Building a Sketch-Based Artificial Intelligence for Interactive Music Generation

Killian Barrere[1], Manuel Bouillon[2], Marcus Liwicki[2], and Rolf Ingold[2]

[1] École Normale Supérieure de Rennes, Univ Rennes, France
killian.barrere@ens-rennes.fr
[2] DIVA Group, University of Fribourg, Switzerland
firstname.lastname@unifr.ch

Abstract. DrAwME is a web app that enables to draw music, which has been developed in the context of the European H2020 project iMuSciCA. This project aims at developing an active learning platform for the teaching of sciences through art, and especially music. This report presents how we developed an artificial intelligence that can co-create music from sketches with the user in DrAwME.
Music generation has greatly improved since the first research started, mostly with the recent deep learning methods. In this report, we propose an original approach to generate music using sketches representing sounds. We use a deep neural network capable to continue sketches of the user to co-create a musical drawing. We generate new datasets of musical sketches and train new models to produce drawings that sounds musical. We also discuss the performance and results of the different generation methods.

Keywords: music generation · handwriting generation · deep learning · artificial intelligence · co-creation · interactive generation · musical sketch

1 Introduction

Music generation is still an open problem that is motivating a lot of research. Some systems are even capable of producing music in cooperation with the users. Meanwhile, *DrAwME* is a web application that let users draw music. The drawing produce sounds according to a temporal axis and an axis for the frequency of the note. It is part of the European *H2020* project *iMuSciCA* that aims to teach students science through arts. While the main goal of *DrAwME* is to draw music, a lot of users also like to simply draw shapes that are not intended to produce music like houses, hearts, etc. Meanwhile, the field of handwriting generation is also producing very good results [13].

With *DrAwME*'s ability to convert drawings into sounds, we wanted to try to generate drawings that produce nice sounds. The main objective is to build an Artificial Intelligence which is able to generate music. Another objective that is important is to enhance the user experience. As the users are mainly young students, it makes sense to build an Artificial Intelligence that is able to co-create with them. We want the user to start drawing something and then ask

the Artificial Intelligence to continue the drawing. Also, because a lot of students like to draw simple skeches, it also makes sense to build a similar kind of Artificial Intelligence that would complete sketches.

This report will introduce how we built such Artificial Intelligence. This work focus on three main ideas: Sketch Generation, Music Generation and the interactivity of the co-creation process. In particular, we will introduce a new model that is able to generate music using an original approach: a musical sketch dataset converted from a dataset of real music.

The remainder of this report is organized as follows. We will start by presenting a short history of the different fields wthat are related to this work in section 2. We will then focus in details on *sketch-rnn* [13] in section 3, because we chose to reuse that system in our Artificial Intelligence. Later, we present every contribution of the report, including how we created the Artificial Intelligence in section 4, and finishing with a discussion of the different results we achieved in section 5.

2 Related Works

2.1 Music Generation

Music generation has been sparking people's interest for a long time. While music was first generated with some primitive solutions, it really started to get interesting with the possibilities computers brought. The term music generation could be used in several fields. First the generation of a piece of audio using already existing instruments and just telling which note is going to be played. It could also refer the action of generating sounds to approach the sounds of already existing instruments [8] or the process of crating new instruments. In this report, we will be focusing on the first kind of music generation we mentionned.

This field has greatly improved thanks to the introduction of Deep Neural Networks. While simple Neural Networks are enough to predict the next note to be played, their inputs are independent from any outputs, and are not able to acquire informations from the past.

Recurrent Neural Networks (RNNs) solves this problem by having some dependance between inputs and outputs. Therefore, they are able to memorize informations and reuse it for a future computation. They are used a lot when the inputs are in a sequence format, like it is the case for text and every input that have something to do with a time representation. They were first widely used, but despite the introduction of new networks, they are still used today [14] [21]. However for pieces of information buried very deep in the history, they struggle to memorize them.

Long Short-Term Memory (LSTM) [16] have been introduced to solve this problem. LSTM has functions doing each a unique task. The differents functions are doing the task of receiving an input, sending an output and also choose what information to keep and which one to lose. They are called respectively input, output and forget gates. By having the ability to loose some information the

network is able to keep only useful information, and therefore useful information from a further distance in the history. Thanks to that, the process of learning and back propagating the gradients is less impacted by the vanishing gradient problem. The introduction of LSTM in the field of music generation greatly helped to improve the musical structures and increased the possibility of music generation [7].

