order to find best performing hyperparameters to improve the training process. As a result, the slightly adapted model generated short novel rhythm patterns in the rock genre.**

***Index Terms*—rock music generation, conditioned GAN, GAN with Genre Ambiguity Loss, deep learning**

I. Introduction

Deep learning is a rapidly growing field in several of application areas, such as computer vision, natural language processing or speech recognition [2], to name just a few. One of the main interests of researchers worldwide is to recreate human consciousness in a machine. That also includes creative tasks such as image or music generation. So far, deep learning algorithms have succeeded at replicating images and have seen an improvement in imitating music from a certain genre. As an example, one of the well-known projects was the *Bach Doodle*, which harmonises with a user-given melody in the style of *Bach* [3]. Further, the band *Yacht* has released an entire album based on the synthetically generated music [4]. But now, when computers have learned to generate art, the question is - what does creativity mean? The author of the paper *"Can GAN originate new electronic dance music genres? Generating novel rhythm patterns using GAN with Genre Ambiguity Loss"* [1], is asking precisely this question and argues that a novel output is a sign of creativity. In other words, can a deep learning-based system create something novel, or is "something novel" always bound to be an attempt to imitate an existing pattern already created by humans? In his approach, the author of the before-mentioned paper studied the emergence of novel rhythm patterns previously not found in the training dataset. As the Generative Adversarial Network (GAN) architecture creates new samples from an underlying data distribution, it is quite debatable whether the model can indeed produce a novel piece. Therefore, to address this problem of originality, the author implemented an extended version of GAN architecture to generate new rhythm patterns. The extended model consisted of a second discriminator D and a **Genre Ambiguity Loss** (section IV-D2). The Genre Ambiguity Loss was used to penalise whenever a classifier was too confident in classifying into one genre. This way, the output would be ambiguous to genres and could be considered as something novel. In our implementation, we used the same design and structure. However, instead of minimising the function in order to penalise clear classification into one genre, we maximised it. This way our new samples were specialised to rock music. Additionally, we used grid computing hyperparameter search in our implementation. This way we could test which hyperparameters were the most effective. We worked with the three optimisers: Adam, RMSProp and SGD (section V-A).

II. Theoretical Background

A. Related Works & Methodology

Throughout history, various approaches were applied to the music generation. One of the earliest computer-based approaches was the use of Markov Models [5]. In this early project (1959), Markov Models had additional rules applied to filter the generated material according to desired properties. The Markov Models were complicated, error-prone and often limited to a specified music genre.

With the advent of deep learning, came a new era for the music generation. The previous deep learning methods used for music generation were RNN sequential models [6]. The RNNs lacked a global structure and hence were unsuitable for learning a music genre. Another candidate, the LSTM, often succeeded in tasks in which RNNs struggled. Eck et al. [7] suggested to apply an LSTM architecture to music generation task. In their work, they successfully generated somewhat novel jazz music. In 2017, Yang et al. [8] created MidiNet, a convolutional GAN model to generate melody either from scratch or conditioned by previous melody bars. Additionally, the Variational Autoencoder (VAE) architecture was successfully applied to music generation task as well. VAE is a refinement of a feed-forward network [6].

GAN [9], an architecture we worked with in our project, takes its origin from the image generation tasks. It (Fig. 1) consists of two model parts:

- A discriminator D which estimates the probability of a sample originating from a real dataset. In other words, it discriminates between synthetic and real data.
- A generator G which produces fake, or synthetic, samples from a noise input. The purpose of the generator G is to learn the real data distribution so well it can duplicate the synthetic samples from that distribution and make it hardly distinguishable from the real ones.

Over the training process, the generator G is trying to deceive the discriminator into believing the fake samples are real, while the discriminator D is trying hard to tell those apart and not be fooled. With training, both model parts get better at their functionalities [10].

