FRIML - Music Generation using Machine Learning

Wiktor Kania, Ewa Kłapcińska, Mateusz Groblewski, Dr Piotr Duch, Dr Tomasz Jaworski

***Abstract*—**The following report aims to describe our song generation application and all the means that were used to create it. The purpose of the application is to enable anyone to generate songs. These songs are generated by artificial intelligence which is based on various models. The paper also describes all the methods that were used to train our models that are later used by the AI.

***Keywords*—**machine learning, LSTM, RNN, music, music generation, midi

I. INTRODUCTION

Throughout our whole life we are surrounded by music - music, which is created by many composers. Music, which - similarly to other fields of culture - is at the same time a chronicle of our world's history. Just like the other areas, it too followed into the world of computer science. Since its debut on Ferranti Mark 1, through creation of the MIDI protocol and until now - the way in which we experience music on our devices has undergone a significant change. Nowadays, digital versions of tracks have completely ruled out the once popular physical storage media.

Looking back, you can not help but wonder - what is next? The moment computers evolved to the point where they allow you not only to enjoy music in high quality, but also help musicians from all around the world create it (effectively replacing synthesizer modules), it should not come as a surprise to think that computers could… create music.

Thanks to the non-stop evolution of artificial intelligence, computers are able to generate music using millions of already existing songs as a base. Computer generated music can be useful to many people. YouTube content creators could easily extend their music library they use in videos as background music. Similarly, beginner game developers who cannot afford to hire a music composer could take advantage of it. Even long time musicians would be able to find some inspiration by listening to computer generated music. It would not mean replacing artists though - we see it as a supplement that can be useful in many fields of content creation. The idea intrigued us so much we decided to create our own AI-trained music generator to gain a better understanding of the subject.

II. LITERATURE ANALYSIS

Before starting work we had to learn more about machine learning. Our team had been focused on building an LSTM (Long Short-Term Memory) network in particular since this is the type of neural network our project is based on.
LSTM is a kind of RNN (Recurrent Neural Network) [1]. The reason why we need this type of neural network is the fact that every music composition has its own theme thanks to which we can not only differentiate the intro and the conclusion of a song, but also recognize the overall style, which keeps the song from being chaotic. When an artist is writing a song, they need to

Wiktor Kania, Lodz University of Technology, Lodz, Poland.
Ewa Kłapcińska, Lodz University of Technology, Lodz, Poland.
Mateusz Groblewski, Lodz University of Technology, Lodz, Poland.
Dr. Piotr Duch (advisor), Lodz University of Technology, Lodz, Poland.
Dr. Tomasz Jaworski (advisor), Lodz University of Technology, Lodz, Poland.

take into account what they have written so far before moving forward - this is the same problem we faced when trying to design a neural network. It needs context and the knowledge of what happened before to compose something that is consistent. This is the type of task RNNs excel at due to the way they are structured - they use a loop of neural networks that could be decomposed to a string of said networks. The example is shown in Fig. 1. The data (our context) is carried from one network to the next [2].

