Procedural Content Generation of Rhythm Games Using Deep Learning Methods

Yubin Liang, Wanxiang Li, Kokolo Ikeda

Procedural Content Generation of Rhythm Games Using Deep Learning Methods

Yubin Liang[1], Wanxiang Li[1] *, and Kokolo Ikeda[1]

School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan
liang_yubin@yahoo.com
wanxiang.li,kokolo@jaist.ac.jp

Abstract. The rhythm game is a type of video game which is popular to many people. But the game contents(required action and its timing) of rhythm game are usually hand-crafted by human designers. In this research, we proposed an automatic generation method to generate game contents from the music file of the famous rhythm game "OSU!" 4k mode. Generally, the supervised learning method is used to generate such game contents. In this research some new methods are purposed, one is called "fuzzy label" method, which shows better performance on our training data. Another is to use the new model C-BLSTM. On our test data, we improved the F-Score of timestamp prediction from 0.8159 to 0.8430. Also, it was confirmed through experiments that human players could feel the generated beatmap is more natural than previous research.

Keywords: Procedural Content Generation · Rhythm Game · C-BLSTM.

1 Introduction

Rhythm game is a genre of music-themed (action video) game in which players play by taking actions in accordance with rhythm and music [8]. Since music and songs are familiar to ordinary people, it is easy to understand how to play such games. In addition, both of easy stage and hard stage can be created from one music, therefore rhythm game becomes a popular game genre in the whole world.

In many cases, the contents (required action and its timing) of rhythm game are hand-crafted by human designers from music material. Also, there are countless pieces of music, but only a part of them have already been used as game contents. Therefore, to generate contents automatically for rhythm game is required. In this research, two models are used to generate contents from music materials for famous rhythm game "OSU!"[1]. One inputs the audio data, outputs timestamps (timing of action), another inputs the timestamp and outputs action type.

As a Machine Learning task, this approach has some difficulties such as: (1) Because same music may be processed by different authors or has different

* Corresponding author

level settings, one music may have many different beatmaps. (2) Proportion of positive/negative samples is ill. To deal with those problems, in this research we adopt that: (1) Handle the difficulty settings as an input feature. (2) Using fuzzy labels to increase positive samples.

Proposed training methods and new model C-BLSTM are evaluated through F-Score, shows a better performance than previous research on timestamp generation. And from human experiments we can find that beatmaps which generated by purposed method are more natural than beatmaps generated by previous research.

2 Background

The system structure of purposed method is shown as Fig.1. The main workflow is from audio files generating the timestamp, and using those timestamps to generate action type by second model. Combining the generated timestamp and action type, we can obtain beatmap for input music.

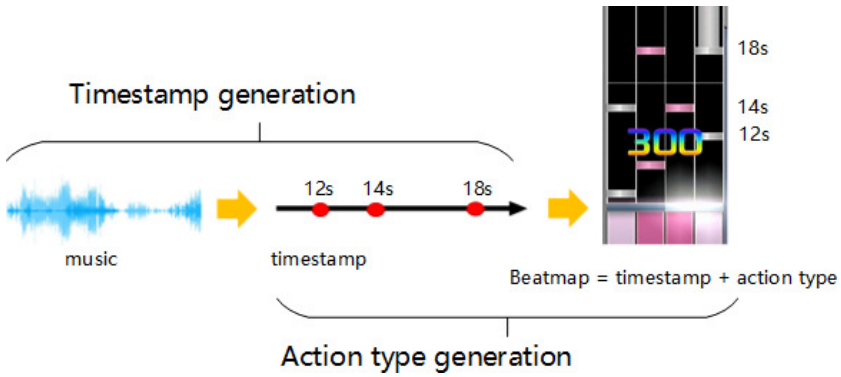


Fig. 1: System structure of purposed method.

2.1 Rhythm game: OSU!

To generate the contents of rhythm game, we use open-sourced game "OSU!" which has over 10 million registered users [1] as our test bench.

"OSU!" consists of many types (or mode) of game, but in this research we foucs on the mode called mania 4k. Players are required to push the button when markers are dropping from the top of screen. When players push the correct button, they would get a high score. On the other hand if the timing or required action was incorrect, system would judge this action as missing. By pushing the buttons, players would feel like they are playing this rhythm by themselves.

These required actions (4 types: click, long push start, long push end and none) and their timing consist the game contents. In "OSU!", beatmap records

these information. Generally, beatmap are usually hand-crafted by human designers. So even if there are countless pieces of music, the number of contents that can be played in the game is limited. An automatic generation method is needed in this case.