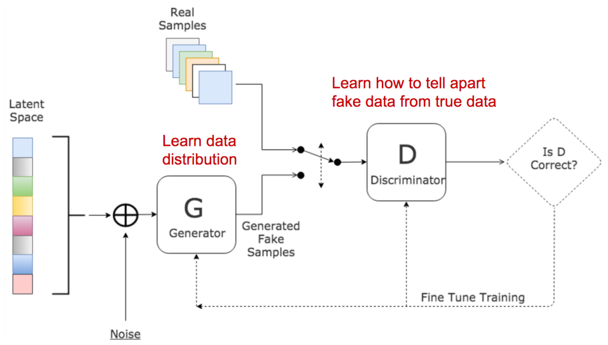


Fig. 1: Overview of GAN Architecture

The following loss function is used to optimise the performance of a classic GAN model:

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

To better understand the loss function, we broke it into the following parts. The $\mathbb{E}_{x \sim p_{data}}(x)[log D(x)]$ has to be maximised by the discriminator, meaning that the closer the output value gets to 1, the better the discriminator D is performing, i.e. accurately telling the real samples apart from the fake ones. The $D(G(z))$ part has to be minimised by the discriminator, i.e. the closer the output value gets to 0, the better again is our discriminator. Next, the smaller the $D(G(z))$ value gets, the bigger the output deducted from 1 becomes $\mathbb{E}_{z \sim p_z}(x)[log(1 - D(G(z)))$. In other words, the latter part of the equation also has to be maximised in order to have a good discriminator. On the other hand, if the output value of $G(z)$ gets bigger, then the generator is performing well, i.e. it's succeeding at deceiving the discriminator into believing the fake samples are real. For the generator to be good, the output value for $G(z)$ as well as the value deducted from 1 has to be big enough in order to prevent maximisation of the second part of the equation.

1) Rhythm Generation: Various music features have been researched in the music generation. Often the focus of deep learning studies was the generation of harmony or melody. For example, the earlier introduced MidiNet model was concerned with melody, while the infamous *Bach Doodle* [3] was concerned with creating counterpoint melodies to a user-given input, all while recreating *Bach's* music style. Rhythm is a fundamental part of music, as it creates structure, pulsation, and specific beats. The stress or subdivisions of beats can form the style of a rhythm. Furthermore, frequent meters can also be signatures of music genres, such as waltz's 3/4 beat [11]. Rhythm plays a significant role in differences between genres and is, therefore, interesting when dealing with tasks that use genre classification. Some studies focus on deep learning in rhythm generation such as symbolic music generation system proposed by Markis et al. [12]. Here, the rhythms are represented as binary words, carrying different information about the beat. In his work, Markis used an LSTM and a feed-forward network. Another example is the research made by Gillick et al. [13], in which a Seq2Seq and a recurrent Variational Information Bottleneck (VIB) models were used to translate rather abstract music ideas, such as scores and rhythm, into expressive performances.

2) Data Type: Music can be represented in different data formats. As listeners, we often perceive music in audio signals, which can be in the raw waveform or transformed. When music is interpreted by a computer, it uses different formats such as piano rolls or MIDI files [14]. In our project, we worked with MIDI files. MIDI (Musical Instrument Digital Interface) files contain event messages which function as protocol. These event messages include real-time note performance data as well as control data. The most important event messages are "Note on" and "Note off". The "Note on" message includes the *channel number*, indicating the instrument or track, the *note number*, indication the pitch and the *velocity*, indicating how loud a note is played. In the "Note off" message, the numbers mean the same, except, the *velocity* indicates how fast the note is released [11]. That means "Note on, 2, 60, 40" translates to "On channel

3, start playing a middle C with velocity 40". Each note is embedded into a data structure containing a delta-time value, representing the time position of the event. This could be both a relative or an absolute time [14]. The excerpt seen in Table I [14] shows an example of a MIDI file. Here the division has been set to 384, i.e. 384 ticks per quarter note.

TABLE I: Excerpt from a MIDI File

2	96	Note-on	0	60	90
2	192	Note-off	0	60	0
2	192	Note-on	0	62	90
2	288	Note-off	0	62	0
2	288	Note-on	0	64	90
2	384	Note-off	0	64	0

Fig. 2 visualises a matrix calculated from one of the MIDI files in our dataset. It shows the rhythm patterns as velocities. The velocity in the context of a MIDI file compares to the notion of intensity, i.e. how hard or fast a button is pushed. For example, a hard hit on a drum would be of higher velocity, while a lighter hit would be of lower velocity. In many interpreters, a higher velocity results in a louder signal. However, in some cases, it can slightly change the pitch too. You can find the beat used in this matrix in our GitHub repository as "data/beats/example_drum_matrix.wav". It has been argued, that one of the drawbacks of the MIDI files is that they do not fully convey the idea of playing multiple notes at the same time [14].

