MellisAI - An AI Generated Music Composer Using RNN-LSTMs

N. Hari Kumar, P. S Ashwin, and Haritha Ananthakrishnan

***Abstract*—The art of composing music can be automated with deep learning along with the knowledge of a few implicit heuristics. In this proposed paper, we aim at building a model that composes Carnatic oriented contemporary tune, that is based on the features of given training song. It implements the task of automated musical composition using a pipeline where various key features of the resultant tune are constructed separately step by step and finally combined into a complete piece. LSTM models were used for creating four different modules namely, the Tune module, the Motif module, the Endnote module, and the Gamaka module. Four models were built namely - Motif, Tune, End Note, and Gamaka whose training accuracy was 86%, 98%, 60%, and 72% respectively. Our work focuses primarily on generating a user friendly Carnatic music composer that accepts the initial user phrase to compose a simultaneous sequence of notes and motifs for the required duration.**

***Index Terms*—RNN-LSTM, composition, deep learning, Carnatic music, AI.**

I. Introduction

The music we hear is often associated with different emotions or vibes based on the psychological relations the sound has with these emotions. The appeal of a song can be controlled by a few concrete sets of attributes of the musical tune, whereas composing a song oriented to a certain feeling requires taking a lot many dynamic factors into consideration. Although art is irreplaceable, we aim to achieve maximum closeness to the musical interpretation of the human brain. Indian Classical Music boasts of one of the vastest and complex musical structures in the world with a repository of protocols and restrictions on note progressions called *Ragas*. *Ragas* provide unique identities to musical phrases. Each *Raga* has a set of note intervals that every phrase of music must adhere to, which can be interpreted as an analogy to the musical scale system in Western Music. Sounds become music when frequencies start possessing a distinct rhythm or timed melody and the sounds signify a certain message or emotion that creates a mellifluous pattern of notes. Music is treated like a puzzle by the brain which seeks to identify any note or rhythm patterns that give rise to some extent of familiarity within. Determining whether the song possesses appeal, is governed by the process of solving this puzzle. Repeating patterns of note lengths play a major role in making the song familiar as it progresses. These groups of repeating note lengths are known as motifs. Motifs were successfully identified in the training corpora and were passed as one of the four main parameters which determine the quality of the outcome of this experiment.

Intuitively, a song can be split into several phrases with noticeable end notes that have a prominent influence over the sounding of the song. Ending notes are primal to the construction of a phrase, as no matter how many variations of note patterns make up the phrase - it all boils down to how it concludes.

The key difference between western music and Indian music is the presence of *gamakas* [1]. *Gamakas* can be understood as continuous pitch movements from one monophonic note to another. They are represented as a combination of a monophonic note and a fixed note. This transition between notes while gliding over other transient notes engendering rapid pitch changes was achieved in our model through Pitch bends. Using an **RNN-LSTM** technology was ideal since every note in the sequence depends upon the context determined by the previous notes, which basically requires the presence of memory. The number of notes determining the context of the tune also varies depending on the *Raga*. Hence the presence of the **forget gate** enabled the training of determining the size of the context i.e. the number of notes considered while predicting the next note in the sequence.

II. Related Works

Over the past few years Music Information Retrieval has progressed like wildfire with the development of music recognition software like Shazam [2] and AI Music composition software such as AIVA (AI generated emotional tracks) (www.aiva.ai) and JukeDeck (https://www.jukedeck.com/), gradually changing the way the world interprets music. Magenta's ML Jam (https://magenta.tensorflow.org/) is an RNN- based music composition software that can generate both drum grooves and melody. Recurrent Neural Networks(RNNs) pave way for the incorporation of long term dependencies in music composition. In CONCERT [3] an RNN based music composition software, the system makes note based predictions trained by 10 popular Bach pieces. This model fell short of grasping the structure of the entire piece due to lack of memory, popularly known as the vanishing gradient issue. To overcome this shortcoming the use of LSTM

Manuscript received August 25, 2019; revised January 10, 2020.

Hari Kumar is with the Ericsson Research Labs, Ericsson, India (e-mail: n.hari.kumar@ericsson.com).

