IE643: Deep Learning : Theory and Practice **Aug-Dec 2020**

End-term Project Report : Raga recognition in Indian Classical Music

Team Name: Daedalus et al. *Team Members: 18D170031, 18D100020*

Abstract

This project report contains the details on our project which deals with the problem of automatic *Raga* recognition for any Indian Classical Music recording. We describe an approach involving deep learning and Long Short Term Memory Networks, and describe the data pre-processing and evaluation methods. We have also experimented with a different model using Gated Recurrent Units, and have reported comparable, even slightly superior, results.

1 Introduction

Raga is the melodic framework for all composition and improvisation in Indian Classical Music (ICM). ICM consists of two main branches - Hindustani and Carnatic music - and has a large amount of cultural influence on the Indian cultural landscape as well as contemporary Indian music. *Raga*, however, is not as straightforward an entity as the scales of European Classical Music, and usually takes a large amount of skill to identify. It consists of *svara* (musical notes), *gamaka* (oscillatory movements about a note), *avarohana* and *arohana* (upward and downward melodic movement) and numerous melodic phrases and motifs. A certain *Raga* functions primarily to portray a specific emotion to an audience, and *Ragas* have both melodic and temporal characteristics. Numerous examples show two *Ragas* having a similar set of notes but are extremely different in their musical effect due to their temporal sequencing. The goal of this project is automatic *raga* recognition of musical recordings. This could empower content-based recommendation systems for both ICM and more contemporary Indian music. This could also be extremely useful in classifying large music libraries.

Considering the complexity that this problem demands, deep learning provides a viable solution due to its flexibility, scalability, ability to learn and adapt, and its ability to consider both spatial and temporal data. In particular, LSTM-RNN (Long Short Term Memory - Recurrent Neural Network) models have been proven to be effective in sequence classification and sequence learning tasks due to their ability to retain memory of data from earlier timestamps while learning from new data. *Raga* classification is, at its essence, a form of sequence classification, so the use of LSTMs is an intuitive approach.

We provide a survey of existing literature in Section 2. Our proposal for the project is described in Section 3. We give details on experiments in Section 5. A description of future work is given in Section 7. We conclude with a short summary and pointers to forthcoming work in Section 8.

2 Literature Survey

Some proposed methodologies for the problem statement of automatic *Raga* classification include that which was presented Parag Chordia and Alex Rae [2]. In their paper, the authors describe a process wherein Pitch Class Distributions (PCDs) and Pitch Class Dyad Distributions (PCDDs) are calculated from the audio signals of ICM recordings.

First a substantial database was assembled to include diversity across the *Raga*, musician, instrument, playing style, recording quality, etc. Commercial recordings as well as the recordings of around twenty hours of unaccompanied *Ragas* were included. Each *Raga* sample was labelled with the frequency of the tonic of that recording, and this was done manually with the help of an oscillator. The tonic can be seen as a reference or base note with respect to which every note in a *Raga* is defined. Pitch was detected using a version of the Harmonic Product Spectrum algorithm[3, 13]. Within a recording, pitch was estimated every 10 ms - these estimations were shown to be robust, with occasional octave errors. In some cases there was confusion with the accompanying instrumentals. The onset of notes within each segment were also detected by thresholding a complex detection function[1].

The PCDs were calculated independent of the onsets simply by taking histograms of the pitch tracks. The bins for this histogram corresponded to each note of five octaves of a chromatic scale centered around the tonic for that particular segment. The five octaves were then folded into one and their frequency values were normalised to create the complete pitch class distribution. This was done to nullify significant octave errors. PCDs were able to capture information from notes held for long periods of time, which proved important in the slow *alaps*. They were also prone to noise, however, as they added every erroneous pitch estimate to the histogram.

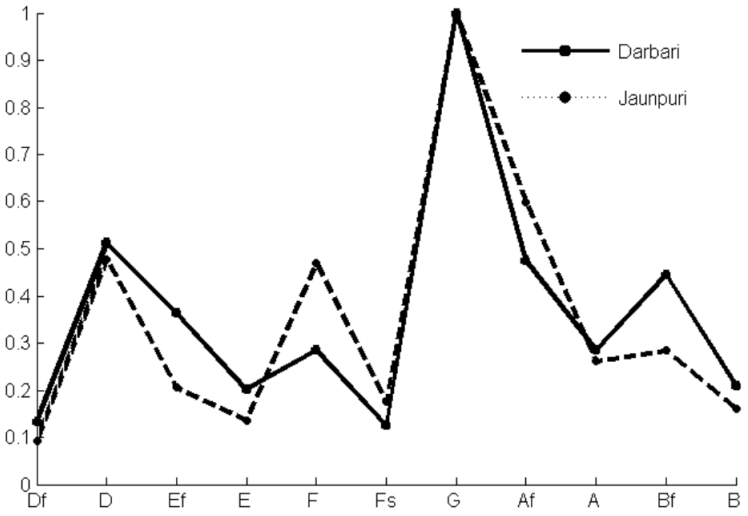

Figure 1: Pitch-class distribution for raag *Darbari* and *Jaunpuri*

PCDDs were determined by using the detected onsets to segment the pitch-tracks into notes. Each note was assigned a pitch class label, and the octaves were folded into one as with the PCDs. The pitch-classes were then arranged in groups of two (dyads). A complication was that PCDDs were unable to reliably account for notes played by sliding up or down from the pitch of the previous note, as in this case, the note onset was not clear.

