# Automatically Generating Beat-maps using Neural Networks

1st Thai Do
*Department of Computer Science*
*Middle Tennessee State University*
Murfreesboro, Tennessee USA
tmd4w@mtmail.mtsu.edu

2nd Jackson Goble
*Department of Computer Science*
*Middle Tennessee State University*
Murfreesboro, Tennessee USA
jlg2av@mtmail.mtsu.edu

3rd Nzubechukwu Molokwu
*Department of Computer Science*
*Middle Tennessee State University*
Murfreesboro, Tennessee USA
num2a@mtmail.mtsu.edu

4rd Levi Shirley
*Department of Computer Science*
*Middle Tennessee State University*
Murfreesboro, Tennessee USA
cls6a@mtmail.mtsu.edu

***Abstract*—Beat Saber allows for anybody to create a beat-map for any song they wish to, but not all of these maps can be considered as fun and playable. There are many maps for more obscure songs that are tuned for very high difficulties. Due to the complexity of both beat-maps and songs, little work has been done to automate the process of generating a beat-map. It is hoped that by exposing a network to many songs and their corresponding beat-maps, it would be capable of generalizing these associated values and generating new beat-maps given a song. We experimented with two different neural networks to automatically generate beat-maps, one using a Long Short Term Memory layer, and the second using only Convolutional layers. Our networks are currently capable of producing a similar beat-map given a song and its corresponding beat-map. Because of these results, we can conclude that our current methology is heading in the right direction of producing beat-maps for any song, but we currently do not have an actual network that is capable of this.**

***Index Terms*—Neural Networks, Convolutional Neural Networks, Recurrent Neural Networks, Long Short-Term Memory, Rectified Linear Unit, Fourier Transformation, Beat-map, CNN, RNN, LSTM, RELU**

## I. Introduction

Rhythm/Music games have been around for a long time. Games such as Dance Dance Revolution, and Guitar Hero are some of the most notable rhythm game titles. In recent years, the advent of virtual reality has given rise to a new game called Beat Saber. In all of these previously mentioned games, developers would can develop levels for specific songs that corresponded to the characteristics of the song. Beat Saber, however, also allows for the players of the game to create levels called beat-maps for any song they wish to, and then play that beat-map. Because these beat-maps can be made by anyone with any level of experience, the quality of the beat-map is dependent on the person that created the map. Some user-developed beat-maps are either impossible to play, or aren't consistent with the characteristics of the song.

Because of the inconsistency among user-developed beat-maps, our goal is to develop and train a neural network that is capable of automatically generating a fun and playable beat-map given any song.

## II. Background

In general, a basic beat-map for Beat Saber is a complex file that contains various hit objects that are placed based on the characteristics of a song. The map itself is a 3x4 grid, thus creating 12 areas that a hit object can be placed on. These hit objects can be notes, bombs, or walls. In Beat Saber, a note consists of two properties: color and direction. A note can either be red or blue, and there are nine possible directions that a note can be in. Bombs and walls are meant to be obstacles that the player needs to avoid. In addition to beat-maps, the songs themselves are also complex structures. Songs contain characteristics such as rhythm, tempo, and frequency [1]. Because each song has varying characteristics, it is difficult for an automatic generator to determine what type of object should appear accordingly.

Based on the previous points, a generator needs to able to determine whether a note, bomb, or wall should be placed. However, we decided that we should ignore bombs and walls, and focus only on notes in our automatic generator. By removing the possibility of bombs and walls, we reduce the complexity of the beat-map we generate and can focus on just notes.

We experimented with two different neural networks to see which would yield better results. For the first neural network, we decided to use both a Convolutional Neural Network (CNN), and a Long Short Term Memory (LSTM), a variation of a Recurrent Neural Network (RNN), to predict whether a note exists or not at a moment in the song and if it does, determine the color and direction of the note. There is a similar project on Github called Osumapper that was built for a rhythm game known as "osu!" [2]. Though the games are different, the concept of classifying and placing a note is the same. This project has a similar network to our first network. Osumapper first inputs a song into a CNN. After

getting an output from the CNN, it then puts the output into a LSTM to classify the type of note that goes with the song. The biggest difference between our first network and Osumapper is that Osumapper feeds its constructed map into a generative adversarial network (GAN), something that our network doesn't include.

Our second network consists of only convolutional and deconvolutional layers. The main difference of this network compared to the second network is that it lacks a LSTM layer. This network lets the CNN layers convolving over the data handle the time dimensional aspect.