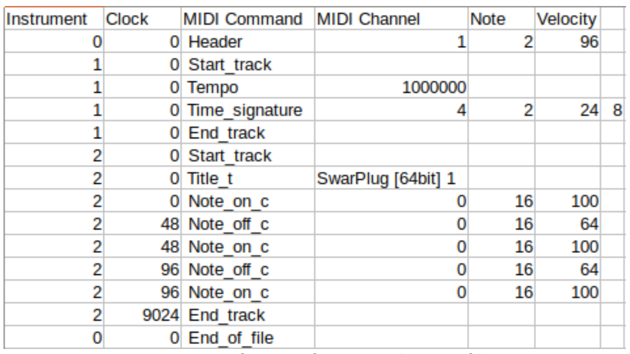
P. S Ashwin and Haritha Ananthakrishnan are with the Computer Science Engineering Department of SSN College of Engineering, Chennai, TamilNadu, India (e-mail: ashwin16015@cse.ssn.edu.in, haritha16038@cse.ssn.edu.in).

(Long short term memory) [4] was advocated and is currently being put into use in computer- generated music. Though these technologies are gaining momentum in the west, the concept of AI-generated eastern classical music is still developing due to the lack of digital musical data for popular eastern pieces. In this paper, we aim to compose Indian classical music with manually crafted data sets in various Carnatic ragas. Extant research in Carnatic music gave us an insight of motif spotting which has been used to identify catchphrases in aalap phrases [5]. Another approach to identify and classify motifs uses dynamic time warping and HMM-based classification applied on a time series of notes [6]. While the mentioned research on motifs is note-based and aims to identify catchphrases, in this paper we have incorporated Motif prediction in the timing domain. The structure of Gamakas has been very efficiently explained as a combination of constant pitch notes and transient notes [7]. Gamakas have rightly analyzed to be a part of a note (svara) and the precision of notes which contain svaras have been studied earlier [8], but not much of research has been done so far on how to predict where Gamakas must be placed in a musical piece and this paper makes a novice attempt at understanding this aspect of Carnatic Music.

III. Data Processing

A. *MIDI Representation of Data*

The ragas that we worked with for our tasks are **Hamsadhwani** and **Mohanam** whose catchphrases were extracted through Raga Surabhi, a Carnatic raga identification platform (http://www.ragasurabhi.com/). The **Hamsadhwani varnam - Jalajakshi** was also MIDI encoded as our training corpus. Digital musical data are represented as MIDI (Musical Instrument Digital Interface) [9]. Music is represented as a series of events represented by MIDI messages that occur at certain time instants. The time is represented using MIDI clock values. Other important information about the song such as tempo and time signature are stored as headers in the MIDI file. Data processing was implemented using the **py_midicsv** library [11] midicsv in python. The first 8 rows of the converted file contain the header information and each of the following rows contain a MIDI message sorted in the order of increasing MIDI clock values.



Instrument	Clock	MIDI Command	MIDI Channel	Note	Velocity	
0	0	Header	1	2	96	
1	0	Start_track				
1	0	Tempo	1000000			
1	0	Time_signature	4	2	24	8
1	0	End_track				
2	0	Start_track				
2	0	Title_t	SwarPlug [64bit] 1			
2	0	Note_on_c	0	16	100	
2	48	Note_off_c	0	16	64	
2	48	Note_on_c	0	16	100	
2	96	Note_off_c	0	16	64	
2	96	Note_on_c	0	16	100	
2	9024	End_track				
0	0	End_of_file				

Fig. 1. CSV format of converted MIDI file.

The CSV file format contains the following attributes (Fig. 1).

– **Track**: Multiple MIDI channel can be present in a single file.

– **Channel**: represents different instruments identified by an unique number.

– **MIDI Clock**: Gives the time instant at which the MIDI event occurred

– **MIDI Command**: Gives the type of MIDI message. MIDI Command has the following values:

- ***Note on c***: Indicates the note being played
- ***Note off c***: Indicates termination of an existing off note.
- ***Pitch bend c***: Gives pitch bend magnitude
- ***End of Track***: Message signifying that the track has ended.

– **Note**: The note played represented by its index number. Lowest note value starts from 0 (Sa note of the lowest octave).