Success rates were found using 10-fold cross-validation (CV). Classification was also attempted using *Raga* excerpts from different performances than those in the training set, called 'unseen' cases. The label for a sample was selected using maximum a posteriori (MAP) rule. The CV experiment yielded results of 99% accuracy using PCD and PCDD features with a Support Vector Machine classifier. In the case of unseen recordings, the accuracy was 75%.

Another solution is proposed in a paper by Gulati et. al [7], where Time-Delayed Melody Surfaces (TDMS) capture the melodic outline of the *Raga*. One of the major shortcomings of the PCD approach is that it disregards the temporal aspects of the melody which are essential for the characterization of a *Raga*. A

TDMS captures the tonal as temporal characteristics of a melody and also has a musically meaningful interpretation. It is also robust to pitch octave errors. The paper outlines a three-step process to compute a TDMS - pre-processing, surface generation and post-processing.

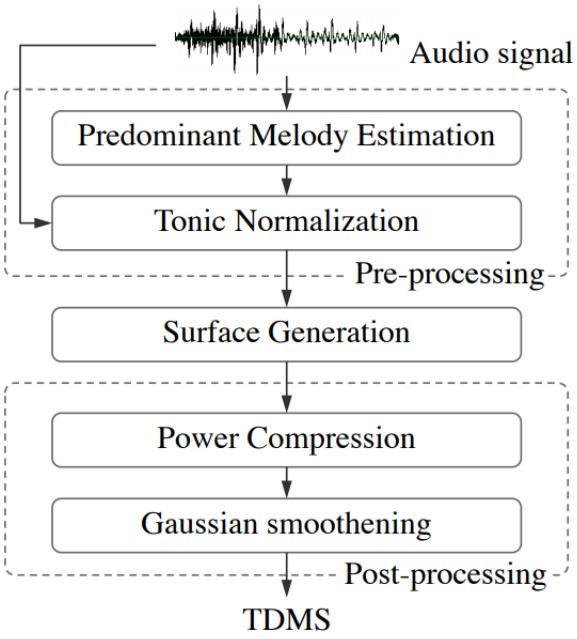

Figure 2: Block diagram for the computation of TDMSs

In pre-processing, the representation of the melody of an audio recording is obtained and normalised with respect to the tonic or base frequency of the piece of music (the tonic pitch of the lead artist). The melody of an audio is represented by the pitch of the predominant melodic source. Then, the predominant melody of every recording is normalised with respect to the tonic pitch ω by the following formula:

$$c_i = 1200 log_2 \left(\frac{f_i}{\omega} \right) \tag{1}$$

for $0 \leq i < N$, where N, is the total number of pitch samples, c_i is the normalised i^{th} sample of the predominant pitch (in Hz).

The two-dimensional surface is generated based on the concept of delay coordinates[9, 14]. A surface $\check{\mathbf{S}}$ is generated of size $\eta \times \eta$ rescursively by computing:

$$\check{s}_{ij} = \sum_{t=\tau}^{N-1} I(B(c_t), i) I(B(c_{t-\tau}), j) \tag{2}$$

for $0 \leq i, i < \eta$, where I, is an indicator function so that $I(x, y) = 1$ iff $x = y$ and is zero otherwise. B is a binning operator:

$$B(x) = \left\lfloor \left(\frac{\eta x}{1200} \right) mod \eta \right\rfloor \tag{3}$$

and τ is a time delay index which is left as a parameter. Here the size $\eta = 120$ is used, corresponding to 10 cents per bin.

In post-processing, element-wise power compression is applied to accentuate values corresponding to transitory regions and reduce the surface's dynamic range:

$$\bar{\mathbf{S}} = \check{\mathbf{S}}^{\alpha} \tag{4}$$

Here, α is an exponent which is left as a parameter. After this Gaussian smoothening is applied by circularly convolving with a two-dimensional Gaussian kernel.

After developing a repository of TDMSs corresponding to each *Raga* or class, classification can be done by using the k-nearest neighbour (kNN) classifier. Three different distance measures were considered to compute the distance between two recordings n and m with TDMS features $\mathbf{S}^n$ and $\mathbf{S}^m$ respectively.

- Frobenius Norm:
$$D_F{}^{(n,m)} = ||\mathbf{S}_n - \mathbf{S}_m||_2 \tag{5}$$
- Kullback-Leibler divergence:
$$D_{KL}{}^{(n,m)} = D_{KL}(\mathbf{S}^{(n)}, \mathbf{S}^{(m)}) + D_{KL}(\mathbf{S}^{(m)}, \mathbf{S}^{(n)}) \tag{6}$$
where
$$D_{KL}(\mathbf{X}, \mathbf{Y}) = \sum \mathbf{X} log\left(\frac{\mathbf{X}}{\mathbf{Y}}\right) \tag{7}$$
where element-wise operations are done and the values are added over all the elements of the resultant matrix.
- Bhattacharya Distance:
$$D_B{}^{(n,m)} = -log\left(\sum \sqrt{\mathbf{S}^{(n)}\mathbf{S}^{(m)}}\right) \tag{8}$$
where element-wise operations are done and the values are added over all the elements of the resultant matrix.

