CLASSIFYING HINDUSTANI INDIAN RAGAS WITH CONVOLUTIONAL NEURAL NETWORKS

BY: RISHOV S. CHATTERJEE

May 14, 2019

Introduction

I have been exposed to Hindustani classical music ever since I was small. I was raised in an Indian household to immigrant parents and my mom is a vocalist and an exponent of Indian classical music. Indian classical music is categorized into two distinct forms: Hindustani and Carnatic, which are practiced in North and Southern India. Unlike western classical music, Indian classical music is very old form and typically doesn't have clear structures but largely depends on the performers or instrument players own elaboration of a melody. Indian classical music is defined by two basic elements – it must follow a Raga (classical mode), and a specific rhythm, the Taal [1]. Most compositions follow a Raga and I have noticed that even experts sometimes have difficulty of telling which Raga a particular song or composition is based on. This is particularly challenging for novices or beginners. Being a data science major, I quickly became attracted to this problem of "Raga detection". My intuition said that machine learning algorithms and techniques could help classify a composition into a main Raga on which it is based. Thus begins my journey to explore and hence this senior thesis.

I will mainly focus on North Indian form which is referred to as the Hin-

dustani classical music. Compositions in Hindustani classical music also are based on a drone, i.e., a continual pitch that sounds throughout the concert, which is tonic [2]. This drone acts as a point of reference as the performer is expected to come back to this home base after a flight of improvisation. The variations and complexity in Hindustani music stems from its use of notes that comprise a Raga. There are seven main musical notes (also called swaras) – Sa, Re, Ga, Ma, Pa, Dha and Ni – along with five intermediate notes (flats and sharps) referred to as "vikrit swaras". The seven notes are referred to as Shuddha and belongs to the saptak (a scale). The flat notes are called "komal" and the sharp notes are called "teevra". A raga consists of at least five notes, and each raga provides the musician with a musical framework within which to improvise [3, 4 5]. The specific notes within a raga can be reordered and improvised by the musician. Ragas range from small ragas like Bahar and Shahana that are not much more than songs to big ragas like Malkauns, Darbari and Yaman, which have great scope for improvisation and for which performances can last over an hour. Each raga traditionally has an emotional significance and symbolic associations such as with season, time and mood [6]. The raga is considered a means in Indian musical tradition to evoke certain feelings in an audience. Hundreds of raga are recognized in the classical tradition, of which about 30 are common [7].

The swaras in a raga can be played in three octaves, the first or lower octave starting from 130 Hz, then middle octave starting at 260 Hz; and upper octave from 520 Hz. The artists are allowed to improvise over the definitions

of raga to create their own renditions. If you listen to two performance of the same raga, they may sound strikingly different to novice ears, though they still retain the rules and defining qualities of ragas.

The rest of the thesis is organized as follows. In Chapter 2, we take a closer look at ragas to understand certain nuances and patterns they exhibit. In Chapter 3, we discuss Librosa, a python package for audio and music signal processing. In Chapter 4, we cover background and related work done on identifying Indian ragas using machine learning and other methods. Chapter 5 will go over the concepts associated with convolutional neural networks. Chapter 6 represents the methodology for raga classification using Convolutional Neural Networks (CNN). Chapter 7 presents the details of the dataset and image data generation. In Chapter 8, I present the data preprocessing steps for the CNN algorithm to be used for raga detection. In Chapter 9 I present the results and analysis of this project. Finally, I conclude in Chapter 10 with a summary of findings and future work.

Indian Classical Ragas

Raga can be identified by various parameters. The particular choice of notes, Ascending and Descending sequences (known as arohana and avarohana pattern), nature of inflexion on different notes (gamaka/meend), characteristic phrases (pakad) all can be helpful to classify a raga. These are further described below:

1. Choice of Notes

   A rāga has a given set of notes (swaras), on a scale, ordered in melodies with musical motifs. The Indian tradition suggests a certain sequencing of how the musician moves from note to note for each rāga, in order for the performance to create a rasa (mood, atmosphere, essence, inner feeling) that is unique to each rāga. Theoretically, thousands of rāga are possible given 5 or more notes, but in practical use, the classical tradition has refined and typically relies on several hundred. For most artists, their basic perfected repertoire has some forty to fifty rāgas [8 -10]. Each raga has a different set of swaras that constitutes it. There must be the notes of the rag. They are the allowd swar. This concept is

similar to the Western solfege. There must also be a modal structure. This is called that in North Indian music and mela in Carnatic music. There is also the jati. Jati is the number of notes used in the rag. Rāga in Indian classic music is intimately related to tala or guidance about "division of time", with each unit called a matra (beat, and duration between beats) [11]. A rāga is not a tune, because the same rāga can yield an infinite number of tunes [12]. A rāga is not a scale, because many rāgas can be based on the same scale. Each raga tends to have a "Vadi" swara, a king swara on which maximum focus is given in a performance [1]. It is also known as the most frequently occurring swara in a particular raga. It is followed by Samvadi (next in importance), then Anuvadi. The swaras that are not allowed in a particular raga are known as Vivadi swaras (enemy notes).