– **Velocity**: The loudness of the note.

IV. Data Cleansing and pre Processing

The universally accepted fundamental unit of music is a **beat** (***talam***), which we have further divided into **steps**, as one beat is divided into **96 MIDI Clocks**. The number of steps making up a bar is determined by the time signature. Depending upon the number of steps per beat, MIDI clock values were quantized to the nearest step and then scaled down to step values. The data set was cleaned by removing noisy channels which did not contribute to the main melody.

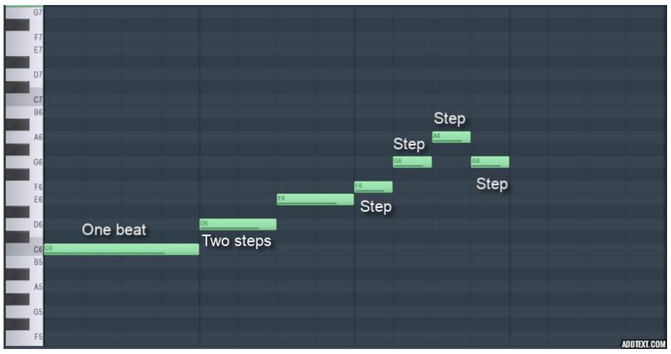


Fig. 2. Units in one bar of music.

An **octave** is a set of 12 semitones that form a perfect interval. Since the musical progression of notes is independent of the octave they are in, all notes were shifted down to the first three octaves to maintain homogeneity. These three octaves were chosen after analysis of contemporary music and human vocal ranges. **Harmonies** were differentiated from the lead tune using note velocities, with the highest velocity being the melody. Notes were made to fill all **empty spaces** present and notes were trimmed without allowing overlap so that we had a sequence of continuous notes without gaps, for the subsequent calculation of note lengths. In *Carnatic* music, every song falls into one of the 12 categories of *Shruti* or scale. **Transpose** refers to the scaling up or down of notes so as to modify the *Shruti* of a song. Table I gives the 12 notes of Carnatic music in comparison to western notes.

In our model, every song present in the training set was transposed, each phrase acting as one separate unit being fed into the training model. This is done with the set of distinctly identifiable end notes that resolve each phrase, giving a conclusion to the previous set of notes. An end note is characterized as the largest note appearing towards the end of a sequence of bars put together as per our logical assumptions.

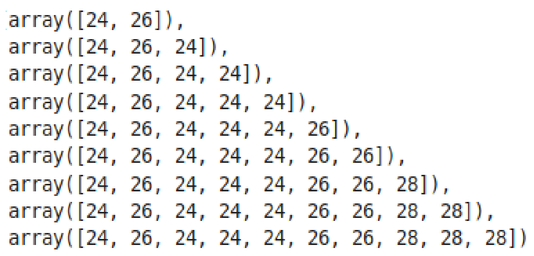
TABLE I: CARNATIC SWARAS IN THE WESTERN MUSIC SCALE

Name	Sa	Ri		Ga		Ma		Pa	Da		Ni	
Carnatic	Sa	Ri1	Ri2	Ga2	Ga3	Ma1	Ma3	Pa	Da1	Da2	Ni1	Ni2
Western	C	C#	D	D#	E	F	F#	G	G#	A	A	# B

V. PROPOSED METHODOLOGIES AND IMPLEMENTATION

A. Tune Model

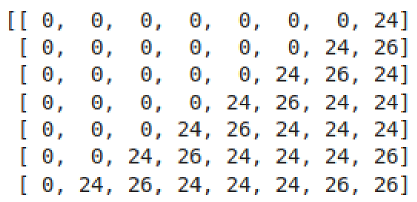
The tune prediction model was constructed to compose a sequence of notes of given length n that resolve into the given input end note. N-gram sequences were generated from phrases starting from $n = 2$, till n = ***sizeOfPhrase***. The size of the input vector (*sizeOfPhrase*) was assumed to be 11 as there are 12 semitones in an octave, and the minimum output span was fixed to be 1. Each n-gram sequence was padded with the required number of zeros to match input vector dimension. Phrases longer than the length of this input vector were fragmented using a sliding window of size 11, which was shifted by one step each iteration. (See Fig. 3)



```
array([24, 26]),
array([24, 26, 24]),
array([24, 26, 24, 24]),
array([24, 26, 24, 24, 24]),
array([24, 26, 24, 24, 24, 26]),
array([24, 26, 24, 24, 24, 26, 26]),
array([24, 26, 24, 24, 24, 26, 26, 28]),
array([24, 26, 24, 24, 24, 26, 26, 28, 28]),
array([24, 26, 24, 24, 24, 26, 26, 28, 28, 28])
```

