AlgoRhythm

Final Report

Jennifer Guo guojenn1
Jason Shang shangji9

Word count: 1995
Penalty: 0%

Goal and Motivation:

The AlgoRhythm is a neural network that is designed to generate music based on its training data. By doing so, the AlgoRhythm would be able to develop many different potential pieces of music, depending on the initial "seed" notes fed into the network to start the generation. This would allow for music to be more easily generated, allowing for faster music development for a variety of purposes. The new generated pieces can also inspire musicians and composers in the creation of their own music. The increased production speed and ease of generation would allow more people to create their own music and also aid in music related multimedia industries, such as movies and video games.

Overview of Software Structure:

The overall software structure of the AlgoRhythm involves a convolutional neural network (CNN), alongside a feed-forward, fully-connected neural network (FCNN). For the training process, the initial training data (consisting of MIDI files) is converted into a set of 3 tensors: a 3D notes tensor, a 1D rests tensor, and a 1D lengths tensor. These three tensors are fed into the CNN, which performs 3D convolution on the 3D notes tensor and 1D convolution on the 1D rests and lengths tensors. The feature maps from these convolutions are reshaped and concatenated in order to be fed into the FCNN, which outputs the network's prediction of the next note.

The generation process is similar to the training process, except instead of using MIDI files, the user enters in 7 "seed" notes. These notes form the initial tensors which are fed into the neural network, and each time the network predicts a new note, that note's data is then added onto the tensors and is refed into the network. This process continues until the user decides enough music has been generated. This process will be explained in more detail in the following sections, and an outline of the entire process can be seen in Fig. 1.

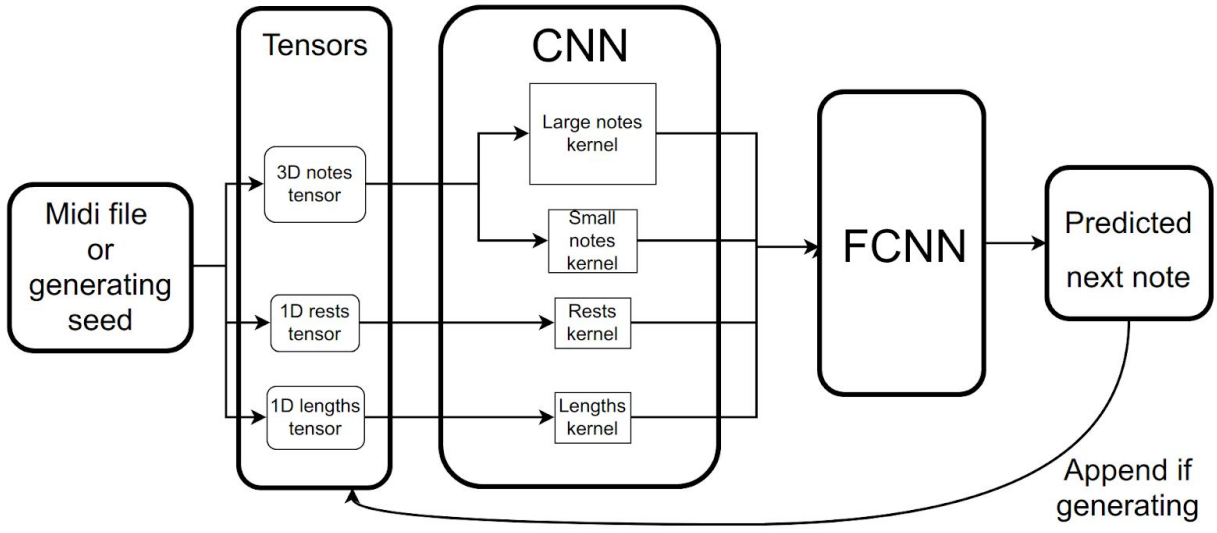

Fig. 1: The diagram for the final model used, showing the process where MIDI files are used to train the model, as well as how seed notes are used to generate new notes.