## III. Methods

### A. Data

*1) Gathering Training Data:* One of the most important aspects of training any neural network is gathering the appropiate training data. Therefore, the first step was to find a source that contains beat-maps that users have developed. For our project, we used the website www.beatsaver.com. This website allows users to upload, share, and rate each others beat-map creations. Because there are so many beat-maps generated by different people, we needed to come up with a set of requirements that a beat-map must pass in order to be used in our training set. The first requirement to be considered an appropiate beat-map is that it needs to be highly-rated by the playerbase of Beat Saber. A map that is highly-rated means that it is well-developed and enjoyed by a majority of players. Secondly, the beat-map level must be categorized as an expert difficulty map. The reason for this is that expert difficulty is commonly considered to offer the proper Beat Saber experience. With these requirements set, we were able to gather a number of "fun and playable" beat-maps.

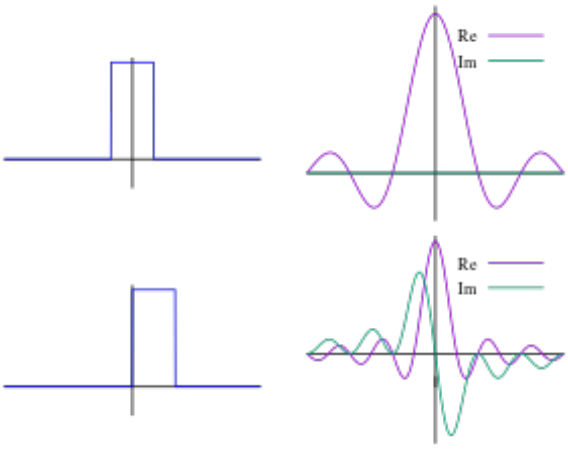


Fig. 1. Example of a Fourier Transformation, from Wikipedia, the free encyclopedia [3]. On the left of this figure is the data before Fourier Transformation. On the right side of this figure is the data after the Fourier Transformation.

*2) Preparing the Input Song:* Our project works with ".ogg" audio files, so we needed a way to load and manipulate such files. For this, we used the Librosa library [4]. With Librosa, we are able to load the audio file, and find the duration of the file in seconds. In addition to that, it also resamples the song, determines the tempo and converts the sample rate into something more manageable. After all this, it divides the audio file into uniform chunks. Librosa then preps the song by generating a Fourier transformation of the audio file. This makes the file much easier to use because a fourier transformation breaks down the audio into simpler constituent frequencies. We then use Librosa to find the absolute value of the transformed data, and then we find the mean-center of it. Figure 1 shows an example of a fourier transformation on a sample audio file.

### B. The First Neural Network

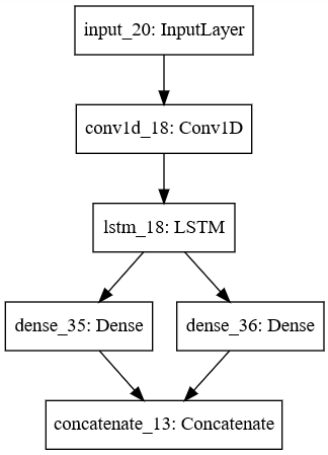


Fig. 2. A visual representation of the first neural network that we built.

For our first neural network, we started with a convolutional layer that takes in the input data, and performs a convolution on it allowing us to access a larger time interval. For the activation function of our convolutional layer, we chose to use the default rectified linear unit (RELU) function. This function is typically the default function used for CNNs. After the convolutional layer, the data will then go into the LSTM layer. The LSTM layer looks at each particular piece of the song while keeping in mind the previous piece and the upcoming piece in order to update its weights appropiately. The end goal of this layer is to generate two outputs, the direction and color of the note. Once the two outputs are produced, they each go into a dense layer. Because these layers are just a simple classification layer, we used a basic softmax activation function. Finally, the two outputs are concactenated together to form the notes that are placed in the beat-map. Figure 2 shows a visual representation of the first network.

### C. The Second Neural Network

Our second neural network is relatively large due to the high number of convoultional and deconvolutional layers in the network. This network handles the time dimension aspect by using several CNN layers to funnel the data and flatten the data into a dense layer. This data is then deconvolved into the necessary output shape. Similar to the first neural network, it

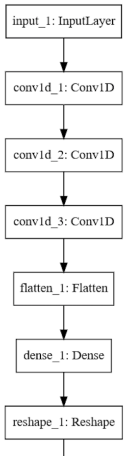


Fig. 3. A visual representation of part of the second neural network that we built.

is split apart into two separate dense layers corresponding to color and direction. Finally, it is concatenated together to form the notes that will go into beat map. Figure 4 shows a visual representation of a part of the second network.