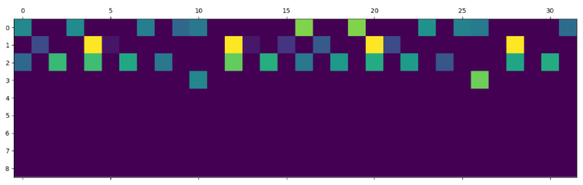
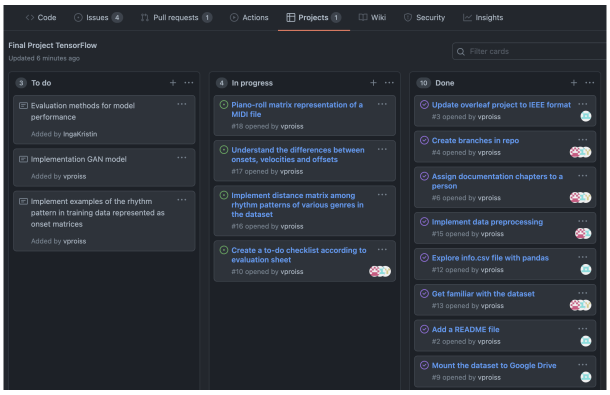


Fig. 2: Examples of the Rhythm Pattern in Training Data Represented as Velocities Matrices

III. Project Organisation

A. Project Organisation and Tools

To organise and implement our project, we used a few technical and project management tools. In terms of the management tool, we used *GitHub Projects* to create an agile scrum-like board to organise our tasks and assignments (see Figure 3). We worked with *Visual Studio* and *PyCharm* IDEs and used *Google Drive* to store the data. Regular meetings were held weekly to share the updates and exchange feedback on completed work. As the technical environment, we chose *Git* for file versioning in combination with *GitHub* for repository hosting. In a pair programming manner, we used the *VScode Live Share* tool, to program together.



Final Project TensorFlow
Updated 6 minutes ago

3 To do

- Evaluation methods for model performance
  Added by IngaKristin
- Implementation GAN model
  Added by vproiss
- Implement examples of the rhythm pattern in training data represented as onset matrices
  Added by vproiss

4 In progress

- Piano-roll matrix representation of a MIDI file
  #18 opened by vproiss
- Understand the differences between onsets, velocities and offsets
  #17 opened by vproiss
- Implement distance matrix among rhythm patterns of various genres in the dataset
  #16 opened by vproiss
- Create a to-do checklist according to evaluation sheet
  #10 opened by vproiss

10 Done

- Update overleaf project to IEEE format
  #3 opened by vproiss
- Create branches in repo
  #4 opened by vproiss
- Assign documentation chapters to a person
  #6 opened by vproiss
- Implement data preprocessing
  #15 opened by vproiss
- Explore info.csv file with pandas
  #12 opened by vproiss
- Get familiar with the dataset
  #13 opened by vproiss
- Add a README file
  #2 opened by vproiss
- Mount the dataset to Google Drive
  #9 opened by vproiss

Fig. 3: SCRUM Board in GitHub Projects

B. Project Progression

The project started with a discussion of the theoretical background behind possible projects. After several brainstorming meetings, we narrowed down our project ideas to GANs and music generation. Following the agreement on the main topics, we conducted literature research on electronic music generation using GANs architecture. Over the following weeks, we have confirmed a paper to re-implement and worked through the defined tasks and challenges.

IV. Project Implementation

A. Dataset

For the project dataset, we decided to give up the paper-introduced dataset, *Groove Monkee Mega Pack GM*, which would only be accessible with a payment. Instead, we chose a freely available *Groove MIDI Dataset*. As the title suggests, the dataset consists of drum performances recorded in a MIDI format with a *Roland TD-11* electronic drum kit. The dataset contains 1.150 MIDI files performed by 10 professional drummers, making approximately 13.6 hours of recorded drum performances with 22.000 measures of drumming. The improvised performances of different styles resulted in a more diverse dataset. Moreover, the drummers were asked to play long and short sequences, i.e. a few minutes of uninterrupted performance or beats and fills. Each recording is annotated with a genre, tempo, and drummer ID (see Table II). Generally, the *Roland TD-11* splits the recording into separate tracks. One has the meta information such as tempo, time, and key signatures, the second track contains information for control changes (hi-hat pedal position), and the third one keeps the notes. Specifically for this dataset, the following simplifications were made:

- all messages were merged into a single track;
- all messages were set to channel 9.

