Classifying Ragams in Carnatic Music with Machine Learning Models: A Shazam for South Indian Classical Music

A thesis presented

by

Hari Narayanan

to

Applied Mathematics

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Faculty Advisor: Professor Gabriel Kreiman

Harvard College

Cambridge, MA

April 1, 2024

Abstract

Ragams, analogous to the Western musical concept of mode, are the cornerstone of Indian classical music. Each song in Indian classical music exists within one of a few hundred common ragams. Learning to identify a song's ragam is a core competency developed during an Indian classical music education, and efforts to develop computational ragam identifiers have thus recently gained traction. In this project, I combine two digital audio signal processing techniques with four deep-learning models to determine whether these models are appropriate tools for classifying ragams in real-world concert settings. I obtained promising results, including 98% model testing accuracy when distinguishing between songs in two different ragams, 94% testing accuracy when classifying songs within a pool of ten ragams, and 86% testing accuracy on a pool of fifteen commonly occurring ragams. My results constitute meaningful strides towards the ultimate goal of engineering a Shazam-like application for reliable real-time song classification in four principal ways:

1. **Dataset Assembly:** Compiled a dataset of over 70,000 Carnatic songs, creating a comprehensive training dataset with over 1 TB of audio files, labeled by ragam, enabling novel features to be extracted and bigger models to be trained.

2. **Improved Model Performance:** Designed Artificial Neural Network (ANN), Long Short-term Memory (LSTM), and 2-D Convolutional Neural Network (CNN) ragam recognition models surpassing benchmarks in ragam classification accuracy and capability from previous studies.

3. **Feature Importance Mapping:** Provided model explainability by highlighting the significance of particular audio features in ragam predictions, enhanced by Carnatic domain knowledge, unlike prior black-box approaches.

4. **Novel Model Application:** Introduced transformer (BERT) models for ragam identification in Carnatic music, marking a novel approach in this research area.

Acknowledgements

This thesis is the result of the guidance, love, and effort of many individuals, to whom I am deeply grateful. In particular, I would like to thank the following people:

First and foremost, to my thesis advisor, Professor Gabriel Kreiman, for his counsel and wisdom, ever since my first week as a student with a budding idea in Neuro 240 almost fifteen months ago, all the way up to the finishing touches of this thesis.

To Vikram, without whose constant support this thesis would never have been written.

To all my professors and TFs over the last four years, for giving me the skills to take on this challenge.

To Amma, Appa, and Sriram, for your endless love and support of my studies throughout my life.

To my guru Tara Aunty and my wonderful grandmother Suja Patti, for giving me the lifelong gift of Carnatic music.

To Sumi Periamma, Sundar Mama, Muthukumar sir, and Ravikanth uncle for their invaluable assistance in providing me with their personal Carnatic music collections.

To Elizabeth Batiuk and Kerry Masteller, for providing me with many hours of audio files from the James Rubin collection of Indian Classical Music at Harvard's Loeb music library.

To the boys of Claverly 406, for their unwavering friendship throughout our years at Harvard College, and for many more to come.

And last but not least, to God, for all these people and everyone else in my life.

Contents

List of Figures

Chapter 1

Problem Statement

What are ragams in Carnatic music and why should we be interested in being able to identify them using computational methods? [1]

1.1 What is Carnatic Music?

Carnatic music is an ancient form of classical South Indian music with its origins dating back to the 12th century AD. Characterized by both its highly structured compositions and improvisations, Carnatic music emphasizes vocal performance, although it also includes a wide range of instrumental accompaniments like the violin, mridangam (a double-headed drum), and the veena (a plucked string instrument). The repertoire of Carnatic music is mainly devotional; many compositions are dedicated to Hindu deities. It is deeply embedded in the cultural and spiritual fabric of South Indian society, with performances commonly taking place in temples, concert halls, and during festivals.

[1] Note that I used ChatGPT to help me write and edit some of the language in this chapter.

1.2 What Carnatic music means to me

Carnatic music has always been one of the most significant influences on my life. I am a fourth-generation performing Carnatic musician, as my mother, grandmother, and great-grandmother were all professional Carnatic vocalists and scholars. My great-grandmother, Ananthalakshmi Sadagopan, was in fact one of the first women to sing on the All-India radio back in the 1940s. I have been learning Carnatic music for 17 years, and have given over 40 concerts in the U.S. and India throughout my performing career. At Harvard, I'm completing a citation in Tamil language and serving as the president of the South Asian Music Association (SAMA), an organization for the appreciation and performance of South Asian Music that has been around for almost 20 years now. We perform all over campus and frequently host the world's top Carnatic musicians on Harvard's campus to perform and speak to the community. Apart from all the lessons in music and life that Carnatic music has imbued in me, it has also been a deeply grounding constant that has kept me connected with my culture, community, and spirituality.

With so many Carnatic musicians, teachers, and fans in my family, Carnatic music has always been the language of my dinner table. Over the years, I have noticed one comment that keeps coming up. "Wouldn't it be nice if there were a smartphone app or something that could tell you the ragam of the song you're listening to?"

In my conversations with family and friends, we have always talked about a ragam identification tool as a kind of holy grail of Carnatic music education and appreciation.

This thesis seeks to design such a tool and evaluate its performance.

1.3 What are ragams in Carnatic music?

In Western music theory, the concept of *ragams* from Carnatic music can be seen as complex, primarily because there isn't a direct equivalent in Western traditions. However, drawing parallels to scales and modes in Western music can provide a foundational understanding,

despite ragams encompassing much more than just a scale structure.

At a basic level, a *ragam* in Carnatic music is analogous to a scale or mode in Western music, serving as a set sequence of notes that provides the melodic framework for a composition. An example of a ragam familiar to a Western music audience would be the major scale, known as the Ionian mode in Western music theory, and *Shankarabharanam* in Carnatic music. This idea is similar to how major and minor scales, as well as modes like Dorian, Phrygian, and Lydian, provide distinct moods or colors in Western music.

However, this comparison holds only to a point. Ragams are not merely scales; they are comprehensive systems that include specific rules and characteristics governing improvisation and performance. While there are only seven modes in Western music theory, there are several hundred ragams in Carnatic music.

Each ragam includes:

1. **Arohanam and Avarohanam:** These are the ascending and descending note sequences of a ragam, akin to the scales in Western music. However, ragams may skip certain notes or use them only in specific contexts, adding complexity beyond Western-scale structures.

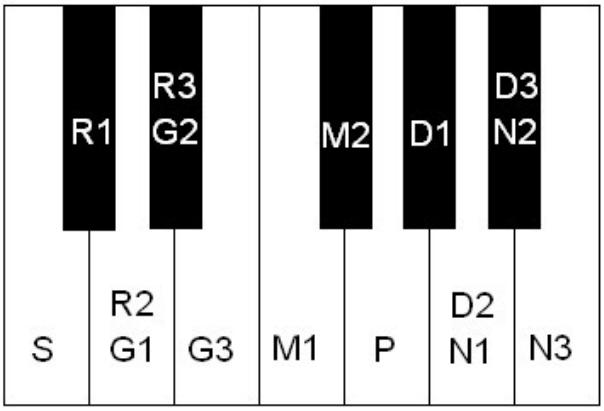
2. **Characteristic Phrases:** Ragams are defined by hallmark phrases or motifs essential to their identity, which must be emphasized in compositions and improvisations, similar to the use of characteristic melodic phrases in certain Western modes.

3. **Emotional and Symbolic Associations:** Ragams often carry specific emotional, temporal, or mythological associations, more specific and codified than the somewhat analogous associations in Western music, such as the pastoral themes or the use of minor keys to convey sadness.

4. **Microtonality:** Carnatic music uses a 22 *shruti* (pitch) system, allowing for nuanced microtonal intervals within an octave, compared to the 12-tone equal temperament

system in Western music, enabling a level of melodic expression not present in Western scales.

In Carnatic music, the seven notes are represented by the solfege syllables *Sa, Ri, Ga, Ma, Pa, Da,* and *Ni*, known as *swaras* [9].



S	sadjam
R1	suddha rishabam
R2 / G1	chatushruthi rishabam / suddha gandharam
R3 / G2	shatshruthi rishabam / sadharana gandharam
G3	anthara gandharam
M1	suddha madhyamam
M2	prathi madhyamam
P	panchamam
D1	suddha dhaivatham
D2 / N1	chatusruthi dhaivatham / suddha nishadham
D3 / N2	shatshruthi dhaivatham / kaisika nishadham
N3	kakali nishadham

Figure 1.1: The seven *swaras* (and their variants) in Carnatic Music

Consider the ragam *Nattaikurinji* as an example. Its *Arohanam* (ascending note pattern) and *Avarohanam* (descending note pattern) are as follows:

Arohanam:

$$S\ R2\ G3\ M1\ N2\ D2\ N2\ P\ D2\ N2\ S$$

Avarohanam:

$$S\ N2\ D2\ M1\ G3\ M1\ P\ G3\ R2\ S$$

To understand the notation used here, consider the following explanations for the symbols:

$$S = Sa\ (Root\ Note)$$
$$R2 = Ri\ (Second\ Note,\ Second\ Variant)$$
$$G3 = Ga\ (Third\ Note,\ Third\ Variant)$$
$$M1 = Ma\ (Fourth\ Note,\ First\ Variant)$$
$$P = Pa\ (Fifth\ Note)$$
$$D2 = Da\ (Sixth\ Note,\ Second\ Variant)$$
$$N2 = Ni\ (Seventh\ Note,\ Second\ Variant)$$
$$S = Sa\ (Root\ Note,\ an\ Octave\ Higher)$$

This notation system reflects the specific intervals and nuances of a ragam, illustrating the complexity and depth of Carnatic music's melodic structures. Peculiar to this ragam, for example, is the feature that on the descent, a musician must access the *Pa* note after the *Ma*, even though *Pa* is a higher frequency. This can be seen in the *Nattaikurinji* ragam *Avarohanam* above. A sweet and reflective ragam, *Nattaikurinji* songs are thought to be suitable for evening time. Keep in mind, that this is just one of several hundred ragams in Carnatic music. [3]

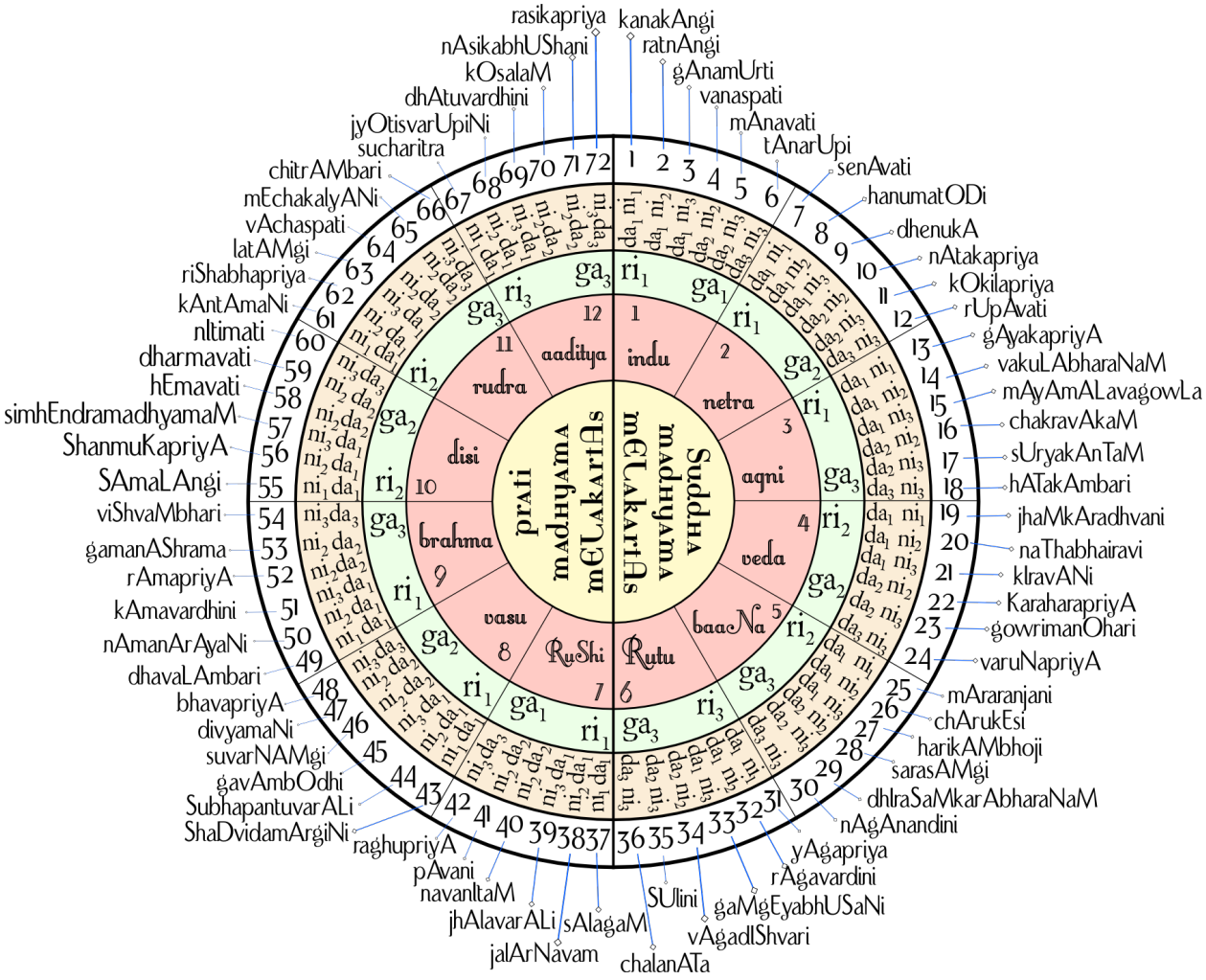


Figure 1.2: A diagram of the 72 possible heptatonic ragams in Carnatic music, known as the *melakartas*, composed of every combination of the 7 *swaras* and their variants

1.4 Ragam Identification

Each song in Carnatic music exists within a distinct ragam, with songs only being able to access the notes/frequencies allowed by their ragam. Ragams are easily identifiable by Carnatic connoisseurs and trained human listeners. Many of my family members can tell you the ragam of a song being played or performed within seconds of it beginning.

However, computational methods still struggle to reliably identify them. In fact, to this day no commercially available tool can tell you the ragam of a Carnatic song being played.

This is an important problem because ragam identification is one of the most important early concepts in Carnatic music education, with beginner students often struggling to understand how their teachers are identifying the ragam of a song being played or performed. Identification is a necessary skill because it represents a student's understanding of the boundaries between distinct ragams, and the ability to adhere to those boundaries during a performance. Additionally, knowing the ragam of a song can also provide critical context enhancing the meaning and beauty of a piece to a listener. For instance, some ragams like *Amruthavarshini* are thought to bring about rain, while others like *Neelambari* are used as lullabies and calming conclusions to concerts. Often, the first question anyone at a Carnatic music concert turns and discreetly asks their neighbor when a song begins, is, "What ragam is this song in?"