2.2 Previous research

In the field of rhythm game and game contents generation research. There are some previous researches which is essential to our approaches.

Capturing the characteristics of music is an important part in rhythm game content generating. Generally, selection of action timings is based on the sounds or rhythm which can be clearly heard by human. Since the beginning or changing of music is most distinguishable to human, we usually set actions around these timing in rhythm games. To extract these timing, Schlüter et.al proposed an research using deep learning methods to detect the beginning of musical notes (musical onsets) [10]. Their method is used to extract features in our research.

Long Short-Term Memory (LSTM) is a kind of Recurrent Neural Network (RNN) which is effective to suppress gradient vanishing over time [5], and it has been proved to be effective for time-based task such as voice detection [7] and translation [6], etc.

Music or rhythm is generally regarded as a type of time-based data, and in the field of rhythm game content generation, LSTM was used to generate game content of a rhythm game called "Dance Dance Revolution" in Donahue et al.'s research [3]. However, to generate a content which correspond to specify difficult level, or some special patterns (e.g. patterns that player good at or not) is still not so gratifying. Hence, this research is aimed at generating game contents that correspond to specific player's level in rhythm game.

In many rhythm games, long press buttons are basically corresponds to a long sound of music. So it is useful for analyzing the melody to generate the long press properly. In Donahue et al.'s research, although the effectiveness of melody information in action generation is examined, but its application has not been taken yet [3]. In this research, we use the method which propose by Salamon et al. [9], for extracting the main melody of music. By using their method, we can detect long sounds in music and attempt to employ them in long press action generation.

3 Preparation of Training Data

3.1 Data and difficulty definition

In this research, all training data is collected from the homepage of "OSU!" [1] on June, 2017. Considering to use the most popular beatmaps as our traning data, only beatmaps which has been played over 100 thousands times will be chosen.

In general, one music file corresponds to various beatmaps depending on the different authors and different difficulties. And players can play a suitable

difficulty according to their ability. The statistical information of used training data is shown as Table 1.

Table 1: Statistical information of traning data

Number of authors*	Number of music	Number of beatmap	Number of all action*	Action per second*
300	473	1655	1690000	7.85

*Approximately.

Since there are various difficulties in the training data, and we aims to generate beatmaps with different difficulties from one music. So a proper definition of difficulty is highly required.

Generally, the difficulty is depending on: (1) Complexity of action combination. (2) Number of actions per second (density of actions). (3) Speed of action marker. (4) The strictness of judging miss (max allowable time difference). (5) Number of miss that could be permitted.

Because (3), (4) and (5) is the more relative to setting of game, we don't consider these factors in this research. Also, individual players have different action combinations that they are not good at, so it is hard to use factor (1) in automatic generation method. More actions in same time means beatmap is more difficult, so in this research we just use density of actions to define the difficulties for each beatmap.

We divide the training data to 10 levels by density, that is (a) number of action under 3 per second is level-0, (b) number of action from 3 to 17 per second is used for level-1 to level-8 equally (e.g. 3 to 4.75 is level-1 and 4.75 to 6.5 is level-2), (c) number of action more than 17 per second is level-9.

3.2 Feature extraction

In order to use the audio data in neural networks, some feature extraction process is needed. Firstly we compute a multiple-timescale short-time Fourier transform (STFT) of every audio file by three window sizes 23ms, 46ms and 93ms and stride of 10ms as 1 frame (that means there are 100 frames per second). Shorter window sizes preserve low-level features such as pitch and timbre while large window sizes provide more context for high-level features such as melody and rhythm[4].

Then we compute the Mel-scale from 27.5Hz to 16kHz with 80 bands of STFT magnitude spectra to better represent human perception of loudness [12].

Finally we prepend and append n frames of past and future context for each frame, so the final audio representation is a 80 * 3 * $(2n + 1)$ tensor.

4 Timestamp generation

4.1 Network structure

Donahue et al. uses C-LSTM to generate timestamp [3], which has the advantage to learn not only the current frame but also the past and future frames. Structure of C-LSTM is shown as Fig.2. We use 15 * 80 * 3 tensor ($n = 7$) as input. First layer is a convolution layer which has the size of 7 * 3 * 3 and 1 * 3 max-pooling, output 9 * 26 * 10 matrix to next layer. Second layer is a convolution layer which has the size of 3 * 3 * 10 and 1 * 3 max-pooling, output 7 * 8 * 20 matrix. Adding the 10 units one-hot density (which present the level of difficulty), we input the matrix to 2 LSTM layer which have 200 units. After LSTM there are 2 full-connected layer with 258 units and 128 units separately. Finally we output the result by a single Sigmoid unit, which presents the possibility of been chosen as a timestamp. LSTM layers use tanh function as activation function, while other layers use ReLU.