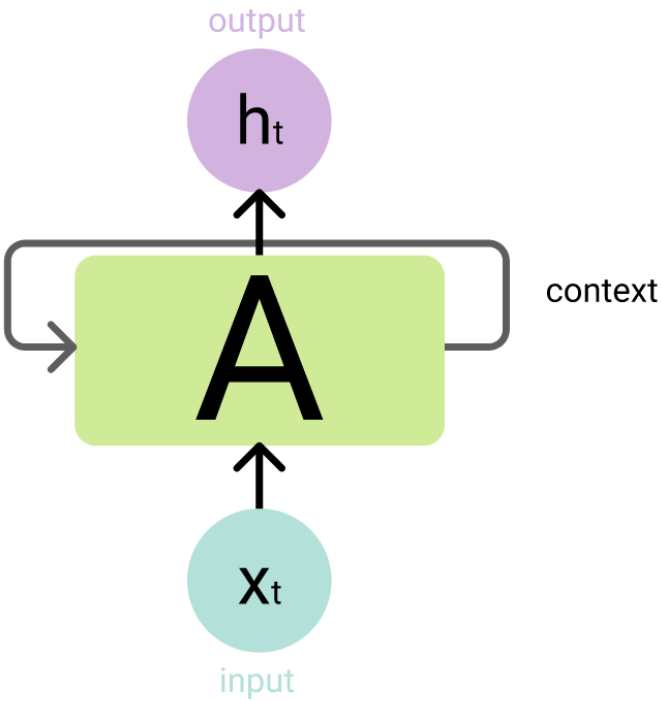

Fig. 1. A simple RNN diagram.

LSTMs are an upgraded version of recurrent networks, which contrary to their predecessor allows for long term dependency. It allows for a context through a longer time period. It was presented by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [3] and has been intensively developed and valued highly by many - this popularity results in it being well-documented. What makes LSTM stand out is the complex structure of the network, which uses the "Forget, Store, Update, Output" method. [4]
The data is filtered through four layers:

- The first one decides what to **forget**, with a result between 0 and 1. 0 means "forget everything" while 1 is "remember everything".
- The second layer decides what new information is to be **kept**.
- The third one **updates** the cell state. Thanks to the previous layers we know what to keep and what to discard, so now what remains is to use that information.
- The last layer produces the **output**.

Thanks to that, new data is generated using updated information while the context is used in the following networks [5].

III. OBJECT, SUBJECT, AND METHODS OF RESEARCH

During our research we were focused on learning how neural networks and music generation work. Machine learning is an important branch in today's market and an interesting topic - interesting enough to encourage us to try to understand it. To accomplish this we developed our own music generator which is the subject of our research.

Our research concerned a few aspects of the project:

- Structure of the network. We had to decide what layers we wanted to build it from. We also had to think about which activation function we would use and how we would calculate the loss. [6]
- Experimenting with datasets. We had to find MIDI sets that would be suitable for machine learning. We also had to check with which parameters they work the best, how good results we got and how we could improve that.
- Optimization. We had to make sure that training models would not use large amounts of memory. This was an important aspect as it was supposed to allow us to learn models with bigger datasets.
- Improving music generation. We could train models and generate music with such little information as note pitches, but if we wanted to get better sounding songs we had to make some improvements. That is why we researched how to add a support of note durations and offsets, and how to transpose input for better harmonization.
- Learning a technology. We had to learn how to use Tensorflow and Keras - common machine learning frameworks - and Music21, a toolkit for musicology that makes supporting musical notations easier. [7-9]

IV. RESULTS

The result of our work is an application that consists of two layers: front-end that users can interact with and back-end that has a structure as presented in Fig. 2.

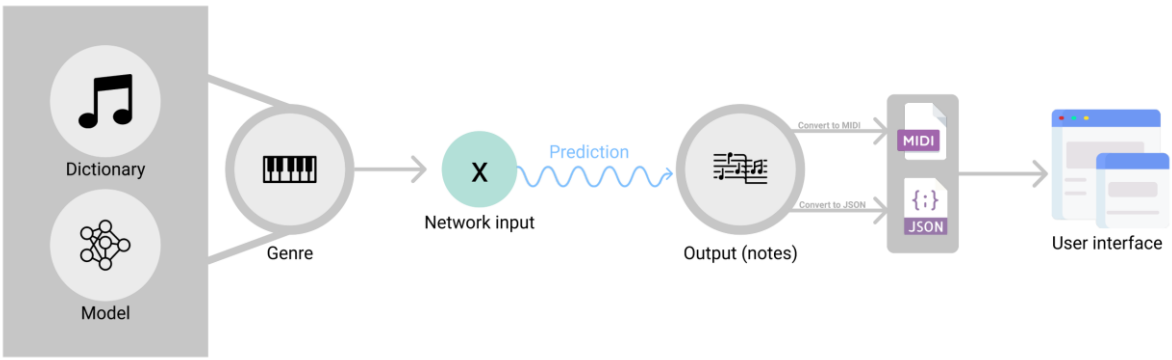

Fig. 2. A diagram showing how the application works.