1.5 Why Computational Ragam Identifiers are Needed

Without strong ragam identification skills, a student can easily wrongly stray into adjacent ragams during renditions of pieces or during the frequent improvisational aspects of Carnatic concerts. This process of learning ragam identification is especially difficult when the instructor is not present, and the student is unsure whether they have correctly identified the song's ragam. Seasoned listeners like my grandmother can easily, consistently, precisely, and independently identify ragams even when listening to a song they have never heard before, or during periods of musical improvisation, but the task of ragam identification is even harder for newer students when they are unfamiliar with the piece being performed.

Thus, creating a computational method to instantly identify the ragam would allow students to verify their thought processes in real time. This would significantly accelerate the iterative process of learning to identify Carnatic ragams, render compositions within their ragam's bounds, and produce creative, aesthetic improvisation demonstrating a solid grasp of the underlying classical concepts. Critically, this computational method would need

to function reliably in real-world concert settings, with all the background noise, low-quality audio, and variabilities these venues introduce.

1.6 What this Thesis Contributes

In this project, I build on recent research illustrating the potential of three different neural network models for this task, and I introduce and develop a fourth model. I apply concepts I have learned across the continuous and discrete breadth courses I have taken for my applied math concentration to introduce and develop these models.

Previous research has demonstrated model efficacy on small pools of studio-quality ragam audio recordings but serious work remains to be done to bridge the gap towards a general-purpose tool that can be deployed on real-world audio data across a wide array of ragams. The eventual goal would be to engineer a Shazam-like app, where you could go to a Carnatic concert, hold up your phone, and have the app tell you the ragam of the song being performed. This project takes steps towards bridging that gap in four principal ways.

1. **Scope of dataset:** By combining thousands of web-scraped audio files, family collections, and Harvard's James A. Rubin Collection of Indian classical music (one of the largest collections of Carnatic music in the world), I assembled a dataset of over 70,000 audio files of Carnatic songs. Representing over 1 terabyte of audio data, and split into millions of audio chunks labeled by ragam, which constitute my model training data, this dataset is far more expansive than any dataset I encountered in previous studies aiming to train ragam classification models. This increased scope enabled both better model robustness in my project and the potential for future research into much bigger models using my dataset. Having a dataset of this scale gets us closer to the goal of a general-purpose tool that can identify hundreds of ragams in the real world.

2. **Improved performance of models previously applied to this task:** I created ANN (Artificial Neural Network), LSTM (Long-Short Term Memory), and 2-D CNN

(Convolutional Neural Networks) models that achieved greater classification accuracy than comparable models from previous studies, and that could recognize a greater number of ragams than models from previous studies.

3. **Mapping of feature importance provides model explainability:** Previous work in this space has largely used black-box models, whereas my models provide an understanding of what specific aspects of the audio file they are considering when making a ragam prediction, by showing the relative weights they assign to each of these features. I further developed intuition for why the models may be assigning these weights based on my Carnatic music domain knowledge. This process allows us to get a sense of what the "computational essence" of a ragam is.

4. **Introduction of a new model not previously applied to this task:** In addition to the successful ANN, LSTM, and CNN model architectures developed in this project, I also demonstrated that transformer (BERT) models can effectively determine ragams in recordings of Carnatic music audio. BERT models have not previously been used for this task, so this was a promising new result.

Chapter 2

Literature Review

2.1 Overview

With the evolution of machine learning models over the past five years, many researchers have begun to apply neural network models to all sorts of audio and music classification problems. In this section, I outline some of the key papers that guided my thinking in this project and provide an overview of prior applications of neural network models to music classification tasks, specifically ragam identification in Carnatic music. Before reviewing the relevant papers, however, it is worth first understanding the two principal ways we can represent audio data and the associated models for each of these representations. [1] [2]

2.2 Image Representation

Mel-spectrograms are a common representation of audio data that show the frequency content of a signal over time. They are created by first computing a spectrogram of the audio signal (which shows the amplitude of each frequency component over time) and then applying a set of frequency-weighting filters that mimic the human auditory system. The

[1] Note that I used ChatGPT to help me write and edit some of the language in this chapter.

[2] Some of the language in this chapter comes from my Neuro 240 final report, which I produced for Dr. Kreiman in 2023, and which is cited in my bibliography below.

mathematics underpinning mel-spectrogram construction will be explained in later sections of this thesis. Mel-spectrograms are widely used in audio analysis, particularly for tasks like speech recognition, music classification, and environmental sound recognition.

Convolutional Neural Networks (CNNs) are a type of neural network that is particularly well-suited to analyzing image data. They are inspired by the structure of the visual cortex in animals and use a series of convolutional layers to extract features from the input image. These layers apply a set of learnable filters to the image, which are shifted over the entire image to produce a feature map. By stacking multiple convolutional layers, CNNs are able to learn increasingly complex features from the input image.[4]

CNNs are well-suited to analyzing mel-spectrograms because they can learn local patterns in the frequency and time domains. By applying convolutional filters to a mel-spectrogram, a CNN can learn to identify important features like harmonics, formants, and other spectral characteristics that are relevant for audio analysis. This makes CNNs a potentially powerful tool for the task of ragam identification.

Convolutional neural networks, which are a class of deep feedforward artificial neural networks, have recently seen wide application in music recognition and more specifically Carnatic ragam identification. A 2019 paper presented at the International Conference on Advanced Computational and Communication Paradigms showed promising results applying CNN's to the problem of Carnatic ragam identification. The study involved creating a visual diagram of a snippet of a song, essentially graphing various sound features against time, and training CNN models on these visual input data, classified by ragam. However, model accuracy dropped from 90% to 80% as the pool of ragams increased from 5 to 11. [6]

With regards to spectrogram construction and inputs for CNNs, a key resource I rely on is the paper "An evaluation of deep neural network models for music classification using spectrograms," published in Multimedia Tools and Applications. This study applied spectrograms for music classification tasks and found architectures that were better able to "process non-speech background audio signals, remove noise, and effectively improve the

accuracy of speech emotion recognition," which is a key objective in my project which aims to more accurately classify real-world audio data by ragam.[12] According to a 2017 paper published in the Journal of Applied Soft Computing, CNNs have been shown to outperform results obtained by using handcrafted features and SVM classifiers in the task of genre classification, which inherently relies on a greater number of parameters than ragam classification, although the differences between tracks may be more pronounced[8]. This is a promising result with regard to the expected success of CNN applications in the task of ragam classification. Another key paper I have found for this project proposal is "Automatic tagging using deep convolutional neural networks," published recently by researchers at Queen Mary University of London, which demonstrates the effects of using deeper CNNs and larger datasets on ROC and AuROC measures in music classification tasks. [7]

2.3 Numerical Features Representation

Mel-spectrograms can be computationally intensive to store and process, as they are essentially large matrices (256 x 256 in my project) capturing every aspect of the audio file. In fact, the original audio file can be reconstructed and played from just the information in the mel-spectrogram. Training models on mel-spectrograms can thus become very costly as complexity scales.

In fact, models can still perform effectively given a very limited slice of information about the audio file. Consider a vector containing several columns of statistics and measurements, such as Chroma Features and the Root Mean Square Energy, that describe the audio file. While one cannot fully reconstruct the audio file from these statistics, they can still provide sufficiently significant information to classification models like Artificial Neural Networks (ANNs). [13]

Artificial Neural Networks (ANNs) are computational models inspired by the human brain, designed to recognize patterns and make decisions by learning from data. They

consist of layers of interconnected nodes or neurons, where each connection can transmit a signal from one neuron to another. ANNs are particularly suited for the ragam classification problem, due to their ability to handle complex, high-dimensional data, such as a dataset with thousands of rows and tens of columns of unlabeled audio statistics. The network can learn to discern the underlying patterns and characteristics of different ragams by training on the dataset, using the final column containing the ragam name as the truth label to guide its learning process. This enables ANNs to effectively classify and predict the ragam of new, unseen musical pieces. [4]

One approach described in a study published in the International Journal of Neural Networks, achieved close to perfect testing accuracy using ANN models with various topologies, and on a pool of 72 ragams, but with datasets involving computer-generated, acoustically-pure, audio files. [15] More recently, a paper entitled "A Comparison of Audio Preprocessing Techniques and Deep Learning Algorithms for Raga Recognition," published last year by researchers at MIT World Peace University used a dataset of live recordings over a pool of 12 ragams, with 40 songs worth of data in each ragam. They were able to achieve north of 90% testing accuracy using spectrogram analysis through ANN and 2-D convolutional neural network methodologies [10]. This is the key paper I relied on for my project as their approach used the most realistic training data for the situations for which I'm most interested in designing a solution: ragam identification in live concerts. [14]

Chapter 3

Data Sources

3.1 Overview

The required data for training machine learning ragam classification models is ragam-labeled audio files. Based on my advisor Dr. Kreiman's advice, I made it my goal to assemble as large a dataset of labeled Carnatic music audio as possible, as a small initial investment of time into creating a larger training dataset can often yield better eventual model accuracy than many hours spent later on advanced feature engineering and hyperparameter tuning.

I relied on three principal sources to assemble such a collection of audio files, which I then labeled by ragam. This section will review the sources, content, and scale of the audio data used to train my models.[1] [2]

3.2 Source #1: Sangeethapriya.com

The first source of audio data for this source was Sangeethapriya.com, the largest online library of Indian classical music in the world, with over 10,000 recordings of Carnatic music concerts [2]. Based on database track frequency and my knowledge of Carnatic music, I

[1]Note that I used ChatGPT to help me write and edit some of the language in this chapter.

[2]Some of the language in this chapter comes from my Neuro 240 final report, which I produced for Dr. Kreiman in 2023, and which is cited in my bibliography below.

defined a set of 140 ragams that I deemed to be the most popular and commonly occurring in concert settings. I then used a brief Python script to scrape and download all individual tracks from Sangeethapriya.com that were labeled with one of the ragams from this set of 140 names. This came out to be 3561 downloaded tracks, of average length 10 minutes and a total size of roughly 500 GB. Based on my review of the literature, this content alone would make my dataset the largest and broadest dataset of Carnatic music used for training a ragam recognition model, with the largest previous being the MIT World Peace University study involving 480 tracks spanning 12 ragams. Importantly, the audio data scraped from Sangeethapriya are from real-world concerts, and are not necessarily recorded under studio conditions where impurities are filtered, silences are trimmed, and applause and background noise are absent. The idea was that using a much larger dataset that is as reflective of real-world listening conditions as possible would be the most effective way to train a robust model for the task of ragam recognition in exactly those conditions.

3.3 Source #2: Harvard University's James Rubin Collection of Indian Classical Music

Shortly after commencing this project, I learned that Harvard is home to one of the world's largest libraries of Indian classical music, the James Rubin Collection.

James Rubin, born in Boston in 1927, was an avid "collector and promoter of Indian music even though he had no formal training," according to the collection's webpage. During his career, he founded and directed an organization called the Pan Orient Arts Foundation, which organized concerts by Indian artists across the United States. A close friend and tour manager for the famous Carnatic singer M.S. Subbulakshmi, the collection webpage states that to this day, Rubin is still fondly remembered by the music community in Chennai "as 'Rubin Mama' (Uncle Rubin) for wearing Indian dress, his love of music, and his avuncular manner."[1]

Physically stored in the Archive of World Music at Harvard University's Loeb music library, this collection is comprised of over 1000 reels of audio tape, containing many thousands of songs recorded by James Rubin over twenty trips to Chennai, India, where my family is from, between 1957 and 1989. Crucially, these recordings are accompanied by documentation containing ragam labels for each song, along with other detailed composition-level information.

This dataset was not easy to access for my project, because much of the collection has not yet been digitized and because the Harvard library system is still designing its protocols around providing music collections for AI research. Furthermore, the files that have been digitized reside in an antiquated storage platform that does not support bulk export. I worked with Dr. Kreiman and librarians at the Loeb library over several weeks to set up a pipeline to access about twenty hours of Carnatic music audio data with accompanying ragam labels for the set of ten ragams I was interested in identifying using my convolutional neural network models, which will be explained in later sections. These ten ragams were: *Mohanam, Nilambari, Ahiri, Amruthavarshini, Bhupalam, Kalyanavasantam, Manirangu, Revati, Simhendramadhyamam*, and *Yamunakalyani*.

While this represents only a slice of the Rubin Collection, these files still added valuable information to my training dataset and allowed me to bring Harvard's resources into my senior thesis. These audio recordings of songs from the Rubin collection were especially valuable to me because they are not available anywhere else in the world, online or physically. This means that the audio data from the Rubin collection would all be new to my model, and would not overlap with any of the public data from Sangeethapriya already in my model's training set. I hypothesized that this would improve the robustness of my models by reducing the likelihood of overfitting, (memorizing of training data), leading to better testing performance on unseen data.

Additionally, this was the first time the Loeb librarians had opened up the Rubin collection for machine learning research. My hope is to continue my research beyond this thesis

to eventually design models trained on the entire scope of the Rubin collection, once it is fully digitized, which is an ongoing effort by the Loeb librarians. Now that we have set up a preliminary working pipeline to access these files, I hope that future researchers interested in extending my work or designing more robust raga-identification models can more easily do so.

3.4 Source #3: Family Collections

The final, and most significant source of training data for this project came from my family members and friends, some of whom are professional Carnatic musicians and teachers who have accumulated massive collections over many decades. My aunt Sumitra Nitin, a Carnatic music teacher based in Bangalore, India, provided me with a dataset of 1,593 audio recordings (one song per track) labeled by ragam, and my uncle, Madurai Sundar, a Carnatic music teacher based in Detroit, MI, provided me with a dataset of 6,985 audio tracks with ragam information. Lastly, my family friend Delhi Muthukumar, a professional Carnatic musician based in Chennai, India, provided me with his collection of 60,663 Carnatic music audio files, also labeled by ragam. As these datasets were all structured and labeled differently, I wrote customized python scripts for each collection to read the documentation and append the ragam label to the file name, for each file in the dataset.

Much as with the Rubin collection, many of the files in these "family collections" were especially valuable to my project due to their being directly recorded at live concerts over many decades by my family members. This means they are unlikely to have been recorded or streamed otherwise, meaning they are very unlikely to be repeated from the public Sangeethapriya database. The addition of this much new data to my dataset was crucial to preventing model overfitting and ensuring the robustness of my model.

3.5 Scale of Dataset

Altogether, my dataset for this project contained 72,812 audio recordings (mp3 files) of Carnatic music audio data, with each file labeled by ragam. Based on my review of preceding research into ragam identification, this is by far the largest dataset used for training ragam identification models assembled thus far.

This dataset not only enabled me to experiment with much larger models than those described in the literature, but my hope is that it will also enable future researchers interested in this problem to extend my work.

Chapter 4

Data Preprocessing

4.1 Overview

In this section, I describe the processes of splitting the dataset, labeling the splits, and performing data augmentation to improve model robustness and ensure efficacy in real-world applications. All audio data, mel-spectrograms, and numerical features were stored in Google Drive folders due to storage constraints on my local drive (the combined size of the audio files was over 1 terabyte). Thus, all code for data pre-processing was written in Google Colab Jupyter notebooks that can read data directly from Google Drive. [1] [2]

4.2 Determining Ragam labels

Defining the training data and labeling them proved to be quite challenging, as the choices had to reflect real-world considerations. A human listener can identify a ragam in just a few seconds of hearing a song. Most ragams also have "tells" or melodic phrases (1-3 second note sequences) that are so unique that a connoisseur could instantly contextually recognize the song as being in that ragam, even if the frequencies of that "tell" are technically within the

[1] Note that I used ChatGPT to help me write and edit some of the language in this chapter.