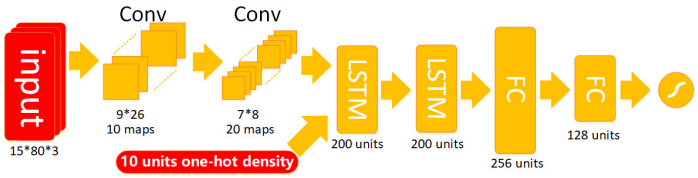


Fig. 2: Structure of C-LSTM.

C-LSTM can only use the information "until now". In this research, "OSU!" has a lot of "long push" action, since most "long push start" action is at the onset of long melody, we need to consider not only the information in past, but also information in the future. One way is to use more frames (i.e. make n bigger), but the high calculation cost and performance reduction is unbearable sometimes. Another way to get more future information without increasing frames is using the C-BLSTM [11].

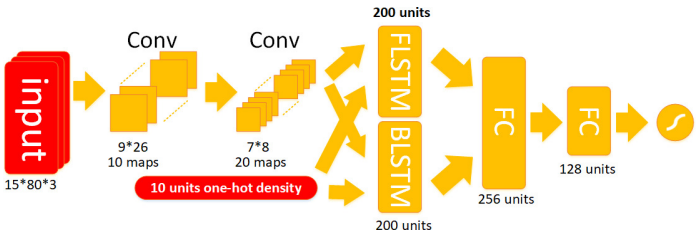


Fig. 3: Structure of C-BLSTM.

The structure of C-BLSTM is shown as Fig.3. C-BLSTM can predict timestamp using more future information, but has the similar calculation cost as C-LSTM.

4.2 Fuzzy label

There are about 7.85 actions per second in our training data. Since we have 100 frames in 1 second, there are just few positive data (less than 8%). One way to solve this problem is to calculate feature with wider stride. For example using 50ms stride will get 20 frames per second, which will make the rate of positive data to 40%. But longer stride will lose some rhythm or melody information.

In this research a method called fuzzy label that can solve the problem is proposed. Shown as Fig.4, the original data use 0 (no action in this frame) and 1 (there is action in this frame) label which have a drastic changing. And fuzzy label modify the data using Gaussian distribution, transform the data from "whether the frame has actions or not" to "the possibility that the frame has actions", makes the changing smoother.

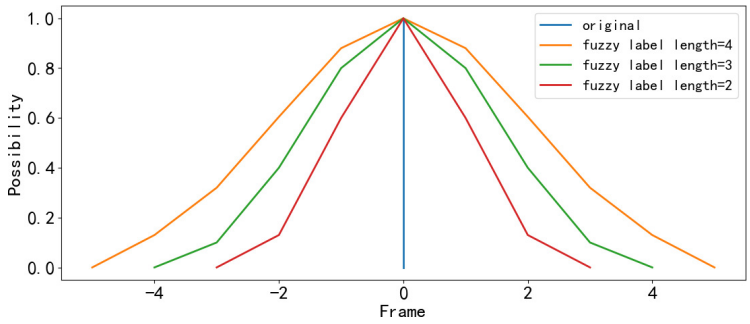


Fig. 4: Example of fuzzy label.

4.3 Threshold selection

Output of network is the "possibility whether one frame should have actions". In practice, we need to transform the possibility into a two-value number (0 or 1), to decide whether there should have actions in this frame. Using some rules such as "when possibility is bigger than a threshold t, there should have actions" can be helpful for judging.

When threshold was too large, output actions would decrease, and output action would increase when it was too small. Both situations are not good for timestamp selection, so a proper threshold is needed. Some data which called "threshold deciding data" is randomly selected to decide the threshold t ,which we will test different threshold in the same model by it, to maximize the F-score. The threshold deciding data is independent to training data and will not take part in the training process.

4.4 Experiment

Difficulty distribution in original data is biased, there are just few low difficulty data and high difficulty data. Such an ill data distribution is harmful and causing unstable learning [2]. To solve this problem, we copy the training data in low and high difficulty to make original data distribution balanced.