Additionally, we had to consider the discrepancies between the *Roland TD-11*, *General MIDI* and *Paper Mapping* specifications (see Table III) - they use differing pitch values.

TABLE II: Dataset Metadata

Field	Description
drummer	A string ID for the drummer of the performance.
session	A string ID for the recording session (per drummer).
id	A unique string ID for the performance.
style	A string style for the performance (primary/secondary).
bpm	Tempo integer (beats per minute) for the performance.
beat type	Either *beat* or *fill*.
time signature	The time signature for the performance.
midi filename	Relative path to the MIDI file.
audio filename	Relative path to the WAV file (if present).
duration	The float duration in seconds (of the MIDI).
split	The predefined split the performance.

TABLE III: Mapping: Roland, GM and Paper

Pitch	Rolland Mapping	GM Mapping	Paper Mapping
36	Kick	Bass Drum 1	Bass (36)
38	Snare (Head)	Acoustic Snare	Snare (38)
40	Snare (Rim)	Electric Snare	Snare (38)
37	Snare X-Stick	Side Stick	Snare (38)
48	Tom 1	Hi-Mid Tom	High Tom (50)
50	Tom 1 (Rim)	High Tom	High Tom (50)
45	Tom 2	Low Tom	Low-Mid Tom (47)
47	Tom 2 (Rim)	Low-Mid Tom	Low-Mid Tom (47)
43	Tom 3 (Head)	High Floor Tom	High Floor Tom (43)
58	Tom 3 (Rim)	Vibraslap	High Floor Tom (43)
46	HH Open (Bow)	Open Hi-Hat	Open Hi-Hat (46)
26	HH Open (Edge)	N/A	Open Hi-Hat (46)
42	HH Closed (Bow)	Closed Hi-Hat	Closed Hi-Hat (42)
22	HH Closed (Edge)	N/A	Closed Hi-Hat (42)
44	HH Pedal	Pedal Hi-Hat	Closed Hi-Hat (42)
49	Crash 1 (Bow)	Crash Cymbal 1	Crash Cymbal (49)
55	Crash 1 (Edge)	Splash Cymbal	Crash Cymbal (49)
57	Crash 2 (Bow)	Crash Cymbal 2	Crash Cymbal (49)
52	Crash 2 (Edge)	Chinese Cymbal	Crash Cymbal (49)
51	Ride (Bow)	Ride Cymbal 1	Ride Cymbal (51)
59	Ride (Edge)	Ride Cymbal 2	Ride Cymbal (51)
53	Ride (Bell)	Ride Bell	Ride Cymbal (51)

B. Dataset Preprocessing

The preprocessing of the data contained two different parts. The first one was to filter the dataset itself, and the second one was to translate the MIDI files into something we could feed into our model. The first step was to filter out the long beats and remove shorter fills as they were insufficient for training the network. After that, we took a look at the different genres. We found that many songs had multiple styles, for example, *latin / brazilian*. To increase our available data, we duplicated the beats we had in each genre category. In the given example, we duplicated the files for *latin* and *brazilian*. With this in mind, we looked at the total number of every genre and selected the five most common genres to continue. We ended up with:

TABLE IV: Genres

Style	Quantity
Rock	212
Funk	63
Latin	53
Jazz	50
Hip-hop	34

Translating a MIDI file into a drum matrix was rather complicated. First, we had to understand how the MIDI file works, then translate this information into a matrix. A MIDI file contains the information of a recording, not as frequencies but rather as a building plan. This plan describes the instruments and notes at a specified time and intensity. In our case, we only had one instrument, the drums. Each MIDI note number, or *Key Number*, corresponds to a different drum sound. Because MIDI can represent 46 of these sounds, we combined similar ones, for example, *High Tom* and *Hi Mid Tom* to just *High Tom*. With this approach, we ended up with a drum matrix containing 9 rows. In each row, we have a different group of drum sounds. The columns represent the beat in quarter notes. Every point inside this matrix represents the drum and the intensity it was played at a given beat and, therefore, represents the played rhythm. An example is shown in Fig. 2.

C. Rhythm Similarity Distance

In our analysis of our results we will use the ”swap distance”, as it has been found to be the most effective way to measure the differences between rhythm patterns [15]. The swap distance is the minimum numbers of swaps required to convert from one rhythm pattern to another. However, the swap distance is limited to patterns with the same length, meaning the same amount of onsets. This is why we are using the edit distance to measure the differences. The edit distance allows three operations: *swap*, *deletion* and *insertion* [1].