Fig. 3. n-gram sequence padding.

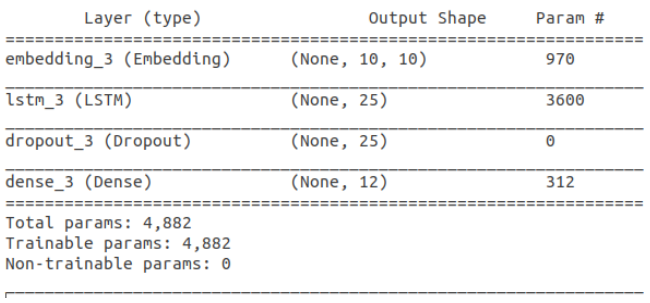
Finally, the padded n-gram sequence was split into X (input) and Y (output) vectors such that last n notes of n-gram sequence would be separated from the n-gram sequence as Applying different output spans, gave us a rough idea of the different kinds of tunes that can be generated by this model. We chose an output span of 3, as we felt it yielded the most favorable results. Each sequence in the total set of n-gram sequences gave rise to a variety of possible combinations in the set of output notes when splitting into X and Y, wherein each combination was labeled with a number. The model was trained with only these combinations of output notes, and hence only the note sequences present in the input song can appear in the tune produced and the similarity to the input song is enforced.



```
[[ 0,  0,  0,  0,  0,  0,  0, 24]
 [ 0,  0,  0,  0,  0,  0, 24, 26]
 [ 0,  0,  0,  0,  0, 24, 26, 24]
 [ 0,  0,  0,  0, 24, 26, 24, 24]
 [ 0,  0,  0, 24, 26, 24, 24, 24]
 [ 0,  0, 24, 26, 24, 24, 24, 26]
 [ 0, 24, 26, 24, 24, 24, 26, 26]
```

Fig 4. Padded N-gram sequences.

The process of selecting an output sequence with respect to class probabilities was randomized to select one from the top three most probable occurrences. This was done to avoid picking the same, most probable class in each iteration which leads to over-fitting and consequently leads to the model constantly composing similar-sounding tunes. The output class is then decoded to retrieve the notes of the melody, and these notes are further appended to the input vector to establish continuity and hence obtain a complete phrase as the output. This process is repeated until the required number of notes are generated.

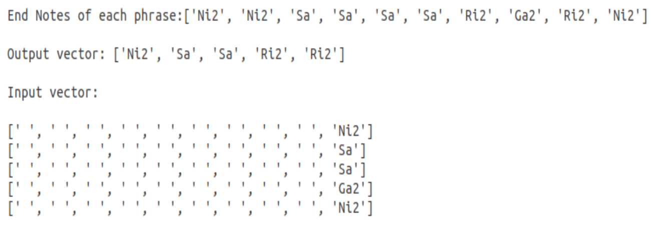


```
       Layer (type)                 Output Shape       Param #
=================================================================
embedding_3 (Embedding)      (None, 10, 10)            970
_________________________________________________________________
lstm_3 (LSTM)                (None, 25)                3600
_________________________________________________________________
dropout_3 (Dropout)          (None, 25)                0
_________________________________________________________________
dense_3 (Dense)              (None, 12)                312
=================================================================
Total params: 4,882
Trainable params: 4,882
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5. LSTM topology description of tune model.

B. End Note Prediction

The end note module was built to make the process of reverse composition by the Tune model more dynamic in nature. As elucidated, the tune module takes the input of an end note and works backward to compose a musical phrase to make the phrase sound sensible and complete.