Validation experiment In this experiment, all model uses 80*3*15 (frame length $n = 7$) tensor as input and same learning hyper-parameters. All networks are trained for 25 epochs (1000 batches per epoch), the mean F-score of each epoch is shown as Fig.5.

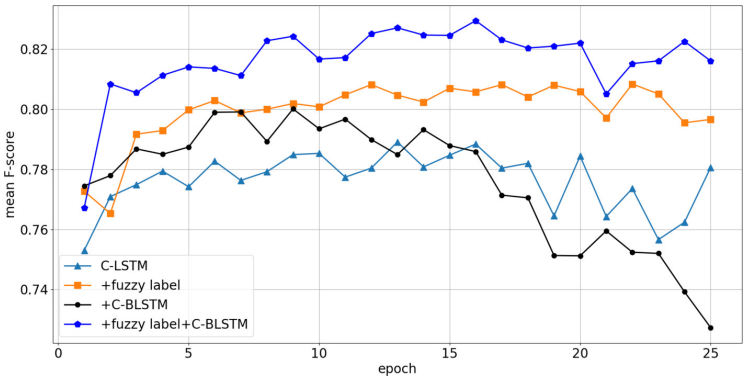


Fig. 5: Result of validation experiment.

From the result, we can find that comparing to the C-LSTM, C-BLSTM shows a better performance in the beginning, but suddenly facing the problem of degradation in performance. We also find that training error of C-BLSTM deceasing follow the process. This means C-BLSTM faced a serious overfitting problem in the mid of training.

When comparing the result of C-LSTM to C-LSTM with fuzzy label (label length is 3), we can find that fuzzy label improved the performance dramatically. Moreover, from the result of C-BLSTM with fuzzy label, we can find that fuzzy label not only improved the performance, but also solved the overfitting problem of C-BLSTM.

Experiment of different fuzzy label We also have experiments on different fuzzy label lengths to compare their performance. In this experiment, all models use C-BLSTM and 80*3*15 (frame length $n = 7$) tensor as input, also, they use same learning hyper-parameters.

Mean F-score of each epoch is shown as Fig.6. We can find that our method improved the naive C-BLSTM comparing to the result with no fuzzy label. Also,

we find best fuzzy label length is 3 on our data. One possible reason is when length was 3, the rate of positive data would be closed to 50%, which is a well-balanced data distribution.

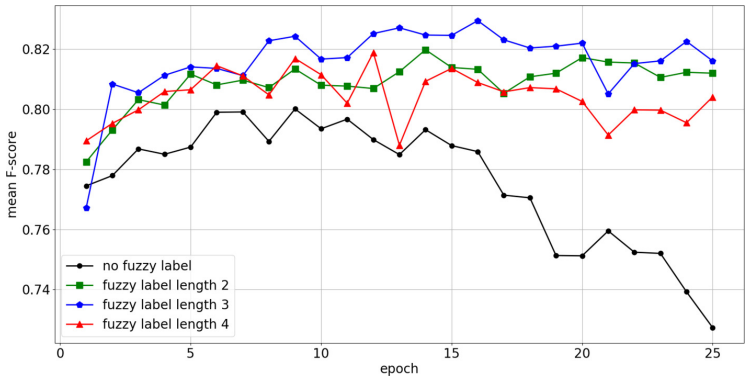
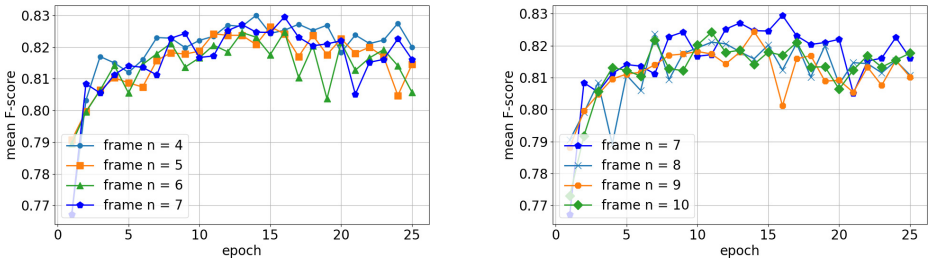


Fig. 6: Result of fuzzy label experiment.

Experiment of different frame length In this experiment, all model use C-BLSTM, same fuzzy label length of 3 and same learning hyper-parameters. The only difference is the frame length, which will influence the input tensor size. For example when frame length n was 7, the input size would be 80*3*15, and when frame length n was 9, the input size would be 80*3*19.