The introduction of Reinforcement Learning also brought some new possibilities to the field [20]. The main differences with the other deep learning methods lie in the way the training is achieved. While others system are trained with a specific target for every sound, in the case of Reinforcement Learning, we only give a reward base on how good we think the generation is. The system then learn by itself to go to an objective that fits what we want. Severals aproaches [11] [18], use Reinforcement Learning in their systems.

Generative Adversarial Networks (GAN) [9] were specifically created to generate data that seems to be created by humans. They work with two networks. The first one has to fulfill the task of generating data, while the second one has to identify generated data from the real ones. They are trained in a process that both are getting better and better. The main objective is to train the first network to generate data that are able to fool the second network. While GANs were mostly used for image generation, they have been also tried for music generation [22] [31] [6] [11].

Autoencoders are used to compress the information. They are able to learn a representation of a high dimensional set of data. This representation uses a lower number of dimensions and is supposed to contain useful features that another system could use to learn something. They are doing very well in the task of generating data. They are now used in almost every music generation system [28] [15].

Some parts of deep architectures useful for handwriting recognition and generation have been re-used to generate musics [5] [26]. Those systems reuse specific features that are using the structures of word and sentences.

Meanwhile, *DrAwME* has been developed in the context of *iMuSciCA* which is an european project that aim to teach students in highschool science, music and arts. *DrAwME* is a *JavaScript* application that provides an interactive interface converting drawings into sounds. The horizontal axis in this interface represents the time, while the vertical axis represents the frequency of the sound to be played and hence the notes. We can choose the waveform of the sound, which is basically like choosing the instrument.

The ability of *DrAwME* to convert drawings into sounds is opening the possibility to use drawing generation and convert it to sounds to have a new generator of sounds.

2.2 Handwriting Generation

Handwriting Generation is a task that consists in creating data that an human could have been drawing with its hand and a pen, a brush, etc. First tries in the

field used Hidden Markov Models to learn and generate sequences of human's drawn shapes [27].

Later, some systems have been developed to complete the task of generating hand's drawings from a picture. Paul the robot [29] is one of the system that are able to produce an hand drawing from images of a camera. For the same task, Reinforcement Learning has also been used [30]. The results are drawings of a good quality and looks to be human.

RNN and Gaussian Mixture Models have then been used in most of the systems following the works achieved by Graves [10]. As for the topic of music generation, LSTMs have been preferred for their possibilities to learn longer-range context. They have been used [32] to recognize and also generates chinese characters. *Sketch-RNN* [13] uses LSTM and Gaussian Mixture Models to generate sketches. It will be presented in more details in section 3.

2.3 Co-creation

This section will try to give a quick history of how systems are used to help users in a cooperative fashion during their creation process. This is one of the main objectives of our Artificial Intelligence.

ChordRipple [17] is one system that is able to help musicians to select chords. They start by giving the chords they want to play, and then the system is able to generate additional chords, or even modify the chords that the user choose. According to the article, it helped musician during their creation process.

Using a Recurent Neural Network, Hadjeres and Nielsen are able to produce a system capable to learn a musical style and then to generate music that match this style while respecting constraints given by the users [14].

With Generative Adversarial Networks, it is possible to create networks capable or generating music that sounds nice and that also could be used to play music in cooperation with the user [6].

The Artificial Intelligence Duet [1] let you play a musical duet with an Artificial Intelligence. The user can start playing some notes and after a few moments, the system start to play notes in response to what the user played.

3 Sketch-RNN

This section is a continuation of section 2. We introduce the *sketch-rnn* [13] system in detail because it is the model we use to produce every result in the next sections.

We will speak about the different data formats (In this case, data are sketches.) in section 3.1. Then we explain in section 3.2 the architecture of the system followed by a detailed look at the loss function used in section 3.3. We then continue by explaining how the number of classes affect the performance of the network in section 3.4. Finally, we briefly show some use-cases of *sketch-rnn* in section 3.5.

For additional informations, we highly recommend reading the original paper [13].

3.1 Sketch formats

In this section we explain the two data representations used in *sketch-rnn*.

The first one is representing each point of the sketch with three values [10]. The first and second values represent the x and y relative coordinates with the previous point. The third value is a boolean that is telling if the pen is touching the paper, or equally if a stroke is being drawn.