2. Arohana/Avarohana

There must also be the ascending and descending sequence of notes. This is called arohana /avarohana. Arohana and avarohana are the descriptions of how the raga moves. The arohana, also called aroh or arohi, is the pattern in which a raga ascends the scale. The avarohana, also called avaroh or avarohi, describes the way that the raga descends the scale. Both the arohana and avarohana may use certain characteristic twists and turns.

3. Pakad

The pakad or swarup, is a defining phrase or a characteristic pattern for a raga. This is often a particular way in which a raga moves; for instance the "Pa M'a Ga Ma Ga" is a tell-tale sign for Raga Bihag, or "Ni Re Ga M'a" is a telltale sign for Yaman. Often the pakad is a natural consequence of the notes of arohana / avarohana (ascending and descending structures). However, sometimes the pakad is unique and not implied by the notes of the arohana /avarohana. It is customary to enfold the pakad into the arohana / avarohana to make the ascending and descending structures more descriptive.

4. Gamakas

Gamakas are better known as ornamentations used in Hindustani music system. These are inflexions and rapid oscillatory movements taken across swaras.

We now take one common raga as a running example and explain how the notes behave with respect to the above definitions and terms. Yaman emerged from the parent musical scale of Kalyan. Considered to be one of the most fundamental ragas in Hindustani tradition, it is thus often one of the first ragas taught to students. Yaman is a heptatonic (Sampurna) Indian classical raga of Kalyan Thaat. Yaman's Jati is a Sampurna raga.

Arohana: Sa Re Ga Ma(Kori Ma/tivra Ma i.e. Ma#) Pa Dha Ni Sa'

Avarohana: Sa' Ni Dha Pa Ma ((Kori Ma/tivra Ma i.e. Ma#)) Ga Re

Sa

The ascending Aaroha scale and the descending style of the avroha includes all seven notes in the octave (When it is Shadav, the Aroha goes like N,RGmDNS' , where the fifth note is omitted; Pa but the Avaroha is the same complete octave). All the scale notes (called swaras) in the raga are Shuddha, the exception being Teevra Madhyam or prati madhyamam.

Vadi and Samavadi

Vadi is G (ga) and Samvadi is N (ni).

Pakad or Chalan

Kalyan has no specific phrases or particular features, many musicians avoid Sa and Pa in ascend or treat them very weakly. You will often hear N0 R G M+ D N S' in ascent and S' N D M+ G R S in descent). Sa is avoided in beginning the ascend such as N0 R G M+ P D N S'. To summarize the rules that make a raga:

A raga has at least 5 notes, including the Shadja. More than 5 is definitely ok. We can then add a further complexity, by choosing different swara in the ascent (Aaroha) and the descent (Avaroha). So the number of combinations can then be 5-5, 5-6,5-7,6-5,6-6,6-7,7-5,7-6 and 7-7. Five is Odhav, six is Shadhav and 7 is Sampoorna. So, an odhav-sampoorna raga will have 5 swar in the aaroha and 7 in the awaroha.

Librosa: Processing Musical Information in Python

The research field of music information retrieval (MIR) has been evolving rapidly. Taken broadly, it covers the area of musicology, digital signal processing, machine learning, information retrieval and library science. As digital music service platforms such as iTunes, Spotify and Pandora have grown, so has been the need for MIR tools which to date has been largely written by programmers as scripts using C++ or MATLAB. In recent years, interest has grown within the MIR community in using (scientific) Python as a viable alternative [ref]. LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems. Here I provide a brief introduction to basic ideas and elements in libROSA as I have used this package in conducting bulk of the work in my thesis.

In general, librosa's functions tend to expose all relevant parameters to the caller. While this provides a great deal of flexibility to expert users, it can be overwhelming to novice users who simply need a consistent interface to process audio files. To satisfy both needs, they define a set of general

conventions and standardized default parameter values shared across many functions.

An audio signal is represented as a one-dimensional numpy array, denoted as y throughout librosa. Typically the signal y is accompanied by the sampling rate (denoted sr) which denotes the frequency (in Hz) at which values of y are sampled. The duration of a signal (d) can then be computed by dividing the number of samples (y) by the sampling rate (sr):

$$d = \frac{y}{sr}$$

```python
# duration_seconds.py
from librosa import load
import os
# Getting all mp3 files of raga Bhairav from dataset
fileList = os.listdir("data/Hindustani/mp3/Bhairav")
# Creating dictionary to store number of samples and sampling rate
sampleDict = {}
# y is number of samples, sr is sampling rate of the audio file
for file in fileList:
  y, sr = load(file)
  sampleDict[file].append(y)
  sampleDict[file].append(sr)
  # calculating signal duration
  duration = y / sr
  sampleDict[file].append(duration)
```

By default, when loading stereo audio files, the librosa.load() function down mixes to mono by averaging left- and right-channels, and then resamples the monophonic signal to the default rate $sr = 22050Hz$. Most audio analysis methods operate not at the native sampling rate of the signal, but over small frames of the signal which are spaced by a hop length (in samples). The default frame and hop lengths are set to 2048 and 512 samples, respectively. At the default sampling rate of 22050 Hz, this corresponds to overlapping frames of approximately 93ms spaced by 23ms. Frames are centered by default, so frame index t corresponds to the slice:

```
y[(t*hop_length - frame_length/2):(t*hop_length+frame_length/2)]
```

where boundary conditions are handled by reflection padding the input signal y. For analyses that do not use fixed-width frames (such as the constant-Q transform), the default hop length of 512 is retained to facilitate alignment of results.