In Fig. 4 the edit distance is 2 for the first one and 4 for the second, as it takes two swaps to get from rhythm A to rhythm B and one deletion and three swaps to get from rhythm C to rhythm D. We also used the edit distance to calculate the differences between the four of the genres, which are: *rock*, *hip-hop*, *funk* and *jazz*.

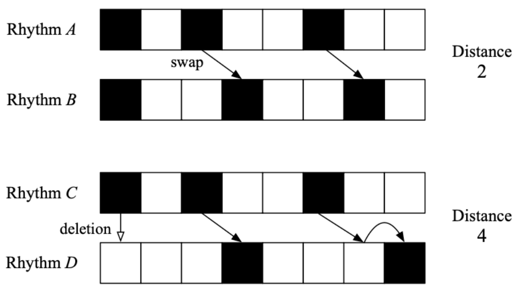


Fig. 4: Edit/Swap Distance Between Two Rhythm Patterns

Looking at the mean differences between the genres, jazz seems to be the most unique, as it requires the most swaps to obtain the other genres. Contrary to that, rock is the genre which requires the least amount of swaps on average. Over all the four genres, the average amount of swaps is 7.66.

TABLE V: Similarity Distance

Genre	Distance
Rock	7.63
Hip-hop	7.67
Funk	7.65
Jazz	7.69

D. Model Implementation

1) Genre-conditioned GAN: As the original paper suggested, we first implemented the GAN model conditioned on the music genre (Fig. 5). This model had to test whether it could reproduce rhythm patterns in a specified genre. In this case, both the discriminator D and the generator G got an additional input y, a scalar number $[0, K)$ representing a condition, or in other words, a genre. The input y, as part of the discriminator, was converted into a dense vector of the same shape as the drum onset matrix through the embedding layer. Next, the output of the embedding layer was concatenated with the drum onset matrix and forwarded to the two-layered bidirectional LSTM. The generator converted input y into the same size as the vector z and performed element-wise multiplication of the genre label with the vector z. This then became the generator's input. The discriminator and the generator then would train competitively.

2) GAN with Genre Ambiguity Loss: The second model had no conditioning input y, but it had a second discriminator D. One of the discriminators D_r served a function as before, i.e. telling apart the fake data from the real one, while the second discriminator D_c classified the genres. The Genre Classification Loss L_c was added to the discriminator loss function. Here, the generator was trying to fool both the discriminator D_r and D_c. At the same time, the generator

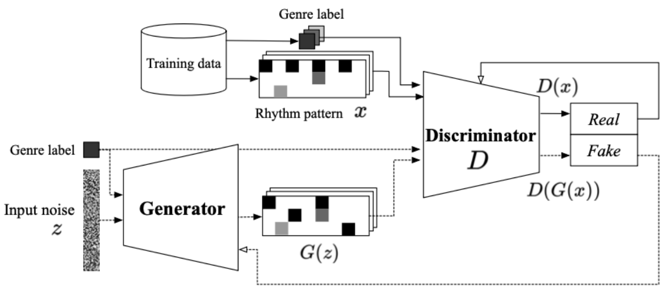


Fig. 5: Overview of Genre-conditioned GAN

was trying to convince the discriminator that the drum rhythm patterns were sampled from the real distribution to prevent the D_c correctly classify the generated genre.

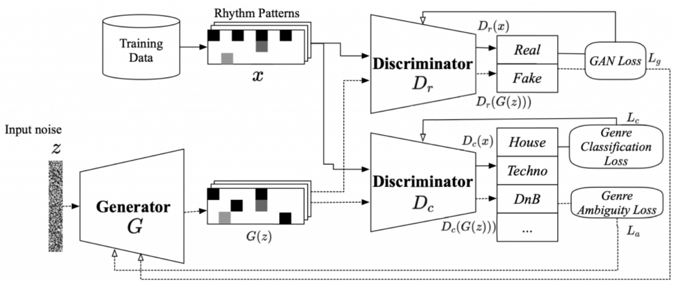


Fig. 6: GAN with Genre Ambiguity Loss - Creative-GAN

E. Model Training

As mentioned before, we trained one generator and two discriminators. For training we used drum matrices with the shape of $(32, 9)$. We shuffled and prefetched the data and used a batch size of 200. Except of a few minor changes, we nearly hold on to the model structure introduced by Tokui [1]. For the first part, that including the grid computing, we trained with 10 epochs.

