BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI
K K Birla Goa Campus

A REPORT ON

CARNATIC MUSIC NOTE GENERATION USING CRNNs

Sangeeth Jayan
2014B5A30418G

Pratik Manojkumar Manwani
2014B5A30677G

Abstract

This project attempts to use a Convolutional Recurrent Neural Network (CRNN) to generate sequences of notes in response to an input sequence, with the constraint that both the sequences are of the same Raga. The CRNN model is created and trained on both instrumental and vocal data collected from various sources online and tested against similar samples. FFT Spectrogram is used as the feature space and LSTM for the hidden neural network layer.

Introduction

Carnatic Music is a system of classical music native to the states of Southern India. It is one of the two main sub-genres of Indian Classical Music with the other being Hindustani Music. Both the systems share similarities and differences and trace their origins back to ancient Hindu and Islamic influences respectively.

Carnatic music is built up on the fundamental concepts of sruti (the relative base pitch), swaras (the sounds of a individual notes), ragas (the modes or melodic formulae) and tala (the rhythmic cycles). The system is distinct from the extensively studied Western Classical systems in the following ways:

The sruti is relative - it depends on the register of the vocalist, unlike the Western system where the equivalent, known as the Tonic, is fixed. For example, an A4 = 440 Hz. But currently, Carnatic system is also gravitating towards a fixed system with the frequency of Sa being set roughly equal to that of the Western C4

The swaras have contextual frequency values - there is a widely accepted set of swaras that are 22 in number. Depending on the raga, the swaras used can be a subset of this. The 22 swaras are categorized into a groups of 7 notes which is roughly equal to those of the Western system.

Carnatic system is based on natural harmonics - the ratio of any note to the root note is factorizable into simple whole number ratios, unlike the Western system, where the notes follow a geometric progression called the equal tempered scale.

The raga which is roughly similar to the Western scale dictates more than just the notes to be used. It provides information about what notes to be used frequently and what to be used sparingly and also about the types of transitions and ornamentations (gamakas) to be used when moving between notes.

Work done previously in similar areas have enabled computers to produce music. But the key difference here is that most of the data used in those studies are MIDI files which are electronically (and mathematically) defined note representations with little error. And because of the nonexistence of gamakas in Western systems, their use as a means of characterizing a raga has also not been studied.

In this project, we have used actual recordings containing instrumental and vocal data to analyze the songs, and extract features from them, using a Convolutional-NN and then understand the time-progression of notes using a Recurrent-NN and both combined together to form a CRN Network.

Theory

The key idea that drives this project is that a learning system should be able to derive a relation between notes that have elapsed and the ones that have not yet. In that respect the model, after training, would be similar to a delay effect with a negative delay - it should produce notes that would chronologically follow the current note. The challenge here is that those notes do not exist. And this is where the neural network comes into picture. Theoretically, by training on a large dataset of note sequences, the model should be able to understand features that dictate what note sequences should look like, and produce notes within this constraint (aka Raga).

Neural Network

Neural Networks receive an input, and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the “output layer” and in classification settings it represents the class scores.

We used two kinds of neural networks for this purpose:
Convolutional neural networks
Recurrent neural networks

Convolutional Neural Networks

Vanilla neural networks cannot be used for problems that deal with image recognition since spatial shifting and changing scale of the object in the image will not be recognised

Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.

Recurrent Neural Networks

The idea behind Recurrent Neural Networks is to make use of sequential information. In a traditional neural network we assume that all inputs and outputs are independent of each other. But for many tasks that’s a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a memory which captures information about what has been calculated so far. For this project, we use a special kind of recurrent neural networks known as LSTMs (Long Short Term Memory).

Implementation

The implementation of this project involved the following steps:

- Collection of data to training and testing the model.
- Preprocessing of data to make it machine feedable
- Designing of a suitable model for the prediction of note sequences.
- Training and testing
- Recreation of audio files from the predicted data.

Data Collection

For the purpose of training and evaluating the model, audio recordings, all pertaining to one raga was required. The raga we chose (for no particular reason) to work with was Kapi, which has a 5-note monotonic arohana (ascend), and a non-monotonic 7-note avarohana (descend).

