COMPARING DEEP COMPOSERS FOR THE EUROVISION AI SONG CONTEST

Otto Mättas 6324363

Thijs Ratsma 6315887

Thijs Hendrickx 6853757

ABSTRACT

In order to contribute to the VPRO Eurovision AI Song Contest team, we explore and compare various options for generating songs using AI. MusAIc (a custom tool developed by a member of the team), commercially available products (e.g. Amazon DeepComposer, SageMaker, Google Magenta) and Folk RNN (a recurrent neural network developed as a composing tool for Irish folk music). While we were unable to generate music using the first two methods, we did generate multiple samples with Folk RNN. After a user study with 27 participants rating these samples, we concluded that Folk RNN cannot generate proper songs but can be used for inspiration or for generating a seed to be used with the musAIc tool. We recommend the VPRO team use the musAIc tool for song generation, as the developer is part of their team.

Index Terms: deep learning, neural network, Folk RNN, musAIc, AWS DeepComposer, Eurovision AI Song Contest

1. INTRODUCTION

In honor of the Eurovision Song Contest 2020 being held in Rotterdam, the Dutch Public Broadcaster VPRO is organizing the first AI Song Contest in the Netherlands. Contestants will be provided with a symbolic data set of nearly 200 different Eurovision songs, which will be used to create a song in the style of Eurovision with the help of AI. The motivation for hosting this contest is the exploration of AI in the context of songs and music.

For this project, we will use three methods of music generation in order to compare outputs of the methods and determine the better one suited for generating a Eurovision melody. We are aiming to measure and reach a conclusion while also opening up doors to further research.

First, we will take a look at a deep neural network composer created by a member of the VPRO AI team, *musAIc* [1]. This composer can be used to train a network based on a collection of music samples, as well as use the trained network to generate new samples.

Secondly, we will explore various commercially available machine learning tools such as *AWS DeepComposer*, a new managed service by Amazon. Training Models requires us to turn to another managed service, *AWS SageMaker*. Initially, it looks like a complete solution with a full feature set for easy training and tunes creation.

Lastly, we will use an existing recurrent neural network called *Folk RNN* [2]. While this network is mainly meant as a composing tool rather than an automatic composer, it is still possible to create coherent melodies.

After samples have been generated, we will evaluate them based on an informal questionnaire posed to 27 students. This evaluation will determine which parameters should be set while training a neural network to achieve the best music sample.

Finally, we will discuss how this research could be improved, as well as further research that could be undertaken based on ours.

2. RESEARCH PROBLEM

We set out to compare three different toolsets for machine song generation: musAIc, commercial composers and Folk RNN. What are the differences between these composers and how do these differences reflect on the outcome of newly generated songs? Furthermore, which features could be most useful for song generation for the purpose of the Eurovision AI Song Contest? To this end, we pose the following research questions:

- How do musAIc, commercial composers and Folk RNN compare?
- What parameters in training deep composers produce the "best" result?

This project provides insight into the current state-of-the-art and the benefits of the various available choices. It also explores if certain algorithm features are more beneficial in the context of the AI song contest.

3. APPROACH

We will now explore our findings on the three different deep composers that we have set out to use for this research. In this section, we discuss the context of the deep composers, the capabilities as well as the outcome of our attempts to generate any Eurovison melodies with them.

3.1 musAIc

musAIc is a deep composing tool that can be used for both training a neural network as well as implementing it to generate new MIDI samples. The software was developed in 2019 by Arran Lyon and has been constantly updated since the start of the Eurovision AI Song Contest, as Lyon is part of the VPRO team.

The architecture of the neural network consists of a custom model that combines models typically used in Natural Language Processing, where LSTMs are used to generate rhythmic sequences, followed by a CNN to fill in the melody line. This melody line can then be accompanied by a chord sequence, which is trained using a separate neural network.

After the neural network has been trained, the GUI of musAIc (see figure 6) can then be used to generate music samples. Generation is done for each individual instrument, which the user can add and remove at will. The user can select a number of measures to generate for each instrumental track. The measures can then be selected and tweaked using the parameters at the bottom of the screen:

- Actions: The main panel for actions of the selected measures. Here, the user can delete, duplicate and to generate (or regenerate) melodies. Measures with an 'X' in the corner are blank and have not been generated, 'O' means that the measure is being generated. After a measure has been generated, it will show a roll of its tones.
- Playback: Sets the octave transpose, note length (0 plays each note at their original length), and the MIDI velocity range.
- Structure: Defines the structure of the main melody. The first number indicates the number of bars in the main melody, while the second number shows the number of repetitions of these bars. If there more than one repetition, the length and the number of alternate endings of the musical phrase can also be defined.
- Lead: Can be used to set which instrument should be set as the lead instrument for the selection, which the selected instrument will then intuitively try to harmonise with. If this setting is set to "None", the selection will generate music independently from other instruments.
- Sample: Various settings for how musAIc will generate the output of the neural network, such as the variety in picked notes and rhythms, likelihood of picking random notes and chord interaction. The number is used to set the maximum number of notes to play at the same time (e.g. '1' would create monophonic output).
- Style: Gives the user more control over which types of rhythms and melodies the network will generate. The user can choose to musAIc lets the network make all of it's own choices or to have some degree of influence on the network, such as scale and type of rhythms. While this will increase the likelihood of these scales and rhythms occurring, it is not guaranteed that the melody will only be in these scales or rhythms.
- Meta: Meant as a random seed manipulator that feeds extra information into the network. The idea is that adjusting these knobs should change the stylistic choices, however this appears to have little effect for now. Future iterations of the network may add more meaning to this setting.

The option to train a neural network, along with the large selection of parameters with which to tweak the to be generated output make musAIc a very suitable network to generate songs for the Eurovision AI Song Contest. Unfortunately, at the time of conducting research, the model is still in development. This, along with the lack of proper documentation, left us unable to generate a melody to use in the evaluation. However, we believe that as the progress on the AI Song Contest progresses, musAIc will see plenty of use, especially since its developer is part of the VPRO AI team.