Following the principle of the first format, a second one has been proposed [13]. It is using five values rather than three. The first two are still the x and y relatives coordinates. The last three values are labelled p_1, p_2 and p_3 and represent various informations about the pen state. p_1 tells if the pen is currently touching the paper and if a stroke will be drawn connecting the current point to the next point. p_2 indicates that the pen will be lifted up after drawing the current stroke and that the next stroke will not be drawn. The next five values will then say where to replace the pen to start the drawing of a new stroke. Finally, p_3 indicates that after drawing the current stroke, the sketch will be finished. The second format is the one used to represent every point of the sequence of the sketch in *sketch-rnn* [13].

3.2 Description of the models

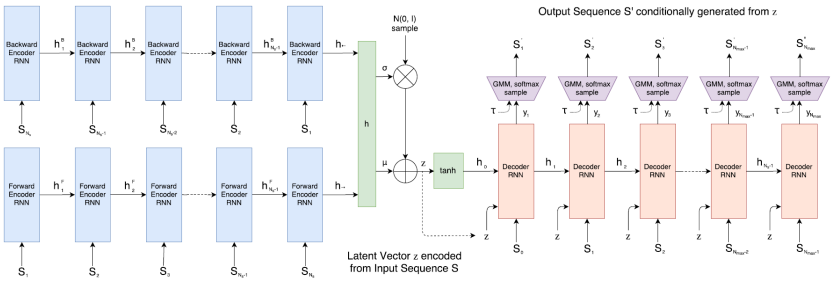

Fig. 1. Picture representing the conditional sketch-rnn model (taken from [13]).

This section presents the two different architectures of *sketch-rnn* introduced by Ha and Eck [13].

Variational Auto Encoder The first one is called the conditional model and works according to the process of Variational Autoencoder [19]. Its main goal is, given a complete sketch drawn by the user, to create sketches that look like the input while keeping features learned during the training. Figure 1 shows a representation of the model.

The *sketch-rnn* model following the Variational Auto Encoder structure is starting with an *Encoder*. This first layer is composed of a Bidirectional Recurrent Network [25] followed by Fully Connected Layers. A Bidirectional Recurrent

Network is the combination of two RNN, one for each direction. In the context of sketch drawing, the first direction represents the drawing of the sketch from the start to the end. The second direction represents the drawing in the reverse order. This *Encoder* is mainly in charge of creating a latent vector that is representing the input with some useful features. This latent vector is more or less randomly modified (depending on a parameter called the temperature τ) to produce a wider range of prediction later.

LSTM is described as the best RNN model for the encoder [13], but others still seem relevant to this task.

The second part of the network is called the *Decoder*. It is an RNN that works with the sequence of the sketch given in the drawing direction. It takes as inputs the same sketch given to the encoder which is concatenated along with the latent vector firstly generated by the *Encoder*. The latent vector is representing the input sketch and also some features about it that are supposed to help the *Decoder* to generates sketches.

For each stroke of the sketch it is outputting parameters to generate the relative coordinate of the next point, that is also representing the next stroke and a pen state. They are parameters for a Gaussian Mixture Model more precisely. Briefly, it is producing means, standard deviations and correlation coefficients for a Bivariate Normal distribution (For more information, please read [13].). This distribution is then able to generate points in two dimensions.

The best models described are first Hyper RNNs [12] because of their ability to change the shape of the network that makes them very competitive when it comes to generation. LSTMs are still producing good results and can be used without loosing too much quality [13].

Decoder Only The second architecture, which is named the unconditional model [13], only use the *Decoder* to generate sketches. The latent vector generated in the conditional mode is replaced by a randomly generated vector following the normal distribution. Strictly speaks, it is not a decoder and only a regular RNN.

In the process of completing sketches given by the users, we will only be using the unconditional mode. To do that, we first generate a random latent vector, and start by feeding the *Decoder* RNN with the input while ignoring the output. Then by using the output of the *Decoder* to generate next points, we can continue to feed the *Decoder* and generate new values.

3.3 Loss Function

In *sketch-rnn*, the loss function used is a weighted sum of two terms. Without detailing each term, the whole equation for the loss look like:

$$Loss = L_R + w_{KL} * L_{KL} \tag{1}$$

The first term is called the reconstruction cost (L_R in equation (1)). It is used in both conditional and unconditional models. It is representing the error between the parameters produced by the decoder and the given input sketch.