Our back-end loads a model and a dictionary (used for translating network output to notes) appropriate for the selected musical genre and then proceeds to feed the model sequences of notes. The model then tries to predict the next note in the given sequence. After receiving a new note it is appended to the sequence and the oldest note in the sequence is forgotten effectively making a slightly different new sequence. This process is repeated until a song of desired length is generated. The song is later converted to MIDI (a file that can be downloaded by the user) and JSON (a file used by our web player) [10].

To generate songs we use sequential models that were trained using a few public datasets, such as “The Lakh MIDI Dataset” that consists of over 45,000 songs from different genres [11] and “NES-MDB” containing 5,200 video game tunes. These MIDI files are first transposed to common keys (separate for majors and minors) to make sure that they will be harmonized. Then they are loaded to the model with the structure shown in Fig. 3.

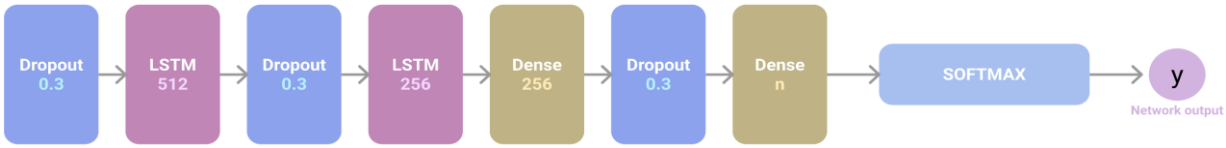

Fig. 3. Structure of our network. Dropout accepts rate as a parameter. LSTM and Dense layers accept units. n value is a number of unique notes.

The model consists of information about notes pitches, durations and offsets. During training we calculate categorical cross-entropy loss which is best suited for our model as it uses a softmax activation function and multi-class classification [12]. The chart of loss for our model is presented in Fig. 4.

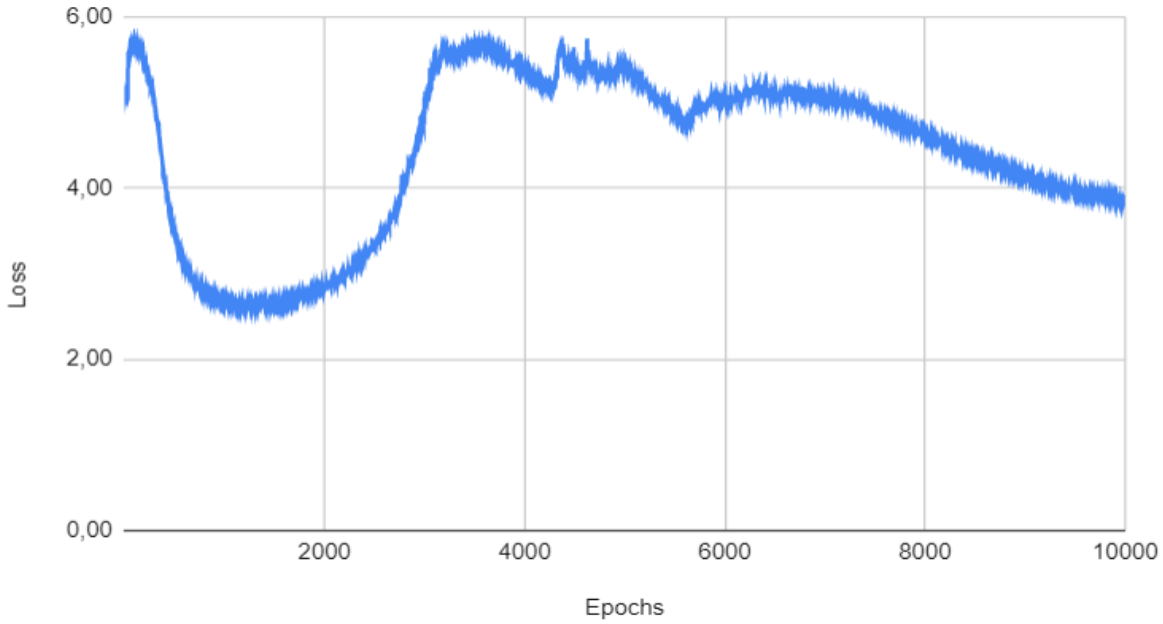

Fig. 4. The loss for the model trained for 10,000 epochs.