```
End Notes of each phrase:['Ni2', 'Ni2', 'Sa', 'Sa', 'Sa', 'Sa', 'Ri2', 'Ga2', 'Ri2', 'Ni2']

Output vector: ['Ni2', 'Sa', 'Sa', 'Ri2', 'Ri2']

Input vector:

[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'Ni2']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'Sa']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'Sa']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'Ga2']
[' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'Ni2']
```

Fig. 6. End notes: input and output vectors.

As the most pivotal part of the tune was the end note, we built an LSTM model to enable relevant endnote prediction for each consequent phrase. The whole song was split into bars of 4 beats, where each beat can contain either three or four steps. An end note score was given to each assignment, depicting the relevance of the combination of bars into one phrase. If the score was greater than 0.5, the split was considered to be valid, else the combination of two or more bars was considered to be a phrase, which is repeated until a desirable score is reached. End notes were calculated to be the longest note towards the end of a phrase. This assumption was made after sufficient research done on various Carnatic and Indian film songs, which yielded the most desirable results. (See (1))

$$\text{EndNoteScore}=\frac{\text{Position}}{\text{BarLength}} * \text{BarsAccumulated} \qquad (1)$$

endNoteScore ranges from 0 to 1 and is the deciding factor of the end note. **BarsAccumulated** signifies the count of bars in the phrase chained together so far.

position is taken as the farthest note which is longer than a beat. Some phrases have a set of introductory notes that are present in the bar preceding the first bar of the phrase. The last few notes of the bar may belong to the following phrase. Hence, **position / lengthOfBar** gives a measure of the length of the note towards the end of a bar.

As the phrase becomes lengthier we are further moving away from the right split and **endNoteScore** is made to be directly proportional to **BarsAccumulated**.

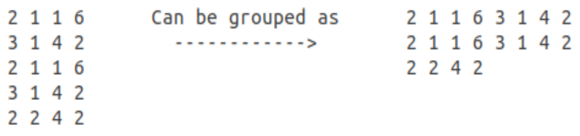
The first note for which *endNoteScore* exceeds **0.5** is taken to be the end note for the corresponding phrase. Once the end notes of each phrase were identified, consequent phrases were paired up to form the X(input) and y(output). X comprised of all the end notes for odd phrases and Y for the end notes of even phrases. To make the working of the model more dynamic there exists a provision for the user to give a leading phrase to begin with, whose endnote will be considered the end note of the predicted song. This way, the semantics of classical music were maintained and aberrations were evaded successfully.

C. Motif Detection

The motif is the smallest structural unit in music, possessing a thematic identity. It is a reflection of the basic feel and idea behind any musical composition, which can be a salient, recurring note or groove pattern. A rhythmic groove is a repeating pattern in any song, which determines it's genre. The motif model aims to compose a groove which can be understood as a vector of note lengths corresponding to the notes generated by the Tune prediction model. Say notes were to appear with the following note lengths (values given in steps) in a set of phrases.

2 2 1 1 2 4 4	-	Phrase 1
2 2 1 1 2 4 4	-	Phrase 2
4 2 2 1 1 4	-	Phrase 3

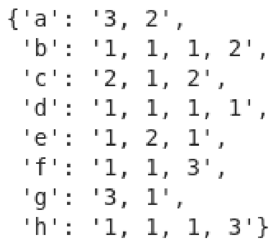
We notice that the pattern, **2 2 1 1** is very common and this can be called as the rhythmic motif of the above 4 lines of music. For model training, the note lengths of all the respective notes are taken. They are grouped into bars which are further grouped by combining the set of notes such that **sum of their note lengths equals 16 steps,** since processed data contains note lengths in units of steps. Due to the unpredictable nature of motifs which can be dispensed across more than one bar, bars were combined so that these are not ignored. At the same time, two separate phrases cannot share a single motif. Hence, we have adopted a different approach to group bars into phrases. Usually, lines in music occur in pairs or triplets, such that each member of the pair or triplet have identical time grooves (see Fig. 7).