1) Beat Generator: The Beat Generator should create new beats from random noise. Therefore, we started with a normal distributed vector of length $288 = 32 \times 9$ as input vector. The first layer was a *dense layer* with 1024 units, followed by a *LeakyReLU* function with $alpha = 0.2$, reshaping to a (32,32) matrix, two LSTMs, both with 512 units, activation function of $tanh$ and returning sequences. The output was generated by a 9 unit LSTM with *sigmoid* activation function and returning sequences as well. As loss function we used *Binary Cross Entropy*.

2) Beat Discriminator: The first Discriminator had the purpose to distinguish between real, man-made and from the generator created beats. To accomplish this, the Discriminator received a drum matrix with the shape of $(32, 9)$ as input. This was first fed into a bidirectional LSTM with 64 units, activation function $tanh$, returning sequences and a dropout rate as well as a recurrent dropout rate of 0.4. This layer was followed by the same layer, but without returning sequences. To gain a single outcome, the model finished with a sigmoid activated *dense layer* with a single unit.

3) Grid Computing: One major difference in our work with this model was, that we used grid computing to find the best hyperparameter. Sadly we just had enough time to focus on optimiser. However, we tried three different optimiser: RMSProp, SGD and Adam. All three with the different learning rates of 0.0001, 0.001, 0.01, 0.0005, 0.005, 0.05.

4) Genre Discriminator: After the first training process we trained another discriminator to specialise the generator more. Here we used as positive labels all drum matrices with the style ”rock”, as negative labels all other styles. To use both, we first had to bring this two datasets to the same length. Therefore we duplicated random samples of the shorter data-set. The model is the same as for the first Discriminator.

V. Results

A. Grid Computing

The results of the grid computing are visualised in figure 9. Shown are all the losses of the discriminator and the generator as well as the accuracy for all three different optimisers. Every line in every graph represents one of the six learning rates. The results shown, were right after the first training, where we trained the first discriminator and generator to create man-made-like drum beats. By analysing the different combinations of optimiser and learning rate we took five combinations and let them run again though the second part of the model. The combinations were RMSProp with learning rates of 0.01 and 0.05, SGD with 0.001 and 0.05 and Adam with 0.001.

B. Beat Creation

The creation of a new beat matrix after the first training is shown in figure 8. Here you can see a drum matrix created from our Generator, after training with the first Discriminator. Meaning the Generator is not jet specialised on a genre. The corresponding *.wav* file is in our GitHub repository under *data/beats/*. Visualised is *RMSProp0.01.wav* because this *wav* file results in most convenient beat. The other optimiser did not create beats, which sounded really naturally at all. Therefore we used the optimiser RMSProp with a learning rate of 0.01 for our further training.

The result of the further training, so the specialisation on the *rock* genre is visualised in figure 8 and audiolised in the *data/beats/final_rock_beat.wav* file. As you can see, in comparison to the beat after the first training step, the drum matrix become way more specified. This could come from just the fact, that it run through more epochs in general. However, it could be as well, that we have here a more specific beat, that's focus on more 'rock' like beats. This two possibilities should be tested, by increasing not just the overall number of epochs, but as the epochs within the two training epochs as well.

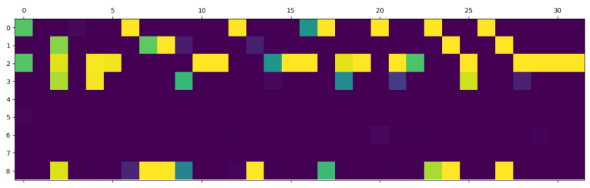


Fig. 7: The drum matrix for RMSProp with learning rate 0.01 after the first training part

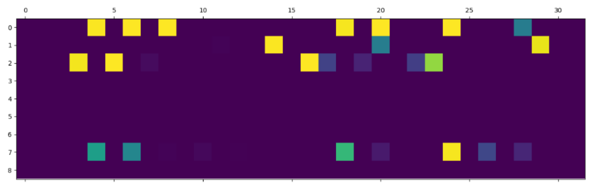


Fig. 8: The drum matrix for RMSProp with learning rate 0.01 after the complete training and genre specification.