The majority of feature analyses implemented by librosa produce two-dimensional outputs stored as numpy.ndarray, e.g., S[f, t] might contain the energy within a particular frequency band f at frame index t. We follow the convention that the final dimension provides the index over time, e.g., S[:, 0], S[:,1] access features at the first and second frames. Feature arrays are organized column-major (Fortran style) in memory, so that common access patterns benefit from cache locality. By default, all pitch-based analyses are assumed to be relative to a 12-bin equal-tempered chromatic scale with

a reference tuning of A440 = 440.0 Hz. Pitch and pitch-class analyses are arranged such that the 0th bin corresponds to C for pitch class or C1 (32.7 Hz) for absolute pitch measurements.

Core Functionality

The librosa.core submodule includes a range of commonly used functions. Broadly, core functionality falls into four categories: audio and time-series operations, spectrogram calculation, time and frequency conversion, and pitch operations [ref]. For convenience, all functions within the core submodule are aliased at the top level of the package hierarchy, e.g., "librosa.core.load" is aliased to "librosa.load".

Audio and time-series operations include functions such as: reading audio from disk via the audioread package [ref], resampling a signal at a desired rate, stereo to mono conversion, time-domain bounded auto-correlation, and zero-crossing detection. Spectrogram operations include the short-time Fourier transform (stft), inverse STFT (istft), and instantaneous frequency spectrogram (ifgram) [Abe 95], which provide much of the core functionality for down-stream feature analysis. Additionally, an efficient constant-Q transform (cqt) implementation based upon the recursive down-sampling method of Schoerkhuber and Klapuri [ref] is provided, which produces logarithmically-spaced frequency representations suitable for pitch-based signal analysis. Finally, logamplitude provides a flexible and robust implementation of log-amplitude scaling, which can be used to avoid numerical underflow and set

an adaptive noise floor when converting from linear amplitude. Since data may be represented in a variety of time or frequency units, a comprehensive set of convenience functions is provided to map between different time representations: seconds, frames, or samples; and frequency representations: hertz, constant-Q basis index, Fourier basis index, Mel basis index, MIDI note number, or note in scientific pitch notation. Finally, the core submodule provides functionality to estimate the dominant frequency of STFT bins via parabolic interpolation (piptrack) [Smith 11], and estimation of tuning deviation (incents) from the frequency reference A440. These functions allow pitch-based analyses (e.g., cqt) to dynamically adapt filter banks to match the global tuning offset of a particular audio signal.

Spectral Features

Spectral representations are the distributions of energy over a set of frequencies which form the basis of many analysis techniques in MIR and digital signal processing in general. Librosa implements a variety of spectral representations, most of which are based upon the short time Fourier transform. The Mel frequency scale is commonly used to represent audio signals, as it provides a rough model of human frequency perception [ref]. Both a Mel-scale spectrogram and the commonly used Mel-frequency Cepstral Coefficients (MFCC) are provided. By default, Mel scales are defined to match the implementation provided by Slaney's auditory toolbox [ref], but they can be made to match the Hidden Markov Model Toolkit (HTK).

While Mel scaled representations are commonly used to capture timbral aspects of music, they provide poor resolution o pitches and pitch classes. Pitch class (or chroma) representations are often used to encode harmony while suppressing variations in octave height, loudness, or timbre. Two flexible chroma implementations are provided: one uses a fixed-window STFT analysis (chroma _ stft) [ref] and the other uses variable-window constant-Q transform analysis (chroma_cqt). An alternative representation of pitch and harmony can be obtained by the tonnetz function, which estimates tonal centroids as coordinates in a six-dimensional interval space using the method of Harte et al. [ref]. Figures 1 and 2 illustrate the difference between STFT, Mel spectrogram, chromagram, and Tonnetz representations.

Display

The display module provides simple interfaces to visually render audio data through matplotlib [ref]. The first function, display.waveplot simply renders the amplitude envelope of an audio signal y using matplotlib's fill_between function. For efficiency purposes, the signal is dynamically downsampled.

Mono signals are rendered symmetrically about the horizontal axis; stereo signals are rendered with the left-channel's amplitude above the axis and the right-channel's below. An example of waveplot is depicted in Figure 3. The second function, display.specshow wraps matplotlib's imshow function with default settings (origin and aspect) adapted to the expected defaults for

13

visualizing spectrograms. Additionally, specshow dynamically selects appropriate colormaps (binary, sequential, or diverging) from the data type and range [ref]. Finally, specshow provides a variety of acoustically relevant axis labeling and scaling parameters. Examples of specshow output are displayed in Figures 1,2, and 3.