The result of n =4, 5, 6 and 7 is shown as Fig.7a. We can find that when n is 4, we would get the best performance, and when n is 7 we would get the similar performance to n =4. From Fig.7b we can find that n =7 showed the best performance. The result shows that we cannot get better performance by simply adding the by frame length.



(a) Result of n =4, 5, 6 and 7 (b) Result of n =7, 8, 9 and 10

Fig. 7: Result of different frames

5 Action type generation

5.1 Input feature and output

In our Action type generation model, we use (1) Action before one frame, (2) Difficulty and (3) Time interval of past/future frame as input. And 4 types of action will be generated and used as input, that is (a) No action, (b) Click, (c) Long push start and (d) Long push end. 4 bit one-hot vector is used to present action on single channel. And since we have 4 channels, the action will be four 4 bit one-hot vectors.

The time interval is the distance between two actions. We use a 8 bit one-hot vector to present it. This 8 bit one-hot vector means the interval is ∼50ms, ∼100ms, ∼200ms, ∼400ms, ∼800ms, ∼1600ms, ∼3200ms and 3200ms∼. Since both the past time interval and future interval is used, the time interval will be presented as two 8 bit one-hot vector. And a 10 units one-hot density label (which present the level of difficulty) is also needed in this model. The input feature is combined with the density, time interval and past action, so input feature has 7 vectors and the data size is 42 bit.

The output is probabilities of all action combinations. Since we have 4 channels and each channel has 4 types of action, the number of all valid actions is 256 (4^4). Finally we will choose one action combination that have biggest probability.

The example of input feature and output probabilities is shown as Fig.8.

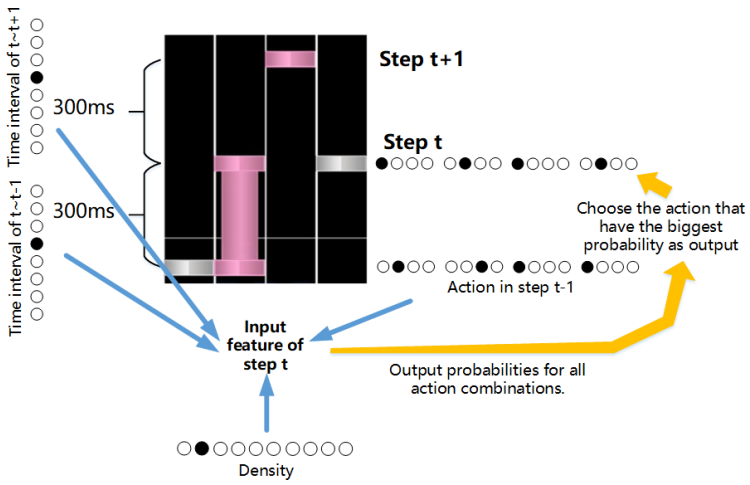


Fig. 8: Example of input features and output.

5.2 Network structure

In Action type generation part, we uses the network frame called LSTM64 [3]. Shown as Fig.9, after input layer the first layer uses 128 LSTM units, and secend

layer uses 128 LSTM units. Finally this network output 256 probabilities of all valid actions.

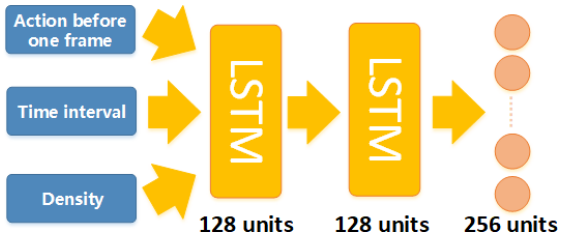


Fig. 9: Structure of beatmap generation model.

5.3 Experiment of predict actions

Same training data as timestamp generation model are used for training this model. Also, since the difficulty distribution in original data is unbalanced, training data in low and high difficulty is copied to make original data distribution balanced.

The finial accuracy of predicting action is 0.4366, which is not a high accuracy. But we find that even in one music, there are different beatmaps. And according to the different designers, the beatmap is various. For example in Fig.10, we can find that about half actions are not same between two human designer, even they have used the same music.

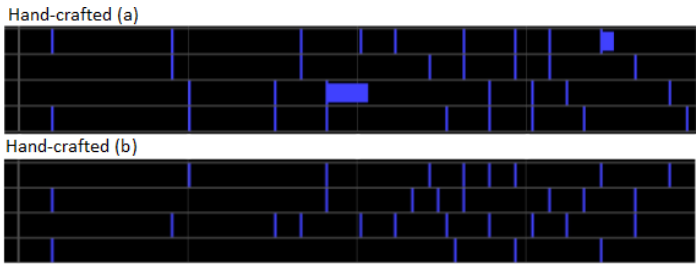


Fig. 10: Compare of two Hand-crafted beatmaps.