Arohaṇa : S R2 M1 P N3 S
Avarohaṇa : S N2 D2 N2 P M1 G2 R2 S

For the sake of convenience, better train-ability and lesser distractions, instrumental music was preferred for the training dataset, though the same was also done for vocal data. All samples for the dataset were taken from the internet as either MP3 or WAV files, and converted to 8000 Hz, 16 bit, Mono wav files for consistency. Noise was removed, and compressed to reduce the prominence of rhythmic percussive elements.

Data Representation

The process of feeding the data into a neural network required the data to be in some representation which is consistent throughout the whole length of the network. Feeding plain wav files was therefore not the best option and would have been a waste of memory and processing as the wav file had a large number of data points (8000 per second) and each point could have had any value within 16 bits of data.
The solution was to use some sort of fixed length vector representation and we chose the Fourier Spectrogram for this.

Fourier Spectrogram

"A spectrogram is a visual representation of the spectrum of frequencies in a sound or other signal as they vary with time or some other variable." (Wikipedia)

The Fast Fourier Transforms algorithm converts data in the time domain to the frequency domain, which being an orthogonal basis, is an good vector representation of the actual data. The Nyquist criteria dictates that the maximum frequency that can be represented by an FFT

$$f_{max} = 1/2\ f_s$$

Where, f_s is the sampling frequency.

With a modest temporal resolution of 10 notes per second (T = 0.1 ms), and a sampling rate of 8000Hz, the FFT algorithm gives a vector of length 400. Analysis of the spectrogram revealed that most of the prominent data lay within 250 Hz (fundamental and a few significant harmonics), and therefore the length of the input feature vector was limited to 256.

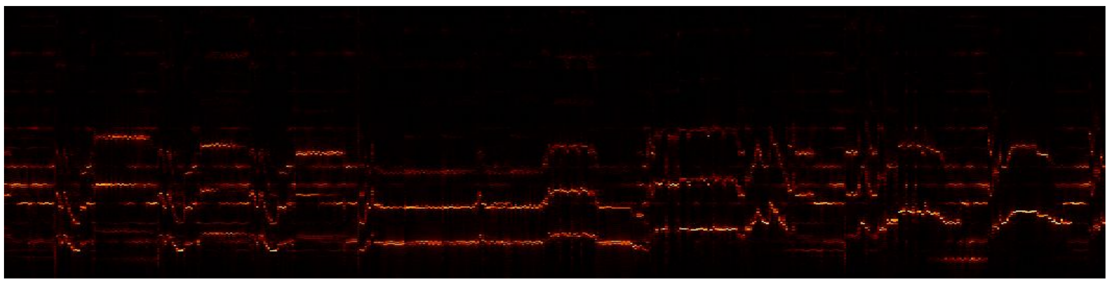

Fig 1: Fourier Spectrogram of a Song in Kapi

The parallel lines are fundamentals and harmonics that follow similar changes in frequencies with time. The x-axis is time and y-axis is frequency (0 Hz-256 Hz)

Preprocessing of Data and Segmentation of Raga Specific Features

An analysis of the spectrogram reveals that the shapes in it follow specific repetitive patterns which are characteristic of the gamakas of the raga. For example, on the left side of Fig1. the pattern can be seen to repeat three times, with three flat note areas and three intermediate transition areas. In the actual audio this corresponds to a note played three times, each one connected to the next by a transition.

For a proper representation of the raga, both the notes and the gamakas need to be categorized. For doing this, the following methods were tried:

Autocorrelation (AC)

"Autocorrelation, also known as serial correlation, is the correlation of a signal with a delayed copy of itself as a function of delay." (Wikipedia)

AC gives the similarity between observations as a function of the delay between them. When operated on a section of the audio file, autocorrelation highlights the fundamental frequency of the section, and for a section of a single note, it gives the actual frequency of the note. This is similar to extracting peaks from a Fourier Spectrogram with the difference being that the autocorrelation gives a better estimate of the note frequency of an interval even when the actual note frequency is not that of the maximum amplitude in the spectrum.