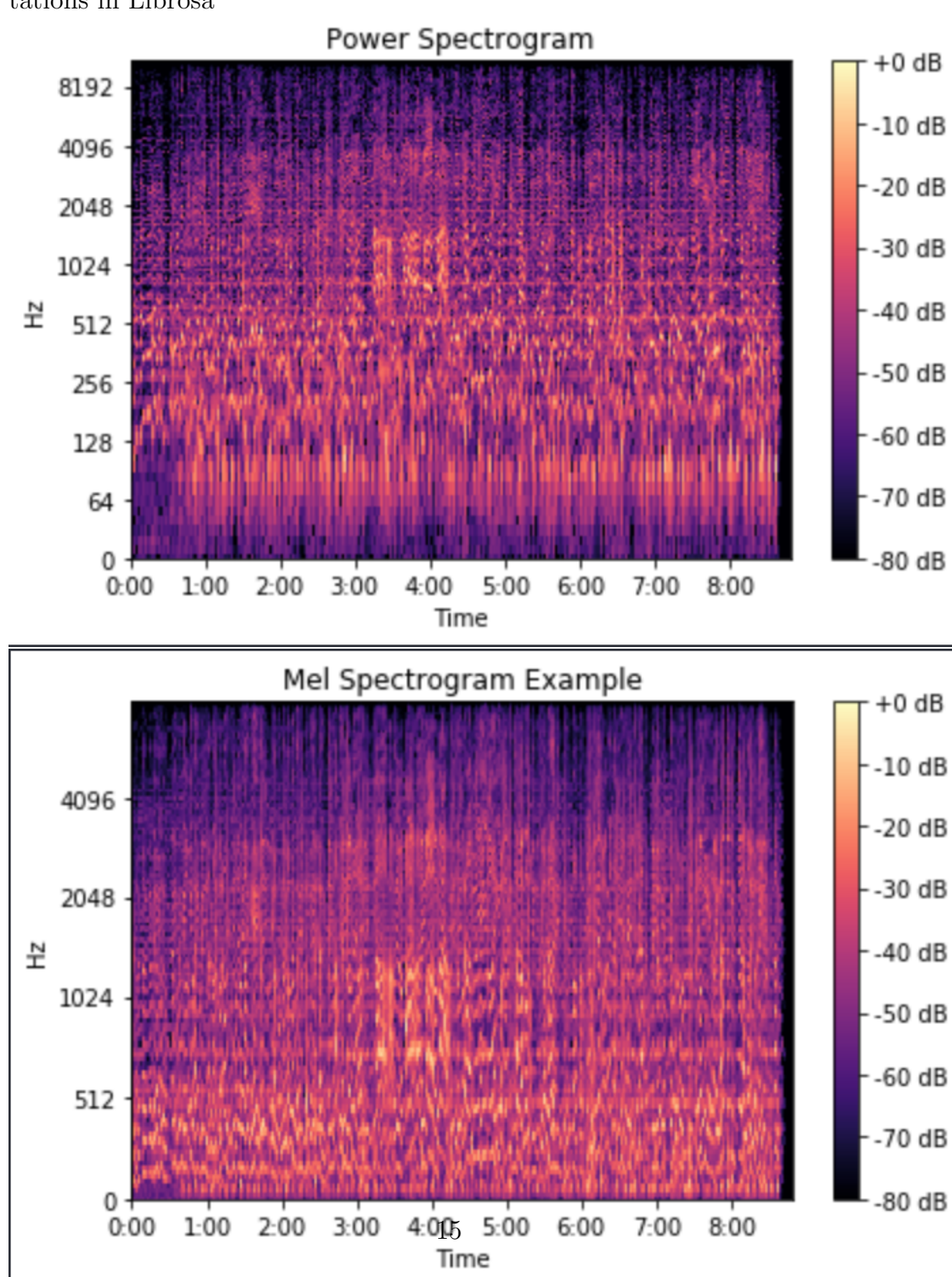Figure 1: Raga Specshow Output for STFT and Mel-Spectrogram Representations in Librosa