[2] Some of the language in this chapter comes from my Neuro 240 final report, which I produced for Dr. Kreiman in 2023, and which is cited in my bibliography below.

bounds of many other ragams. Therefore, simply detecting the pattern of these frequencies would not be sufficient for a model to predict a distinct label– it would have to learn the association between that "tell" and the ragam.

There were also issues with the labeling of each track that made ground truth labeling and annotating challenging. Many file names in the dataset contained missing or incorrect ragam names, requiring me to listen to a few seconds of the track before hand-annotating the ground truth label for that track. For the majority of the labeling task, however, I wrote a script found in the SplitGenerator.ipynb file in my GitHub repository. Some ragam names, like "Sree" or "Shyama" are also extremely common words found in the names of Carnatic music songs, and my labeling script was wrongly labeling songs with the words "Sree" and "Shyama" as being in those ragams. This misclassification rate was sufficiently high to justify removing these two ragams from the bank of available labels, thus excluding all tracks in these two ragams. The script then iterated over all the file names in the dataset, searching for a match between the tokens in the filename and one of the ragams in our bank of 138 remaining ragams.

To provide a sense for the fraction of the original tracks classified as belonging to a ragam after conducting this process, 3316 out of the original 3561 Sangeethapriya tracks were able to be definitively labeled as belonging to one particular ragam. Upon inspecting the remainder, I found that most unlabeled tracks fell into one of the exclusion categories listed above, and would not have been useful for teaching a model to recognize ragams as it is unlikely a trained human listener would have been able to identify much in those tracks either.

I then wrote another Python script that processed each track's sample rate and audio time series, and cut each track into ten-second split files named by their ragam and split number in that ragam. This yielded a total of 261,341 ten-second audio files from the Sangeethapriya set alone, with each file titled according to its ragam. The decision to use a ten-second split was informed by the MIT World Peace University study using five-second

splits. My intuition was that a ten-second split contained strictly more signaling features about a ragam that could be captured as learning information by the CNN, and the fact that I had so many more tracks in my dataset than in their study eased the concern that I would have fewer training data points if I used longer time intervals. The number of splits was not standardized by ragam. For some extremely common ragams, there were nearly 15,000 splits, and for some of the rarer ones, the splits produced numbered in the low 100s.

As I discussed with Professor Kreiman, while class imbalance is typically avoided, in this problem, the imbalance reflects the real-world disparity between the frequencies of performance of certain ragams and is valuable intuition for the model to pick up when predicting output probability vectors. If there are a few ragams that sound somewhat similar, but one of them dominates the others in terms of performance frequency, the model would correctly bias towards that ragam prediction, in line with our desired outcome. For the purposes of this project, all audio files were converted to .wav format.

4.3 Preparing the Mel-Spectrograms for the Model

After generating these audio splits, the next step was to generate mel-spectrograms for each split. Mel-spectrograms, as described in the Literature Review section, are a visualization of the frequency content of an audio signal over time. They are commonly used in music classification tasks involving Convolutional Neural Networks to analyze and classify different genres, emotions, and instruments in a piece of music. In the next section, I will briefly introduce the mathematical basis for constructing spectrograms, based on material I encountered in Applied Math 104: Complex Analysis and Fourier Analysis. This analysis informs the data pre-processing and augmentation decisions I made for this dataset, as described below.

4.4 Mathematical Review of Spectrograms

Complex Analysis Background

In complex analysis, a complex number is represented as $z = x + iy$, where x and y are real numbers, and i is the imaginary unit with the property $i^2 = -1$. Complex analysis involves the study of functions that operate on complex numbers.

Fourier Analysis Background

Fourier analysis decomposes a function (in our case, a sound signal) into its constituent frequencies. The basic tool for this is the Fourier Transform, which transforms a time-domain signal into a frequency-domain representation.

Given a continuous time-domain signal $x(t)$, its Fourier Transform $X(f)$ is defined as:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-2\pi i f t} \, dt$$

where:

- $X(f)$ is the complex-valued Fourier Transform of $x(t)$,
- f is the frequency in hertz,
- t is time in seconds,
- i is the imaginary unit.

Spectrogram Calculation

A spectrogram is obtained by computing the squared magnitude of the Short-Time Fourier Transform (STFT) of the signal. The STFT is a series of Fourier Transforms computed over

short, overlapping time windows, which allows for the analysis of non-stationary signals (like most sounds).

Given a sound signal $x(t)$, the STFT $X(\tau, f)$ at time τ and frequency f is defined as:

$$X(\tau, f) = \int_{-\infty}^{\infty} x(t) w(t - \tau) e^{-2\pi i f t} \, dt$$

where $w(t)$ is a window function centered at zero, typically a Hanning window or a Gaussian window, which tapers off towards the ends to minimize edge effects. [13]

The spectrogram $S(\tau, f)$ is then the squared magnitude of $X(\tau, f)$:

$$S(\tau, f) = |X(\tau, f)|^2 = X(\tau, f)\overline{X(\tau, f)}$$

where $\overline{X(\tau, f)}$ denotes the complex conjugate of $X(\tau, f)$.

The duration of the audio clip is represented in the STFT through the temporal length of the analysis window and the overlap between consecutive windows. The total number of STFT windows N depends on the window duration D, the overlap O, and the total duration T of the audio clip:

$$N = \left\lfloor \frac{T - D}{D - O} \right\rfloor + 1$$

This equation shows how the duration of the audio clip influences the number of analysis windows in the STFT, which in turn affects the temporal resolution of the STFT representation. [13]

What a Spectrogram Shows

The spectrogram, through the STFT, provides a time-resolved frequency analysis, showing how the energy of different frequency components of the sound signal varies over time. The complex numbers in the STFT capture both the amplitude and phase information of the

signal at each time and frequency point. The magnitude squared operation converts this complex-valued representation into a real-valued one, representing the energy at each point in the time-frequency space, which is what the spectrogram visualizes. [13]

By analyzing the spectrogram, one can identify and study the temporal evolution of various frequency components in the sound, such as the harmonic structure of music, which makes it an optimal graph to describe ragams in a piece of Carnatic music audio data over a period of time.

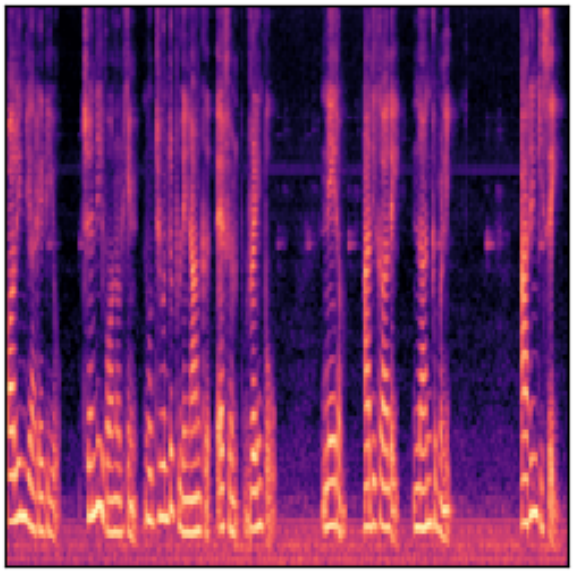


Figure 4.1: A mel-spectrogram representing a short audio clip of my singing the ragam *manirangu*

Modifications to the STFT Algorithm

Several mathematical modifications can be made to the STFT algorithm to accommodate various considerations about the audio file. In this section, I explain these modifications and their implementation in my audio-processing code.

1. Window Function Modification

The choice of window function $w(t)$ can be altered to balance the time-frequency resolution trade-off:

$$w(t) = \text{Hann}(t, D) \quad \text{or} \quad w(t) = \text{Hamming}(t, D) \quad \text{or} \quad w(t) = \text{Gaussian}(t, \sigma)$$

Different window functions may be chosen based on the desired properties of the STFT. In Carnatic music, ragams are characterized by their unique melodic and rhythmic patterns, often exhibiting complex tonal variations and subtle ornamentations. The choice of window function directly impacts the ability to capture these nuances effectively. I chose to apply a Hann window, implemented in `librosa`, as it is beneficial for maintaining a balance between time and frequency resolution, essential for capturing both the rapid ornamentations (*gamakas*) and sustained notes typical in Carnatic music. [10]

The Hann window function, also known as the Hanning window, is represented by the equation:

$$w(n) = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N-1}\right)$$

where n ranges from 0 to $N-1$, and N is the length of the window.

The Hann window function maintains a balance between time and frequency resolution by smoothly tapering the signal toward zero at both ends. By gradually reducing the amplitude of the signal towards the edges, the Hann window effectively balances time and frequency localization, preserving the resolution of both domains.[13]

Other options, like a Gaussian window function might be better suited for emphasizing specific frequency ranges and could be better for particular ragams with microtonal variations, but the Hann window function would be generally effective for processing most Carnatic audio data.

2. Overlap Adjustment

The overlap O between consecutive windows can be adjusted to change the temporal resolution and redundancy of the STFT representation:

$$O = \alpha D$$

where $0 < \alpha < 1$ is the fraction of the window that overlaps. Increasing α improves temporal resolution but increases computational complexity. Ragams in Carnatic music often feature rapid transitions between different notes, intricate rhythmic patterns, and dynamic tempo variations. Adjusting the overlap between consecutive windows enables the spectrogram to capture these temporal variations more accurately. A higher overlap ratio ensures smoother transitions between successive frames, thereby preserving the continuity of melodic and rhythmic elements.[13] This is particularly relevant in Carnatic music, where subtle variations in pitch and tempo can convey distinct emotional expressions and aesthetic nuances characteristic of different ragams. This ratio is determined by the `hop_length` parameter in `librosa`, which I set to 75%.

3. Zero-Padding

Zero-padding can be applied to each windowed segment before performing the Fourier Transform to increase the frequency resolution:

$$X(\tau, f) = \int_{-\infty}^{\infty} x(t)w(t-\tau) \cdot \text{ZeroPad}(\cdot)\, e^{-2\pi i f t}\, dt$$

This does not increase the actual information content but interpolates the frequency bins, making it easier to identify peaks in the frequency domain. Carnatic music is renowned for its rich harmonic content, often incorporating complex frequency modulations. Zero-padding before Fourier Transform interpolation enhances the frequency resolution of the spectrogram, making it easier to identify spectral peaks corresponding to different ragams. This is essential

in Carnatic music, where the identification of subtle tonal variations between notes is crucial for distinguishing between closely related ragams. [10]

4.5 Spectrogram generation in Python

Thus, I applied the Python library `librosa`, which is a popular library used for audio analysis, processing, and feature extraction, and is widely used in music information retrieval research, for computing these mel-spectrograms. One can think of these mel-spectrograms as 2D arrays of pixel intensities. I developed a custom function that wrote a mel-spectrogram and corresponding ragam label to a pickle file, derived from the split file name, for each split file within a ragam folder. The pickle file enabled easy data loading into my models after one-time generation. Otherwise, loading in the mel-spectrograms into my model each time I wanted to compile it could take up to 30 minutes, for each ragam. This was due to the mel-spectrogram computation time, the combined size of the mel-spectrograms across the splits for each ragam (several gigabytes), and the fact that these audio files were being stored in Google Drive, resulting in slower read/write times due to network latency and speed considerations.

During this stage, each mel-spectrogram was standardized and resized to a 256x256 array, keeping with the methods employed by Hebbar and Jagtap, and a subset of ten ragams in the dataset were selected for computation, given the large size of the full dataset. These power spectrograms were then scaled to decibel units, using a standardized audio sample rate of 16,000 and with all audio samples scaled by a factor of 1/32768, which are figures in line with those used in the surveyed literature during data preprocessing for music classification. The ten ragams, randomly selected, were *ahiri, amruthavarshini, bhupalam, kalyanavasantham, manirangu, mohanam, nilambari, revati, simhendramadhyamam*, and *yamunakalyani*, and had an average number of 1000 splits each. While mostly consistent in sample size, the length of each ragam's pickle file did in some cases reflect the real-world class imbalance

captured by the dataset, with the commonly performed *mohanam* ragam having 3000 splits and the lesser-known *ahiri* ragam having only 600. An X array containing the computed mel-spectrograms for each split, and an accompanying y array containing the ragam name, was written to a pickle file for each of these ten ragams.

4.6 Removal of Misleading and Irrelevant Spectrograms

There were a few situations in the dataset that could make ragam detection difficult for humans and neural networks alike. For instance, there are some pieces, called *ragamalikas* where the artist may choose to rapidly switch ragams, giving the listener only limited time to orient themselves. Secondly, during percussion solos within songs, there is no melody, and classifying the song's ragam is impossible. In a similar vein, there were often speeches and applause throughout sections of certain tracks. And because the audio splits containing these elements came from a larger labeled track, they were labeled as a ragam, even though no ragam could be detected by listening to them, introducing error into the dataset. Having silences, speech, and percussion labeled as being in a particular ragam could greatly impede the model's ability to learn the actual features of that ragam.

To combat these issues, I manually filtered out all *ragamalika* pieces and wrote a script in Python to detect spectrograms with extensive silences, percussion solos, and speeches, and remove them from the dataset.

To clarify this process, I have provided a brief mathematical representation of the features my code was identifying when deciding whether to remove a spectrogram from the training set.

Representation of Silence

Silence in an audio signal is characterized by minimal energy across all frequencies. In the STFT representation:

$$X_{\text{silence}}(\tau, f) \approx 0$$

for all f within the audible range, indicating a lack of significant frequency components in the signal during silent periods. Practically, this involves summing the squared magnitude of the spectrogram values and determining whether it falls below a silence threshold, which I implemented in Python.

Representation of Percussion

Percussion sounds, such as drum beats, are transient and broadband in nature. They contain a wide range of frequencies with rapid onset and decay times. In the STFT representation, percussion elements appear as:

$$X_{\text{percussion}}(\tau, f) = \begin{cases} \text{High magnitude}, & \text{for short } \tau \text{ duration across a wide range of } f \\ 0, & \text{otherwise} \end{cases}$$

This indicates sharp, short bursts of energy spread across a broad frequency spectrum.

Representation of Melody

Melodic components, such as those from a singing voice or a musical instrument playing a tune, are characterized by harmonic structures and are more sustained than percussive sounds. In the STFT representation, the melody appears as:

$$X_{\text{melody}}(\tau, f) = \text{Harmonically related peaks over sustained } \tau \text{ durations}$$

In practical coding terms, removing the percussion-dominant splits involved removing

splits with a majority of energy peaks occurring within short-duration windows, and with many windows having below silence threshold energy.

4.7 Preparing the Numerical Feature Vectors for the Model

As mentioned in the Literature Review section, models can still perform effectively given a very limited slice of information about the audio file. For the non-CNN models evaluated in this thesis, I constructed 1-D input data, essentially a vector of statistics describing the audio split. This is much less computationally intensive to generate and store than a 256x256 matrix (spectrogram). In this section, I will introduce a mathematical background for the statistics I selected, and explain how each one could provide useful information to a ragam identification model.