```
2 1 1 6      Can be grouped as      2 1 1 6 3 1 4 2
3 1 4 2        ------------>        2 1 1 6 3 1 4 2
2 1 1 6                             2 2 4 2
3 1 4 2
2 2 4 2
```

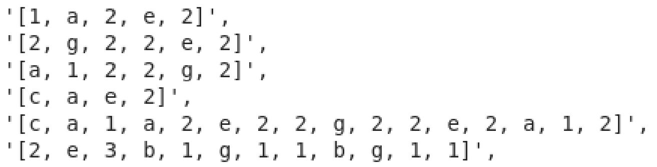
Fig. 7. Motif grouping.

Hence the bars can be grouped into phrases such that the phrases form pairs or triplets with other phrases. The phrases which do not find pairs are allowed to exist as singleton phrases. Motifs that start at the first count of a beat (Samam) and those that occur with a delay of few or more steps sound different and ought to be contrasted. Hence sub sequences are generated starting from positions that are on the beat. Motifs can be of any duration and any number of notes. Hence all on beat sub sequences of all lengths are generated and their number of occurrences are counted. Top n most occurring motifs are finalized, with n being the number of motifs which is a variable that can be changed according to requirements. Each finalized motif is assigned an alphabetical label and their occurrence in the phrases are replaced with their corresponding alphabetical labels.



```
{'a': '3, 2',
 'b': '1, 1, 1, 2',
 'c': '2, 1, 2',
 'd': '1, 1, 1, 1',
 'e': '1, 2, 1',
 'f': '1, 1, 3',
 'g': '3, 1',
 'h': '1, 1, 1, 3'}
```

Fig. 8. Motif nomenclature and note lengths.