3.2 Amazon Web Services (AWS)

AWS is committed to democratizing machine learning. Their goal is to make it widely accessible to customers with different levels of training and experience, and to organizations across the board. AWS innovates rapidly, creating services and features and Machine Learning services are no exception. In the diagram below, you can see how the current AWS Machine Learning services map to the other common AI diagrams. (See Figure 7.)

3.2.1 AWS DeepComposer

Currently missing from the scene is AWS DeepComposer which was announced late last year while the service is not yet delivered to the general public. Users will be able to get hands-on with a musical keyboard and the latest machine learning techniques. Designed specifically to educate, AWS DeepComposer includes tutorials, sample code, and training data that can be used to get started building generative AI models, all with a promise of having not to write a single line of code.

While not all details are known yet, they are making use of Generative Adversarial Networks (GANs). This Generative AI technique pits two different neural networks against each other to produce new and original digital works based on sample inputs. With AWS DeepComposer, you can train and optimize built-in GAN models. To customize even further, you can train models with another tool within the ecosystem - Amazon SageMaker.

Currently known features of the DeepComposer toolset include:

- Input with keyboard - User can input a melody by connecting the AWS DeepComposer MIDI-compatible keyboard to their computer, or play the virtual keyboard in the AWS DeepComposer console. You can also export the MIDI files to your favorite Digital Audio Workstation (DAW) and get even more control over the input to the ML model.(See Figure 8.)
- Manage with console - User can generate an original musical composition using the pre-trained genre

models in the AWS DeepComposer console. Initial choices include models for rock, pop, jazz, classical, or building your own custom genre. User will have ability to tweak the model hyperparameters such as tune and tempo.

- Bring your own music data set in MIDI format and create your own custom models in SageMaker. But you don't need to as there are four pre-trained models for use.

- Saving and exporting your musical creations in MIDI for additional processing using external tools, or in WAV or MP3 format for sharing. Also, the user can share their creations by publishing the tracks directly to SoundCloud.

- DeepComposer is a cloud service, connection to the internet is required to run inference against models for musical creations.

There are two ways to get started. First, one can create music using one of the sample models that have been pre-trained on musical inputs from multiple genres, such as pop and classic. After inputting a melody either from an example sample, or a keyboard, DeepComposer performs inferences to generate an output with a 4 part accompaniment in that genre.

Secondly, learning can be taken further by creating own custom models using one of DeepComposer's publicly available SageMaker notebooks. You can also bring your own data set and build a customized machine learning notebook in SageMaker to learn and create your own Generative AI models. This potentially allows the toolkit to be used for such a purpose as generating music for the Eurovision AI Song Contest.

In addition to lowering the learning curve, the company also gives leeway in terms of costs for the user. For getting started, AWS DeepComposer provides a 12-month Free Tier for first time users, so one can try it out by generating new music using their sample models at no cost. The Free Tier of up to 500 inference jobs lasts for 12 months and starts when you first use the AWS DeepComposer service.

After the year long period, one will be charged based on their usage, as there are no minimum fees or upfront costs. When you use it to train models and generate music compositions (known as inference), costs will be accumulated based on an hourly rate.

As the VPRO team might be interested in generating custom models, we dedicated time investigating the capabilities of AWS SageMaker aswell.

3.2.2 Amazon SageMaker

Amazon SageMaker is a fully-managed machine learning service that enables developers and data scientists to quickly and easily build, train, and deploy machine learning models at any scale. The service includes different functionality from managing data sets to deploying models to production workflows: (See Figure 9 and Figure 10.)

- Amazon SageMaker Studio - a fully integrated development environment (IDE) for machine learning which provides a single, web-based visual interface where the user can perform all ML development steps. SageMaker Studio gives complete access, control, and visibility into each step required to build, train, and deploy models. One can quickly upload data, create new notebooks, train and tune models, move back and forth between steps to adjust experiments, compare results, and deploy models to production all in one place. All ML development activities including notebooks, experiment management, automatic model creation, debugging, and model drift detection can be performed within the unified SageMaker Studio visual interface. (See Figure 11.)

- Amazon SageMaker Ground Truth helps build training data sets quickly and accurately using an active learning model to label data, combining machine learning and human interaction to make the model progressively better. After a few experiments, we can conclude that this tool is most suitable for managing data sets for supervised learning.

- Fully-managed and pre-built Jupyter notebooks are provided to address common use-cases. The services come with multiple built-in algorithms, and there is the AWS Marketplace for Machine Learning containing more than 100 additional pre-trained ML models and algorithms. Own algorithms and frameworks that are built into a Docker container can be used as well. We used GluonNLP Sentence Generator for generating a model on the data set, namely the files in abc-format. Also, Amazon SageMaker's implementation of XGBoost to create a highly predictive model was used to train models.

- Amazon SageMaker also includes built-in, fully managed Reinforcement Learning (RL) algorithms. SageMaker supports RL in multiple frameworks, including TensorFlow and MXNet, as well as custom developed frameworks.

- SageMaker sets up and manages environments for training, and provides hyperparameter optimization with Automatic Model Tuning to make the model as accurate as possible. SageMaker Neo allows you to deploy the same trained model to multiple platforms. Using machine learning, Neo optimizes the performance and size of the model and deploys to edge devices containing the Neo run-time. AWS has released the code as the open source Neo-AI project on GitHub under the Apache Software License. SageMaker deployments run models spread across availability zones to deliver high performance and high availability. We experimented with the tool in order to make the model available for music generation.

- There are many more high-level features to list but were excluded from research in the scope of this

project. For example - Amazon SageMaker Experiments, Amazon SageMaker Debugger, AWS' TensorFlow optimizations, Managed Spot Training, one-click model deploy onto auto-scaling Amazon ML instances across multiple availability zones for high redundancy, Amazon SageMaker Model Monitor, Integration with Kubernetes for orchestration and management.

As part of the AWS Free Tier, starting with Amazon SageMaker is free. For the first two months, first time users are offered a monthly free tier of 250 hours of t2.medium or t3.medium notebook usage for building models, plus 50 hours of m4.xlarge or m5.xlarge for training, plus 125 hours of m4.xlarge or m5.xlarge for deploying your machine learning models for real-time inferencing and batch transform with Amazon SageMaker. (See Figure 12.)