The frontend enables users to generate songs. It is divided into four stages.

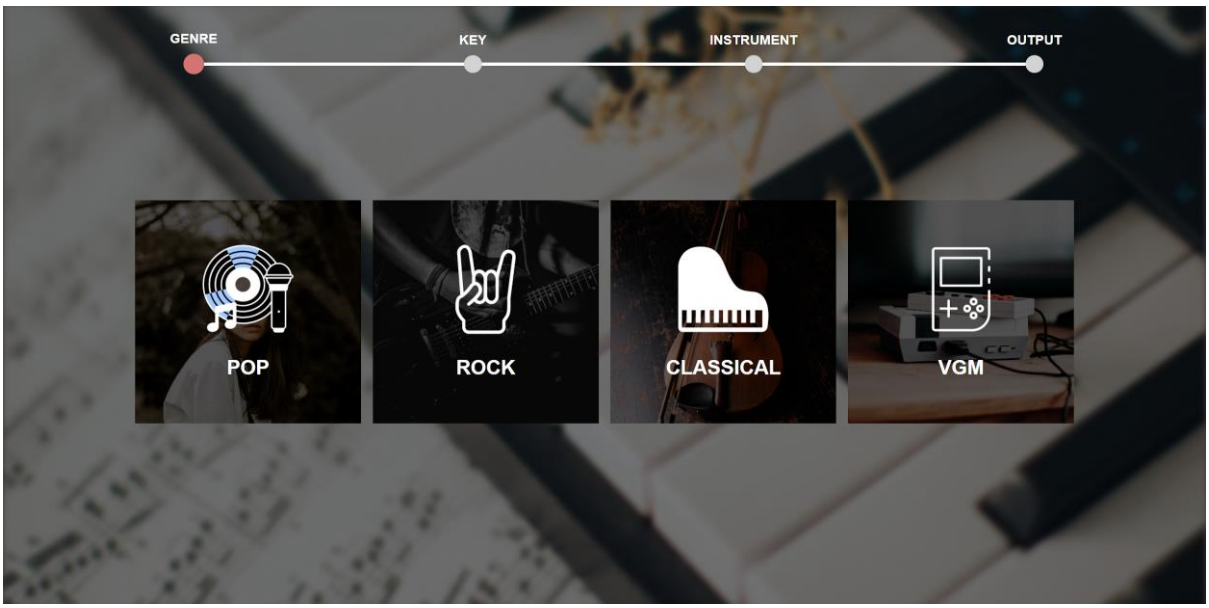

Fig. 5. Genre selection screen.

First, you need to choose the genre on the genre selection screen shown in Fig. 5. The genre defines which trained model will be used to generate the requested song.