C. Training Time

We want shortly mention, how long the training took place. First of all, already the prepossessing, so the translation from Midi file to a drum matrix took for 416 data points around 50 minutes. For the first training period the training took around 4 minutes per epoch resulting in 40 minutes with 10 epochs. The second training was however, faster, but takes another 30 min to complete. Therefore we end up with over two of training time.

VI. Discussion

A. Findings

As the author of the originally paper we were able to translate a MIDI file into a drum matrix, train a network and transform the result into a listenable *.wav* file. However, we choose a slightly different approach in the second step of training,meaning not to create a complete new style, but to specialise on a certain, chosen genre. Even though we had a much smaller dataset, we created in the first training process something, that could be man-made, and even sounds for us humans kind of natural. Our approach to use hyperparameter optimisation to overcome our lack of data, did not payed out, because we actually more or less ended up with the same optimiser (RMSProp) as the original paper.

B. Limitations

One of the major limitations were of course the calculation time. The calculation power was much higher than we expected. Therefore we weren't able to train for many epochs. Even though we used grid computing, we couldn't

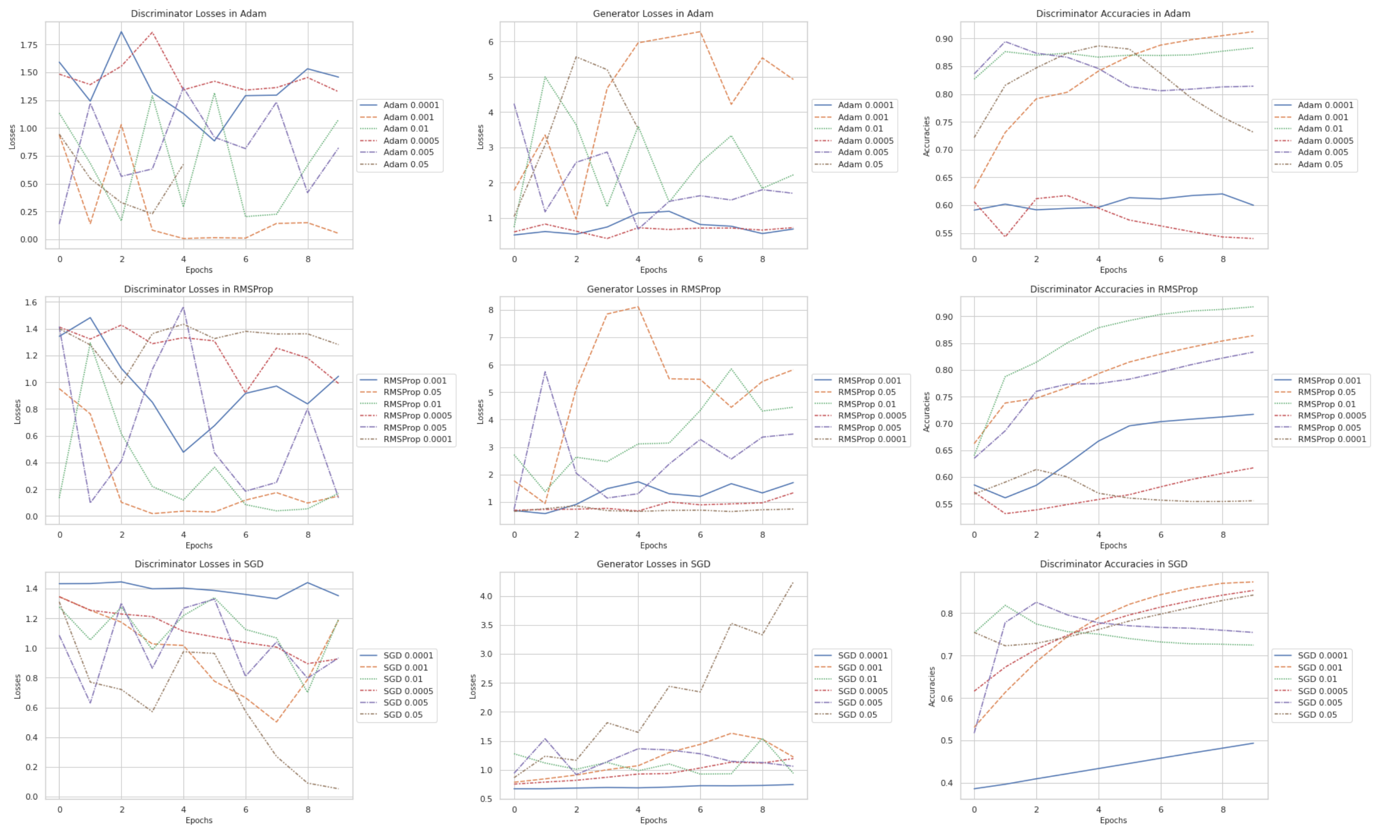


Fig. 9: Optimiser Comparison Graph: Adam, RMSProp and SGD

optimise our model the way we would have liked.