## IV. Results

### A. First Neural Network Result

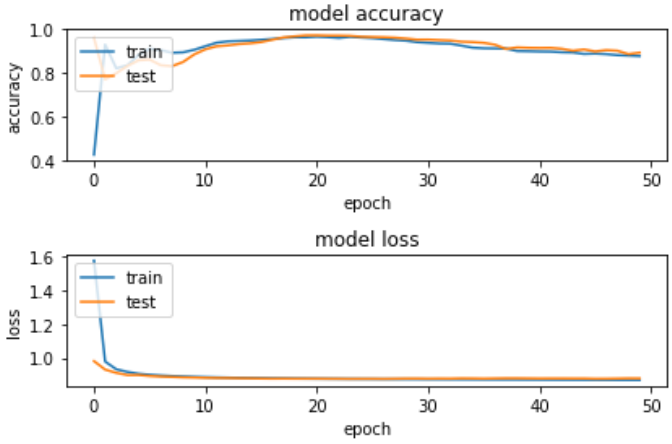


Fig. 4. The training and testing results of our first neural network.

To train this neural network, we give it a song that has been prepped by Librosa and broken up into separate soundwaves, and its corresponding beat-map tied to those particular moments. After it finishes training on those soundwave chunks, it validates the newly generated beat-map with the inputted beat-map. Figure 4 has a visual graph of our training results along with validation. The training accuracy average is 88 percent, and the testing accuracy average is 92 percent. This model doesn't perform as well as the second network as you'll see later on.

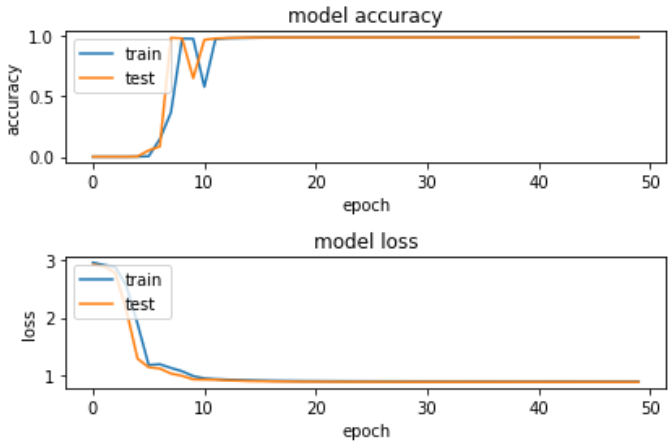


Fig. 5. The training and testing results of our second neural network.

### B. Second Neural Network Result

This network trains and validates in a similar way to our first neural network. The main difference being that this network removes the LSTM layer and instead handles the time aspect entirely through the CNN layers. Figure 5 shows our training results with validation. As you can see, training and validation accuracy is higher than the first network. Both the average training and training accuracy is 98 percent. So the initial attempt to input a song and its corresponding beatmap in order to generate a similar beatmap has been successful. Based on this, we can verify that we have a well-trained network that is capable of producing a valid beat-map.

## V. Discussion

Both of the resulting networks show promise. Each of the networks have high training and validation accuracy towards generating a similar beat-map for the given song. However, this may not be indicative of a well-formed beat-map. Further testing needs to be done in order to verify that the generated beat-map is actually similar to the given beat-map. In retrospect, our current networks show that it is possible to create similar beat-maps based off the original maps. Therefore, it is reasonable to assume that given enough songs and their corresponding beat-maps, we will be able to train a network that is able to successfully generalize and be able to generate new beat maps for any song. So while we were not able to meet this goal, we can conclude that we are moving in the right direction. Our future considerations would consist of specifying song genre when inputting songs and their beat-map into the neural network as well as trying to generate maps for the varying difficulties in Beat Saber.

## References

[1] Y.-Y. Chang and Y.-C. Lin, "Music tempo (speed) classification," 2005.
[2] kotritrona, "osumapper," 2018. [Online]. Available: https://github.com/kotritrona/osumapper
[3] Wikipedia, "Fourier transformation," 2019.
[4] B. McFee and O. Nieto, "librosa: Audio and music signal analysis in python," *SciPy 2015*, 2015.