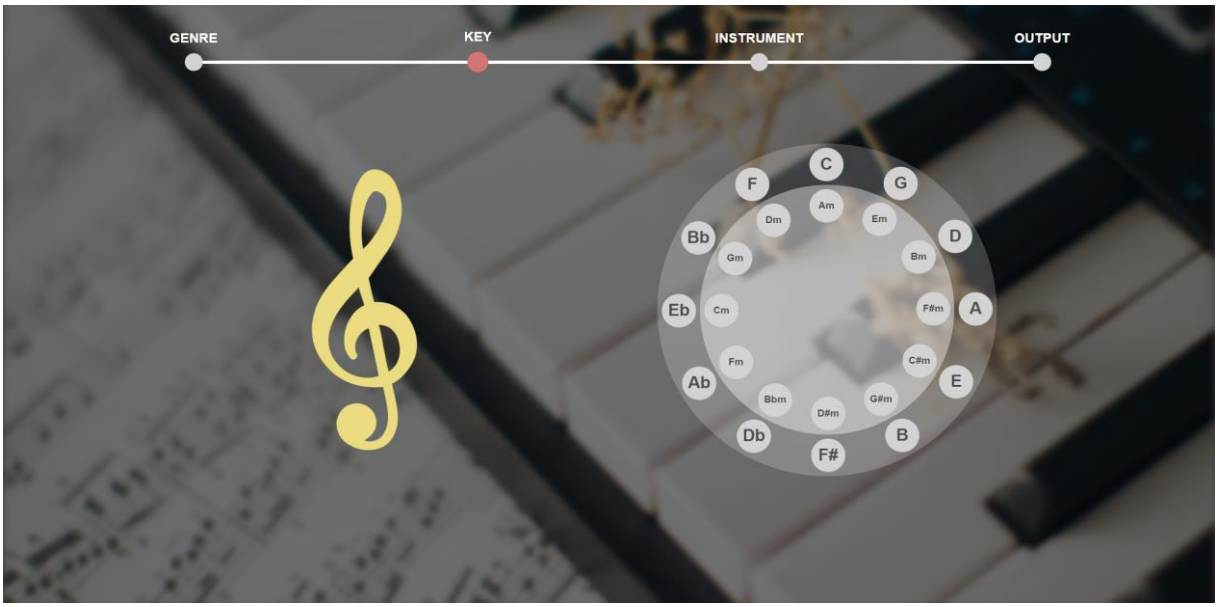

Fig. 6. Key selection screen.

Next, you need to specify the key on the screen presented in Fig. 6. You can choose keys from both minor and major scales. The generated song will later be transposed to the selected key.

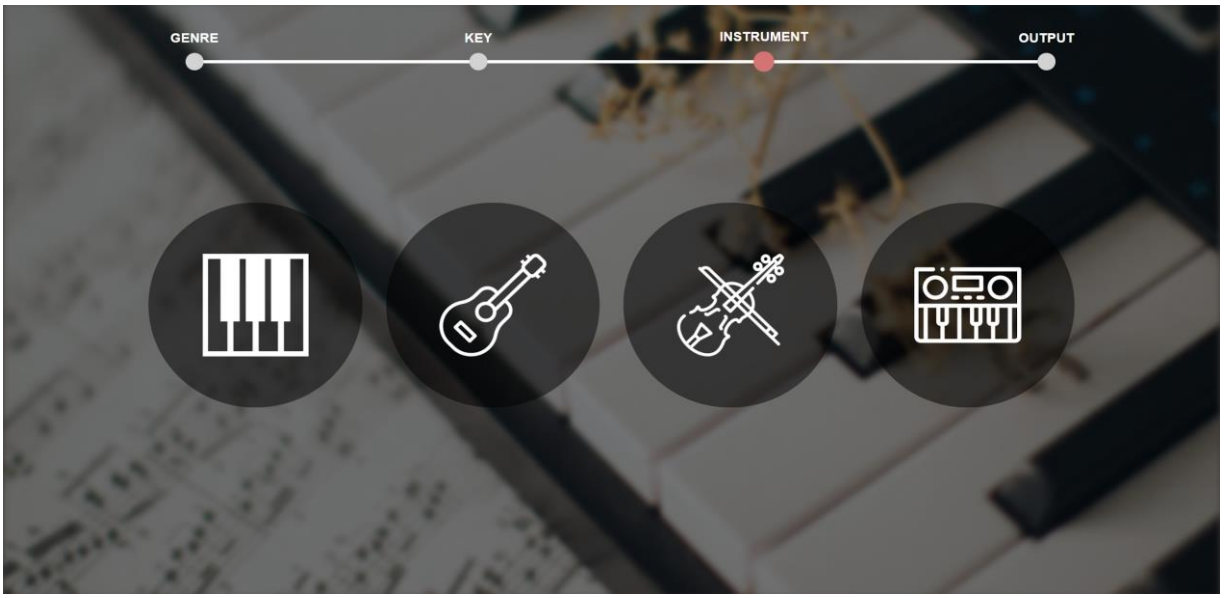

Fig. 7. Instrument selection screen.