SageMaker provides all of the components used for machine learning in a single toolset so models get to production faster with much less effort and at a lower cost. Unfortunately, our experience with GAN methods was lacking to employ the toolset fully. In the time scope of the project, we were able to generate models from our initial data set but did not manage to generate audio in order to compare. This is where AWS DeepComposer promises to lower the curve for future work and make generating AI music more accessible.

3.2.3 Magenta - A Google Research Project

Magenta is an open source research project exploring the role of machine learning as a tool in the creative process, developed at Google. It splits into two directions:

- Python - Magenta is distributed as an open source Python library, powered by TensorFlow. This library includes utilities for manipulating source data (primarily music and images), using this data to train machine learning models, and finally generating new content from these models.
- JavaScript - Magenta.js is an open source JavaScript API for using the pre-trained Magenta models in the browser. It is built with TensorFlow.js, which allows for fast, GPU-accelerated inference. If you are interested in seeing how Magenta models have been used in existing applications or want to build your own, this is probably the place to start!

In February 2019, Magenta Studio was released. It is a collection of music creativity tools built on Magenta's open source models, available both as standalone applications and as plugins for Ableton Live. Each of the plugins lets you utilize Magenta.js models directly within Ableton Live. The plugins read and write MIDI from Ableton's MIDI clips. Or if you do not have Ableton, you can just use MIDI files from your desktop. The first release includes 5 apps: Generate, Continue, Interpolate, Groove, and Drumify. (See Figure 13.)

- Generate uses MusicVAE to randomly generate 4 bar phrases from its model of melodies and drum patterns learned from hundreds of thousands of songs;
- Continue can extend a melody or drum pattern you give it;
- Interpolate can combine features of your inputs to produce new ideas or create musical transitions between phrases;
- Groove is like many "humanize" plugins, but it adds human-like timing and velocity to drum parts based on learned models of performances by professional drummers;
- Drumify is similar to Groove, but it can turn any sequence into an accompanying drum performance.

Given how simple these plugins make it to interact with complex machine learning models, we plan on using this platform to try if it holds value in the scope of our current project.

The plugins are built using Electron and Max For Live. Using Electron is said to bring a few advantages: it allows for the use of Tensorflow.js and Magenta.js for GPU-acceleration without requiring users to install any additional tools or GPU drivers. It also gives users the ability to develop the interface with familiar tools such as Polymer. Finally, Electron lets a standalone version of each plugin be created for people who do not have Ableton Live.

Beyond this initial set of plugins, the researchers have a lot of other interesting models in the pipeline – including polyphonic transcription, transformer piano performances, and synthesis – that are hoped to be released soon, in addition to their ongoing and future research. The researchers have expressed excitement about the opportunity to allow artists to personalize models through local training, for example by applying developed techniques on latent constraints.

In the scope of our project, we were not able to import our own data set or pre-generated models. Even though the plugins work as expected and bring a lot of value for exploration in music, we were not able to apply the toolset on our particular problem of music generation from the Eurovision data set.

3.2.4 Other

- Musenet - a deep neural network that can generate 4-minute musical compositions with 10 different instruments, and can combine styles from country to Mozart to the Beatles. MuseNet was not explicitly programmed with human understanding of music, but instead discovered patterns of harmony, rhythm, and style by learning to predict the next token in hundreds of thousands of MIDI files. MuseNet uses the same general-purpose unsupervised technology as GPT-2, a large-scale transformer model trained to predict the next token in a sequence, whether audio or text. For generating on novel data sets, this toolset is not usable.

- Azure Machine Learning - like Amazon Web Services, Azure Machine Learning can be used for any kind of machine learning, from classical ML to deep learning, supervised, and unsupervised learning. The code options for development include Python, R code and zero-code/low-code options such as the designer. The toolset allows to build, train, and track highly accurate machine learning and deep-learning models within the ecosystem.

3.2.5 Conclusions

So, what are the tools out there meant for users rather than researchers? We looked at many and here are some takeaways:

- It is the easiest to generate your first piece of music with Musenet while SageMaker gives you the most power and freedom.
- It was becoming clear, though that the market gravitates towards two categories - fun tools others have built for research and fully-capable general purpose AI/ML studios with a lot of customization options.
- We generated many models and tried to reach the end goal of having music but due to limitations of the tool and our experience with the full landscape (for studios), we did not get to usable output yet.

We can therefor not give a recommendation for the VPRO team on using one of these toolsets.

3.3 Folk RNN

Folk RNN is a recurrent neural network developed for the purpose of generating samples that could help in the composing of music. It consists of three hidden layers with Long Short-term Memory (LSTM) nodes. The creators did not intend for the tool to be used to complete complete songs. Originally it was trained on Irish folk tunes and has produced various music samples that various composers and performers have used to create folk songs with. We wanted to see if it could produce usable results when trained on the Eurovision song data set.

Since Folk RNN has been out for 4 years, a decent amount of documentation is available which helped in getting the tool to run. Folk RNN consists of a training script and a sample script. The training script generates a model based on the data set and configuration it is provided. The sample script generates a sample with this model, and allows for a custom starting seed (read starting melody).

We trained Folk RNN on the Eurovision song data set using a variety of training parameters. The following parameters could be altered in the configuration.

- Use One-Hot encoding: encoding method that prevents issues when categorizing different classes.
- Number of layers: number of layers of the neural network.
- **RNN size**: number of neurons per layer.
- Dropout: ratio of neurons that will drop out during training to improve learning.
- **Learning rate**: rate at which the neural network will try new random inputs.
- **Learning rate decay**: rate at which the learning rate will start decreasing per iteration.
- **Learning rate decay threshold**: iteration at which learning rate decay will set in.
- **Batch size**: number of training samples used per batch.
- **Max epoch**: maximum number of epochs the training will take place for.
- Gradient clipping: setting used to prevent exploding gradients.
- **Validation fraction**: fraction of worst training results that will be discarded after a validation.
- **Validate every x iterations**: number of iterations that occurs before a validation happens.

We decided to only alter the parameters in bold, as the others were either widely accepted industry standards or we did not know exactly what they did and could therefor not provide reasoning for altering said parameter.