Sources of Data and Preprocessing:

The training data for the model consists of MIDI files of various Bach compositions from bachcentral.com, which in total, consist of approximately 100000 notes. We believed that it would be easier for the NN to learn meaningful patterns in the music if its training set was constrained to be from only one composer.

In order to convert these MIDI files into a format that can fed into the neural network, a Python library called Music21 was used to extract the note and chord objects stored in the MIDI files. As mentioned previously, the music was converted into a 3D notes tensor plus a 1D rests tensor and a 1D lengths tensor (Fig. 2). At each timestep in the song, the notes that are playing at that moment in time are represented as a 2D note tensor, which has dimensions of pitch and octave. This 2D tensor is then stored as a "slice" of the 3D tensor. This format allows multiple notes that occur at the same time to be represented in single object. If there are no notes playing in that moment of time, it is recorded in the rest variable. The length of that time-step,which is the time until another another note is played or a current note ends, is stored in the length variable.

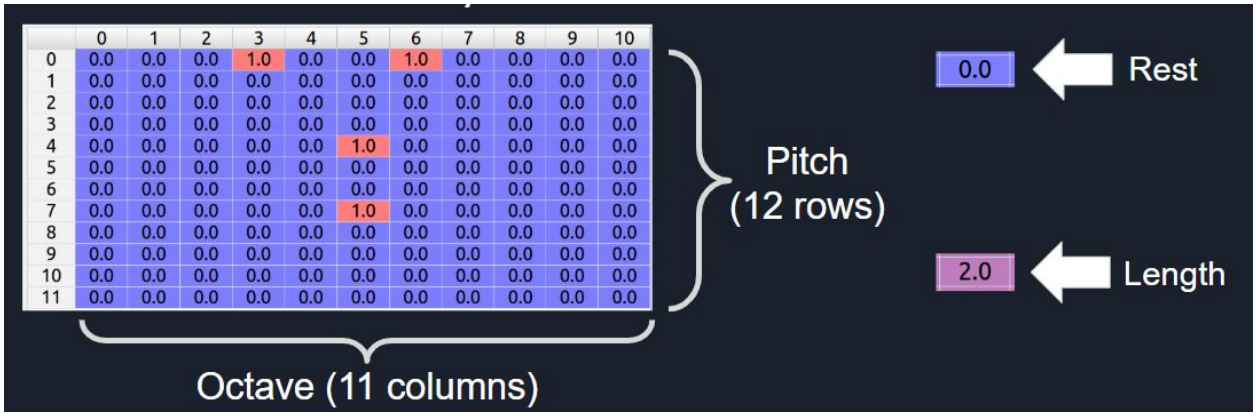

Fig. 2: An example of a possible time-step. The notes are represented in a 2D tensor, while the length of the time-step and whether or not that timestep was a rest are stored as variables.

The data from the time-step is then stacked onto the data from the previous time-steps, resulting in a 3D note tensor, a 1D rest tensor showing if the time-steps are rests, and a 1D length tensor, which shows the lengths of each time-step (Fig. 3).