Figure 2: Raga Specshow Output for Chroma_CQT and Tonnetz Representations in Librosa
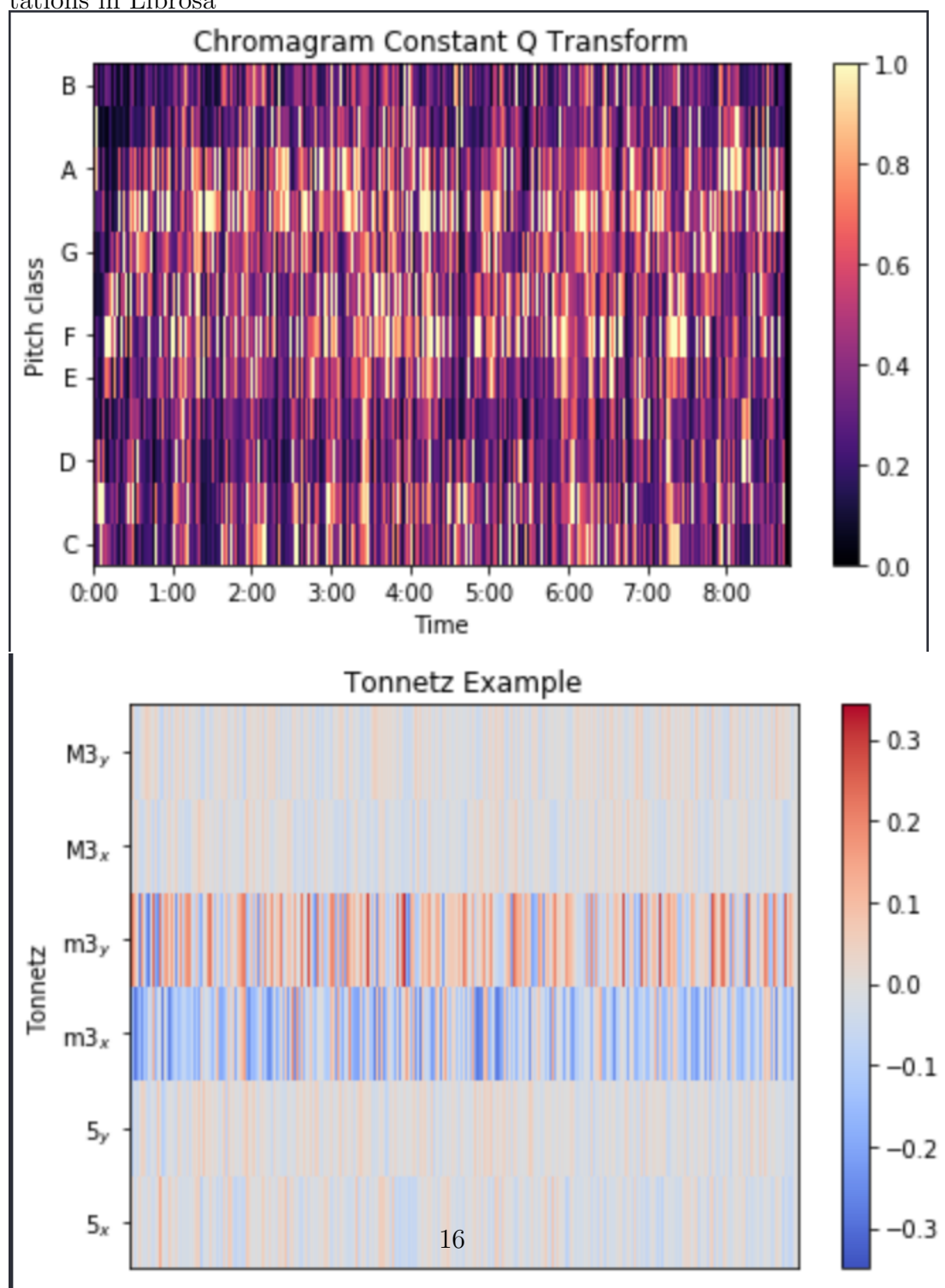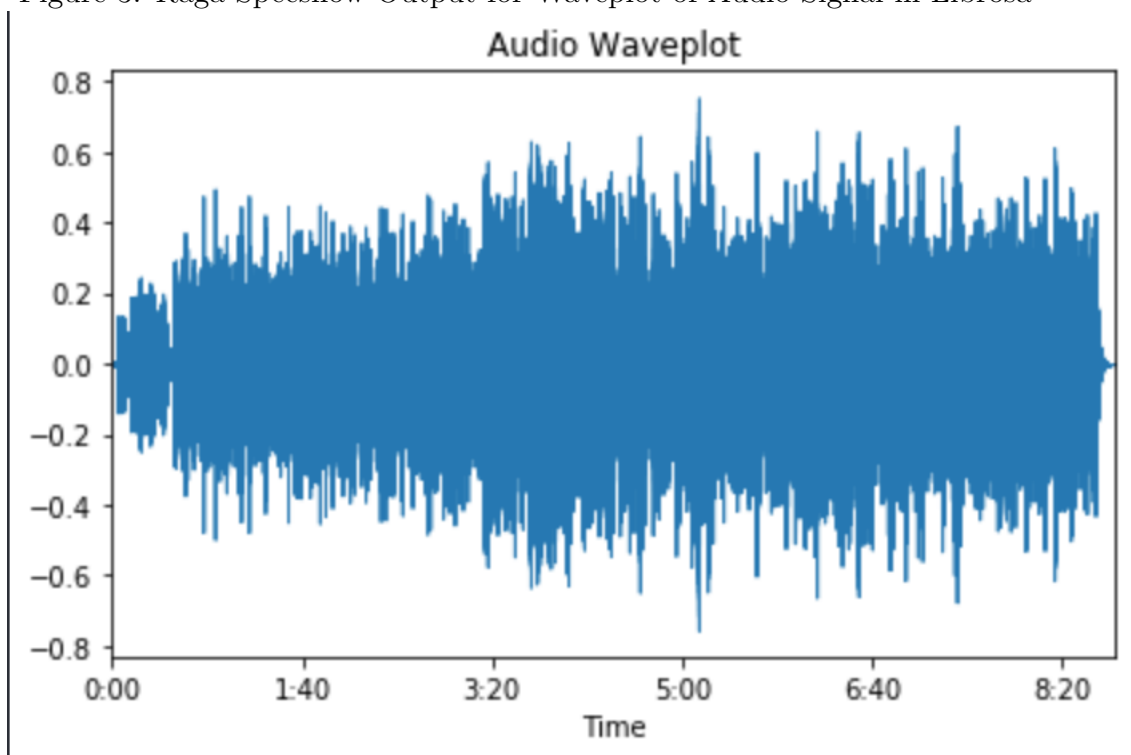

16

Figure 3: Raga Specshow Output for Waveplot of Audio Signal in Librosa

Background and Related Work

Scholars and researchers have used different mechanisms to identify ragas. In a system called Tansen, Pandey [ref] created a system in which a raga is automatically identified based on Hidden markov model. Pendekar et. al., [ref] were able to identify a raga by segmentation of audio signal via spectral flux and thereby identifying raga by using its pitch frequency. Certain other researchers such as Chelpa [ref] proposed a fuzzy set theory for generation of alap patterns. In [ref], Sreedhar et. al., created a database of ragas and used the scale of the raga performance as a similarity metric using nearest neighbors algorithms. Within a scale, notes are matched with the existing sets of notes in the ragas in the database. The closest raga in the database is given as output for the test raga.