Lastly, on the instrument selection screen shown in Fig. 7, you have got to pick the instrument you want the song to use. This setting only changes the output sound.

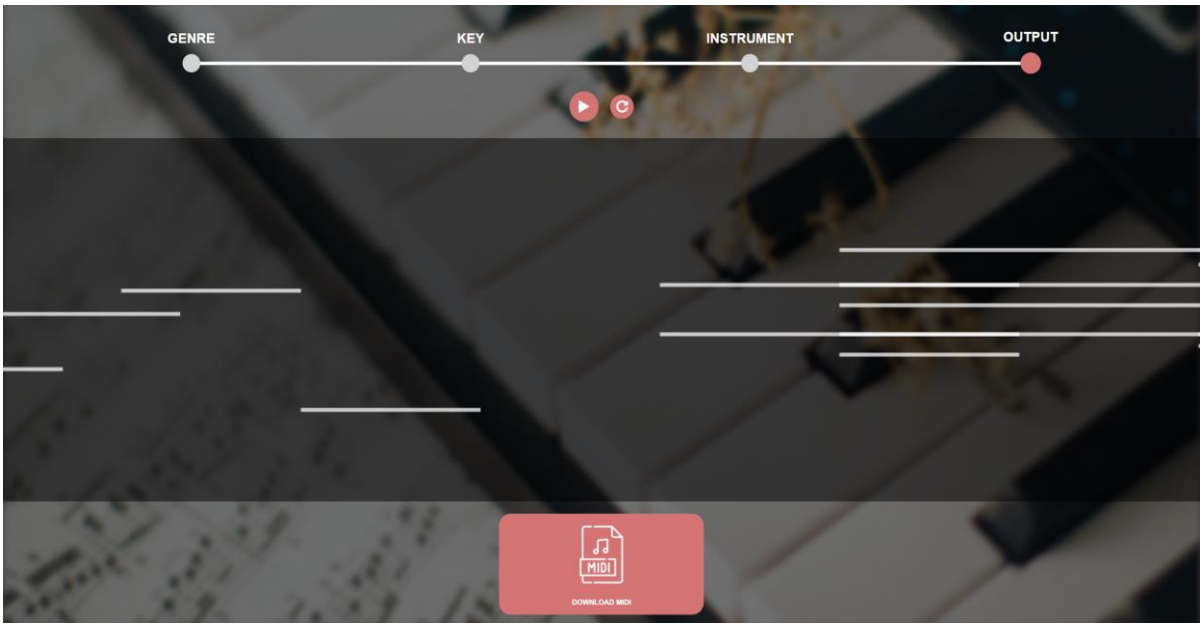

Fig. 8. Output screen.

At the end you will be presented with the generated song. You can play it in your browser and download the generated midi file on your computer. If you choose to play it in your browser a simple visualization in the form of a piano roll will be shown like the one in Fig. 8.

The app utilizes three different web servers. The first one is used to serve static web files (HTML, CSS, Javascript) to the browser. The second one was written in Javascript and is running on Node.js, it acts as an HTTP and Websocket server. The third server was written in Python. It acts as an HTTP server and has access to Tensorflow generated models. When the Python server receives an HTTP request it loads the needed model and starts generating a new song. The newly generated song is then sent back to the Node.js server. The server written in Javascript acts as a bridge between front-end (the browser) and back-end (the Python server). Clients are connected to the Node.js server through the Websocket protocol where they are queued. The Node.js server is responsible for managing the queue and sending all requests to the Python server when possible (the Python server is capable of executing one request at a time) as well as sending back the data generated by the Python server to the users. Fig. 9 presents the general idea:

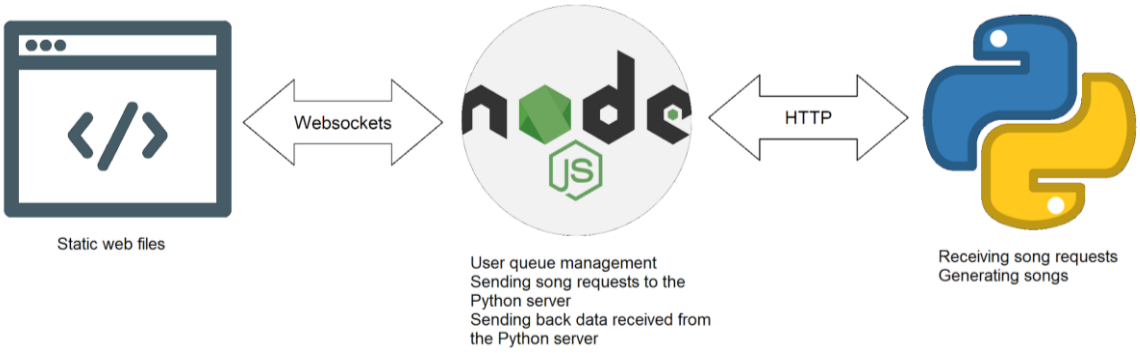

Fig. 9. An image describing the relationships between the servers.

This solution ensures that multiple clients can use the app at once without the Python server being overwhelmed (song generation is a demanding task).

To test the effectiveness of our project we decided to conduct a survey. We played a few samples to a small group of people and we asked them if it was generated by our program or composed by a human. Results are presented in Table 1.

Table 1. Results of a survey. Answers where majority guessed wrong are marked in red.

	Number of people who answered it is composed by a human	**Number of people who answered it is generated by FriML**	**The answer**
Sample 1	4	7	Human
Sample 2	3	8	FriML
Sample 3	3	8	FriML
Sample 4	8	3	Human
Sample 5	5	4	FriML

V. CONCLUSIONS

Nowadays, more and more developers opt to use machine learning to develop their applications. This method not only shortens the development time, but also allows us to create software that is near impossible to do with conventional methods.

Thanks to machine learning we were able to develop an application for music generation. It can be used by anyone - whether you are a musician looking for inspiration, a content creator in need of songs for your videos or you are just curious and want to see what today's technology can achieve - FriML has got you covered.

REFERENCES

[1] Douglas Eck, Jürgen Schmidhuber, *A First Look at Music Composition using LSTM Recurrent Neural Networks*, 2002

[2] Sanidhya Mangal, Rahul Modak, Poorva Joshi, *LSTM Based Music Generation System*, International Advanced Research Journal in Science, Engineering and Technology, Volume 6, Issue 5, May 2019

[3] Sepp Hochreiter, Jürgen Schmidhuber, *Long Short-term Memory*, Neural Computation, Volume 9, Issue 8, November 1997

[4] Ava Soleimany, *Recurrent Neural Networks | MIT 6.S191*, MIT Introduction to Deep Learning 6.S191, January 2020

[5] Christopher Olah, *Understanding LSTM Networks*, colah's blog, August 27, 2015. Available on: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning (Adaptive Computation and Machine Learning series)*, The MIT Press, November 2016

[7] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, TensorFlow: *Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[8] Nikhil Ketkar, *Introduction to Keras*, In book: Deep Learning with Python, Springer, 2015

[9] Michael Scott Cuthbert, Christopher T. Ariza, *music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data*, Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010), August 2010

[10] Deborah Nolan, Duncan Temple Lang, *JavaScript Object Notation*, In book: XML and Web Technologies for Data Sciences with R, Springer, November 2014

[11] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, Yi-Hsuan Yang, *Lakh Pianoroll Dataset*, Music and AI Lab, Research Center for IT Innovation, Academia Sinica

[12] Raúl Gómez Bruballa, *Understanding Categorical Cross-Entropy Loss, Binary Cross-Entropy Loss, Softmax Loss, Logistic Loss, Focal Loss and all those confusing names*, Raúl Gómez blog, May 23, 2018. Available on: https://gombru.github.io/2018/05/23/cross_entropy_loss/