```
'[1, a, 2, e, 2]',
'[2, g, 2, 2, e, 2]',
'[a, 1, 2, 2, g, 2]',
'[c, a, e, 2]',
'[c, a, 1, a, 2, e, 2, 2, g, 2, 2, e, 2, a, 1, 2]',
'[2, e, 3, b, 1, g, 1, 1, b, g, 1, 1]',
```

Fig. 9. Different grooves with motifs.

This helps since when the group of notes constituting a motif is substituted by a single alphabet, it becomes a single entity when viewed by the LSTM and this is later decoded to be a vector of corresponding note lengths. The starting note length is arbitrarily picked from all the starting notes of the labeled set of phrases, where the probability of selecting a start note depends upon its frequency of occurrence. Since the labeled phrases are in use, a possibility of the starting note being a motif alphabet is also introduced. When the LSTM predicts a note of odd length, every following even note lengths occur at odd intervals disrupting the groove. As the LSTM cannot possibly learn to bring the groove back to even intervals, when the LSTM generates an odd note length, it is complimented with another odd note based on a certain probability. This probability is calculated based upon the occurrence of odd notes in the input bars.

D. Gamakam Module

Gamakas are ornamentations applied to the basic tune in Indian classical music. They are similar to the usage of **grace notes** in western music. In contrast to western music where the use of grace notes are quite infrequent, their use is much more frequent and laid out with more complexity in Indian classical music. *Gamakams* are variations in the pitch of the note involving oscillations between the primary note and the secondary or transient notes. Each raga lays down a specific framework within which *gamakams* can be used which include what specific notes and specific types of *gamakams* can be used on them.

This module is the final module in the pipeline, which takes the end tune produced by the modules as mentioned earlier and adds *gamakams* according to the notes present in the main tune. There are more than 15 types of *gamakams* in present-day Carnatic music. In the viewpoint of how the model learns, these 15 types can be narrowed down into 4 types (Fig. 10)

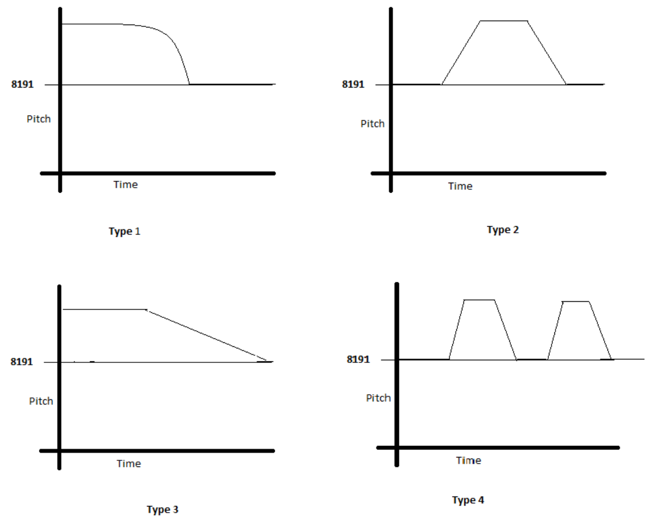


Fig. 10. Types of gamakams.

- **Type 1**: Involves a subtle nudge of the transient note before rapidly returning to the primary note. The pitch stays constant at the transient note initially for a transient period for the transient note's presence to be felt
- **Type 2:** The pitch oscillates once from primary to transient and then returns to the primary note.
- **Type 3:** A slow and sliding transition from the transient note to the primary note.
- **Type 4:** Persistent oscillation between the transient note and the primary note. The oscillation begins with the primary note and ends with the primary note. This can be interpreted as vibrato in western music.

The transition from one note to another in pitch follows a straight line, whose slope can be modified to achieve different desired results. *Gamakams* can be emulated using straight notes which are quick and short spanned. When it comes to adding *gamakams* during the composition stage, the use of the pitch bend facility provides better results. **Pitch bend** signals are given along with corresponding magnitudes at instants of MIDI clocks which add or subtract the overall pitch of the instrument's output equally irrespective of which note is played or the number of notes being played. A pitch bend of magnitude **8191** is equivalent to the change of one-octave pitch. Hence pitch bend of **8191/12** equals one semitone or one note of pitch change.
The required change in pitch is achieved by (2)

$$\text{Change}_{\text{pitch}} = 8191 + \frac{n*8191}{12} \tag{2}$$

where *n* is the number of notes.

The model predicts the class that may appear in the corresponding context and this numerical class label is translated into the *gamakam* note and *gamakam* type. Based on the transient Note, the **pitch bend range** value that would transpose the current note to the *gamakam* note is calculated as follows (3):

$$\text{NumberOfSemitones=GamakamNote} - \text{PrimaryNote}$$
$$\text{pitchbendRange=}8191 + (\text{NumberOfSemitones} * 8191/12) \tag{3}$$

VI. Consolidated Pipeline

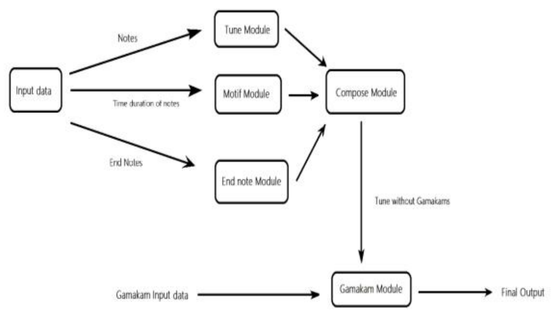


Fig. 11. Pipeline of modules.

The above figure (Fig. 11) depicts our end to end working. The input corpora is MIDI encoded and sent to the Tune module, End Note Module and Motif Module. Different features of the same data set are extracted in these three modules. The End note model extracts the end notes of the given song, and predicts the end notes of N-following bars using it's LSTM model. These predicted notes,form the onset of tune composition. The tune module then extracts the entire note sequence, which is fed into the LSTM Tune model, designed to predict N-notes, reversed based on the predicted End Note. The motif module simultaneously extracts the note lengths from the input dataset, and identifies the most frequent rhythm patterns and generates motifs for the predicted tunes. These three independent modules are amalgamated into one in our Compose method. This tune is then fed into the *Gamakam* module which has it's own pitch bent input dataset of the same raga. The *Gamakam* Module then predicts where, and which type of *Gamakams* need to occur.