Mel Frequency Cepstrum Coefficients (MFCCs)

MFCCs are derived from the real cepstrum of a windowed short-time signal. They are based on the linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency [13]:

$$\text{MFCC}[i] = \sum_{k=0}^{N-1} \log\left(S[k]\right) \cos\left[i\left(k - \frac{1}{2}\right)\frac{\pi}{N}\right], \quad i = 1, \ldots, M \tag{4.1}$$

Here, $S[k]$ is the power spectral density of the signal, computed as $|X[k]|^2$, where $X[k]$ is the Fourier transform of the signal. The mel scale is defined to mimic the human ear's response more closely than the linearly-spaced frequency bands, typically using the formula:

$$\text{Mel}(f) = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{4.2}$$

where f is the frequency in Hz. The inverse mel scale, used to convert back from mels to frequency, is given by:

$$\text{Mel}^{-1}(m) = 700\left(10^{\frac{m}{2595}} - 1\right) \tag{4.3}$$

where m is the frequency in mels. The N mel filters used in the computation of MFCCs are triangular filters spaced evenly on the mel scale, and M is the number of cepstral coefficients we wish to retain.

In Carnatic music, MFCCs hold particular importance due to their ability to encapsulate the timbre and texture of musical sounds. Carnatic ragams are defined in part by their *gamakams*, which, as I have mentioned above, add a distinctive color to the ragam and convey its emotional essence. MFCCs, by capturing the spectral properties of these nuances, allow for a representation of the intricate melodic contours of *gamakams*. Consequently, when employing neural networks for ragam classification, MFCCs serve as a fundamental feature set, enabling the model to discern the complex spectral characteristics that distinguish one ragam from another. In my code for this section, I retained 19 cepstral coefficients for each audio split using librosa functions.

Chroma Features

Chroma features aggregate all the spectral information within each of the 12 distinct semitone (pitch class) bands, regardless of the octave. For the Chroma STFT, we first compute the Short-Time Fourier Transform (STFT) of the signal to obtain the frequency spectrum [13]. The Chroma feature for each pitch class c is then computed as:

$$\text{Chroma}[c] = \sum_{n=0}^{N-1} |X(n)| \cdot \mathbb{I}\left[\text{PitchClass}(n) = c\right] \tag{4.4}$$

where $|X(n)|$ is the magnitude of the STFT at bin n, $\mathbb{I}$ is the indicator function (which

is 1 if the condition is true and 0 otherwise), and PitchClass(n) determines the pitch class of the n-th bin, mapping it to one of the 12 semitones ($c \in \{C, C\#, D, \ldots, B\}$).

Chroma features, encompassing both Chroma STFT and Chroma CENS, are particularly aligned with the modal nature of Carnatic music. As explained in the problem statement, each ragam in Carnatic music is characterized by a unique set of *swaras* (notes) that form its scale. The ragam's identity is deeply rooted in these *swaras* and their sequential progression. Chroma features effectively abstract the pitch content of music into 12 distinct semitone bands, making them highly relevant for capturing the essence of a ragam's scale, irrespective of the octave. This octave invariance is crucial in Carnatic music, which frequently employs elaborate octave jumps. By analyzing the energy distribution across these chroma bands, a neural network can learn to identify ragams based on their foundational *swara* structures, even in the presence of complex ornamentations. Once again, I used librosa functions to calculate and store the 12 Chroma STFT and Chroma CENS features.

Root Mean Square Energy (RMSE)

RMSE is a measure of the power of the audio signal and is computed over short frames of the signal. For a frame of N samples, the RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}|x(i)|^2} \tag{4.5}$$

where $x(i)$ is the amplitude of the i-th sample within the frame. This measure gives an estimate of the average power across the frame, providing insight into the signal's amplitude and energy content. [13]

The dynamic range of a Carnatic music performance, captured through RMSE, provides insights into the emotive intensity of a composition. In Carnatic music, dynamic variations serve as a key expressive tool in delineating different sections of a performance, such as the explorative *alapanai* and the rhythmic *tanam*. These sections vary both in melodic content

and energy levels. For instance, an *alapanai* might start with a soft, meditative energy, gradually building up to a climax. By quantifying these energy variations, RMSE aids in distinguishing ragams that might employ similar scales but differ in their expressive delivery, thereby enriching the feature set for neural network-based ragam classification. This statistic is easily calculated by means of the `rms` function in `librosa`.

Spectral Centroid

There are three spectral features I was interested in calculating for each audio split. Firstly, the spectral centroid represents the center of mass of the spectrum and is computed as the weighted mean of the frequencies present in the signal, weighted by their amplitudes. [13] The spectral centroid for a frame is given by:

$$\text{Centroid} = \frac{\sum_{k=0}^{N-1} f(k)|X(k)|}{\sum_{k=0}^{N-1} |X(k)|} \tag{4.6}$$

where $f(k)$ represents the frequency of the k-th bin in the spectrum, and $|X(k)|$ is the magnitude of the k-th bin in the Fourier transform of the frame. The spectral centroid is a measure of the "brightness" or "sharpness" of the sound, with higher values indicating a brighter sound with more high-frequency content.

The spectral centroid is a measure of the brightness or tonal center of a sound, which in the context of Carnatic music, correlates with the *shruti* (the foundational pitch around which a performance is centered). The spectral centroid captures these subtleties by providing a mean frequency value, representing the tonal center that the artist navigates around [13]. This is particularly useful in distinguishing ragams that may occur more frequently around certain *shruthis*, thereby serving as an essential feature for neural network models aimed at ragam classification.

Spectral Bandwidth

Secondly, spectral bandwidth quantifies the spread of the spectrum around its centroid, effectively measuring the width of the spectrum. The p-th order spectral bandwidth is defined as:

$$\text{Bandwidth} = \left(\frac{\sum_{k=0}^{N-1} (f(k) - \text{Centroid})^p |X(k)|}{\sum_{k=0}^{N-1} |X(k)|} \right)^{\frac{1}{p}} \tag{4.7}$$

Typically, p is set to 2, in which case the spectral bandwidth measures the standard deviation of the spectral distribution, reflecting the spread of the energy around the spectral centroid.

Spectral bandwidth, which measures the spread of the spectrum around its centroid, captures the extent of frequency modulations in a musical piece. [13] In Carnatic music, *gamakas* significantly influence the spectral bandwidth of a piece. A ragam that heavily employs wide-ranging *gamakas* will exhibit a broader spectral bandwidth compared to a ragam with more restrained ornamentations. Therefore, spectral bandwidth is a critical feature in capturing the unique textural characteristics of each ragam for the models to learn as they train.

Spectral Rolloff

The third spectral feature I captured was spectral rolloff, representing the frequency below which a certain percentage (e.g., 85%) of the total spectral energy is contained. [13] It is defined as the minimum frequency for which the cumulative sum of the spectrum reaches a certain threshold:

$$\text{Rolloff} = \min \left\{ f(k) : \sum_{n=0}^{k} |X(n)| \geq \theta \sum_{n=0}^{N-1} |X(n)| \right\} \tag{4.8}$$

where θ is the threshold percentage (e.g., 0.85 for 85%). The spectral rolloff gives an

indication of the "skewness" of the spectrum, with higher rolloff values indicating more high-frequency content.

Spectral rolloff points to the frequency below which a specified percentage of the total spectral energy is contained. This feature is indicative of the harmonic content and the textural richness of a Carnatic music piece. Compositions in certain ragams might be rendered with a rich harmonic backdrop, while others might focus on the purity of the consistent melodic line. The spectral roll-off captures these variations by identifying the point in the frequency spectrum where most of the musical energy is concentrated.

Thus, capturing the distribution of musical energy across the frequency spectrum through these three spectral features could greatly assist a model in learning the harmonic features that distinguish ragams. All three of these spectral features were calculated for each audio split using built-in librosa functions and then appended to the vector of numerical features.

Zero-Crossing Rate (ZCR)

The final statistic I was interested in capturing for each audio split was the zero-crossing rate. The zero-crossing rate measures how frequently the audio signal changes sign, effectively quantifying the rate at which the waveform crosses the horizontal axis. For a frame of N samples, the ZCR is computed as:

$$\text{ZCR} = \frac{1}{N-1} \sum_{i=0}^{N-2} \mathbb{I}\left[x(i)x(i+1) < 0\right] \tag{4.9}$$

where $x(i)$ is the amplitude of the i-th sample, and $\mathbb{I}$ is the indicator function, which is 1 if the product $x(i)x(i+1)$ is negative (indicating a sign change) and 0 otherwise. ZCR is a simple measure of the signal's frequency content, with higher rates indicating more frequent sign changes and potentially more high-frequency content. [13]

ZCR, a measure of the frequency of sign changes in a signal, is particularly relevant in capturing the rhythmic and percussive elements of Carnatic music. The intricate rhythms,

articulated through the mridangam and other percussion instruments, along with the rhythmic syllables of *konnakol*, contribute to the distinctive rhythmic structure of Carnatic compositions. Moreover, the vocal or instrumental rendering of fast-paced passages significantly influences the ZCR. A higher ZCR may indicate a piece with brisk, articulated note sequences, characteristic of certain ragams or specific sections within a Carnatic music performance. Thus, ZCR provides essential information on the tempo and rhythmic texture, complementing the melodic and harmonic features in the neural network's toolkit for ragam classification. It would later be interesting to determine how the model weighed ZCR, a percussion/tempo feature, relative to the harmonic features, when making a ragam classification prediction.

4.8 Constructing and Storing the Numerical Feature Vectors

As mentioned in the previous section, I then generated a vector of the aforementioned numerical features for each audio split, painting a rich picture of the audio content in a much less storage-intensive format than the mel-spectrogram representation. Across the MFCC, Chroma, RMSE, Spectral, and ZCR features, the fully appended vector was of dimension 1 x 50, with the fifty-first column containing the ragam label for that audio split. These features and labels would later be loaded into the model as the X and y data arrays, respectively. Much as with the image representation, I stored these vectors in pickle files grouped by ragam, for easy retrieval and data loading during model training. I selected the most commonly occurring ragams (sorted by # of splits) in my dataset and generated a pickle file for each of the 33 ragams with at least 2000 ten-second audio splits. Each ragam's pickle file contained 2000+ rows corresponding to each of its ten-second audio splits. And once again, class imbalances reflected the real-world relative incidences of certain ragams.

4.9 Data Augmentation Techniques

After creating the audio splits and labeling them by ragam, the next step was to increase the amount of audio training data being fed to the models in ways beyond simply adding new recordings to the dataset. This is critical to ensuring model robustness and preventing overfitting, leading to better performance on unseen testing data and real-world audio. Following Dr. Kreiman's advice, I performed data augmentation in four principal ways: volume variation, speed variation, semitone shifting, and background noise introduction. The idea here was not simply to increase the number of tracks in the training set, but also to ensure the model would learn to recognize ragams no matter how loud or fast the musician chooses to perform, at whatever *shruthi* (pitch) they choose to perform, and in whatever environmental conditions the music is produced.

Mathematical Representation of Audio Data Augmentation

1. **Volume Variation:**

   - Increasing or decreasing the volume of an audio clip can be represented by multiplying the time-domain signal by a scalar factor. Let's denote this scalar factor as v, where $v > 1$ for increasing volume and $0 < v < 1$ for decreasing volume.
   - Mathematically, this can be represented as:

$$x_{\text{aug}}(t) = v \cdot x(t)$$

     - Where:
       * $x(t)$ is the original time-domain signal,
       * $x_{\text{aug}}(t)$ is the augmented time-domain signal.

- *Mel Spectrogram Adaptation*: Multiplication by a scalar v scales the magnitude in the STFT, and thus each cell in the mel spectrogram is scaled by v, altering the intensity without changing the structure.
- *Carnatic Music Context*:
  - Models the dynamic range variations in live performances.
  - Trains the model for ragam recognition across different acoustic environments.
- *Librosa Implementation*: Use `librosa.effects.time_stretch(audio, rate=1.0/v)` to adjust the volume by inversely modifying the speed. Using `librosa`, I created 2 new versions of each split: one 25% louder, and one 25% quieter.

2. **Speed Variation:**

- Speed variation, or time stretching/compression, can be achieved using time-domain resampling techniques. Let's denote the speed factor as s, where $s > 1$ for increasing speed (shortening the duration) and $0 < s < 1$ for decreasing speed (lengthening the duration).
- Mathematically, this can be represented as:

$$x_{\text{aug}}(t) = x(st)$$

  - Where:
    - $x(t)$ is the original time-domain signal,
    - $x_{\text{aug}}(t)$ is the augmented time-domain signal.
- *Mel Spectrogram Adaptation*: Speed modification by a factor s changes the time axis in the STFT, affecting the tempo and time resolution in the mel spectrogram.
- *Carnatic Music Context*:
  - Mimics natural tempo variations in performances by different artists.
  - Enhances robustness to tempo changes in raga recognition.

- *Librosa Implementation*: Apply `librosa.effects.time_stretch(audio, rate=s)` with s as the speed factor. Using `librosa`, I created 2 new versions of each split: one 25% faster, and one 25% slower.

3. **Semitone Shifting:**

- Semitone shifting involves changing the pitch of the audio clip by a certain number of semitones. This can be achieved by modifying the frequency content of the signal while keeping the time axis intact.
- Let's denote the semitone shift factor as n, where n represents the number of semitones to shift.
- Mathematically, this can be represented as:

$$x_{\text{aug}}(t) = x(t) \cdot e^{(2\pi i \cdot n \cdot t)}$$

  - Where:
    - $x(t)$ is the original time-domain signal,
    - $x_{\text{aug}}(t)$ is the augmented time-domain signal,
    - n is the number of semitones to shift.
- *Mel-spectrogram Adaptation*: Pitch shifting by n semitones translates frequency bins vertically in the mel-spectrogram, changing the pitch without affecting duration.
- *Carnatic Music Context*:
  - Reflects the adaptability of ragas to different tonic notes.
  - Improves raga recognition across various pitch levels, especially given that most of the original data is clustered around the keys of C and G, which are the most popular male and female vocal *shruthis* respectively (instrumentalists always tune to the *shruthi* of the performing vocalist). This augmentation improves the model's robustness and efficacy when faced with audio data in an unfamiliar key.

- *Librosa Implementation*: Use `librosa.effects.pitch_shift(audio, sr, n_steps=n)` where n is the semitone shift. Using `librosa`, I shifted each ten-second audio split up by two semitones, and down by two semitones. The intuition here was to generate more data around the keys of D, F, and A, which can be encountered in Carnatic concerts, especially ones with instrumental leads, but were previously under-represented in the dataset.