All our iterations can be categorized into three main groups based on the goals we had for those iterations. Our first goal was to see what would happen if we trained Folk RNN using the default parameters that the authors used when training on their data set of 24000 Irish folk tunes, and compare this with scaled parameters to match our data set. Next we focused on determining if we should increase the data set size by appending it with an equal amount of Dutch folk tunes. Lastly, we started changing the parameters in order to determine what (to us) led to the most coherent result. The parameters of each iteration can be seen in Table 1. Parameter values in red indicate a change compared to the previous iteration.

4. RESULTS

4.1 Session 1

In session 1 we compared the default parameter set with a scaled parameter set based on our data set size. We noticed that the default parameter set was overfitting heavily on the data set, which made sense considering that the batch and node sizes were too large for a data set of 200 songs. Our parameter set generated a way more coherent piece of music, which is why we decided to work with these parameters for our next session. We experimented with a higher epoch count, but this did not make a noticeable difference in the result so we went back to 100 epochs.

4.2 Session 2

We hypothesized that a data set size of 200 was considered very low, especially compared to the 24000 songs the Folk RNN authors used. In order to evaluate if we could get more workable results from a larger data set, we appended the Eurovision data set with 200 additional Dutch folk tunes from the Nederlandse Liederenbank. This doubled the data set size, which we could then compare to only using the Eurovision data set. The result showed us that instead of intermingling the two various styles of songs in the mixed data set, it would pick either the folk or Eurovision style for each individual music segment. This resulted in a non-coherent piece of music with different styles all over the place. We therefor concluded that we would continue to use the non-altered Eurovision data set for our final session.

4.3 Session 3

Now that we scaled our parameters and picked the best data set, we could start tweaking the parameters. We started off by investigating if a larger RNN node size and more frequent validations would have a desired effect. This resulted in an impressive music sample with some original parts that worked well together. The overall music structure was also more coherent than we had seen before. Based on this we decided to stick with a node size of 64. Next we wanted to explore how changing the validation rate would affect performance, as we hypothesized that increasing or decreasing the validation could affect how much the model was generalizing the data set. We started with a baseline of no validation and strong a validation fraction with less frequent validation. We compared these 2 results with I4 to determine if we should have no validation, less frequent but stronger validation or more frequent but softer validation. Out of these three, we thought I4 (more frequent but softer validation) was better. We then tested the extreme of this direction, training a model with an extremely high validation frequency and extremely low validation fraction. This did not improve on our result of I4 so we decided to leave the validation parameters like I4. Lastly we wanted to test if increasing the learning rate would prove beneficial. This did not turn out to be the case though, and we got a similar quality piece to I4.

4.4 User study

Up until this point we have been determining the course of our training purely subjectively with our opinion of the pieces. In order to bring this more into scientific perspective, we conducted a user study with 27 participants. We selected 3 samples from our third training session based on which we thought were "best" sounding and played them one by one for the participants. The participants were then asked to fill out a form which asked them to grade each sample on a scale of 1 to 5, and which sample was their favourite.

As can be seen in Figure 4 and Figure 5, the participants graded the I4 and I6 samples way higher than I7. This is in

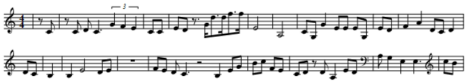

Figure 1. Score of I7 used in the evaluation.

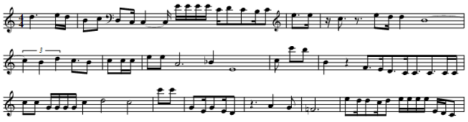

Figure 2. Score of I6 used in the evaluation.

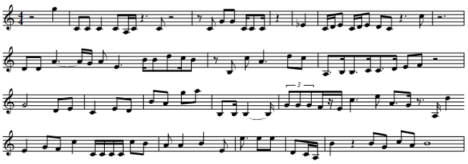

Figure 3. Score of I4 used in the evaluation.

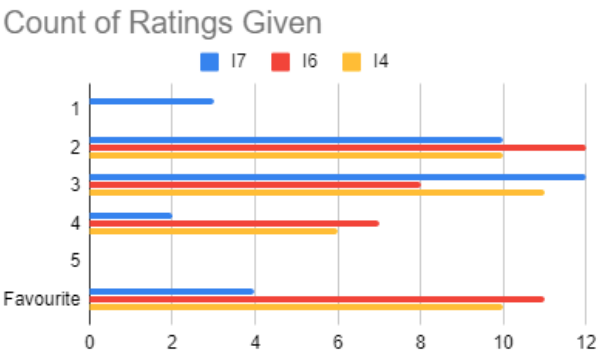

Figure 4. Score and favourite count per sample.

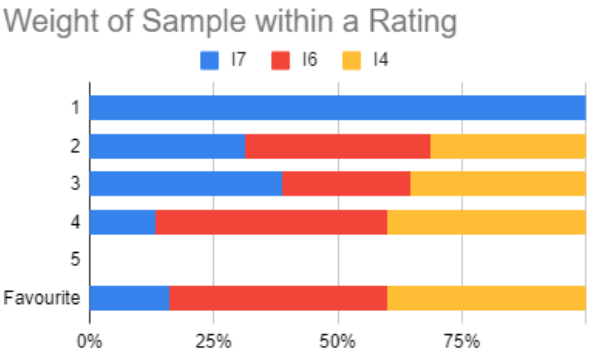

Figure 5. Weight per sample for each score value and favourite.

line with what we concluded, that the extremely increased validation frequency and decreased validation fraction had a negative impact on the song quality. Since I4 and I6 had similar preferences, we conclude that validation parameters near the middle of the spectrum based on the Folk RNN authors' defaults worked best.

Overall the scores were quite low, with many additional comments that the music pieces did not sound that great and coherent. This is reflected in the mean score for all the samples which was 2.5 for the I7 sample and 2.8 for the other two. The individual segments within each music piece were more positively received however. This is once again in line with what the Folk RNN authors also established themselves. As the individual segments were positively received, we do think that Folk RNN can be used by the VPRO Eurovision AI song contest team as inspiration or starting seed input for a more fleshed out song generation algorithm.