Tzanetakis [ref] has also proposed various schemes in the English music classification based on their moods and styles of the performer as well as songs genre classification. Clustering is suggested as the classifier [ref]. Sentiment analysis of movie review based on naïve Bayes and genetic algorithm is suggested in [ref]. Since this methodology depends on the likelihood it can be connected to a wide assortment of spaces and results can be uti-

lized as a part of numerous ways [ref]. Shetty et. al., [ref] has identified raga based upon arohana-avorahana pattern on different ragas using neural network technique.

In a 2015 paper [ref], Sharma and Bali compared several ML classifiers to dataset of music labeled by 4 ragas: Des, Bhupali, Yaman and Todi. The audio performances are converted into .wav extension and chroma features are extracted using MIR toolbox in Matlab. A hop factor of 0.025 second is selected which gives 4719 frames. Then they were able to extract the Vadi swara, also known as the king swara whose magnitude and pitch are relatively greater than notes (swaras). Then they used the WEKA tool which includes a comprehensive collection of machine learning algorithms. The dataset of different ragas is classified using machine learning classifiers in WEKA. Classifiers they used are Random Forest, C4.5, Bayesian network and K-star. After performing comparison of classifiers on ragas, they observed that K-star gives the largest accuracy of 93.38%, on dataset of ragas followed by the random forest with 92.64%.

In another work, Kumar et. al., proposed a method to identify the ragas of an Indian Carnatic music signal. They discuss why this problem is hard due to i) the absence of a fixed frequency for a note, ii) relative scale of notes, iii) oscillations around the note, and iv) improvisations. In this work, they framed the raga classification problem in a non-linear SVM framework using a combination of two kernels that represent the similarities of a music signal using two different features pitch-class profile and n-gram distribution

of notes. This differs from the previous pitch-class profile based approaches where the temporal information of notes is ignored. They evaluated the proposed approach on their own raga dataset and CompMusic dataset and show an improvement of 10.19% by combining the information from two features relevant to Indian Carnatic music.

In a recent article, Ale Koretzky [ref] provides valuable guidance on music signal processing that is helpful for the work that I have to do. Ale addresses the problem of how can we extract vocals out of a mixed track? This is formally known as Audio Source Separation. When we look at a audio source file (,m3 or ,wav), we are visualizing a waveform in the time domain. All we have access to are the amplitude values of the signal over time. While it is possible to extract features like envelopes, RMS values, zero-crossing rate etc., but these features aren't strong discriminators. To extract vocal content from a mix, we must somehow expose the structure of human speech. This is where the Short-Time Fourier Transform (STFT) comes to our rescue.

What the STFT plot shows us are:

i a fundamental frequency (f0) determined by the frequency of the vibration of our vocal cords.

The above example shows Ariana is singing in the 300-500 Hz range.

ii A number of harmonics above f0 following some shape or pattern

iii no unvoiced speech, which includes consonants like 't', 'p,', 'k', 's', breaths,

Figure 4: Short Term Fourier Transform of Vocal Audio Sample

etc.

They manifest as short bursts in the high frequency.

The basic idea now is to figure out some sort of mask that when applied (element-wise multiplication) to the magnitude STFT mix gives us an approximate reconstruction pf the magnitude of STFT of the vocals. If this idea works, then we can begin to look at Convolutional Neural Networks (CNN) and what they have been able to achieve on images. Since STFT exhibits spatial patterns (in the time versus frequency space), CNNs should be able to learn from.

Using CNNs on spectral information extracted from librosa to learn patterns has not been explored much in prior work. This is the approach I am going to take in the thesis. My research question becomes:

Can CNNs learn and classify ragas based on spectral features represented within chromagrams?

Before answering this question, it is first important to review what a convolutional neural network is as well as its underlying mechanics for recognizing images in the next chapter.

Convolutional Neural Networks

To understand Convolutional Neural Networks, we have to understand the following foundational ideas:

- Convolution

- Pooling

- Jargon: padding, stride, filter, etc

In computer vision, we have used diverse techniques in the past on images to do object detection and image classification. One major problem with computer vision problems is that the input data can get really big. Suppose an image is of the size 68 X 68 X 3. The input feature dimension then becomes 12,288. This will be even bigger if we have larger images (say, of size 720 X 720 X 3). Now, if we feed this big input to a neural network, the number of parameters will swell up to a very large number (depending on the number of hidden layers and hidden units). This will result in more computational and memory requirements – not something most of us can deal with.