4. **Adding Background Noise:**

- Introducing background noise, such as tanpura drones, crowd chatter, or static, can be represented by adding a noise signal to the original audio signal.
- Let's denote the background noise signal as $n(t)$.
- Mathematically, this can be represented as:

$$x_{\text{aug}}(t) = x(t) + n(t)$$

  - Where:
    - $x(t)$ is the original time-domain signal,
    - $x_{\text{aug}}(t)$ is the augmented time-domain signal,
    - $n(t)$ is the background noise signal.
- *Mel Spectrogram Adaptation*: Adding noise $n(t)$ introduces additional energy across frequency bands depending on the noise characteristics, simulating ambient sounds in the spectrogram.
- *Carnatic Music Context*:
  - Simulates ambient noise in outdoor performances or recordings.
  - Prepares the model to focus on musical content amidst extraneous sounds.
- *Librosa Implementation*: Add a scaled noise signal to the audio before computing the spectrogram: `audio_augmented = audio + scaled_noise`. Using `librosa`, I

added a background noise signal of static, one at 15% audio split amplitude, and one at 30% audio split amplitude, to each track, to simulate various levels of background noise, typical of amplification systems at Carnatic music performance venues, known as *sabhas*, in India.

Each of these four data augmentation methods added two new versions of each original ten-second audio split to the dataset. Thus, this data augmentation process yielded a 9x increase in the size of our dataset

I did not pursue more than one layer of augmentation to avoid overlapping data. That is to say, I did not create any augmented data where more than one of these four parameters had been adjusted, to avoid feeding highly repetitive data into the training process and risking the model memorizing the input data.

Chapter 5

Methods

5.1 Overview

In this section, I will introduce the mathematical basis and implementation details for the four machine learning models I designed and evaluated on the task of ragam identification in Carnatic music. I will begin with the three models trained on the numerical feature vectors, before describing the Convolutional Neural Networks trained on image data. As mentioned in the previous section, all code for these models was written in Google Colab Jupyter notebooks that can read data directly from Google Drive. I used high-ram, A100 GPU-enabled runtimes capable of handling these data and computationally intensive model training processes. [1] [2]

Mathematical Introduction to Neural Networks

Neural networks are a cornerstone of modern machine learning, inspired by the biological neural networks that constitute animal brains, as will be further explained later in this report. At their core, neural networks are composed of layers of interconnected nodes or

[1] Note that I used ChatGPT to help me write and edit some of the language in this chapter.

[2] Some of the language in this chapter comes from my Neuro 240 final report, which I produced for Dr. Kreiman in 2023, and which is cited in my bibliography below.

"neurons," each capable of performing simple computations. The power of neural networks lies in their ability to approximate complex functions through the collective operations of these neurons, making them highly effective for tasks such as image recognition, natural language processing, and audio classification, as in this problem.

Mathematical Foundations

To understand neural networks from a linear algebra perspective, we begin with the concept of a neuron. Mathematically, a neuron's operation can be represented as a weighted sum of its inputs, followed by the application of an activation function. If we consider a neuron with n inputs $x_1, x_2, \ldots, x_n$ and corresponding weights $w_1, w_2, \ldots, w_n$, the output y of the neuron can be described as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Here, b represents a bias term, and f is the activation function, which introduces non-linearity into the model.

From Single Neurons to Layers

In a neural network, neurons are organized into layers: an input layer, one or more hidden layers, and an output layer. The output of one layer becomes the input of the next. This structure can be efficiently represented using matrices and vectors, facilitating straightforward computations

Consider a layer with m neurons and an input vector $\mathbf{x} \in \mathbb{R}^n$. The weights connecting the inputs to each neuron can be represented as a matrix $W \in \mathbb{R}^{m \times n}$, and the biases for each neuron as a vector $\mathbf{b} \in \mathbb{R}^m$. The operation of the entire layer can then be expressed as:

$$\mathbf{y} = f(W\mathbf{x} + \mathbf{b})$$

where f is applied element-wise if it is a non-linear function, and $\mathbf{y}$ is the output vector of the layer.

Network Training

Training a neural network involves adjusting the weights and biases to minimize the difference between the network's output and the true outputs for a set of training data. This process is typically performed using backpropagation and gradient descent, where backpropagation computes the gradient of a loss function with respect to each weight and bias in the network, and gradient descent uses these gradients to update the weights and biases to minimize the loss.

The loss function quantifies the difference between the predicted and true outputs. For a given set of training examples, the goal of training is to find the set of weights and biases that minimize this loss function.

Artificial Neural Networks and Dense Layers

Artificial Neural Networks (ANNs) are a class of neural network models designed to simulate the way a human brain analyzes and processes information. ANNs are the foundation of deep learning algorithms and are particularly well-suited for problems that involve complex, non-linear relationships.

Dense Layers in ANNs

Dense layers, also known as fully connected layers, are a fundamental component of ANNs. In a dense layer, every neuron is connected to every neuron in the previous layer. This dense interconnectivity allows the layer to learn deep representations of the input data.

Mathematically, the operation of a dense layer can be described as follows:

Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the input matrix to the dense layer, where n is the number of samples

and d is the number of features for each sample. The weight matrix of the dense layer is denoted as $\mathbf{W} \in \mathbb{R}^{d \times k}$, where k is the number of neurons in the layer. Each neuron in the dense layer also has a bias term, represented as a vector $\mathbf{b} \in \mathbb{R}^k$.

The output of the dense layer, $\mathbf{Y} \in \mathbb{R}^{n \times k}$, is computed as follows:

$$\mathbf{Y} = f(\mathbf{X}\mathbf{W} + \mathbf{b})$$

Here, f represents the activation function applied element-wise. Common choices for f include ReLU (Rectified Linear Unit), softmax, sigmoid, and tanh functions, each introducing non-linearity to enable the learning of complex patterns. I will not go into the math behind all of these activation functions, but the softmax function is defined as:

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

where: - e is the base of the natural logarithm. - z_i is the score for class i. - K is the total number of classes. - $\text{softmax}(\mathbf{z})_i$ is the probability of class i.

This function converts logits $\mathbf{z}$ into a probability distribution, ensuring each element is in the range (0, 1) and their sum is 1. The exponential function applied to each logit introduces non-linearity, amplifying differences between logits and allowing interpretation as probabilities. The element with the highest assigned probability corresponds to the model's ragam prediction. [4]

Importance of Dense Layers

Dense layers are crucial for learning high-level features in the data. In deep learning architectures, multiple dense layers are often stacked together, allowing the network to learn a hierarchy of features. Lower layers might learn basic patterns, while deeper layers can learn more abstract representations.

Mathematical Perspective on ANN Training

Training ANNs involves finding the optimal weights $\mathbf{W}$ and biases $\mathbf{b}$ that minimize the loss function. This is achieved through optimization algorithms like stochastic gradient descent (SGD) or Adam. I will not go into the mathematical background of SGD or Adam in this report, but the backpropagation algorithm plays a key role here, calculating the gradient of the loss function with respect to each weight and bias in the network, which is then updated accordingly.

Given a loss function L, the update rule for the weights can be expressed as:

$$\mathbf{W}_{\text{new}} = \mathbf{W} - \eta \frac{\partial L}{\partial \mathbf{W}}$$

And for the biases:

$$\mathbf{b}_{\text{new}} = \mathbf{b} - \eta \frac{\partial L}{\partial \mathbf{b}}$$

Here, η is the learning rate, a hyperparameter that controls the step size during optimization.

Considerations

While dense layers are powerful, they also come with challenges. Overfitting is a common issue, where the model learns the training data too well, including its noise and outliers, leading to poor generalization. While I will not explain the full mathematical background of these concepts here, I used techniques like regularization, dropout, and early stopping to mitigate this. Furthermore, dense layers can significantly increase the number of parameters in a model, leading to increased computational cost and memory usage. Therefore, careful architecture design and hyperparameter tuning are crucial in developing effective ANN models.

With that being said, overall, ANNs are well-suited for tasks where the input data can

be effectively represented as feature vectors, as in this case the numerical features extracted from audio signals for ragam identification. As I have described, ANNs are adept at learning complex mappings between input features and output labels, making them suitable for classification tasks. In the context of ragam identification, ANNs can efficiently capture the relationships between the extracted audio features and the corresponding ragam labels, leveraging their capacity for nonlinear transformations. Additionally, ANNs are relatively quick and computationally straightforward to train and interpret, making them a practical choice for this task, especially when dealing with the kind of structured input data and discrete output labels I have generated.

5.2 Artificial Neural Network Implementation Details

After loading in the numerical feature vectors and labels for all audio splits for a pool of ten ragams into X and y data-frames in Python, I performed a 90% - 10% train-test split of the the data. These ten ragams were *kalyani, bhairavi, mohanam, carukesi, kapi, varali, nattakurinji, kamavardhini, shankarabharanam*, and *hindolam*. They were randomly selected from the set of the 33 most commonly occurring ragams, and the labels were manually encoded as integers 0-9 instead of ragam labels.

Model Architecture

I implemented my ANN models for this task using the `keras` library in Python, a popular framework for deep learning tasks. The first model I attempted to build was a simple linear model with one dense layer, seeking to categorize ten ragams. My goal was to get a sense of baseline performance using this simplistic model.

The first layer in this model was a *Flatten* layer, which was used to convert the 2D input of shape (50, 1) into a single-dimensional array of 50 elements. This is necessary because a linear model does not inherently process 2D or 3D data as is, but rather works

with flat input vectors. The `input_shape` parameter was set to (50, 1), matching the shape of the input data, excluding the batch size dimension. Following the flattening of the input, the model employed a Dense layer, which is a fully connected neural network layer. This layer had 10 units, corresponding to the 10 possible ragam classifications. The `softmax` activation function was used, which is typical for multi-class classification tasks as it outputs a probability distribution over the classes summing to 1. [10]

Finally, the model was compiled with the Adam optimizer, a popular choice for many types of neural networks due to its adaptive learning rate capabilities, making it efficient for problems with large datasets and parameters, like this dataset of thousands of rows of numerical feature vectors, with fifty columns each. The learning rate was set to 0.001 and the loss function used was `sparse_categorical_crossentropy`, which is suitable for multi-class classification tasks where the labels are provided as integers (as opposed to one-hot encoded vectors). The model's performance was evaluated using the accuracy metric.

However, in the first few passes of the network, I noticed that the validation loss stalled around 2.2 and the validation accuracy did not increase past 33% due to early stopping after 27 epochs. While 33% accuracy is not anything to write home about in a 10 ragam classification problem, it is considerably higher than the random guessing score of 10%, and along with the initially decreasing validation loss for the first 20 epochs, suggested that the model was actually learning something from the data. This suggested that I needed a more sophisticated model capable of learning deeper feature representations and more resistant to overfitting.

Restructuring and Hyperparameter Tuning

To achieve this, I made several adaptations to my next model. The first enhancement was the addition of a Dense layer with 64 units and ReLU (Rectified Linear Unit) activation. The ReLU activation introduces non-linearity to the model, allowing it to learn more complex patterns in the data. This layer served as a hidden layer, increasing the model's capacity to

represent and learn from the input data. I next included a dropout layer, which is a form of regularization used to prevent overfitting. Dropout works by randomly setting a fraction of input units to 0 at each update during training time, which helps to prevent neurons from co-adapting too much. Next, I added a new dense layer with 32 units, also with ReLU activation, further increasing the model's depth. The final layer was left unchanged, as the task was still about classifying ten ragams and generating a probability distribution over them. During compilation, I reduced the learning rate to 0.0003, thinking the lower learning rate might offer more fine-grained updates during training, potentially leading to better convergence.

However, during training, I noticed that while training accuracy was slightly higher, validation accuracy was essentially unchanged from the previous model, suggesting overfitting. Training and validation loss also decreased slightly faster.

To improve the performance of the model, I significantly increased its complexity in the following ways, akin to the approach used by Hebbar and Jagtap. I began the model with a Dense layer comprising 512 units, which also serves as the input layer due to the `input_shape` parameter being specified. This layer has a substantial number of neurons, allowing the model to learn a high level of detail and complexity from the input data. Following the first dense layer was another with 256 units, a third dense layer with 128 units, and a fourth with 64. Again, ReLU was the chosen activation function, consistent with the model's strategy to stack layers with non-linear activations. This design choice gradually reduced the dimensionality of the representations, focusing the model's learning on the most salient features. Once again, I utilized a softmax activation and output layer with ten classes, compiled the model with the Adam optimizer set to default parameters, and adjusted the early stopping callback to have a patience of 5 epochs to prevent the model from learning noise in the training data.

This time, during the training process, the training loss significantly decreased from 8.2653 in the first epoch to 0.1764 by the 50th epoch, indicating substantial learning and

improvement in the model's performance. Validation accuracy saw a notable increase from 20.94% in the first epoch to 94.89% in the final epoch, reflecting the model's enhanced ability to generalize to unseen data. This consistent improvement in both training loss and validation accuracy over the epochs suggests that the model was effectively learning from the data, capturing the underlying patterns without significant overfitting occurring.

Encouraged by these training results, I maintained this model architecture but expanded to a set of fifteen ragams. These ragams were *kalyani, bhairavi, mohanam, carukesi, todi, varali, nattakurinji, kamavardhini, shankarabharanam, hindolam, shanmukhapriya, ritigaula, arabhi, anandabhairavi*, and *simhendramadhyamam*. These are some of the most frequently occurring ragams in Carnatic concerts, so a model capable of recognizing them could immediately be useful for real-world demonstrations to users familiar with Carnatic music. When training this fifteen ragam model, over the course of 42 epochs (out of an initially planned 250), the training loss decreased from 5.1784 to 0.2451, indicating that the model was effectively learning from the training data. Furthermore, the validation accuracy improved from 17% to 87%, showcasing the model's increasing proficiency in generalizing to unseen data. However, training was halted prematurely at the 42nd epoch due to early stopping, triggered by a lack of improvement in the validation loss, suggesting that further training might not have led to better generalization.

Setting up Feature Importance Mapping

Lastly, I re-appended the original numerical feature names (Chroma, RMSE etc.) to a list, as they had been removed in the training data, such that they could later be mapped to the weights to which the trained model had converged. This would allow us to understand what specific aspects of the audio file the model had been considering when making its prediction, and hypothesize what aspects of Carnatic music might explain these relative weights. In some sense, this exercise could help me understand the "computational essence" of a ragam. This is an interesting philosophical question, as Carnatic music teachers always emphasize

capturing the "essence" of the ragam while singing, and this feature map would let me understand the quantitative physical phenomena to which they are probably referring.

The results of these ANN model implementations, their performance on unseen testing data, and the relative feature importance will be discussed in the results chapter of this report.

5.3 Recurrent Neural Networks (RNNs)

RNNs introduce the concept of memory into neural networks, allowing for the processing of sequences of data. The key feature of an RNN is the hidden state $\mathbf{h}_t$, which captures information from all previously seen elements in the sequence.

The basic RNN update equation is:

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}) \tag{5.1}$$

where:

- $\mathbf{h}_t$ is the hidden state at time t,
- $\mathbf{x}_t$ is the input at time t,
- $\mathbf{W}_h$ is the weight matrix for the hidden state,
- $\mathbf{W}_x$ is the weight matrix for the input,
- $\mathbf{b}$ is the bias.

[4]

5.4 Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks are a special kind of Recurrent Neural Network (RNN) capable of learning long-term dependencies. They are designed to avoid the long-

term dependency problem in traditional RNNs, making them particularly useful for tasks involving sequential data such as time series analysis, natural language processing, and audio data.

LSTMs enhance RNNs with mechanisms called gates, enabling the network to better regulate the flow of information. An LSTM unit typically includes a cell state $\mathbf{c}_t$, and three gates: forget gate $\mathbf{f}_t$, input gate $\mathbf{i}_t$, and output gate $\mathbf{o}_t$.