The music collection used is compiled as part of the CompMusic project, and comprises both a Hindustani Music Dataset (HMD) and a Carnatic Music Dataset (CMD), which comprise 130 and 124 hours of commercially available audio recordings respectively. CMD comprises 40 *Ragas* while HMD comprises 30 *Ragas*.

This method was evaluated and compared with the PCD method by Chordia & Senturk (denoted $\mathcal{E}_{PCD}$), along with another method by Gulati et. al involving Vector Space Modelling (denoted $\mathcal{E}_{VSM}$). The results for the TDMS method using the Frobenius Norm ($\mathcal{M}_F$), Kullback-Leibler divergence ($\mathcal{M}_{KL}$)and Bhattacharya distance ($\mathcal{M}_B$) in comparison to the PCD and VSM methods are shown in Figure 3.

Data set	$\mathcal{M}_{\mathrm{F}}$	$\mathcal{M}_{\mathrm{KL}}$	$\mathcal{M}_{\mathrm{B}}$	$\mathcal{E}_{\mathrm{PCD}}$	$\mathcal{E}_{\mathrm{VSM}}$
HMD	91.3	**97.7**	**97.7**	91.7	83.0
CMD	81.5	**86.7**	**86.7**	73.1	68.1

Figure 3: Accuracy (%) of the three proposed variants, $\mathcal{M}_F$, $\mathcal{M}_{KL}$ and $\mathcal{M}_{BC}$, and the two existing state-of-the-art $\mathcal{E}_{PCD}$ and $\mathcal{E}_{VSM}$

This method outperformed the PCD and VSM methods, which were considered state-of-the-art at the time. However, there was considerable confusion among *Ragas* belonging to allied sets which share a common set of *svaras* and similar phrases. There were also some errors in the estimation of tonic pitch.

Another method that makes use of deep learning is one proposed by Sathwik Tejaswi Madhusudhan and Girish Chowdhary[11]. A human trying to identify a *Raga* will often break it up into smaller subsequences as they listen and classify these into a *Raga*. Through data augmentation, this method attempts to do something similar. It also aims to be tonic-independent - while the previously discussed methods depended

on the availability of the tonic pitch of every sample, this method does not. While this method doesn't match up to the previously discussed methods with respect to the accuracy achieved, it offers much potential, as well as a new line of thinking.

This approach uses Convolutional Neural Networks (CNNs). The success of CNNs in text classification demonstrate their capability in handling temporal sequences, and *Raga* identification, as mentioned before, can be viewed as sequence classification.

First, pitch tracking is performed to represent the given audio as an array of frequencies. The audio is a continuous waveform, and to be analyzed as an array of frequencies, discretization is required. This is done by converting a given frequency into the corresponding Musical Instrument Digital Interface (MIDI) Note by using the formula:

$$MIDI = 69 + 12 * log_2\left(\frac{f}{440}\right) \tag{9}$$

The range of 21 to 4186 Hz are then represented by 88 discreet levels. To prevent loss of information, α additional levels (called cents) are defined between two MIDI notes, resulting in $88 * \alpha$ possible levels. Every note is therefore represented as a tuple (M, C), or 'C' cents above the MIDI note 'M'.

The classifier is fed the data in every possible tonic, regardless of whether or not it is available, in order to make the classification independent of the tonic. This is done by a novel data augmentation technique. The sequence of frequencies obtained via pitch tracking is $S = S_1, S_2, ...S_T$ where T is the number of time steps in the audio. S is then converted into a sequence of (M, C) tuples to obtain $MC = (M, C)_1, (M, C)_2$, ..$(M,C)_T = MC_1, MC_2, ..MC_T$. $(M, C)_i$ can then be transformed to $M + C$. If *minv* and *maxv* are the minimum and maximum value observed in the sequence, a suitable step size δ could be selected, and MC could be regularly decremented by this step size until *minv* is reached, or incremented by this step size until *maxv*. For every increment/decrement, a new sequence is obtained which represents how the audio would look in another tonic. Each of these sequences is stored.

The dataset chosen for this purpose is quite small but diverse. The created dataset DBICM has recordings of 8 artists and 10 different tonics. After the data augmentation, 300000 10-second-long clips are obtained. None of the selected recordings from the same *Raga* have the same tonic. To emulate the way a human would classify *Raga*, subsequences are repeatedly sampled ($MC_{sub} = MC_i, MC_{i+1}, ...MC_{i+1000}$), so that i is selected randomly) and the model is trained on these subsequences. Due to the small size of the training data, overfitting was a serious concern, and a high dropout rate was employed to help the model sufficiently generalise the test dataset. It was found that the degree of overfitting was inversely proportional to the value of δ. The network architecture is shown in Figure 4.