VII. Results and Evaluation

The pipeline (Fig. 11) of our project started with the training data, a mixture of various ragas being fed as input. The input was split into phrases and simultaneously fed to the Tune model which extracted the N-gram note sequence, the Motif Model that trained with the patterns of note lengths and finally the End note model whose training set was the phrase's end notes. The output of the combined model was further fed into the Gamakam module. The data was then test-train split and the respective models built, whose test and training accuracy was computed as shown in Table II.

TABLE II: Train and Test Accuracy for the Models with Converging Loss Value

Model	**Train Accuracy**	**Test Accuracy**	**Loss**
Tune Model	91.88%	80.95%	0.2409
Motif Model	63.56 %	75.003 %	0.9964
End Note Model	63.23%	73.45%	0.9878

VIII. Conclusion

This paper elucidates our implementation of generating Carnatic raga based tunes using RNN-LSTMs. We have used four different prediction modules for every aspect of our software and were successfully able to auto generate various phrases of music, depending on the user's input. Future scope for our model includes expanding our data set, incorporating multiple timing and note patterns in various other ragas and research on other deep learning algorithms that could improve our model accuracy. We also believe that there is vast scope for this project in the commercial arena once the prototype can be converted into a product.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Haritha Ananthakrishnan and PS Ashwin actively participated in the implementation of the project and worked simultaneously on separate models and integrated them in the end. Ashwin worked on the Motif and Gamakam modules, and Haritha worked on the Tune model integration and End Note models. Ashwin helped with content procurement for the paper, and Haritha drafted the paper in accordance to the templates and consolidated the outputs and screenshots.

Acknowledgment

We would like to thank Mrs. Vedavalli, a Veena teacher and Music director Kavin for their valuable inputs regarding our project. Mrs. Vedavalli gave us an insight about how to further elaborate our data set with different kinds of songs and ragas and encouraged us to take the implementation further, while Mr. Kavin suggested a few additions to make our product market ready, and to maximize usage by film and music directors.

References

[1] R. S. Jayalakshmi, "Gamakas explained in Sangita-sampradaya-pradarsini of Subbarama Diksitar," Doctoral dissertation, Madras University.
[2] A. Wang, "The Shazam music recognition service," *Communications of the ACM*, vol. 1, 2006.
[3] P. Rao, J. C. Ross, V. Pandit *et al.*, "Classification of melodic motifs in raga music with time-series matching," *Journal of New Music Research*, vol. 43, no. 1, pp. 115-131, 2014.
[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.
[5] V. Ishwar, S. Dutta, A. Bellur, and H. A. Murthy, "Motif spotting in an alapana in carnatic music," *ISMIR*, pp. 499-504, 2013.
[6] I. Simon and S. Oore. (2017). Performance RNN: Generating music with expressive timing and dynamics. [Online]. Available: https://magenta.tensorflow.org/performance-rnn
[7] V. S. Viraraghavan, R. Aravind, and H. A. Murthy, "A statistical analysis of gamakas in carnatic music," *ISMIR*, pp. 243-249, 2017.
[8] V. S. Viraraghavan, R. Aravind, and H. A. Murthy, "Precision of sung notes in carnatic music," *ISMIR*, pp. 499-505, 2018.
[9] International MIDI Association. MIDI musical instrument digital interface specification 1.0. Los Angeles. 1983.
[10] Walker J. Midi-csv. URL: http://www. fourmilab. ch/webtools/midicsv/(h ämtad 2017-05-20). 2008.

N. Hari Kumar is a senior researcher in artificial intelligence at Ericsson Research. He received his bachelor's degree in information and technology from Anna University, India.

He is currently focusing on big data, IoT and machine intelligence technologies. He joined Ericsson in 2008 as a software engineer. He holds 11+ patents in his area of expertise. He is an active member of 3GPP 5G Standardization body.

P. S Ashwin is pursuing her bachelor of engineering at SSN College of Engineering, Chennai, Tamil Nadu India, majoring in computer science engineering.

He has participated and won many national level Hackathons, like the Smart India Hackathon, and VIT Hackathon.

Haritha Ananthakrishnan is pursuing her bachelor of engineering at SSN College of Engineering, Chennai, Tamil Nadu India, majoring in computer science engineering.

Ms. Ananthakrishnan has worked on multiple projects in the domain of machine learning, natural language processing and deep learning, and has published a paper with CLEF in the CEUR-WS journal on Early detection of anorexia and self harm using RNN-LSTMs and SVM Classifiers, Author profiling in Arabic, and Classification of insincere questions for FIRE 2019.