The second term is called the Kullback Leibler Divergence cost (L_{KL} in equation (1)). It is telling how close the encoder is of producing a random latent vector. This term is only used with the conditional model.

w_{KL} then represent the weight attributed to the Kullback Leibler Divergence cost (L_{KL}). L_{KL} is only used while using the conditional model (introduced in section 3.2). The goal of w_{KL} is to control by how much the *Encoder* can produce random latent vectors. When $w_{KL} \to 0$, the model is getting closer and closer to a model using only the *Decoder*.

Every detail on how each term is computed can be found in the original articles [13] [10].

3.4 Number of Class

As the task of *sketch-rnn* is more about sketch generation than predicting the class of the sketch the user drawn, the number of class that the model is train to generate impact more the model. Usually each model of *sketch-rnn* is trained on a single class. For instance, a model could be trained only with drawings of cat and will be able to generate drawings of cat. By using a single class, the network usually generates nice and coherent sketches. Training a model on multiple class is not that easy. The main reason is that the model has to learn to be coherent in the drawing it is generating. While the model is producing mostly coherent sketch for a small number of classes (less than 10) , it is producing strange results with a large number of classes (more than 10). These drawings more or less look like a mean of every sketch.

3.5 Results obtained

This section will show some use-cases of *sketch-rnn*. Again, here we will briefly introduce some possibilities. More possibility are shown in the original paper [13].

Conditional Reconstruction Using an input from a user, the model is able to produce similar looking sketches. In order to do that, we first need to feed the *Encoder* (introduced in section 3.2) with the input of the user. Then we add to the extracted features some randomness and feed everything to the *Decoder*. By using different (thanks to the random part) but still close latent vectors, the *Decoder* is able to produce similar looking sketches. By modifying the temperature τ (which is in charge of how big the random changes are) , the network can produce from very strict looking sketches to completely different sketches.

Predicting Endings of Sketches One other possibility is to use the unconditional model (which is introduced in section 3.2) to predict the ending of an incomplete sketch started by the user. We start by generating a random latent vector from the normal distribution and feed it concatenated with what the user started to draw to the *Decoder*. We then start generating points when the last

point of the input sketch has been feed, and reuse these points to draw the next strokes on screen and also to continue feeding the *Decoder*.

This section explained quickly what was *sketch-rnn*. More informations about *sketch-rnn* can be found in the original paper [13]. We will be using this model in the section to come. We will focus on the prediction of incomplete sketches for the rest of the paper as this is something very close to what we want.

4 Contribution

This section is introducing the different contributions I brought to *DrAwME* which is a *JavaScript* application converting user's drawing into sounds. Presented in section 2.1, *sketch-rnn* and the others contributions that are not directly linked to these two. The main objective was to integrate *sketch-rnn* within *DrAwME* and the different contributions are always connected to this objective. The contributions will be introduced in three main axes. The first one is focusing on the process of co-creation and the user experience. It is presented in section 4.1. Then section 4.2 introduces the second axes speaking about sketch generation and models. Finally, section 4.3 explains our contributions to the generation of music and more precisely, musical sketch generation.

4.1 Co-creation in DrAwME

In this section, we mainly discuss about how we integrated *sketch-rnn* within *DrAwME* (introduced in sections 3 and 2.1 respectively) before speaking of the different decisions to improve the process of co-creation and the user experience.

To add the implementation of *sketch-rnn* to the *DrAwME* application, we used the implementation in *JavaScript* of the system that was coded by the authors of the original article [13]. They provided a script predicting the end of the users' incomplete sketches which are close to the task we desire to fulfill. For that task we used the model in an unconditional mode which means that *sketch-rnn* is mostly working with only its *Decoder* (this model is introduced in section 3.2).

We reused that code so that it is taking the drawings of the user in *DrAwME* as the input of the model and redirected the output of *sketch-rnn* to the application's screen. In both processes, it was required to convert the data to the good datatypes. In addition we added a "magical-wand" button that is mostly telling the Artificial Intelligence to complete the drawing of the user.

We think that completing an user incomplete sketch directly toward its end is not creating a good interaction with the user. We imagined the process of co-creation to looks like a reciprocation between letting the user drawing and giving the pen to *sketch-rnn*. We then decided to modify the code so that it is doing something that show more interactions.