Layer Name	Layer_Specifications
Feature Embedding	Embedding Layer (Embedding Size = 80)
Sequential1	1D Conv (kernel = 20, feature maps = 16, Relu activation), BatchNorm 1D, Dropout
Sequential2	1D Conv (kernel = 15, feature maps = 32, Relu activation), BatchNorm 1D, Dropout, MaxPool
Sequential3	1D Conv (kernel = 10, feature maps = 64, Relu activation), BatchNorm 1D, Dropout
Sequential4	1D Conv (kernel = 5, feature maps = 128, Relu activation), BatchNorm 1D, Dropout, MaxPool
Sequential5	1D Conv (kernel = 5, feature maps = 256, Relu, activation), BatchNorm 1D, Dropout
Dense	Fully Connected Layer and Softmax output

Figure 4: Network Architecture

The data had an equal number of samples for each class and was hence quite balanced, and accuracy was taken to be indicative of the model's performance. The model was able to achieve an accuracy of 77.1% on the test data using dataset D1 (3 *Ragas* in 2 tonics, 70000 training instances and 30000 test instances) and 72.8% on the test data using dataset D2 (7 *Ragas* in 8 tonics, 140000 training instances and 70000 test instances).

Our approach to solving the problem at hand is inspired by the solution offered by Sathwik Tejaswi Madhusudhan and Girish Chowdhary[12] which uses LSTMs. This approach used a method of pitch tracking, tonic normalisation, random sampling, and then feeding to an LSTM network follwed by fully connected layers to get class scores for *Ragas*. This method beat the preformance of the TDMS approach, which produced state-of-the-art performance on the Carnatic Music Dataset (CMD) at the time. The comparison of the accuracies achieved via the LSTM method (SRGM1 and SRGM Ensemble) with those of the previous methods (TDMS, VSM and PCD respectively) is shown in Figure 5. SRGM1 is the simple model, while SRGM-Ensemble was created using 4 sets of weights for the same model architecture, obtaining the outputs of these 4 models and combining them by summing the log of the outputs to obtain the final prediction vector for a sample. These results have been evaluated across two datasets, CMD-10 which consists of only 10 *Ragas* from CMD, and CMD-40 which consists of all 40 *Ragas* from CMD.

Method	**CMD-10 Ragas**	**CMD-40 Ragas**
SRGM1	95.6%	84.6%
SRGM1 Ensemble	**97.1**%	**88.1**%
$\mathscr{M}_F$	-	81.5 %
$\mathscr{M}_{KL}$	-	86.7 %
$\mathscr{M}_B$	-	86.7 %
$\mathscr{E}_{VSM}$	91.7 %	68.1 %
$\mathscr{E}_{PCD}$	82.2 %	73.1 %

Figure 5: Summary of the performances of various models on the CMD dataset and its 10 Raga subset

Methods such as that with PCDs ignore temporal qualities of the melody entirely, while others heavily rely on pre-processing of the data and feature extraction. The LSTM method is more flexible in this regard. It takes into account the time-based characteristics of a recording and only uses pitch tracking and tonic normalisation before random sampling to the network. It also uses only a single LSTM block at the core of its architecture and is simpler, more intuitive and easier to implement than several of the previously discussed methods.

3 Methods and Approaches

As a part of our method, *Raga* recognition is treated as a sequence classification problem solved through LSTM (Long Short Term Memory) Networks. RNNs (Recurrent Neural Networks) can be thought of as multiple neural networks stacked back-to-back, with the output of one treated as the input to the next. LSTMs are a type of RNN, but they are better than RNNs at appreciating long-term dependencies, or when two pieces of data across a large timestamp correlate. They are able to retain memory of previous data longer and thus mimic the human brain slightly better - as humans often learn new data in the context of previous learning. Since *Raga* is such a complex temporal sequence, LSTMs are especially helpful.

In most of our experiments, we have tested the classification problem only considering 10 of the 40 *Ragas* available in the Carnatic Music Dataset. This is due to our memory and processing power constraints - training to classify between a larger number of classes would take too much time given the capabilities of our machines. Each *Raga* has 12 recordings associated with it. These recordings have undergone preprocessing as elaborated in Section 4 , and 200 randomly sampled subsequences from each recording are taken from this processed data to to train the model. Our model has also been built in accordance with the recommendations of our reference paper, with a few changes where information was not provided. The LSTM layer is at the core of the model, and we have taken a single LSTM layer with a hidden size of 768. Prior to the LSTM layer is an embedding layer which converts each note in the subsequence to a