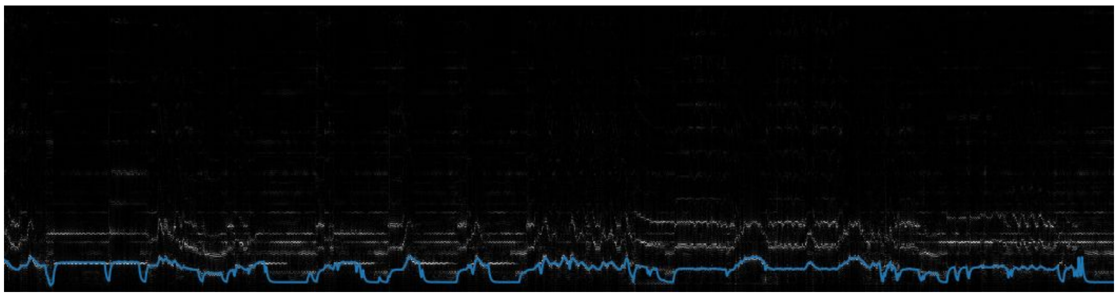

Fig 2: Autocorrelation Output (Blue) of a section of Raga Kapi

The x-axis is time and the y-axis is the frequency of the fundamental of the temporal section. The blue line is seen to follow the white ones which are the fundamentals and overtones of the same song segment.

For segregating the notes, noises, silence periods and the gamakas from each other, some segmentation algorithm was required.

Slope-Based Segmentation

It can be observed from Fig 2. that the AC output has flat areas where notes occur and non-monotonic areas where gamakas occur. Splitting the song into segments based on positions where this difference occurs separates the song into notes and gamakas. The issue here was that due to microtonal variations and vibratos, the algorithm was able to effectively segment the song. It was taking into account even the tiniest flat areas(those atop a transition, etc) and marking sections which should have ideally formed part of a gamaka as different notes in series.

Pairwise Distance Matrix

Pairwise distance matrix is a matrix formed with every element a_{ij} = distance between ith element and jth element of a vector. Here the vectors are a (M x N) matrix, where $M = f_sT$, and T is the total duration of the sample and f_s , the sampling frequency. N = 256. The distance metric used is 'cosine proximity' with

$$a_{ij} = \cos\theta = v_i.v_j / |v_i||v_j|$$

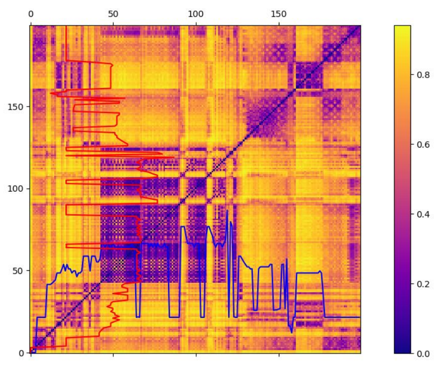

Fig 3: Pairwise Distance Matrix Heatmap overlaid with the Autocorrelation Output

The blue areas are regions with smaller P-Distances or greater similarity. The box patterns indicate recurring note patterns

This approach also suffers from the issue of gamakas being broken down and considered as any other series of notes, but not a single entity.

Due to these reasons, an analytical breakdown of the song into notes and gamakas is erroneous and time consuming. The solution to this is the use of neural networks, CNNs to be specific, which can extract features out of images, like edges and curves. For the purpose feeding an RNN, the entire training dataset is treated as an image and fed into a CNN which produces the features as output, on which the RNN can be trained.

Convolutional Recurrent Neural Network Architecture

The CRNN uses the Keras library atop the Tensorflow backend. The objective of the model is to extract relevant features from the spectrogram and train the time dependence.

Model 1

The first model we used was a Sequential model, with two 1-dimensional Convolutional layers (Conv1D), one Hidden LSTM layer and a Dense Layer. The Conv1D had a 32 filters, and kernel size of 10. The LSTM was configured to return sequences. The Dense layer was TimeDistributed so that there would be one layer for each of the timestep of the input data.