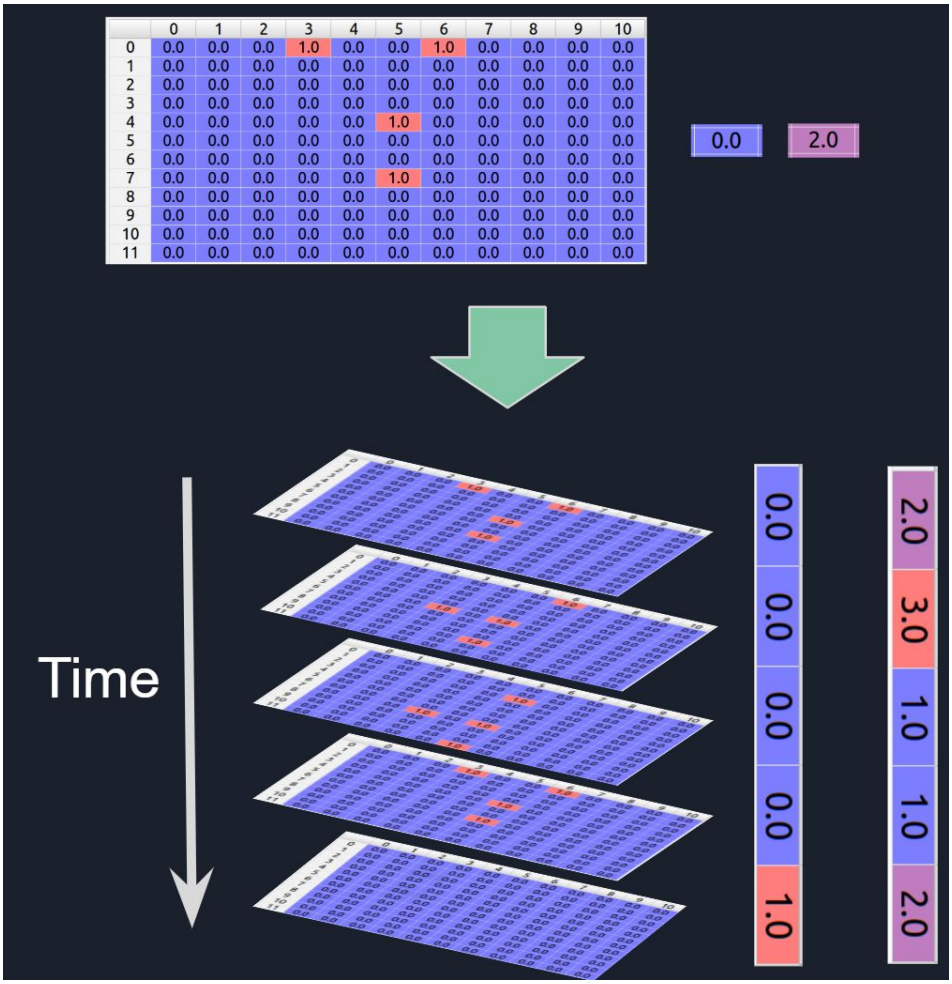

Fig. 3: A diagram showing how data from timesteps are appended to each other to create a 3D note tensor, a 1D rests tensor, and a 1D lengths tensor.

Machine Learning Model:

The machine learning model itself consists of 2 components, a 3D CNN and a FFNN. A convolutional architecture was chosen because music is invariant to shifts in time, pitch, and octave, which makes convolution an excellent choice.

Alternative Models:

Before we arrived at the final model, various other models were also tested. The first model that was tested was a RNN model that used a GRU, which was fed the notes that composed the melody of the song. However, when the result from the GRU was compared to the result from the FFNN, the FFNN's result sounded more natural and pleasing to the ear, while the RNN result was rather erratic. Later, 2 types of CNNs were used to try and encapsulated the various patterns that may be found in the music. The first type of CNN was a 2D CNN with one axis as time, and the other as pitch and octave. Since octaves just separate pitch by a constant, the second axis represented the 128 pitch-octave combinations that the MIDI format supports, encoded in a one-hot format. The other CNN type, which was used in the final model,

was a 3D CNN with dimensions of time, pitch and octave. When both models were tested, the 3D CNN performed better and produced more coherent melodies. This may be because the 2D CNN also included more convolution kernel sizes, greatly increasing the number of parameters and slowing down training. In the end, due to time constraints and the promising results that the 3D CNN model showed, we opted to use a 3D CNN in our final model.

Final Model:

During the convolution process of our final model, 3D kernels are convolved over the 3D note tensor. For the kernels, two different kernel sizes were used because they would have different-sized receptive fields and thus be able to capture patterns on a different size scale, both in the time and pitch dimension. The larger kernel had size 7 (time) x 12 (pitch) x 2 (octave), and the smaller kernel was 4 (time) x 4 (pitch) x 2 (octave).