5. DISCUSSION

While the results show that sample I7 scored the lowest of all three samples, samples I6 and I4 score about even. Furthermore, the scores for all three samples were below average, which was lower than we were hoping. None of these samples seemed to be up to Eurovision standard, according to our fellow students. Why was this?

For starters, Folk RNN was never meant to be used as a deep composer, but rather as an automated composing tool to spark creativity during a composing session by a human composer. We mainly wanted to include Folk RNN in this research to see how it would perform compared to software that had composing as its main purpose, but due to complications with the other deep composers this was unfortunately not possible. For future research, we propose at least one other deep composer that focuses on music generation should be trained alongside Folk RNN to compare their results, preferably AWS DeepComposer or musAIc.

Our training parameters may also have been determined too naively. Our initial settings were based on the settings used in the original Folk RNN paper [2], from which we tweaked individual parameters until we felt we achieved a satisfactory result. The original parameters were determined based on a training set of nearly 24,000 folk songs, however. Scaling these parameters down to nearly 200 Eurovision songs might be a more complex and precise task than we had originally envisioned. Furthermore, a data set of nearly 200 songs to train on is likely a bit on the short side. The variety in this data set could be enriched by adding more songs or by splitting melodies within each song. This way, the neural network will train on generating a melody rather than trying to piece a whole song together.

6. CONCLUSION

When we first started this research, we set out to explore the differences between three different deep neural network composers to see which one would be most suited for use in melody generation for the Eurovision AI Song Contest. After exploring and reviewing musAIc and AWS DeepComposer and realizing their current limitations, we instead shifted our research to exploring parameter settings for training Folk RNN to produce the best-sounding melody. After an evaluation performed by 27 fellow students, we found that no melody scored significantly better than the other two. While we did not find the optimal set of parameter settings to generate the best Eurovision song, we did get a better understanding in what parameter settings have an influence on the result and what research could be conducted in the future to achieve a definitive result. For now, we advise the VPRO AI team to continue using musAIc, as the lead developer of the software is part of their team. The team can also use Folk RNN to generate short samples for inspiration or starting seeds for musAIc.

7. CONTRIBUTIONS

Thanks to Arran Lyon for his work on the development of musAIc, as well as his assistance and support with our research.

8. REFERENCES

[1] Arran Lyon. musaic github repository. In *https://github.com/al165/musaic*, 2019.

[2] Bob L. Sturm, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. In *arXiv preprint arXiv:1604.08723*, 2016.

9. APPENDIX

	Session 1			Session 2		Session 3				
	Default	**I1**	**I2**	**I3-E**	**I3-M**	**I4**	**I5**	**I6**	**I7**	**I8**
RNN Size	512	32	32	32	32	64	64	64	64	64
Learing rate	0.003	0.003	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.01
Learning rate decay	0.97	0.97	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.98
Learning rate decay after x iterations	20	20	40	40	40	40	40	40	40	40
Batch size	64	8	16	16	16	16	16	16	16	16
Max epoch	100	100	200	100	100	100	100	100	100	100
Validation fraction	0.05	0.05	0.05	0.05	0.05	0.05	0	0.1	0.01	0.05
Validation after x iterations	1000	60	60	60	60	30	0	60	10	30

Table 1: The parameter values for every training iteration.

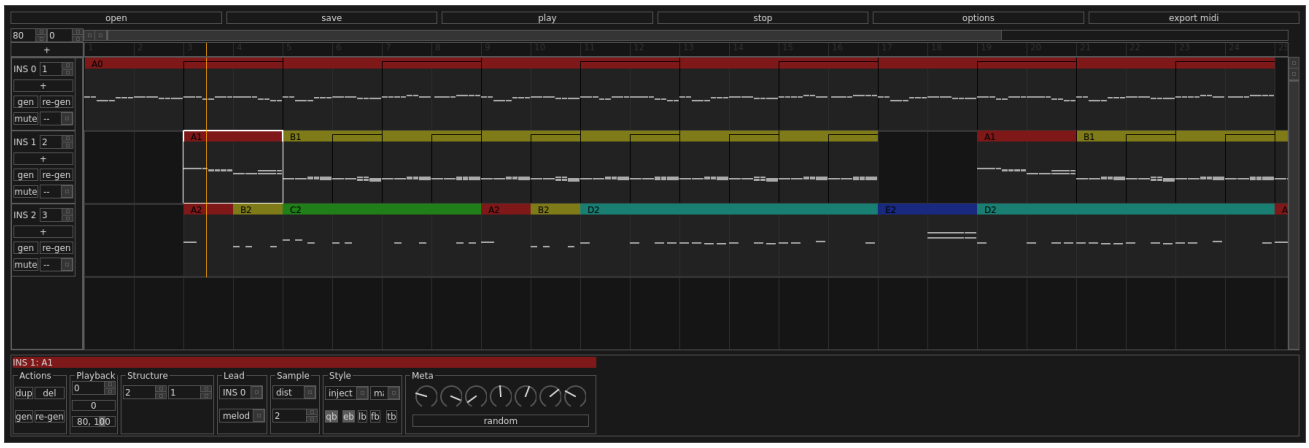

Figure 6. musAIc's GUI.