Because the implementation of *sketch-rnn* is working in the unconditional mode, the main component of the model is an RNN (introduced in section 2.1). RNNs are working step by step and produce one output based on one input at

every step. That said, we are then able to produce the points one by one until we are satisfied of the produced drawing.

Therefore, we decided to feed the model with what is already drawn on screen and let it produces points until the model decided to lift up the pen. We then print the result on the screen. The user can decide at any time to push the "magical wand button" to tell the Artificial Intelligence to draw one stroke based on the drawing of the user, leading toward its completion.

We therefore added a button in the user interface that allow you to choose the number of strokes generated at each time step. We found it useful if we want to swap to a model predicting a different class that requires to draw more strokes.

4.2 Sketch Generation

In this part, we discuss about the contribution regarding sketch generation which are based on *sketch-rnn*.

After the integration of *sketch-rnn* within *DrAwME*, which was using one of their pre-trained mode [13], we decided to train a model by ourselves one model. For that we used the *QuickDraw* [2] dataset (See section 5.1). We trained the model with a class composed of only sketches of cats. We trained the model with its native implementation in *Tensorflow* [3].

One important contribution is also the portability of the *sketch-rnn* model from *Tensorflow* to *PyTorch* [23]. We decided to implement it in *PyTorch* mainly because it is allowing us to have a lot more possibility than the original implementation with *Tensorflow*. An important reason is the lack of portability of the *Tensorflow* implementation of *sketch-rnn* with GPU's drivers, whereas in *PyTorch* this is not an issue. Therefore, we could gain a huge speed factor during the training of models (half a day with a GPU while more than a week when using CPU). We chose to implement it using *DeepDIVA* [4], a framework that allow the easy reproduction of experiments. We implemented both the conditional and unconditional model.

Because of a lack of time, we did not manage to finish the implementation. Therefore, we could not use it to train more models. We believe that the implementation is almost finished and just require a little more time to debug it.

4.3 Musical Sketch Generation

That section is showing the different points that have been developed including the conversion of a music dataset to a sketch dataset and how the musical model works inside the *DrAwME* web application.

As *DrAwME* is (at the moment) only capable to play sounds of a maximum duration of 4 seconds, we first decided that for every example of the training, validation and training set, the maximum duration would be 5 seconds. We also decided to use a single instrument for each example. However, in the different

sets, there is not a label, color or other indication saying which instrument is drawn. In *DrAwME*, the color influences the shape of the signal being played, and that is resulting in different instruments. For the conversion we decided likes in the application that the horizontal axis is representing the time and the vertical axis the frequency of the notes. The notes are added to the sketch in a time-ordered fashion, meaning that we first add the first notes to be played. We used the same scale for every note, which mean that between two consecutives notes, there is always the same number of pixels. Lastly, there is a maximum of range. The notes go from C_{-2} to C_8, while *DrAwME* only uses 2 ranges.

For the integration within *DrAwME*, there is only a few change in comparison to section 4.2. We decided that by default it is more interesting for the process of co-creation to generates more than one note at each step. We found that generating 10 strokes (notes) is a good choice. This number can be set by using the user interface.

5 Experimental Validation

This section presents the different experiments and the results obtained. An important point we would like to emphasize is the difficulty to have quantitative results. While a classification task produces results that can be quantified, leading to accuracy score, loss, etc., the task of generation cannot rely on those scores and then requires human feedback. While asking different people to give their advices on our result mixed with a baseline is a good solution, it is not so easy to find testers and could be expensive. We will mainly give our personal impressions while keeping a comparison with results belonging to the state of the art.

Firstly, section 5.1 introduces the different datasets used for the experiments. We present both sketch datasets and music datasets in section 5.1. The section 5.2 explains the different results of how interactive our approach is. Then we discuss about the results of sketch generation in section 5.3. To finish, we explain the results we obtain by training a model using a custom dataset of musical sketch (music converted to sketch) in section 5.4.

5.1 Datasets

Sketch Datasets To train models on the task of sketch generation, we used the *QuickDraw* dataset [2]. It is composed of more than a hundred classes of sketches. We used one version of the dataset which provides 70K examples for training, 2.5K for validation and 2.5K for the testing process. This is the same dataset as the one used by *sketch-rnn* [13].