vertical edges

horizontal edges
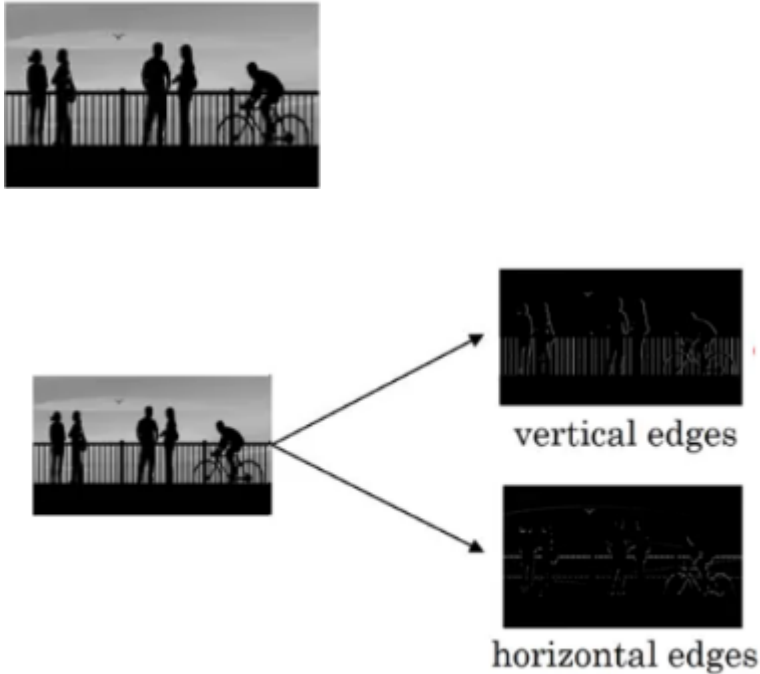
We begin by looking at edge detection as a simple example. The early layers of a neural network detect edges from an image. Deeper layers might be able to detect the cause of the objects and even more deeper layers might detect the cause of complete objects (like a person's face).

Edge Detection Problem

In this section, we will focus on how the edges can be detected from an image. Suppose we are given the figure below:

As you can see, there are many vertical and horizontal edges in the image. Therefore, the first thing to be done is to detect these edges.

How do we detect these edges? The first thing to do is assuming the image is BW, we represent it by a pixel map of gray scale values. Assume it

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

1	0	-1
1	0	-1
1	0	-1

3 X 3 filter

6 X 6 image

is a 6 x 6 matrix.

The values in these cells show pixel values in grayscale. Next, we convolve this 6 X 6 matrix with a 3 X 3 filter. They are also sometime referred to as feature detector or kernel.

We take the feature filter and apply it over the original image. By placing it on the left upper corner, we get:

So, we take the first 3 X 3 matrix from the 6 X 6 image and multiply

3^1	0^0	1^{-1}
1^1	5^0	8^{-1}
2^1	7^0	2^{-1}

0^{1}	1^{0}	2^{-1}
5^{1}	8^{0}	9^{-1}
7^{1}	2^{0}	5^{-1}

-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

it with the filter. Now, the first element of the output (which is a 4 X 4 matrix) will be the sum of the element-wise product of these values, i.e. 3*1 + 0 + 1*-1 + 1*1 + 5*0 + 8*-1 + 2*1 + 7*0 + 2*-1 = -5. To calculate the second element of the 4 X 4 output, we will shift our filter one step towards the right and again get the sum of the element-wise product:

Similarly, we will convolve over the entire image and get a 4 X 4 output:

The shifting of the filter is also called a stride. A stride of 1 shifts it one cell to the right, while a stride of 2 will shift it 2 cells to the right. So, convolving a 6 X 6 input with a 3 X 3 filter gave us an output of 4 X 4. This output is sometimes referred to as feature map. While the above explains what convolve operation does, we will take another example which can clarify how edges can be detected. If high pixel values represent bright areas and low values represent darker areas of an image, then look at this following example.

Higher pixel values represent the brighter portion of the image and the

26

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

*

1	0	-1
1	0	-1
1	0	-1

=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

lower pixel values represent the darker portions. Looking at the 4 x 4 output matrix, we can detect a vertical edge in an image.

In mathematics, functional analysis, convolution is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other. The convolution of f and g is written f∗g, using an asterisk or star. It is defined as the integral of the product of the two functions after one is reversed and shifted. As such, it is a specific kind of integral transform:

$$[f * g](t) = \int_0^t f(\tau)g(t - \tau)d\tau$$

27

Kernel Types

A number of well-known filters or kernels have been used in prior work. Some common ones include Sharpen, Blue, Edge-Detect, or Edge-Enhance.