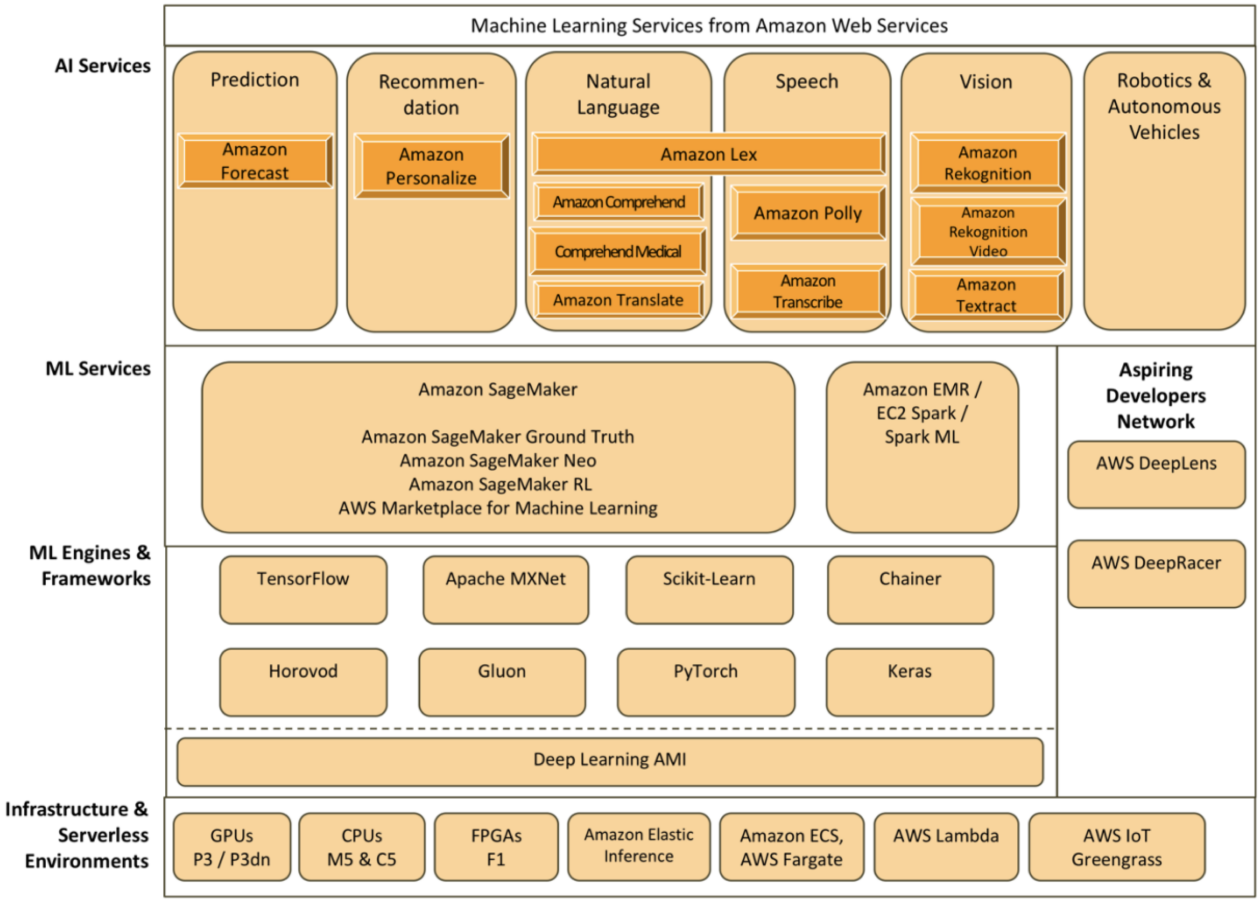

Figure 7. AWS Machine Learning landscape.

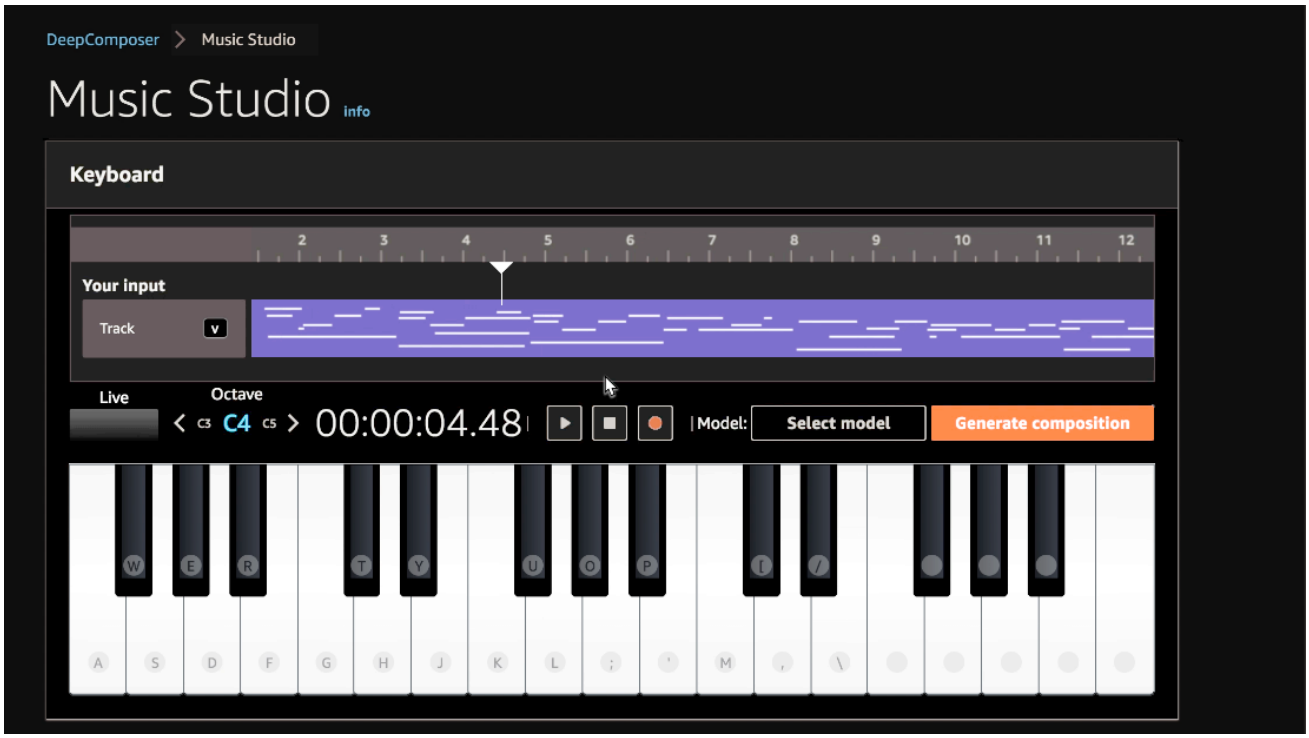

Figure 8. AWS DeepComposer virtual keyboard.