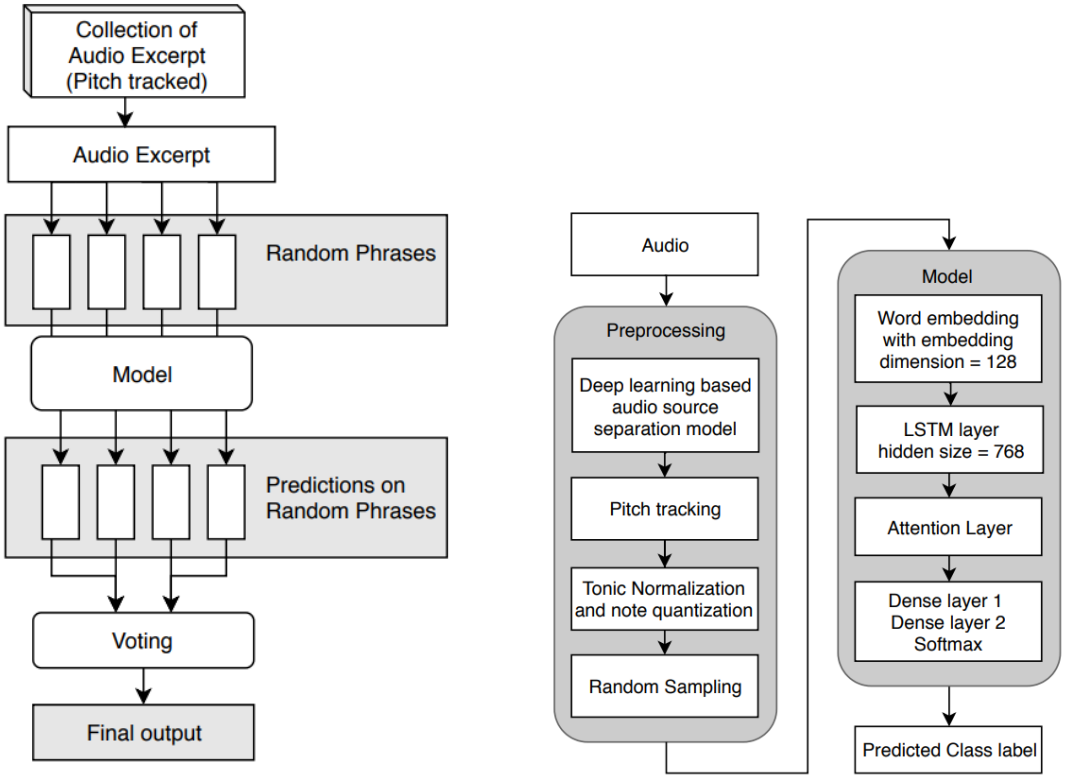

Figure 6: Overview of the *Raga* Inference Process and Model

128-dimensional vector. The LSTM layer is followed by an attention layer[5]. The attention layer gives the model the ability to focus on a subset of notes in a subsequence which are the most important to its classification, i.e. it selects specific inputs. This is followed two fully connected layers with ReLU activations, the first of which has 384 hidden units and the second of which has 10 (equal to the number of classes). Dropout is employed after the first fully connected layer to aid the generalisation of the model. This is then followed by a softmax layer to convert the output into a probability distribution.

Our model has been trained over 30 epochs, with a batch size of 40 (40 subsequences are fed to the model with each training step) and an initial learning rate of 0.0001. The epochs and batch size were decided using the ballpark of the parameters mentioned in our reference paper, while also taking into account our computing power and memory constraints, as we did not want training to happen over too long a time period (our model took a little over 3.5 hours to train). As the classification problem involves more than one class, the loss function used was the Cross Entropy Loss function, computed as follows:

$$CCE = -\sum_{i=1}^{N} y_{i,j} * log(p_i) \tag{10}$$

where N is the number of classes, $y_{i,j}$ is an indicator function which is 1 when $i = j$, j is the training label for a given subsequence and p_i is the i^{th} value in the probability vector.

The Adam optimizer[10] is used to update the gradients during the training phase. After obtaining the predictions for all 200 subsequences relating to a single audio, voting is performed to determine the majority class prediction. We took the threshold for a majority to be 60% - that is, if more than 60% of the pre-

dictions for the subsequences of a particular recording point to the true *Raga* label, we label the classification as correct. If less than 60% of these predictions give the true *Raga*, then the classification is deemed incorrect.

3.1 Work done before mid-term project review

Prior to the mid-term project review, most of our efforts went into thoroughly reading the paper from which we drew our major inspiration, as well as two related papers, and understanding all aspects of the problem statement as well as the various methods used to tackle it. RNNs were previously unknown to us, and we understood not only the basic models of these, but also advanced models such as LSTMs and GRUs. We learned the basics of the Pytorch framework to make it easier to use ahead, and also started work to procure the usable dataset. The dataset was available in a hierarchical online directory, and its procurement as well as its conversion into a processed form that we could use had to be handled in Python. We also started modelling the LSTM network architecture as described in Mudhusudhan and Chowdhary's paper in Pytorch.

3.2 Work done after mid-term project review

After the mid-term project review, we progressed with the construction of the model in Pytorch and consolidated the dataset. The code for to access the files in the dataset and manipulate them to accomplish all pre-processing goals was written in Python. This included code to form the pitch tracked arrays in numpy, normalise them with respect to their respective tonic frequency, and randomly sample subsequences of fixed length from the given arrays and store them as training and test data. The construction of the model was also completed in Pytorch. The model in the paper was used as a basic reference framework, but specifics were modified to account for limited storage space and computing power. The training code was then written in Pytorch to train the model across 30 epochs, as well as the code to perform the majority voting to determine the predicted class label. GPU support was obtained to train the model, and training was completed and results were reported. Additional experiments were also done, like testing various voting thresholds at the time of determining the predictions, and substituting the LSTM block with a GRU. We also experimented with classification on all 40 ragas by training our model for 10 epochs (required around 6 hours) and achieved promising results. The literature survey was also written, for which an additional paper related to the problem statement had to be read and understood. The project report completed with descriptions of our results and conclusions.

4 Dataset Details

Madhusudhan and Chowdhary's paper uses the Carnatic Music Dataset made available by the CompMusic group. This dataset consists of high-quality recordings of ICM concerts, that is, 124 hours of 480 commercially available recordings. These recordings are all stored in an mp3 format. This collection consists of 40 different *Ragas*, with 12 full-length recordings per *Raga*. They feature numerous artists and a wide variety of compositions and the *Ragas* are also diverse in their melodic attributes.