The artificial creation of music is on many different levels complicated. We looked in our project just on one, small aspect of that, the beat generation. And even here we had to make major constrains. Normally MIDI files of drums can contain up 60 different channels, we reduced them on only 9. In addition, a song has not just one rhythm over the whole song, it can differ, has fillins, brakes and more specific reactions. All this cannot be caught in just 8 tact's, 32 notes. However, with more notes, the matrices are getting larger as well, and so does the computational effort, don't imagine for complete songs.

Another limitations were the epochs, with more epochs the results would be much better as well. For all this three problems the major limitations was the computational power. A training on a GPU would have maybe solved some of this problems but definitely not all. Our training took place only on a CPU.

The last limitations was the dataset. Here we had, even though we splitted every beat into 8 tact's sequences, not enough data and definitely not enough different styles. The only possible genre we could specify on was the *rock* genre. But, of course, more data would have resulted into more computational effort as well.

C. *Perspective*

As already mentioned before, the perspective of future work are broad. Beginning with a bigger dataset for example the one which was originally used by Tokui. Different specified Generators are possible as well. On the MIDI scale, the training could use longer sequences of for example 16 tact's or even more, and use more different channels.
On the model level are optimisations possible as well. With the already used grid computing, much more hyperparameters could be tried out and even the model structure could be varied to find a better fitting model.

VII. Conclusion

Overall, we were able to train a GAN model to generate rhythm patters for rock music while analysing which optimisers were the optimum one to use. There are still limitations in our work, such as a rhythm not staying exactly the same across a song, and of course our time constraints in training, as a longer training time would have led to better results. In general, there is still a lot of space for improvement in the state of the art in music generation, as so far it is not possible to manage to produce a novel complex music piece. Focusing on rhythm could be a good area of emphasis, as rhythms are now an integral part of most music and therefore a necessary complement to melodies and harmonies.

References

[1] N. Tokui, "Can GAN originate new electronic dance music genres? – generating novel rhythm patterns using GAN with Genre Ambiguity Loss."

[2] J.-P. Briot and F. Pachet, "Music Generation by Deep Learning - Challenges and Directions," *Neural Computing and Applications*, vol. 32, no. 4, pp. 981–993, 2020.

[3] "Google: Celebrating Johann Sebastian Bach."

[4] M. N, ""how YACHT fed their old music to the machine and got a killer new album "."

[5] I. L. Hiller LA, *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill, New York, 1959.

[6] J.-P. Briot, "From Artificial Neural Networks to Deep Learning for Music Generation: History, Concepts and Trends," *Neural Computing and Applications*, vol. 33, no. 1, pp. 39–65, 2021.

[7] D. Eck and J. Schmidhuber, "A First Look at Music Composition Using LSTM Recurrent Neural Networks," 2002.

[8] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang, "MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation."

[9] J. P.-A. Ian J. Goodfellow, "Generative Adversarial Nets," 2014.

[10] L. Weng, "From GAN to WGAN."

[11] "Deep Learning Techniques for Music Generation – A Survey."

[12] "Combining LSTM and Feed Forward Neural Networks for Conditional Rhythm Composition," in *Engineering Applications of Neural Networks* (G. Boracchi, L. Iliadis, C. Jayne, and A. Likas, eds.), (Cham), pp. 570–582, Springer International Publishing, 2017.

[13] J. Gillick, A. Roberts, J. Engel, D. Eck, and D. Bamman, "Learning to Groove with Inverse Sequence Transformations,"

[14] J.-P. Briot, G. Hadjeres, and F.-D. Pachet, *Deep Learning Techniques for Music Generation*. Cham: Springer International Publishing, 2020.

[15] G. Toussaint, "A Comparison of Rhythmic Similarity Measures.," 01 2004.