LSTM Update Equations

The LSTM update equations are as follows:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f) \tag{5.2}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_i[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i) \tag{5.3}$$

$$\tilde{\mathbf{c}}_t = \tanh(\mathbf{W}_c[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_c) \tag{5.4}$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \tag{5.5}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o) \tag{5.6}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \tag{5.7}$$

[4]

where:

- $[\mathbf{h}_{t-1}, \mathbf{x}_t]$ denotes the concatenation of $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$,
- $\odot$ denotes the Hadamard product (element-wise multiplication),
- $\mathbf{W}$ and $\mathbf{b}$ with subscripts denote the weight matrices and bias vectors for each gate,
- $\tilde{\mathbf{c}}_t$ is the candidate cell state.

5.5 LSTM Applicability

LSTMs, with their complex gating mechanisms, provide a powerful tool for modeling sequential data, building on the fundamental principles of linear algebra and neural network design. Their ability to capture long-term dependencies makes them a cornerstone of modern sequential data processing and prediction tasks. Despite their typical association with sequential data, LSTM models can still be effective for tasks involving non-sequential vector data, such as the numerical features extracted from audio files in ragam identification. LSTMs are renowned for their ability to capture long-term dependencies and temporal dynamics, which can be crucial for understanding the intricate nuances present in audio signals. In the context of Carnatic music, where melodies unfold over time with intricate variations, LSTMs can leverage their memory cells to retain relevant information from past observations, allowing them to discern subtle patterns and transitions that are characteristic of different ragams. By learning to associate specific combinations of numerical features with ragam labels over time, LSTMs can effectively capture the underlying structure and complexity of Carnatic music compositions, leading to accurate identification of ragams based on the extracted audio features.

5.6 LSTM Implementation Details

As with the ANN models, I loaded in the numerical feature vectors and labels for all audio splits for a pool of two ragams into X and y data frames in Python, once again performing a 90% - 10% train-test split of the data and converting all ragam names to integers in the y data-frame. These two ragams were *nilambari* and *todi*.

The first LSTM model I designed was a binary classification model with the goal of distinguishing the *Todi* from *nilambari*. These ragams were both among the 33 most commonly occurring ragams mentioned earlier, and I thought they would be good candidate ragams for the first experiment as they sound very different. Todi, with its flattened second and sixth

degrees, resembles the minor scale in Western music, with a darker and more melancholic sound. On the other hand, Nilambari, with its flattened third and sixth degrees, tends to evoke a sense of tranquility and serenity.

This two ragam LSTM model was once again implemented using `keras`. The model was constructed using the Sequential API in Keras, tailored for sequential data processing. It began with a first LSTM layer featuring 128 units, an input shape of (50, 1), a dropout rate of 5%, and a recurrent dropout of 25%, set to return full sequences. This was followed by a second LSTM layer with 64 units, configured to output a single vector by setting `return_sequences=False`. A dense layer with 2 units and a softmax activation function was added as the output layer, suitable for binary classification. The model was compiled using the Adam optimizer with a learning rate of 0.0003, employing `sparse_categorical_crossentropy` as the loss function and tracking accuracy as the performance metric. An early stopping callback was initialized to monitor the validation loss and halt training if no improvement was observed over 5 epochs, to mitigate overfitting. [10]

Due to LSTM units being more computationally intensive, this model took significantly longer to train than the ANN models. Over 18 epochs, the model's training loss decreased from 1.6930 to 0.1385, and its training accuracy improved from 40.94% to 95.74%, indicating effective learning and adaptation to the training data. Meanwhile, the validation accuracy increased from 53.33% to 94.29%, with validation loss reducing from 0.9955 to 0.2224, showing the model's growing ability to generalize to unseen data. Despite these initial improvements, early stopping was triggered at the 18th epoch, suggesting that further training did not significantly enhance validation performance.

Given the success of this model, I scaled up to five ragams, *kalyani, bhairavi, latangi, nattaikurinji*, and *mohanam*. During training, over 52 epochs, at which point early stopping was triggered, the training loss of the model decreased from 1.5902 to 0.3286, and its training accuracy increased from 24.05% to 88.63%. Similarly, the validation loss decreased from 1.5409 to 0.2857, while the validation accuracy improved from 28.44% to 90.56%. These

changes indicate that the model effectively learned from the training data, as evidenced by the consistent decrease in loss and increase in accuracy. Additionally, the model's ability to generalize to unseen data improved, as shown by the enhancements in validation accuracy.

With that being said, this represented a roughly 5% drop in validation accuracy from the two-model case, which is to be expected given the increased problem complexity. To accommodate the increasing problem complexity, in my next edition of the model, which aimed to classify ten ragams, I incorporated a larger LSTM input layer with 256 units. These ten ragams, once again selected from the pool of the thirty-three most commonly-occurring ragams, were *kalyani, bhairavi, mohanam, nattakurinji, carukesi, kapi, varali, kamavardhini, shankarabharanam*, and *hindolam*. During training, over the course of the 94 epochs prior to early stopping, the model's training loss steadily declined from 2.2713 to 0.3099, and its training accuracy increased from 13.67% to 90.28%. Concurrently, the validation loss decreased from 2.2428 to 0.2471, while the validation accuracy improved from 16.94% to 92.61%. These changes signify that the model was effectively learning from the training data, as evidenced by the consistent decrease in loss and increase in accuracy across both training and validation sets.

The results of the LSTM models described here, as well as their performance on unseen testing data, will be discussed in the results chapter that follows.

5.7 Bridge from LSTM Networks to Transformer Models

Transformers use parallel processing and self-attention mechanisms to handle sequences, significantly improving over RNNs and LSTMs in many aspects. A transformer model consists of an encoder and decoder, each containing multiple layers of multi-head attention and feed-forward networks, along with positional encoding. Carnatic music ragams are intricate melodic frameworks that require understanding nuanced sequences and patterns,

making them ideal candidates for Transformer-based models.

Self-Attention Mechanism

The self-attention mechanism allows each position in the sequence to attend to all positions in the previous layer simultaneously:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Head Attention

Multi-head attention runs several attention operations in parallel:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

[5]

5.8 BERT and Sequence Classification

BERT (Bidirectional Encoder Representations from Transformers) is a Transformer-based model pre-trained on a large corpus, which has shown great success in various NLP tasks, including sequence classification. BERT's ability to capture deep contextual relationships within sequences makes it well-suited for classifying ragams in Carnatic music. The model's bidirectional nature and fine-tuning capabilities allow it to learn the subtle nuances and complex patterns characteristic of ragams, offering a promising approach to this unique classification task. [5]

Transformers, despite being primarily designed for sequential data processing, offer unique

advantages for handling non-sequential vector data in tasks like ragam identification. Transformers are renowned for their attention mechanisms, which enable them to focus on relevant information within the input data, irrespective of its sequential nature. In the context of ragam identification, where the relationships between different numerical features may be complex and non-linear, Transformers can leverage their attention mechanisms to identify salient features that are indicative of specific ragams. By attending to relevant feature combinations and capturing global dependencies within the input data, Transformers can effectively learn to differentiate between different ragams based on the extracted numerical feature vectors. Additionally, Transformers' capacity for hierarchical representation learning allows them to capture abstract relationships and patterns within the data, enabling them to discern subtle distinctions between ragams that may not be apparent from individual feature values alone. Overall, Transformers offer a powerful framework for modeling complex relationships within non-sequential vector data, making them well-suited for tasks like ragam identification in Carnatic music compositions.

5.9 BERT Implementation Details

I once again implemented this model in a Colab python notebook, using the Hugging Face `transformers` library. Hugging Face is a company and online community that provides a vast collection of pre-trained models designed to simplify the use of state-of-the-art natural language processing (NLP) models. `BertForSequenceClassification` is a model architecture provided by Hugging Face's `transformers` library, specifically designed for the task of sequence classification, such as sentiment analysis or spam detection. It leverages the BERT model, fine-tuning it for classification tasks by adding a classification layer on top of the pre-trained BERT model's output.

Much as with the prior two models, a data frame was created from feature data X and labels y, with the dataset subsequently split into training and test subsets using the same

90% - 10% split as before. The training data was then separated into features (`X_train`) and labels (`Y_train`), with a similar process for the test data. The features were reshaped to match the expected input format for BERT, and a dictionary once again mapped ragam names to integer labels.

The numerical features were converted into a text format, which was then tokenized using the BertTokenizer from the Hugging Face library. This tokenizer was configured to handle the text data, ensuring uniform sequence lengths and converting the data into a format compatible with BERT. The tokenized data was wrapped in TensorDataset objects, and DataLoader instances were prepared for efficient batch processing during training and evaluation.

BertForSequenceClassification was then instantiated from the pre-trained 'bert-base-uncased' model, configured for a binary classification task with two output labels: the ragams *kalyani* and *bhairavi*, two extremely popular and common ragams. The AdamW optimizer was employed to optimize the model parameters, with a predefined learning rate and epsilon value.

The training loop involved setting the model to train mode, iterating over batches of data, computing the loss, performing backpropagation, and updating the model parameters using the optimizer. After each epoch, the average training loss was calculated and reported. During training, I observed that over the course of four epochs, the BERT model demonstrated significant learning and improvement in performance, as evidenced by the consistent decrease in training loss. Initially, the loss was relatively high at 0.6350 in the first epoch, but as the model underwent training, it rapidly assimilated the patterns in the data, leading to a reduction in loss to 0.4361 by the second epoch. This trend of decreasing loss continued in subsequent epochs, with the loss further reducing to 0.3542 in the third epoch and ultimately reaching 0.2863 by the fourth and final epoch. This progression indicates effective learning and optimization of the model's parameters.

While the two ragam transformer model showed promise and evidence of learning, I

was not able to replicate this success for larger pools of ragams. Training loss decreased much more slowly for a three ragam model, and essentially flat-lined for a five ragam model. Compounding the issue was the length of time each BERT model took to fine-tune and train. Transformer models, like BERT, typically take longer to train than LSTM and simpler ANN models due to their complex architecture and the extensive number of parameters involved. Transformers use self-attention mechanisms to process input data in parallel rather than sequentially, which, despite being efficient at capturing long-range dependencies in my ragam audio data, requires significant computational resources. Additionally, the pre-training and fine-tuning stages of models like BERT involve processing vast amounts of text data, further contributing to the training duration.

The results of the Transformer models described here, as well as their performance on unseen testing data, will be discussed in the results chapter that follows.

5.10 Why LSTMs and Transformers for Non-Sequential Data?

Given that LSTMs and Transformers are typically applied to sequential, time-series data, their application here to the numerical features vector input data instead of the spectrogram data might be surprising. After all, the mel-spectrograms are in fact sequential data, whereas there is no inherent order to the fifty columns of statistics in the numerical features vector. You could scramble the order of the columns in the vector, so long as you keep the ground truth ragam label in column 51. So why am I training these models on non-sequential data?

Feeding non-sequential data to LSTM and Transformer models can lead to superior performance for a number of reasons. Firstly, these models are adept at representation learning, capturing essential patterns and relevant information regardless of data sequence. Their proficiency in understanding temporal context and complex feature interactions allows them to discern nuanced patterns, making them particularly effective for tasks requiring a

deep understanding of temporal dependencies, such as ragam identification. Additionally, the flexibility and hierarchical feature learning capabilities of these models, especially transformers, enable them to process information across various abstraction levels.

However, I noted that LSTM and Transformer models either failed to train on or performed poorly with spectrogram data, likely due to several factors. Spectrograms, which represent audio signals in the time-frequency domain, present unique challenges that may not align well with the sequential processing strengths of LSTMs and Transformers. These models might struggle with extracting meaningful features from the frequency information in spectrograms due to their inherent design for temporal data. The high dimensionality of spectrograms poses additional challenges, potentially leading to inefficient data processing and overfitting. Moreover, the transformation into spectrograms might result in the loss of crucial temporal dynamics, impeding the models' ability to leverage their strength in capturing long-range dependencies. It's also possible that numerical features derived from spectrograms could more effectively encode key information, making them more suitable for LSTM and Transformer models. Lastly, the suboptimal performance might stem from inadequate model architecture and hyperparameter optimization on my part, for handling spectrogram data effectively.

5.11 Convolutional Neural Networks (CNNs)

Now, we will introduce the mathematics behind Convolutional Neural Networks and the implementation details for this model in my project. The three models described up until this point all made use of the numerical features vector representation of the audio splits; the CNN model will make use of the mel-spectrogram image data that I have described in detail in chapter four of this report.

Convolutional Neural Networks (CNNs) are a specialized kind of neural network for processing data that has a known grid-like topology, such as images. CNNs make use of a

mathematical operation called convolution, which, in the context of neural networks, can be thought of as a weighted sum of inputs like in traditional neurons, but with a key difference: the weights (or the convolutional filter) are shared across the input space.

This weight sharing significantly reduces the number of parameters that need to be learned, making CNNs much more efficient than fully connected networks for tasks like image recognition. Additionally, CNNs are designed to automatically and adaptively learn spatial hierarchies of features, from low-level edges and textures to high-level patterns and object classes.

In a CNN, a convolutional layer consists of several convolutional filters that are applied to the input images (or feature maps from the previous layer) to produce feature maps. These feature maps then pass through non-linear activation functions, just like in traditional neural networks. Pooling layers are often added after convolutional layers to reduce the dimensionality of the feature maps, helping to make the network more computationally efficient and less prone to overfitting.

The transition from fully connected layers to convolutional layers marks a significant shift in neural network architecture, enabling the effective processing of spatial data and leading to breakthroughs in fields such as computer vision. Through the lens of linear algebra, both fully connected and convolutional layers can be understood as transformations of input data, guided by learned weights, to capture and utilize complex patterns in the data.

1. Convolution Operation

The convolution operation is central to CNNs. It involves sliding a filter or kernel over the input data and computing the dot product of the filter and local regions of the input. This operation is crucial for feature extraction from the input data.

2. Mathematical Representation

Consider an input image represented as a 2D matrix I of size $M \times N$ and a filter F represented as a 2D matrix of size $K \times K$. The convolution operation is defined as:

$$(S * F)(i, j) = \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} I(i + u, j + v) \cdot F(u, v) \tag{5.8}$$

where $(S * F)(i, j)$ is the output of the convolution at position (i, j). [4]

3. Feature Learning

Through the convolution operation, the CNN learns to detect various features in the input data. Multiple filters can be used to detect different features, creating a set of feature maps.

4. Pooling

Pooling layers follow the convolutional layers and serve to reduce the dimensionality of the feature maps. This reduction helps in achieving abstraction in feature representation.

5. Max Pooling

Max pooling is a common pooling operation that selects the maximum element from the region of the feature map covered by the pool. Mathematically, it is represented as:

$$P_{\max}(i, j) = \max_{0 \leq u < P, 0 \leq v < P} S(i \cdot P + u, j \cdot P + v) \tag{5.9}$$

where P is the size of the pooling region. [4]

6. Fully Connected Layers

After several convolutional and pooling layers, CNNs typically include fully connected layers. These layers connect every neuron in one layer to every neuron in the next layer and are

crucial for classification tasks.