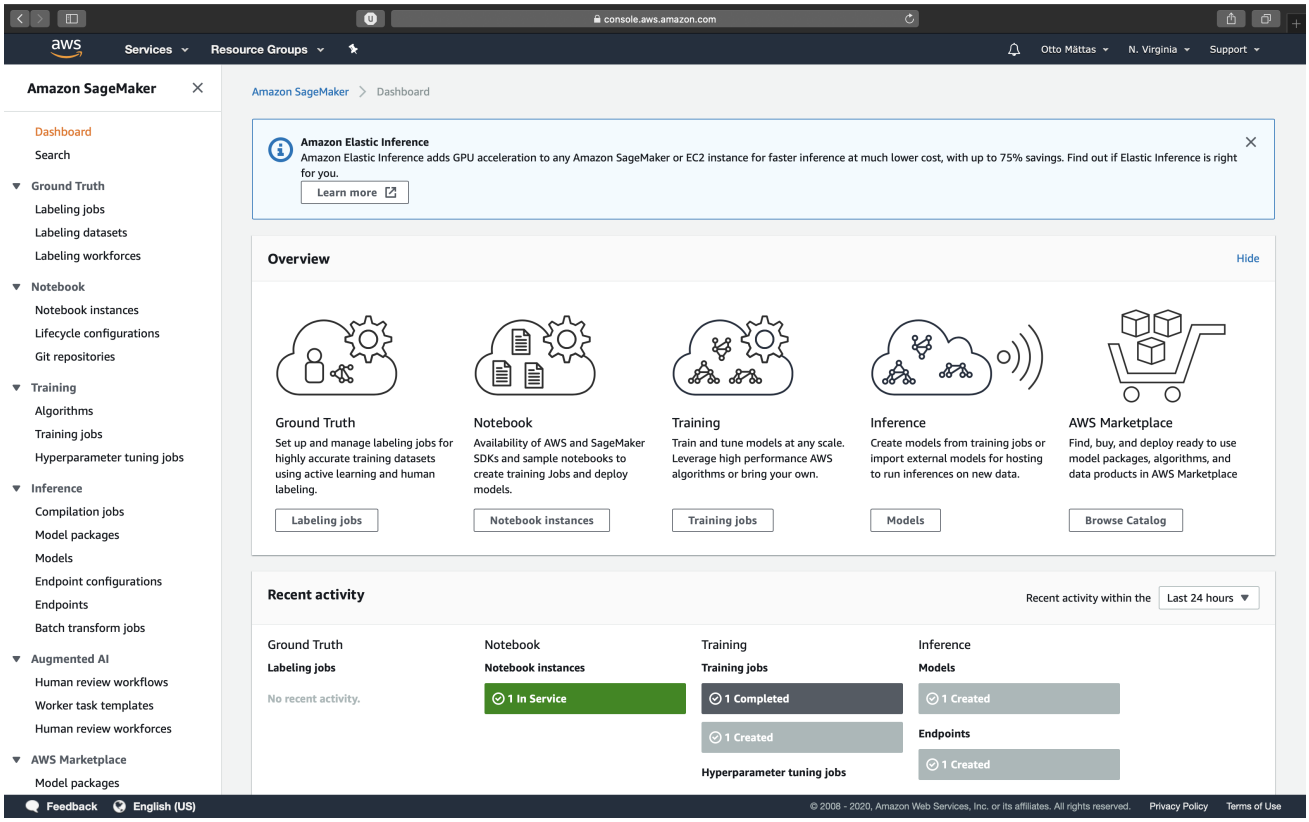

Figure 9. Amazon SageMaker console.

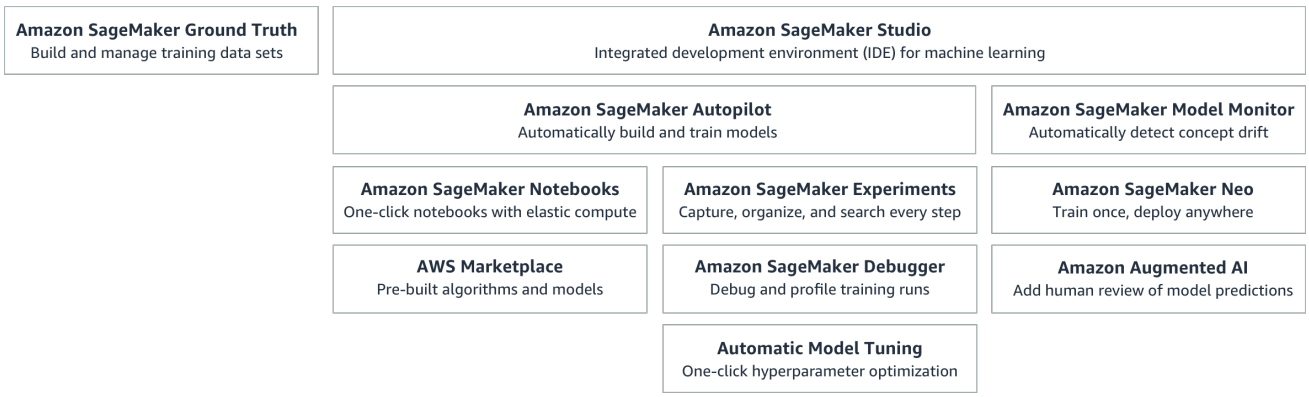

Figure 10. Amazon SageMaker service topology.