1	0	-1
2	0	-2
1	0	-1

Sobel
filter

3	0	-3
10	0	-10
3	0	-3

Scharr
filter

The Sobel filter puts a little bit more weight on the central pixels. Instead of using these filters, we can create our own as well and treat them as a parameter which the model will learn using backpropagation. In order to understand the next details of a CNN, let us take a look at the entire series of steps that happens on a single input image.
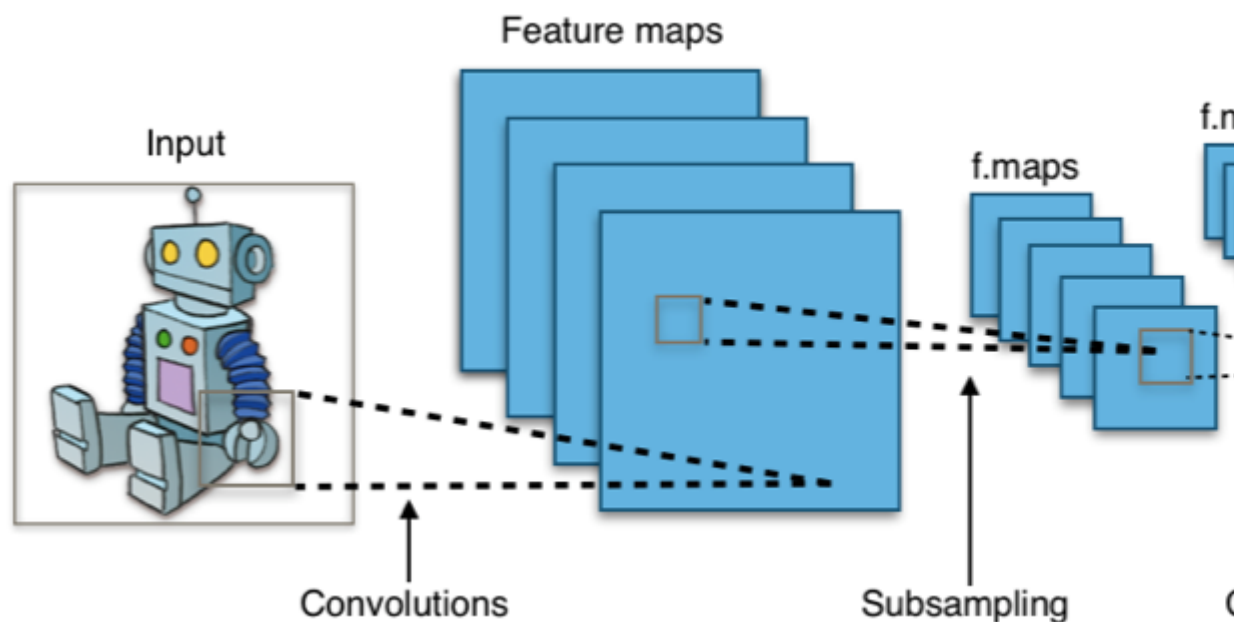
It is important to note that a single image can generate many feature maps using different kernels. We do this so that different features are extracted out. This first step creates a convolutional layer consisting of many feature maps.

Padding in Convolution

We have seen that convolving an input of 6 X 6 dimension with a 3 X 3 filter results in 4 X 4 feature map. We can generalize it and say that if the input is n X n and the filter size is f X f, then the output size will be (n-f+1) X (n-f+1):

- Input: n X n

- Filter size: f X f

- Output: (n-f+1) X (n-f+1)

    There are two disadvantages to this approach:

1. Every time a convolution is applied, the size of the image shrinks.

2. Pixels present in the corner of the image are used only a few number of times during convolution as compared to the central pixels. Hence, not much focus is put on the corners since that can lead to a loss of information.

In order to overcome these issues, we can pad the image with an additional border, i.e., we add one pixel all around the edges. This means that the input will be an 8 X 8 matrix (instead of a 6 X 6 matrix). Applying convolution of 3 X 3 on it will result in a 6 X 6 matrix which is the original shape of the image.

This is where padding is very useful:

- Input: n X n

- Padding: p

- Filter size: f X f

- Output: (n+2p-f+1) X (n+2p-f+1)

There are two common choices for padding.

1. Valid: No padding at all. If we are using valid padding, the output will be (n-f+1) X (n-f+1)

2. Same: Here, we apply padding so that the output size is the same as the input size, i.e., n+2p-f+1 = n ;therefore, p = (f-1)/2

We now know how to use padded convolution. This way we don't lose a lot of information and the image does not shrink either. It is also important to note that the ReLU function (Rectifier Linear Function) is applied during the kernel application step since images are often very non-linear and we

want to ensure that convolution does not create something too linear. Hence applying the ReLU rectifier function helps to solve this problem.

Pooling or Subsampling

Pooling layers are generally used to reduce the size of the inputs and hence speed up the computation. Consider a 4 X 4 matrix as shown below:

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2

Applying max pooling on this matrix will result in a 2 X 2 output:
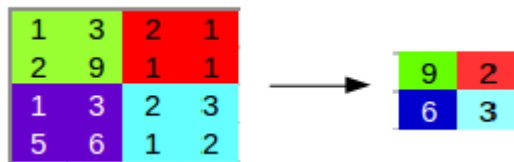
It is called max pooling as for every consecutive 2 X 2 block, we take the maximum number. Here, we have applied a filter of size 2 and a stride of 2. These are the hyperparameters for the pooling layer. Apart from max pooling, we can also apply average pooling where, instead of taking the max of the numbers, we take their average. In summary, the hyperparameters for a pooling layer are:

1. Filter size

2. Stride

3. Max or average pooling

If the input of the pooling layer is nh X nw X nc, then the output will

be:

$$\frac{(nh{-}f)}{s+1} \times \frac{(nw{-}f)}{s+1} \times nc$$

Methodology

Initial Dataset and Image Data

Data Preparation and Processing

Results and Analysis

Future Steps

Bibliography

Appendix