7. Mathematical Representation

In a fully connected layer, the operation can be represented as:

$$Y = f(WX + b) \tag{5.10}$$

where X is the input vector, W is the weight matrix, b is the bias vector, and f is the activation function. The output Y is used for classification or regression tasks.

Applications to Ragam Identification

Overall, CNNs are highly effective for ragam recognition due to their capacity to extract hierarchical feature representations from audio data. In Carnatic music, CNNs can capture the intricate melodies of ragams through operations like convolution, non-linearity, and pooling. By analyzing the frequency-time domain of mel-spectrograms, CNNs can discern nuanced ragam-specific motifs, enabling automated classification of ragams based on their unique musical characteristics.

5.12 2-D Convolutional Neural Network Implementation Details

The first model attempted was a binary classification CNN model aiming to distinguish between clips in the ragam *nilambari* and the ragam *bhupalam*. The code for this model begins in the cell of the notebook linked here. The architecture is a sequential `Keras Convolution2D` model with two layers, `ReLu` activation functions, flattening, and 2 dense layers with `softmax` output classification activation function. The `Keras sgd` optimizer was

used to calculate gradient with a learning rate of .01. Binary cross entropy and accuracy were used for the loss and accuracy metrics during training with a batch size of ten, twenty epochs, and a validation split of 10% (the existing train-test-split was 9:1). Categorical accuracy is simply defined as the number of correct predictions divided by the total number of predictions, a standard measure of accuracy in previous studies in ragam recognition. The labels were encoded from the ragam names 'nilambari' and 'bhupalam' to the numbers 0 and 1. However, in the first few passes of the network, I noticed that learning was not occurring, and that accuracy was remaining consistently at .3, well below even the random guess percentage of 50% that one would expect to be the lower bound of a binary classification problem. I also noticed that the validation loss remained relatively uniform over the epoch progression, suggesting that overfitting was likely occurring. After discussions with Dr. Kreiman, I altered the model parameters by adding max-pooling layers, changing the loss metric to categorical cross-entropy, and reducing the learning rate to .001.

Thus, the next experiment was to expand the architecture of the binary classification problem to accommodate multi-category classification, specifically for classifying songs spanning the pool of ten ragams randomly selected for CNN experimentation mentioned above. Using the same architecture as in the binary classification case leads to vastly reduced learning and accuracy scores, averaging about 35% validation and testing accuracies after 20 epochs in this newly expanded pool. To address this, I changed the model parameters by adding two more 2D CNN layers with `ReLu` activation functions and MaxPooling to improve learning, adding a larger dense layer to prevent overfitting, changing the output dense layer to size 10, and changing the kernel initializer to sample from a uniform distribution. While this improved learning to over 80%, I noticed that validation loss remained constant and testing accuracy was still too low.

To address this issue, in my next and final CNN model, I added an even bigger 2D CNN layer to improve feature learning and implemented early stopping with dynamic learning rate adjustment. Essentially, learning would be terminated if validation loss did not drop

for four epochs, and the learning rate would be adjusted by a factor of 0.1 for every epoch where validation loss did not directly decrease.

Applying similar processes to the data as described above, including encoding the ten ragams under consideration by the digits 0-9, and deploying this larger model architecture with early stopping and dynamic learning rate reduction achieved better results, which will be described in the results chapter of this report that follows. The ragams included in this ten ragam model were *mohanam, nilambari, ahiri, amruthavarshini, bhupalam, kalyanavasantam, manirangu, revati, simhendramadhyamam*, and *yamunakalyani*.

Chapter 6

Results

6.1 Overview

Results across all four models were quite promising. The models I created yielded test accuracies (performance on unseen data) north of 90%, with some above 95%, suggesting they could be highly effective at detecting Carnatic ragams in the real world. In this section, I state the results and walk through some salient visualizations for each model.

While reading through this section, please refer to the methods chapter for implementation details of any models whose results are stated here. [1] [2]

6.2 Artificial Neural Network (ANN) Model Results

10-Ragam Model

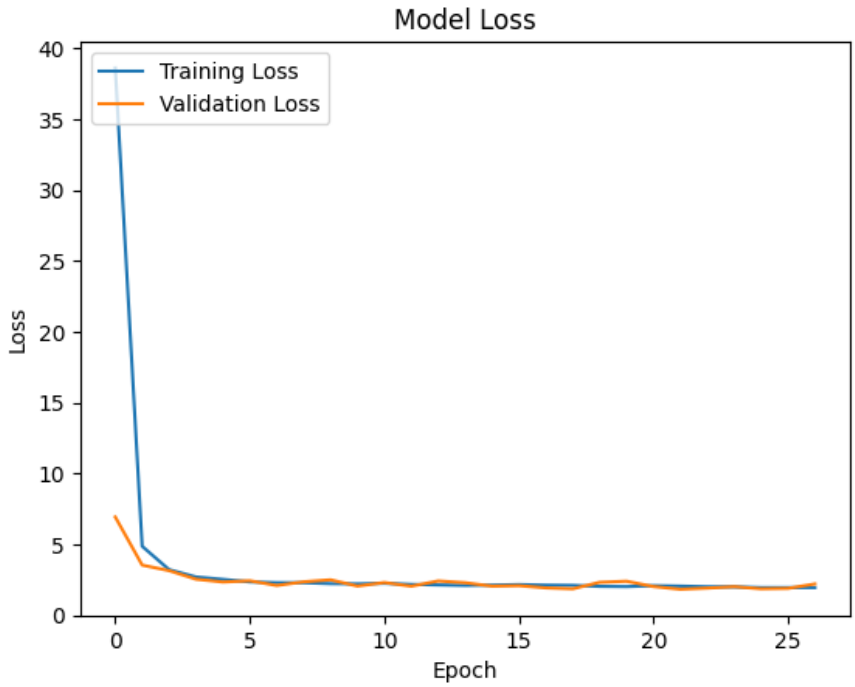
Initial Loss and Accuracy Curves:

As stated in the methods section, in the initial ANN models with simpler architectures, training and validation loss stalled, and the accuracy did not increase beyond 33%, which

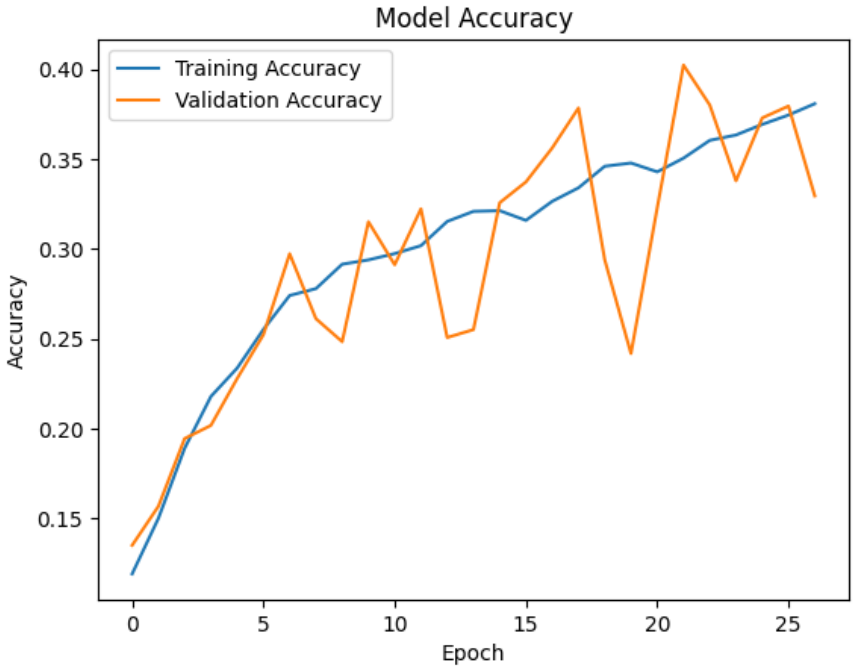
[1] Note that I used ChatGPT to help me write and edit some of the language in this chapter.

[2] Some of the language and visualizations in this chapter come from my Neuro 240 final report, which I produced for Dr. Kreiman in 2023, and which is cited in my bibliography below.

while better than a random guessing score of around 10% for a 10 ragam classification model, was not useable. The figures below demonstrate the loss and accuracy curves from one of these initial, simplistic models.



(a) Loss Curves for Initial ANN Models



(b) Accuracy Curves for Initial ANN Models

Figure 6.1: Loss and Accuracy Curves for Initial ANN Models

Confusion Matrix:

However, after significantly scaling up the complexity of the ANN models, validation accuracy increased to 95% and training loss decreased markedly. As can be seen in the following picture, testing accuracy for the ten-ragam model was evaluated to be 93.6% on the unseen data, showing that this ten-ragam model was highly effective at identifying ragams under real-world conditions.

```
63/63 [==============================] - 0s 2ms/step - loss: 0.2595 - accuracy: 0.9360
63/63 [==============================] - 0s 1ms/step
```

Figure 6.2: Testing accuracy printed by the model during evaluation

This highly effective performance of the 10-ragam ANN model on unseen testing data can be seen in greater detail in the following confusion matrix. The numbers along the diagonal reflect correct classifications with the off-diagonal numbers reflecting the incorrect classifications.

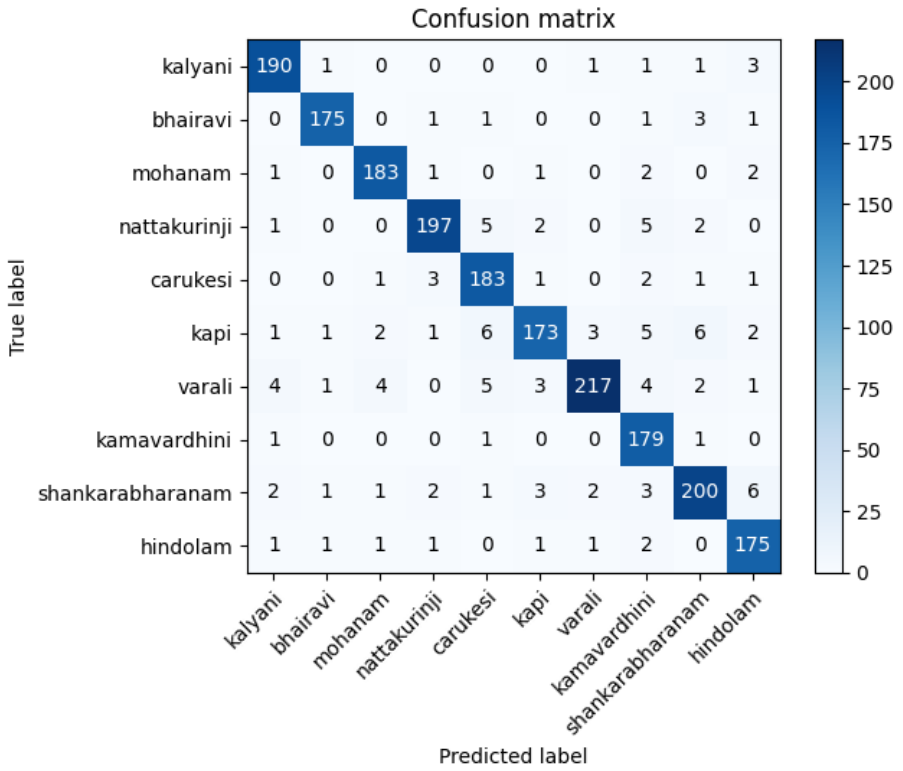


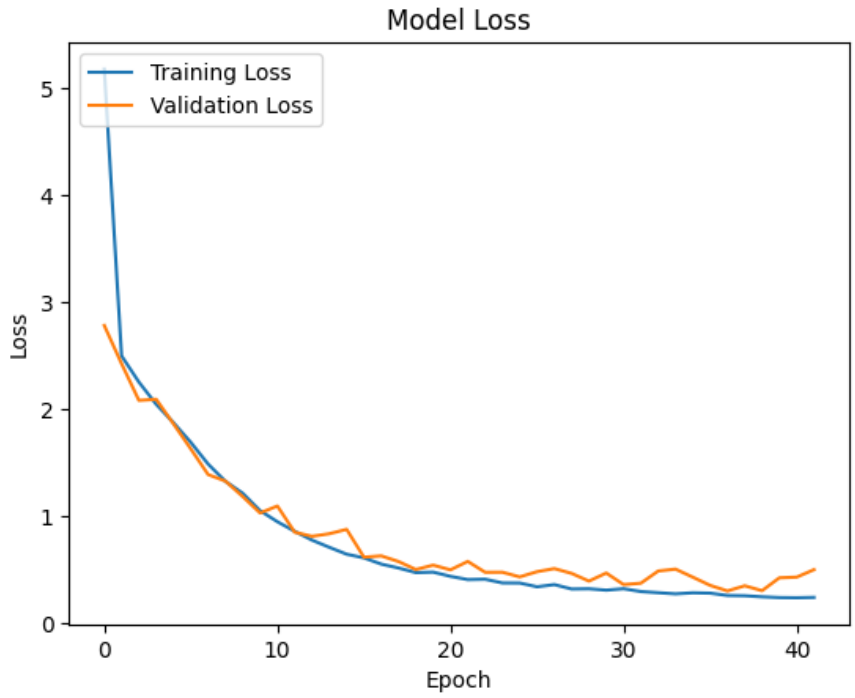
Figure 6.3: Confusion Matrix showing the True vs. Predicted labels in the unseen testing data

As you can see from the high counts across the diagonal in the confusion matrix above, the ten-ragam model reliably classified most audio splits of unseen, real-world testing data into the correct ragams.

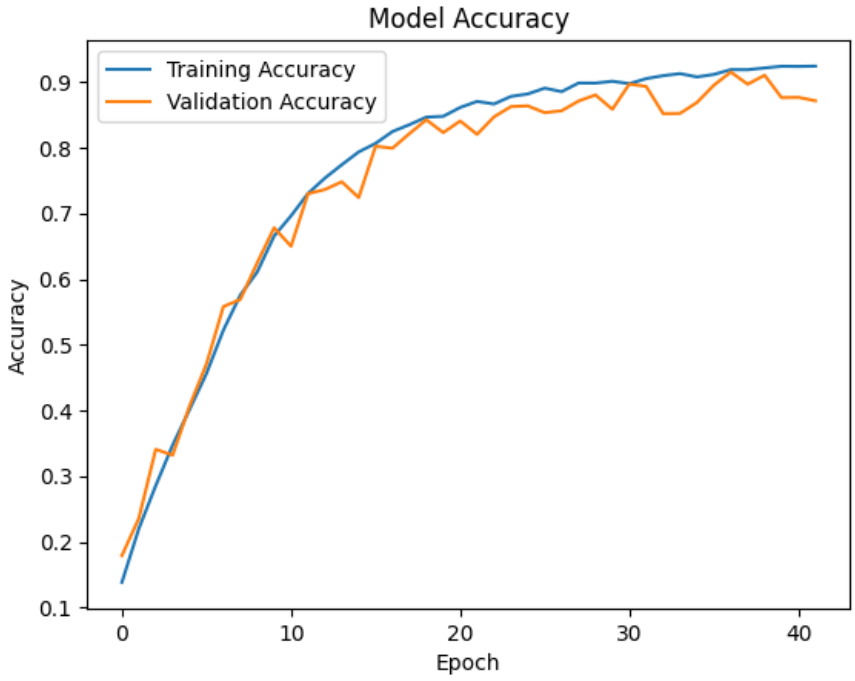
15-Ragam Model

Loss and Accuracy Curves:

As stated in the implementation section in the methods chapter, training accuracy greatly increased and loss was significantly reduced with the new model architecture. This is demonstrated by the curves below.