Music Datasets In order to train the model to generate music, we started to use the *Lakh Pianoroll Dataset* [6] [24]. This dataset contains 174154 piano-rolls of one or more instruments.

Each piano-roll have a quit long duration of 2 to 3 minutes in average. This is far longer than the duration that *DrAwME* can play (see section 2.1). Currently, *DrAwME* is able to convert drawing to song of a maximum of 4 seconds. We then decided to split each pianoroll first by instrument and then by some short period of 5 seconds which are a bit more than what *DrAwME* is able to play. We also removed the small part of 5 seconds without a single note inside.

5.2 Interactive Creation

This section will mostly discuss about the result of the integration of *sketch-rnn* within *DrAwME*. The first thing that need to be said is how much the system requires resources. The model is taking from our observations two seconds to be loaded in a web browser, which is taking a very long time compared to loading *DrAwME* without the system. Because the application is working locally, the model uses a lot of resources, and for low configuration, it may result in failure while loading the model or even later when using it. One solution will be to make servers (that may use GPU) running the models and by transferring the different strokes, the resource usage of the client would be lighter. But that also could result in an another problem if the servers are overloaded with demands. The same thing happens if you want to change the model dynamically.

Speaking of how good the user experience is, we think that the prediction stroke by stroke or more strokes at any step (as you can manually change that parameter) is the best solution. However, for a real conclusion, we will have to ask different user what they think of the different possibility to generate completion of sketches.

5.3 Sketch Generation

In this section, we will compare the results of the different models we tested. As a comparison basis we will use the results from *sketch-rnn* [13]. All those results are on the task of predicting the ending of a sketch. To do that, we used a script written in *JavaScript* that let the user start a drawing and then draw the predicted ending of the drawing.

In our experiment, we first used a pre-trained model. As this model is one provided by *sketch-rnn*, the results are the same as those presented in the article [13]. All we can say is that globaly it is well predicting the ending of a drawing. Sometimes the prediction could seem a bit weird, but generally, the predicted ending are coherent with the drawings.

We trained a new model for the task of generating cat drawings. For that model, we used an unconditional model with a *Decoder* being an LSTM of size 1024. We trained it using the *Tensorflow* implementation provided by *sketch-rnn* [13]. The model has been trained for a week on a 16 cores CPU.

The model we trained showed good and coherent results when it comes to the task of predicting the end of a drawing (figure 2 is showing some results). However the different predictions are more often not matching the original drawings. Sometimes for instance, the model we trained predicts that the beginning

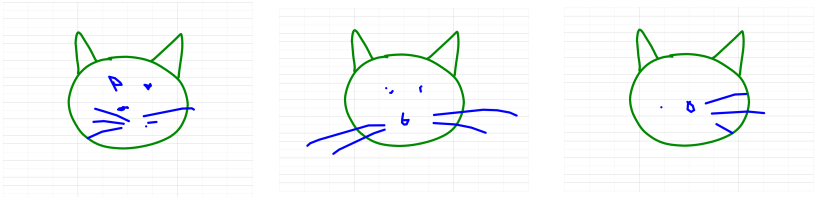

Fig. 2. Different predictions for the same beginning of a sketch using the cat class in *DrAwME*. In green, the contour of the face and its two ears have been drawn. The model predicted the blue strokes. Test it yourself: https://unifri.imuscica.eu/drawme/test/

of a cat's face is an eye of the cat and then draw the stroke of the cat's face all around the first strokes. The model looks less performant than the one from *sketch-rnn*. It is possible that the model we trained still has to learn some useful thing to generate sketches. The difference could be explained by the different training, and also in the different hyperparameters used. While we only tested with the default parameters for an unconditional model, the model trained by *sketch-rnn* is probably one with very good hyperparameters.

For future works, it would be interesting to train new architectures (using for instance Convolutional Neural Network), and see how good the new models are able to generate sketches. We could later also try the new architectures on music generation.

5.4 Musical Sketch Generation

We trained the musical model with the same parameters for sketch generation. We used only the decoder with an RNN's size of 1024 and trained it on a 16 cores CPU in 9 days.