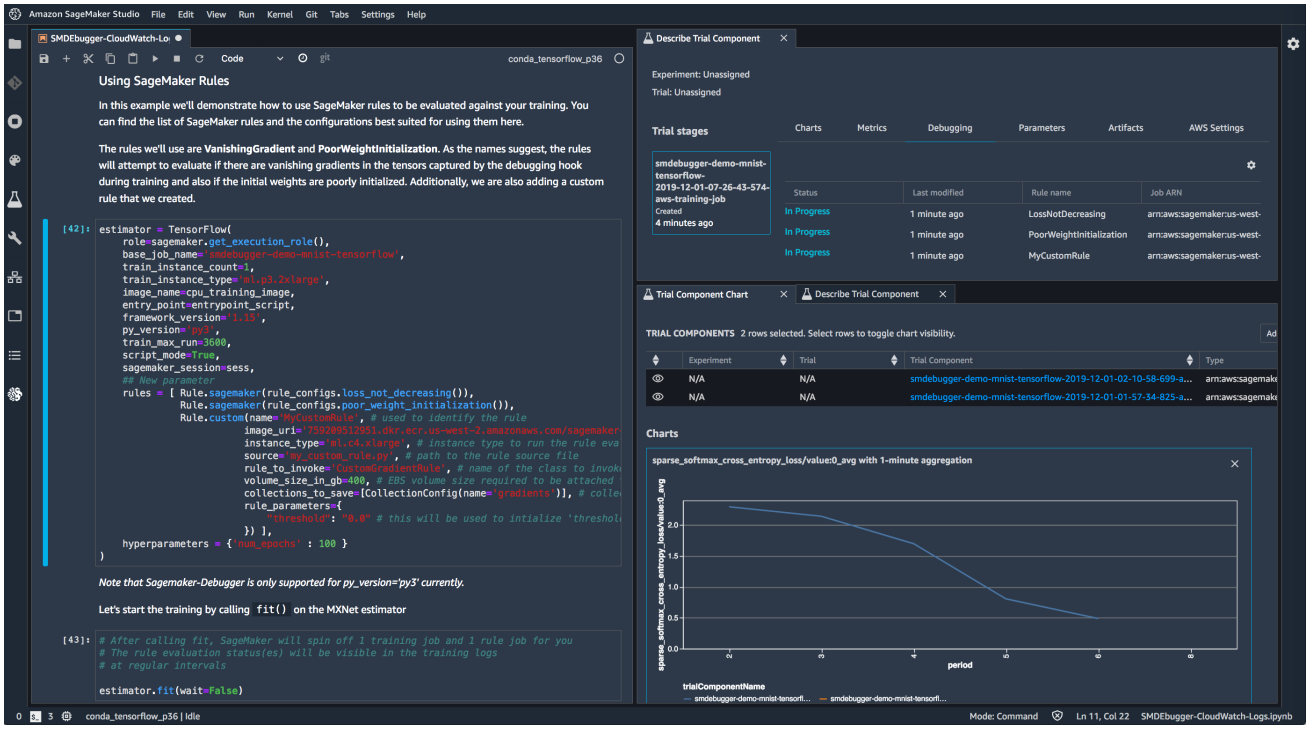

Figure 11. Amazon SageMaker Studio IDE.

EC2 instance	vCPU	Memory (GiB)	Network Bandwidth (Gbps)	EBS Bandwidth (Mbps)
t2.medium	2	4	*Low to Moderate Performance*	-
t3.medium	2	4	Up to 5	-
m4.xlarge	4	16	*High Performance*	750
m5.xlarge	4	16	Up to 10	Up to 4,750

Figure 12. Amazon EC2 instance specs.

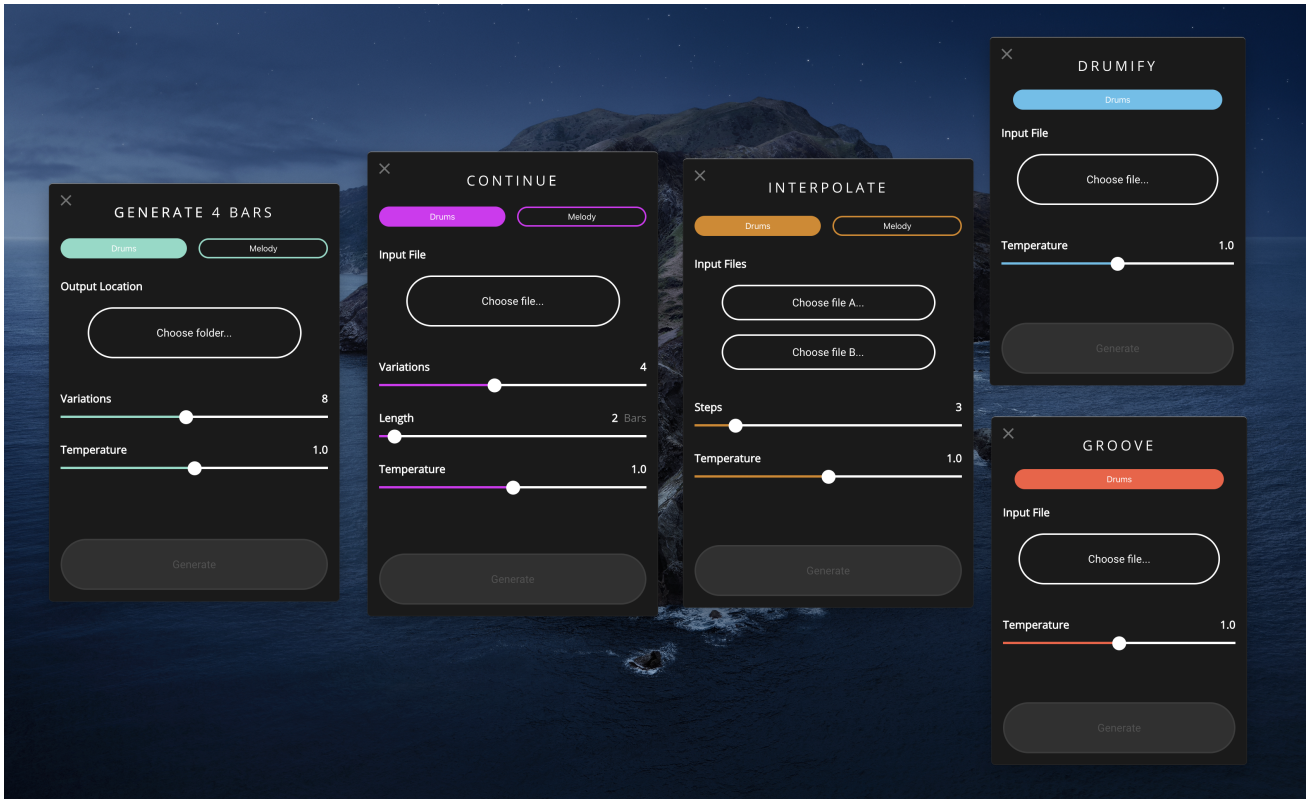

Figure 13. Magenta Studio GUI.