Wraparound was performed in the pitch dimension to enable the kernel to identify patterns that are similar but appear in different keys (e.g. C major vs. D major), as a sequence of notes may be shifted in terms of pitch throughout songs. Intuitively, performing wraparound encodes the notion of pitch being circular, as shown in Fig. 4. Wraparound was performed on-line (i.e. during training) in order to save space, as pre-wrapping around all of the data would have increased the size of the training data by almost three times.

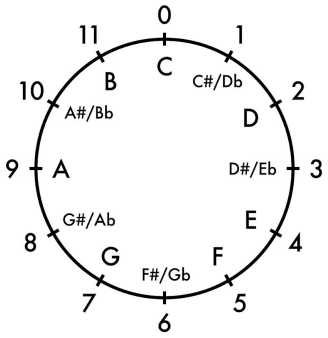

Fig 4: A diagram showing how pitch can be described to be "circular" [1].

To perform wraparound (visualized in Fig. 5), three copies of the original tensor were concatenated together in the pitch dimension. Then, the excess was trimmed off of each end of the concatenated tensor. (As an example, for the smaller kernel which had size 4 in the pitch dimension, 3 wrapped-around pitches were kept on each side. Therefore, 9 pitches were trimmed off of each side.)

1D convolution was also performed on the rest tensors and the length tensors. The feature maps from all of these convolutions represent the locations where the NN recognized patterns that appeared in the music. and are reshaped and concatenated in order to be fed into the FFNN.

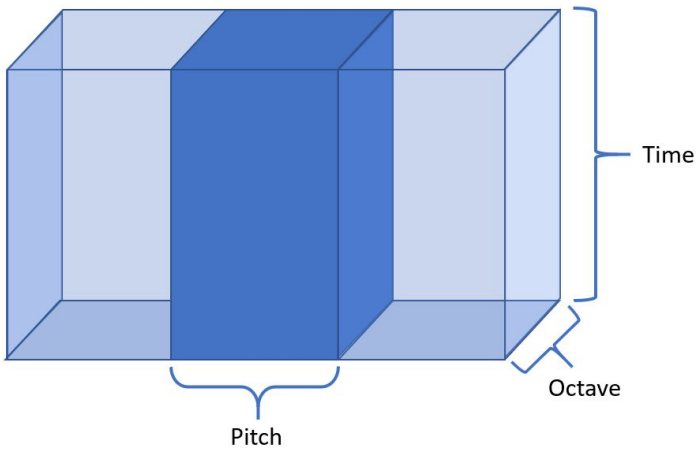

Fig. 5: Diagram demonstrating the wraparound performed on the note tensor in the pitch dimension.

The feature maps from all convolutions are then concatenated and fed into an FFNN, which outputs probabilities for each note for the next timestep. The FFNN consists of 2 fully connected linear layers of size 100 and an output layer. All convolutions and layers used a LeakyReLU activation function, except for the last layer, which used a sigmoid activation. (Sigmoid was chosen instead of softmax because the NN is predicting multiple notes, while softmax is suited only for prediction of a single note.) To obtain the predicted notes from the probabilities, the k most likely notes are chosen. (The user sets k depending on how many simultaneous notes they would like; we found k=2 or 3 worked best.)

Training, Validation and Test:

The models were trained on a set of MIDI files of Bach's compositions. To evaluate performance, we used number of correct notes as a proxy for accuracy. Since we are predicting multiple notes and comparing the prediction to multiple notes, then using a naive accuracy metric (i.e. requiring all predicted notes to exactly match the correct notes) would mask the very common case where some but not all notes are correct. Since we wish to include these partially-correct predictions, the average number of correct notes was used as a proxy.

During the training of the final model, the training and validation curves plateaued at approximately 60%, which can be seen in Fig. 6. Although 60% may seem low, this is reasonable since music by its very nature has many different possibilities, and thus a sequence of notes can be followed by many different plausible notes that would sound equally pleasant.