Fig.11 shows a result of beatmap generated by network and human designer for same music. We can find that even some actions are different, these two beatmaps are similar, which player may not notice the difference. According to these evidence, we think the result that predicting accuracy is 0.4366 is not bad. Prediction accuracy is not enough for estimating how human players can enjoy the generated beatmaps, so it will be an interesting future work to propose such a measurement.

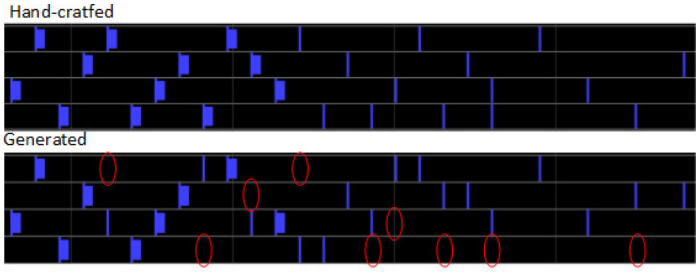


Fig. 11: Compare of Hand-crafted and purposed method.

6 Evaluation of naturalness

We conduct a experiment though 10 human subjects. 15 beatmaps (5 of them are hand-crafted by human designer, 5 of them are generated by previous research and 5 of them are generated by proposed method) are used in this experiment. "Purposed method" data and "Existing method" data use same action type generation model, but "Purposed method" data use C-BLSTM model with fuzzy label (length=3), frame size n is 7 to generate timestamp. timestamp of "Existing method" data are generated by C-LSTM without fuzzy label, and frame size n is 7.

All Beapmaps are randomly watched by every subjects, and subjects are asked to judge the naturalness for each beatmap. The result is hand-crafted beatmaps get 4.52 points, existing method get 3.30 points and purposed method get 3.72 points. From the result we can find that even beatmaps which generated by purposed method are not as natural as hand-crafted beatmaps, the naturalness is better than beatmaps generated by existing method.

7 Conclusion

Overall, a new method "fuzzy label" is introduced in game contents generation of rhythm game, and new model C-BLSTM is used in our timestamp generation method. We have proofed that purposed method improved the performance of Timestamp generation, and the F-Score is increased from 0.8159 to 0.8460.

Moreover, a generation model from timestamp to beatmap is introduced. And by experiment on human subjects, we have proofed that beatmaps generated by purposed method is more natural than previous research.

Acknowledgments. This research is financially supported by Japan Society for the Promotion of Science (JSPS) under contract number 17K00506.

References

1. Osu! `https://osu.ppy.sh/home`, accessed June, 2017

2. Buda, M., Maki, A., Mazurowski, M.A.: A systematic study of the class imbalance problem in convolutional neural networks. Neural Networks **106**, 249–259 (2018)
3. Donahue, C., Lipton, Z.C., McAuley, J.: Dance dance convolution. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70. pp. 1039–1048. JMLR. org (2017)
4. Hamel, P., Bengio, Y., Eck, D.: Building musically-relevant audio features through multiple timescale representations (2012)
5. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
6. Kalchbrenner, N., Danihelka, I., Graves, A.: Grid long short-term memory. arXiv preprint arXiv:1507.01526 (2015)
7. Parascandolo, G., Huttunen, H., Virtanen, T.: Recurrent neural networks for polyphonic sound event detection in real life recordings. In: 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 6440–6444. IEEE (2016)
8. Pasinski, A.: Possible benefits of playing music video games (2014)
9. Salamon, J., Gómez, E.: Melody extraction from polyphonic music signals using pitch contour characteristics. IEEE Transactions on Audio, Speech, and Language Processing **20**(6), 1759–1770 (2012)
10. Schlüter, J., Böck, S.: Improved musical onset detection with convolutional neural networks. In: 2014 ieee international conference on acoustics, speech and signal processing (icassp). pp. 6979–6983. IEEE (2014)
11. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing **45**(11), 2673–2681 (1997)
12. Stevens, S., Volkmann, J., Newman, E.: The mel scale equates the magnitude of perceived differences in pitch at different frequencies. Journal of the Acoustical Society of America **8**(3), 185–190 (1937)