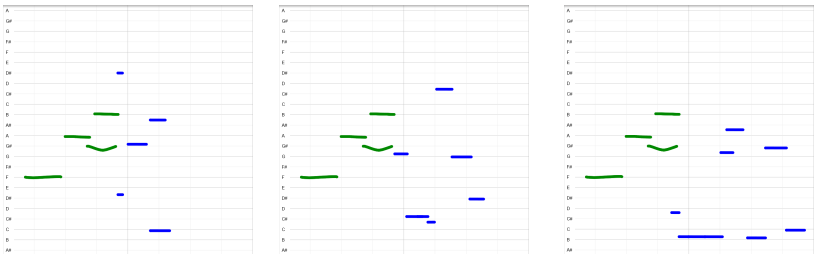

Fig. 3. Different predictions for the same beginninng of a sketch using the music class in *DrAwME*. In green, the four left most notes have been drawn. The model predicted the blue strokes. Test it yourself: https://unifri.imuscica.eu/drawme/test/

The sketches produced by the model we trained is able to produce sketches that looks like a regular pianoroll (see figure 3). Despite the fact that the model

has been trained on different range of notes, we think that the generated notes are still coherent with the drawing of the user. The model is not producing notes with exact frequencies, mainly because of the different numbers of range between the training dataset and the *DrAwME* interface.

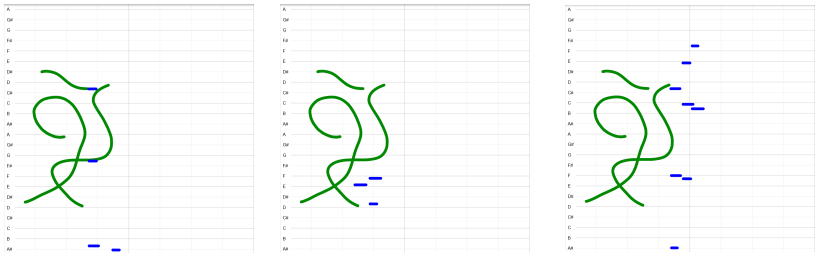

Fig. 4. Different predictions for the same beginning of a sketch using the music class in *DrAwME*. Here, we show what the network is producing with drawing that do not looks like usual notes. In green, the strokes that are not horizontal have been drawn before the generation by the model. The model predicted the blue horizontal strokes. Test it yourself: https://unifri.imuscica.eu/drawme/test/

One interesting point is how the model reacts to different styles of inputs. While it is generally generating strokes that looks like notes, in figure 4, we show how the model generates the next strokes with unusual drawing according to the class. As the model has learned the structure of music, it is always producing horizontal strokes. However, because it is not expecting such drawings, we think that the quality of the music generated could be affected by this change.

One important point that need to be tested is how good the music produced is and also how interactive the whole process is. For that, we would need to ask different people what they think.

We also have to try with further experiments on how we convert the music dataset to a dataset of sketch representing musics. First we will have to try with fewer musical styles. As *sketch-rnn* is not doing well when it comes to generate sketches of multiple classes, we think that the musical model could face the same problem. We will then have to try with a single musical style, and also with only one instrument. Also, because the notes have been added to the drawing in the order of their apparition on the music (from left right), one problem of the generation is that the network is only generating notes at the right of the last drawn stroke. For future works, it would be important to solve this problem so that the model could complete music before the first notes and also in the middle of what have been drawn.

6 Conclusion

Using the *sketch-rnn* [13] system, we successfully built an Artificial Intelligence that is able to generate drawing and music. It is capable to work in a process

of co-creation with the user, completing a drawing or a music step by step. To do that, we have been training new models using either a sketch dataset or a musical sketch dataset that we created.

They are still tasks to be completed especially for the validation of our results. We will need to make an user-study to see how good our Artificial Intelligence is at generating music. It is opening a lot of opportunities to work on. For a model that uses conversion between drawings and music, we could try how well Convolutional Neural Networks, or others architecture perform for this task. Also, there is still a lot to do to have a relevant conversion from music to drawing. We still have to better define how the examples are restricted.