The ICM recordings include vocalists accompanied by various instruments, but for the purposes of *Raga* recognition, analysing just the vocals is sufficient. Therefore, Mad-TwinNet[4] (a deep neural network based approach of recovering vocals from a single channel track) is used to separate the vocals. The pretrained model made publicly available by the paper's authors was used.

This vocal track is then pitch tracked using the Python API[8] for Praat[6], an open source software for

speech and sound analysis. This converts the audio into a series of pitches in Hertz against the timestamp of their occurrence in the audio. We obtained the pitch tracked data for the Carnatic Music dataset from the CompMusic site. This contained files with the pitch tracked arrays of all 480 recordings, along with files containing a single value representing the tonic for each recording.

The next stage in preprocessing is the tonic normalisation and note quantisation. Here, the pitch tracked array must be normalised with respect to the tonic of the recording, which is provided with the dataset, using the formula:

$$f_n = 1200 log_2 \left(\frac{f}{T} \right) \tag{11}$$

where f is the frequency after the pitch tracking in Hz, T is the frequency of the tonic in Hz and f_n is the tonic normalised frequency.

The above equation results in 100 levels of frequency in a half step of music, but in the paper being consulted it was observed that merely 5 were enough. Therefore, the expression for the tonic normalised note we used in our pre-processing was:

$$f_n = 1200 log_2 \left(\frac{f}{T} \right) * \frac{k}{100} \tag{12}$$

where k is the number of desired levels in a half step, which we took as 5.

We did not utilise every recording in the dataset for majority of our experiments, but only used the recordings of 10 *Ragas*. Hence our classification problem is that of only 10 classes. Furthermore, the pitch tracked arrays are extremely long, with over a million notes in some cases, and also varying depending on the length of the recording. The paper refers to this as well, and mentions the need to train the network on randomly sampled subsequences, and expresses that an ideal length for these is 4000-6000. We have taken our subsequence length as 5000, and have randomly sampled 200 subsequences from each audio recording. The paper suggests about 400 samples per recording but owing to our resource constraint, we decided to go with half the data. As there are 12 recordings per *Raga* and we have included 10 *Ragas* from the 40, we end up with 24000 subsequences of 5000 steps each. Out of these 12 recordings in a *Raga*, we have taken 9 as the training data and 3 as the test data. Therefore, 90 recordings are sampled (18000 subsequences) for training and 30 recordings (6000 subsequences) for testing. For the last experiment with all 40 ragas, we use the same strategy of 200 subsequences per sample.

5 Experiments

We used the GeForce GTX 1080 Ti GPU for training the models in all our experiments.

After having successfully trained the model as described in the paper using an LSTM, the first experiment that was done was changing the LSTM block of the model to a GRU (Gated Recurrent Unit). The LSTM consists of a cell state, which can be thought of as a conveyer belt running through all the LSTM cells and a hidden state. The information to be removed and added to the cell state is regulated by structures known as gates. They consist of a sigmoid neural net layer and a pointwise multiplication function, and they are a way of optionally letting information through. An LSTM cell has an input (to decide what to update), forget (to decide what information to retain) and output gate (to decide what to output).

A GRU combines the forget and the input gates into a single update gate, and merges the cell state and hidden state into a reset gate. The operations among these are also modified accordingly. As the number of gates are reduced, the resulting model is far less complex. It is also approximately as good as an LSTM

despite the reduction in complexity and is generally used when training needs to be done quickly and with decent accuracy, or if there are infrastructural constraints. Our experiment involves replacing the LSTM block of our model with a GRU in Pytorch. We hypothesized that this could speed up training without compromising on accuracy. When implementing the GRU model, all other model parameters and layers were kept the same as before.

We also experimented with different voting majority thresholds on the final sample predictions of the model, to see their effect on the reported model accuracy. In addition, after we were able to train with 10 classes successfully, we tried with the full 40 classes in the dataset. The data consolidation code had to be changed accordingly - we continued with the use of 200 random samples of length 5000 from each recording, bringing the total number of random samples used in training up to 72000 and those used in testing to 24000.

6 Results

With 60% majority voting (as initially proposed), an accuracy of 93.33% was obtained for our LSTM model on the test set. When the majority voting threshold was reduced to 50%, the accuracy on the classification of the test set rose to 100%. The plot of the average loss of the LSTM model vs the epochs completed is shown in Figure 7. The accuracy reported in the original paper for the LSTM based model on the 10-raga was 95.6%. This was achieved with approximately 400 subsequences per sample and 40 epochs of training. The difference between this and our accuracy (93.33%) might be due to the following reasons:

- We used only 200 subsequences per model due to our resource constraint whereas the authors used twice as much data.
- We trained for only 30 epochs, as opposed to 40 by the authors.
- The authors did not specify which ragas they considered in their 10-raga subset. The discrepancy might be due to us classifying on a different subset of ragas.
- Random sampling of subsequences can also cause variance in performance.