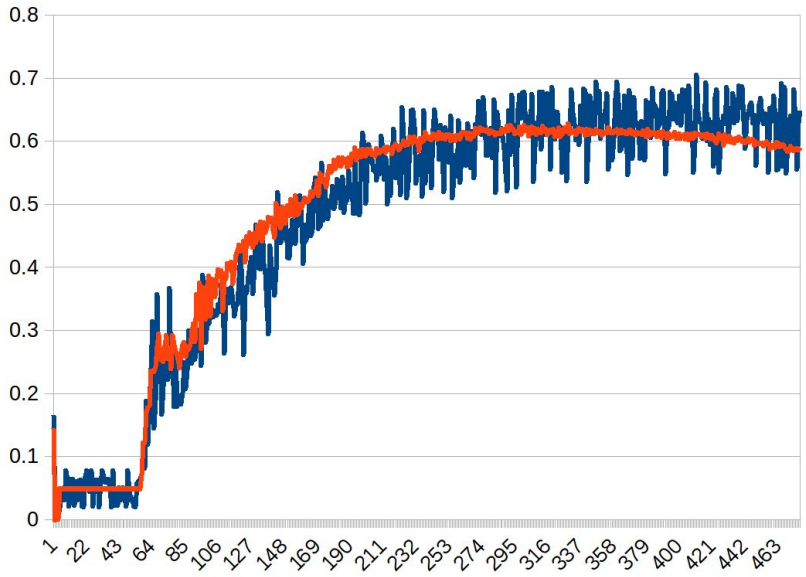

Fig. 6: Graph showing the model's training and validation accuracy over number of epochs trained.

Ethical Issues:

As with any new technology, the AlgoRhythm comes with some ethical concerns. The main concern with the AlgoRhythm is that, if it becomes sufficiently advanced, it might cause musicians and composers to lose their jobs. However, this will not become a possibility for many years, as the AlgoRhythm's capabilities are currently nowhere close to those of even amateur musicians or composers. When, or if, the AlgoRhythm approaches such capabilities, society may be much changed by then, and we would need to reevaluate its ethical impact based on society's attitude towards AI at that time. In the meanwhile, since the AlgoRhythm produces music that is very different from that composed by humans, it can inspire composers to produce unique music based off of the style of the AlgoRhythm's melodies, making it easier and faster to think of new musical themes and creating a wider variety of musical styles.

In this way, people in the music industry can benefit from the improved production speed, and casual musicians now have a fun new way to develop new music.

Key Learnings and Observations:

During the development of the AlgoRhythm, one major lesson learned was that larger models (such as the CNN versions of the AlgoRhythm) can require large amounts of RAM and time to train, especially on a computer without a GPU. Therefore, next time we do a similar project we would first learn how to use GPUs and cloud services such as Google Cloud, to speed up the training process. Another lesson was that in group coding situations, comments are vital to allow each person to understand the purpose of another person's code. Therefore, in future group coding projects, we will strive to provide more detailed comments in our code.

An interesting observation was that, while training CNN3D, the subjective quality of the generated music seemed to correlate well with validation accuracy. In particular,higher the validation accuracy resulted in more pleasing generated chords and greater rhythm variation. This trend held both in the early phase of training, when both validation accuracy and generated music quality increased quickly; as well as during the later phase, when validation accuracy started dropping and music quality decreased along with it. Since validation accuracy only captures whether the predicted note was right or wrong, and does not distinguish between a wrong note that was reasonable and one that was completely implausible, this is a surprising and nontrivial observation and suggests that validation accuracy could be a suitable quantitative metric for how well the model is trained, reducing the need to constantly monitor the generated music and evaluate it subjectively.

References:

[1] Integer-Circle.001.png. [Online] Available: https://davidkulma.com/wp-content/uploads/2016/08/Integer-Circle.001.png. [Accessed: 2-December-2018]