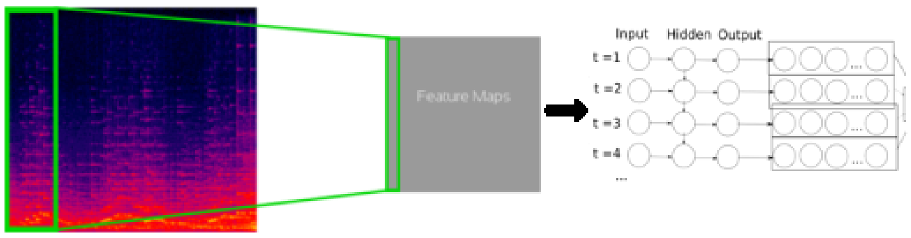

Fig 4: The CRNN Architecture

The Conv1D layer makes a feature map by traversing along the time axis. Its output will be a set of feature all concatenated into a feature map, which is then fed to the LSTM. The output of the LSTM goes to the time distributed Dense layer, the output of which is of the same dimensions as the input. The system is therefore expected to learn and produce output data in the same domain as the input. A heat map of the output should resemble that of the input.

Training Dataset

The training dataset was prepared such that five seconds of the input gives one second of output which is compared against the sixth second of the input for calculation of the loss. The input set moves one second forward and the process is repeated. The interval of 5 seconds was chosen because it is roughly the time required for a transition to go from one note to another in most of the samples. Having more than one gamaka in the same input set should not be an issue as a moving window is used, which splits the gamakas when their positions coincide.

Training and Evaluation

The CRNN model created is trained using the dataset created as described earlier.
For the loss function, “mean_squared_error” was used, with “RMSprop” optimizer with a learning rate of 0.02 and decay of 1.0E-05. The model was trained in batches of 10, upto an epoch count of 100.. Finally the model’s training accuracy and validation accuracy was calculated, and the model saved.

For the implementation of the original objective of the project, the saved model and weights can be loaded and the output predicted for any input of similar dimensions as the training dataset.

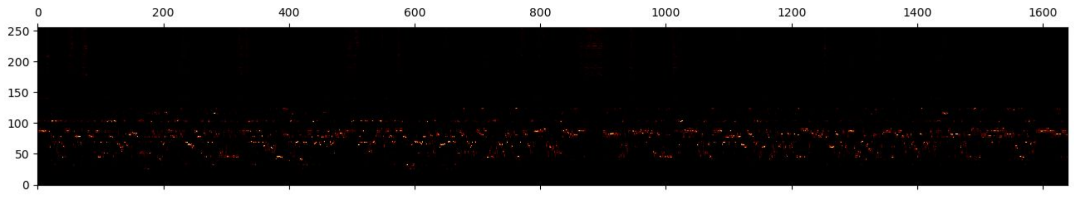

Fig 5: The Predicted Data After Initial Training

The model can be seen to have identified the root/tonic of the raga as a large number of notes are visible around 100 Hz

Recreation of Audio From the Predictions

Since the output is a spectrogram with same number of FFT bins(N = 256), recreation of the audio is the Fourier reconstruction on the basis components. The magnitude of the output waveform at any instant t is

$$y(t) = \sum_{i}^{N} a_i sin(\omega_i t)$$

where, a_i is the magnitude of the ith component of the output vector, and ω_i is the frequency of the ith FFT bin (here, $\omega_i \sim i$) and t is the time, between an interval T = 0.01ms. y(t) gives one waveform every 1/10th of a second.

Results and Conclusions

The model was trained on a set of 5 sample of varying lengths, all of compositions (instrumentals) in the same Raga Kapi. The training accuracy achieved was ~80% and the testing accuracy was ~10%. This could be increased by increasing the size of the training dataset as more and more features characteristic of the Raga would be learnt. When a portion of the training data was given as test input, the accuracy increased significantly. The model was also trained and tested against vocal samples with slightly lesser accuracy.

The program could be extended to include other Ragas as well by having an initial network that classifies the raga and loads the required model and weights accordingly.