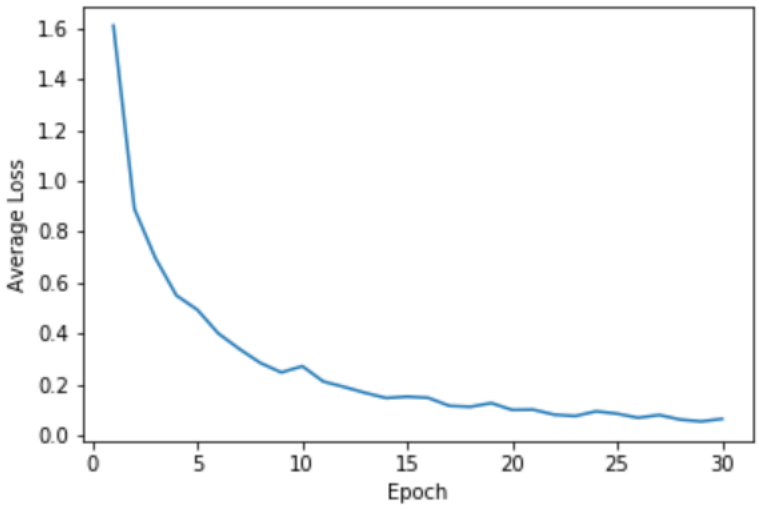

Figure 7: Average Loss vs Epoch for the LSTM Network

When the LSTM block was swapped with a GRU and the voting threshold was kept at 60%, the accuracy remained at 93.33% on the test set. However, there was a case of a test set sample which was previously

misclassified by the LSTM, being correctly classified by the GRU. As the GRU has a simpler architecture and fewer parameters, it did indeed train around 20% faster than the LSTM network. That is, it took 20% less time for this network to achieve the same performance as the LSTM network did after 30 epochs. The loss vs epoch plot of the GRU was also smoother than that of the LSTM. The comparison of these plots relating to the GRU and LSTM is shown in Figure 9. It's clear from the figure that GRU also learns faster than the LSTM, achieving lower losses on similar training epochs.

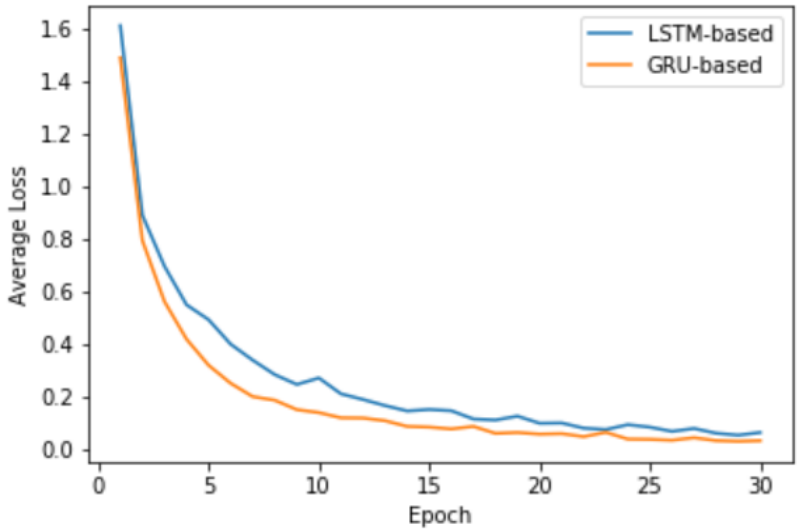

Figure 8: Comparison of the Average Loss vs Epoch plots for the LSTM and GRU Networks

Evidently, the GRU-based network experiences a dip in loss faster and more steadily than the LSTM-network, and, after 30 epochs, it reported a slightly lower loss than the LSTM-based network on the training data.

Our last experiment was classification on the full 40-raga dataset using the LSTM-based network. Our results after only 10 epochs show 62.5% accuracy with the 60% majority voting threshold and 75.83% accuracy with a 50% majority voting threshold. In comparison, the authors report an accuracy of 84.6% accuracy on the 40-raga dataset with the 60% majority voting threshold, after training for 40 epochs, with 400 subsequences (we took 200) for each recording in the training set. Even working with half the data, 10 epochs required about 6 hours, hence we stopped there, but even the initial stages of training show very promising results.

7 Future Work

Things that can be explored to get more reliable and robust performances are:

- We can use bi-directional LSTMs and GRUs to better capture the relationship between musical notes in close vicinity of each other. While a regular LSTM takes into consideration a note's relationship with notes preceding it, a BiLSTM will also take into consideration a note's relationship with notes following it. This will help the model distinguish better between different Ragas
- Current state of the art methods for various NLP problems, especially text classification, use attention based Transformer-based architectures. We can try a similar approach to classify ragas.