References

1. A.i. duet, a piano that responds to you., https://github.com/googlecreativelab/aiexperiments-ai-duet
2. The quick, draw! dataset, https://quickdraw.withgoogle.com/data
3. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)
4. Alberti, M., Pondenkandath, V., Würsch, M., Ingold, R., Liwicki, M.: DeepDIVA: A Highly-Functional Python Framework for Reproducible Experiments (apr 2018)
5. Choi, K., Fazekas, G., Sandler, M.: Text-based lstm networks for automatic music composition. arXiv preprint arXiv:1604.05358 (2016)
6. Dong, H.W., Hsiao, W.Y., Yang, L.C., Yang, Y.H.: Musegan: Symbolic-domain music generation and accompaniment with multi-track sequential generative adversarial networks. arXiv preprint arXiv:1709.06298 (2017)
7. Eck, D., Schmidhuber, J.: A first look at music composition using lstm recurrent neural networks. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale **103** (2002)
8. Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., Norouzi, M.: Neural audio synthesis of musical notes with wavenet autoencoders. arXiv preprint arXiv:1704.01279 (2017)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)
10. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850 (2013)
11. Guimaraes, G.L., Sanchez-Lengeling, B., Outeiral, C., Farias, P.L.C., Aspuru-Guzik, A.: Objective-reinforced generative adversarial networks (organ) for sequence generation models. arXiv preprint arXiv:1705.10843 (2017)
12. Ha, D., Dai, A., Le, Q.V.: Hypernetworks. arXiv preprint arXiv:1609.09106 (2016)
13. Ha, D., Eck, D.: A neural representation of sketch drawings. arXiv preprint arXiv:1704.03477 (2017)
14. Hadjeres, G., Nielsen, F.: Interactive music generation with positional constraints using anticipation-rnns. arXiv preprint arXiv:1709.06404 (2017)
15. Hadjeres, G., Nielsen, F., Pachet, F.: Glsr-vae: Geodesic latent space regularization for variational autoencoder architectures. arXiv preprint arXiv:1707.04588 (2017)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)

17. Huang, C.Z.A., Duvenaud, D., Gajos, K.Z.: Chordripple: Recommending chords to help novice composers go beyond the ordinary. In: Proceedings of the 21st International Conference on Intelligent User Interfaces. pp. 241–250. ACM (2016)
18. Jaques, N., Gu, S., Bahdanau, D., Hernández-Lobato, J.M., Turner, R.E., Eck, D.: Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. arXiv preprint arXiv:1611.02796 (2016)
19. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
20. Le Groux, S., Verschure, P.: Towards adaptive music generation by reinforcement learning of musical tension. In: Proceedings of the 6th Sound and Music Conference, Barcelona, Spain. vol. 134 (2010)
21. Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., Bengio, Y.: Samplernn: An unconditional end-to-end neural audio generation model. arXiv preprint arXiv:1612.07837 (2016)
22. Mogren, O.: C-rnn-gan: Continuous recurrent neural networks with adversarial training. arXiv preprint arXiv:1611.09904 (2016)
23. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
24. Raffel, C.: Learning-based methods for comparing sequences, with applications to audio-to-midi alignment and matching. Columbia University (2016)
25. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing **45**(11), 2673–2681 (1997)
26. Shin, A., Crestel, L., Kato, H., Saito, K., Ohnishi, K., Yamaguchi, M., Nakawaki, M., Ushiku, Y., Harada, T.: Melody generation for pop music via word representation of musical properties. arXiv preprint arXiv:1710.11549 (2017)
27. Simhon, S., Dudek, G.: Sketch interpretation and refinement using statistical models. In: Rendering Techniques. pp. 23–32 (2004)
28. Teng, Y., Zhao, A., Goudeseune, C.: Generating nontrivial melodies for music as a service. arXiv preprint arXiv:1710.02280 (2017)
29. Tresset, P., Leymarie, F.F.: Portrait drawing by paul the robot. Computers & Graphics **37**(5), 348–363 (2013)
30. Xie, N., Hachiya, H., Sugiyama, M.: Artist agent: A reinforcement learning approach to automatic stroke generation in oriental ink painting. IEICE TRANSACTIONS on Information and Systems **96**(5), 1134–1144 (2013)
31. Yang, L.C., Chou, S.Y., Yang, Y.H.: Midinet: A convolutional generative adversarial network for symbolic-domain music generation. In: Proceedings of the 18th International Society for Music Information Retrieval Conference (ISMIR'2017), Suzhou, China (2017)
32. Zhang, X.Y., Yin, F., Zhang, Y.M., Liu, C.L., Bengio, Y.: Drawing and recognizing chinese characters with recurrent neural network. IEEE transactions on pattern analysis and machine intelligence **40**(4), 849–862 (2018)