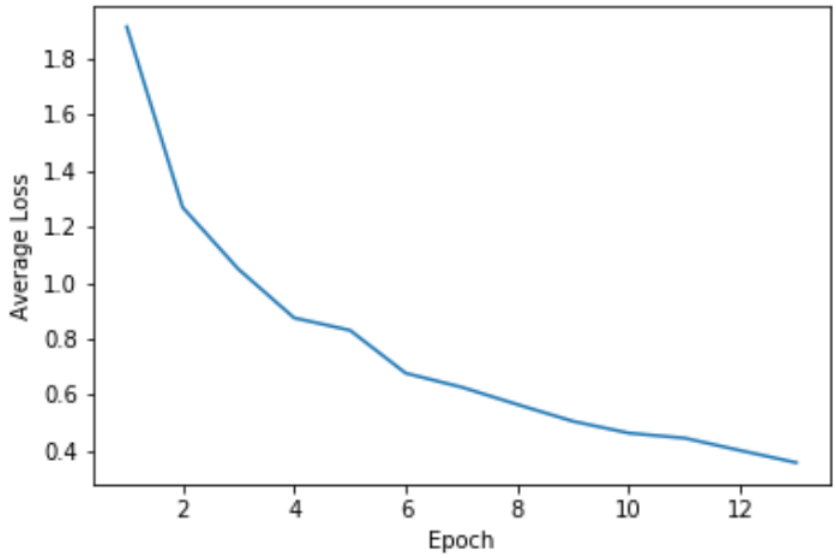

Figure 9: Average Loss vs Epoch plot for the LSTM Network with 40 *Ragas*

8 Conclusion

In this project we researched and implemented solutions to the problem statement of automatic *Raga* classification of an Indian Classical Music recording. For any recording supplied, an algorithm or methodology is desired by which the *Raga* of the recording should be automatically identified and reported with minimal error, regardless of the complexity of the melody, the number of instruments or the temporal variations. After exploring the previous work done in this field involving Time-Delayed Melody Surfaces, Pitch Class Distributions and Convolutional Neural Networks, we took inspiration from a solution involving Long Short Term Memory networks, which are a class of Recurrent Neural Networks. We also aimed to implement our own modifications in efforts to improve the performance of the proposed algorithm. We procured and pre-processed an ICM dataset comprising 40 *Ragas* and used this to train an LSTM-based neural network which achieved 93.33% accuracy while identifying the *Ragas* of the test samples if voting threshold was to be taken as 60%, and 100% accuracy with a 50% voting threshold. The modification we proposed was substituting the LSTM with GRU, which we hypothesized would reduce the parameters and complexity of the network, therefore reducing training time without compromising on accuracy. Our expectations were met, as the GRU-based model achieved the same accuracy on the test set while also taking around 20% less time to reach the same level of loss as the LSTM network had after 30 epochs. Therefore, GRUs offer a reliable method of achieving the same classification results with a less complex network architecture and training time. We also attempted training the network using the full 40-*Raga* dataset. Despite not being able to carry out training past 10 epochs, the model achieved 63.5% accuracy with the 60% majority voting threshold and 84.6% accuracy with the 50% majority voting threshold. The result for the model in our reference paper for 40 *Ragas* after training over 40 epochs and with twice the amount of data was 84.6% with the 60% majority voting threshold, so we took these results to be promising and an indication of the correctness of our approach. Future work that can be carried out to improve *Raga* classification methods are the use of bidirectional LSTMs and GRUs to relate data in both linear directions rather than just front-to-back, as well as the use of Transformer-based architectures, to carry on the theme of similarity between this problem statement and those of Natural Language Processing.

References

[1] M. Davies C. Duxbury, J. P. Bello and M. Sandler. A combined phase and amplitude based approach to onset detection for audio segmentation. In *Proc. of the 4th European Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS–03)*, pages 275–280.

[2] Parag Chordia and Alex Rae. Raag recognition using pitch-class and pitch-class dyad distributions. In *Proceedings of the 8th International Conference on Music Information Retrieval*, 2007.

[3] Patricio de la Cuadra, Aaron Master, and Craig Sapp. Efficient pitch detection techniques for interactive music. In *Proceedings of the International Computer Music Conference*, 2001.

[4] Konstantinos Drossos, Stylianos Ioannis Mimilakis, Dmitriy Serdyuk, Gerald Schuller, Tuomas Virtanen, , and Yoshua Bengio. Mad twinnet: Masker-denoiser architecture with twin networks for monaural sound source separation. In *International Joint Conference on Neural Networks (IJCNN)*, 2018.

[5] Kyunghyun Cho Dzmitry Bahdanau and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. 2014.

[6] Paul Boersma et al. Praat, a system for doing phonetics by computer. In *Glot international*, 2002.

[7] Sankalp Gulati, Joan Serrá, Kaustuv K Ganguli, Sertan Şentürk, and Xavier Serra. Time-delayed melody surfaces for raga recognition. 2016.

[8] Yannick Jadoul, Bill Thompson, and Bart De Boer. Introducing parselmouth: A python interface to praat. In *Journal of Phonetics*, 2018.

[9] H. Kantz and T. Schreiber. *Nonlinear time series analysis*. Cambridge University Press, 2004.

[10] Diederik P Kingma and Jimmy Ba. Adam. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[11] Sathwik Tejaswi Madhusudhan and Girish Chowdhary. Tonic independent raag classification in indian classical music. 2018.

[12] Sathwik Tejaswi Madhusudhan and Girish Chowdhary. Deepsrgm - sequence classification and ranking in indian classical music with deep learning. 2019.

[13] Xuejing Sun. A pitch determination algorithm based on subharmonic-to-harmonic ratio. In *A pitch determination algorithm based on subharmonic-to-harmonic ratio*, 2000.

[14] F. Takens. Detecting strange attractors in turbulence, warwick 1980. In *Dynamical Systems and Turbulence*, pages 366–381.