(a) Loss Curve for 15-ragam ANN Model



(b) Accuracy Curves for 15-ragam ANN Model

Figure 6.4: Loss and Accuracy Curves for 15-ragam ANN Model

AUC-ROC Curve:

This 15-ragam model ultimately achieved 86% testing accuracy on the unseen audio splits in the testing data. This is a promising result, as I did not encounter any models in my survey of the literature that could reliably identify 15 ragams, let alone 15 of the most popular ragams in all of Carnatic music, as this model does. [10]

Below is an Area Under the Receiver Operating Curve (AUC-ROC Graph), showing the AUC to be .93. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at different classification thresholds. The AUC, or "Area Under the Curve," represents the degree to which the model is capable of distinguishing between classes (ragams). The higher the AUC, the better the model is at correctly predicting ragams as themselves. The score ranges from .5, suggesting random guessing, to 1, which represents perfect predicting power.

An AUC of 0.93, as we have here, means that the model has a 93% chance of correctly distinguishing between a randomly chosen positive instance and a randomly chosen negative instance. This is generally considered to be a good performance, indicating that the model has a high predictive accuracy in classifying these 15 ragams.

The ROC curve helps visualize the trade-off between the true positive rate and the false positive rate. A model with high classification accuracy will have a ROC curve that hugs the upper-left corner of the plot, indicating a high true positive rate and a low false positive rate across various threshold values.

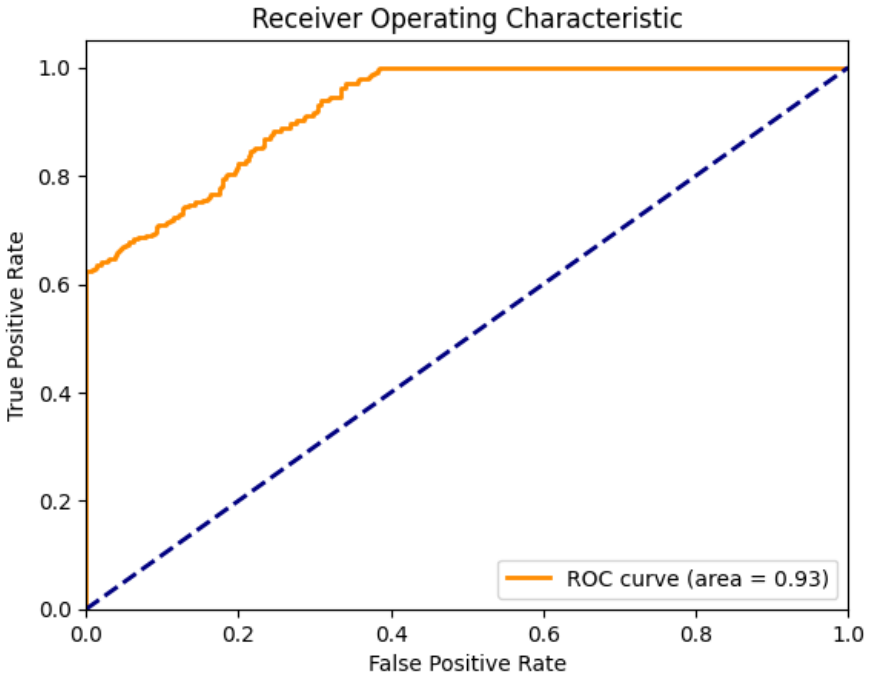


Figure 6.5: Area under the ROC Graph

Confusion Matrix:

The performance of the model on the testing data can further be seen in the confusion matrix below.

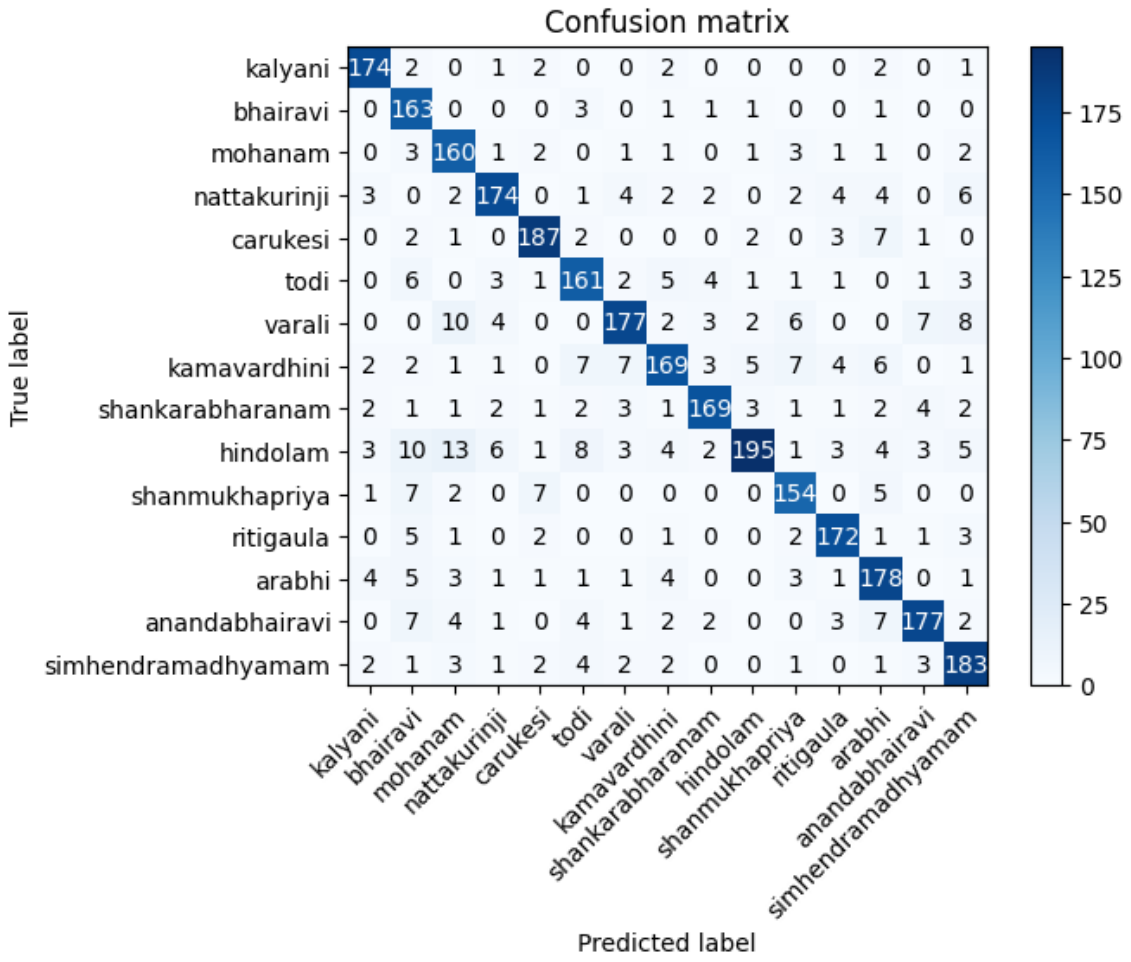


Figure 6.6: Confusion Matrix showing the True vs. Predicted labels in the unseen testing data

Note the high counts across the diagonal in the confusion matrix above, showing that the model reliably classified unseen testing data audio clips into their correct ragams.

Feature Importances:

After evaluating the 15-ragam ANN model, it became possible to visualize the relative importance of the various numerical features to the model's prediction. Below is a graph of the relative importance of all the features extracted.

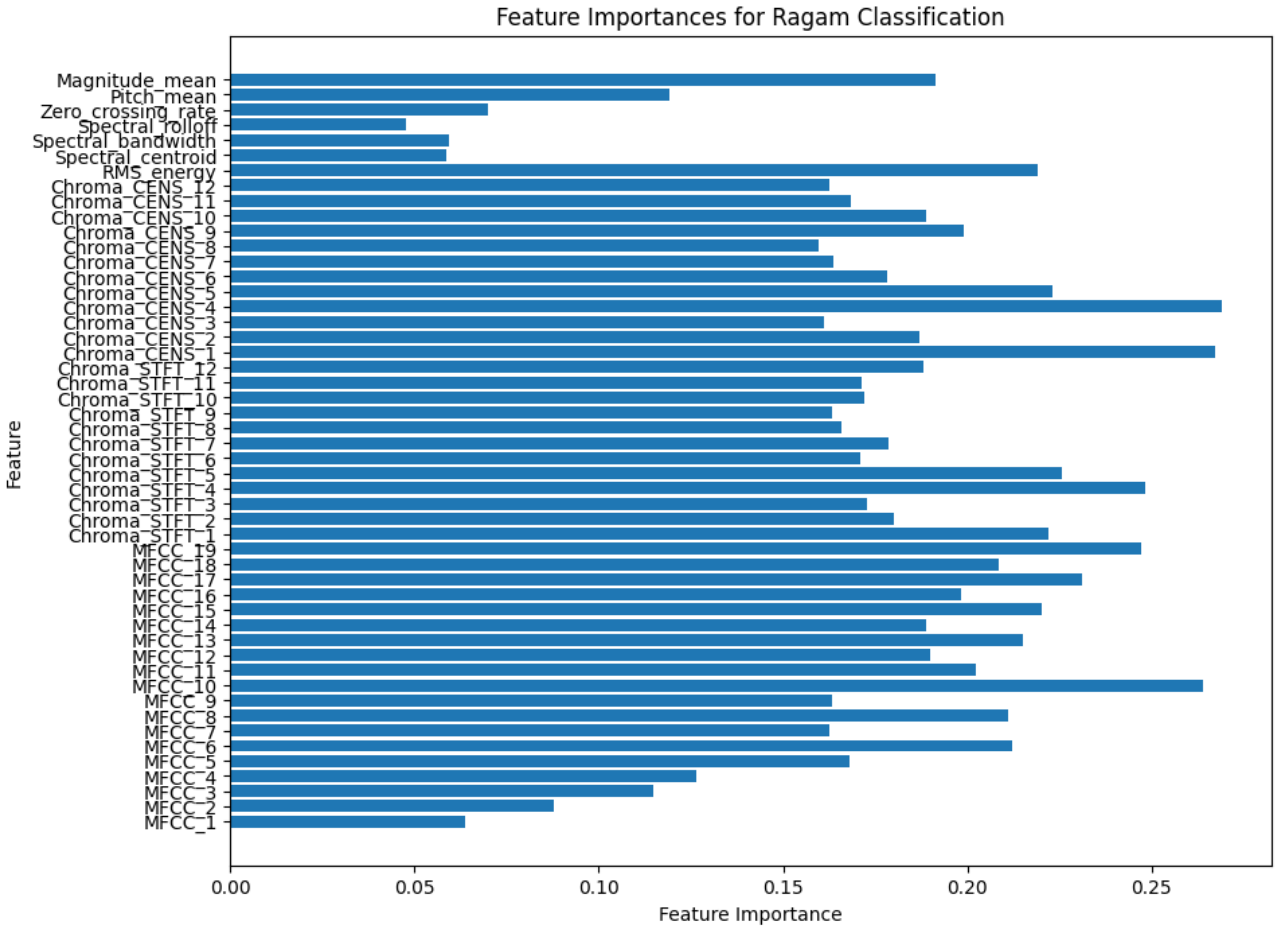


Figure 6.7: Feature Importance Map, showing which numerical features most inform ragam predictions

As I have mentioned earlier in this paper, this process allowed me to develop an understanding of the "computational essence" of ragams, by seeing what aspects of the audio file the model deems important to the classification. I found these results fascinating and developed some intuition for why the model might be assigning these relative weights based on my knowledge of Carnatic music.

Features with High Importance:

The high importance of the 10th MFCC (0.26378018) could indicate that this particular cepstral coefficient captures a critical aspect of the variation between different ragams. MFCCs are known to be effective in capturing timbral and spectral properties of audio signals, and

certain coefficients might be more relevant than others for distinguishing between specific types of music or vocal styles present in different ragams. Chroma CENS features with high importance scores (2nd: 0.26727563, 4th: 0.2691906) suggest that these specific pitch classes play a significant role in differentiating between ragams. The prominence of these specific Chroma CENS features could be related to the tuning or the characteristic intervals used in the ragams.

The RMS energy feature (0.19124618) having a relatively high importance score could relate to dynamics and loudness variations within the performances of different ragams, and the mean of magnitudes (0.19124618) also showing high importance suggests that the overall intensity of spectral magnitudes, possibly related to the loudness or energy of the signal, plays a significant role in classifying ragams, which makes sense as certain ragams such as *athana* are naturally much more strident than others.

Features with Low Importance:

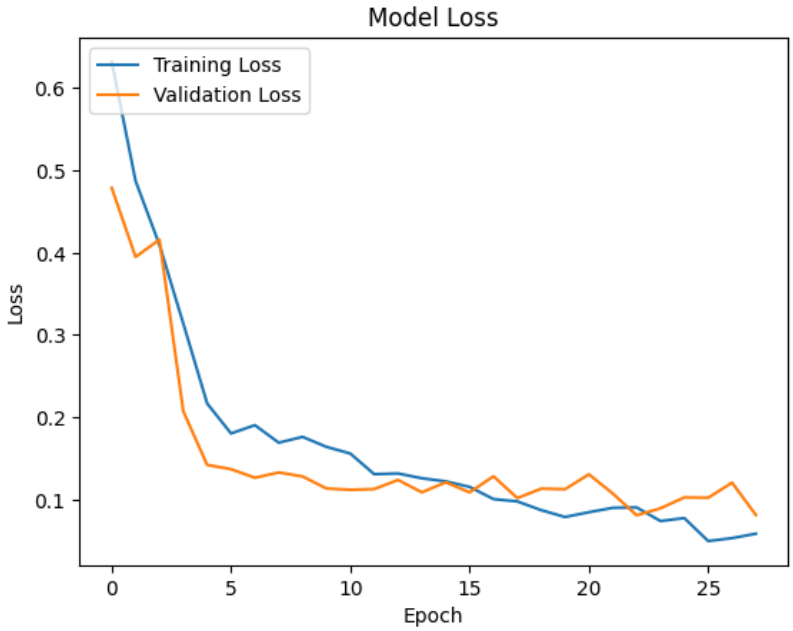
The relatively low importance of spectral roll-off (0.04749999) might indicate that the point below which a certain percentage of the total spectral energy is contained is not as distinctive for the classification of these particular ragams. This could be because ragams, given their harmonic complexity, might not be as effectively differentiated by the spectral shape's skewness alone. The relatively low importance assigned to the Zero-Crossing rate also makes sense, as the percussive and tempo elements, which ZCR captures, are far less likely than the Chroma or CENS features to be capturing melodic information from which a ragam prediction can be generated.

6.3 Long Short-Term Memory (LSTM) Model Results

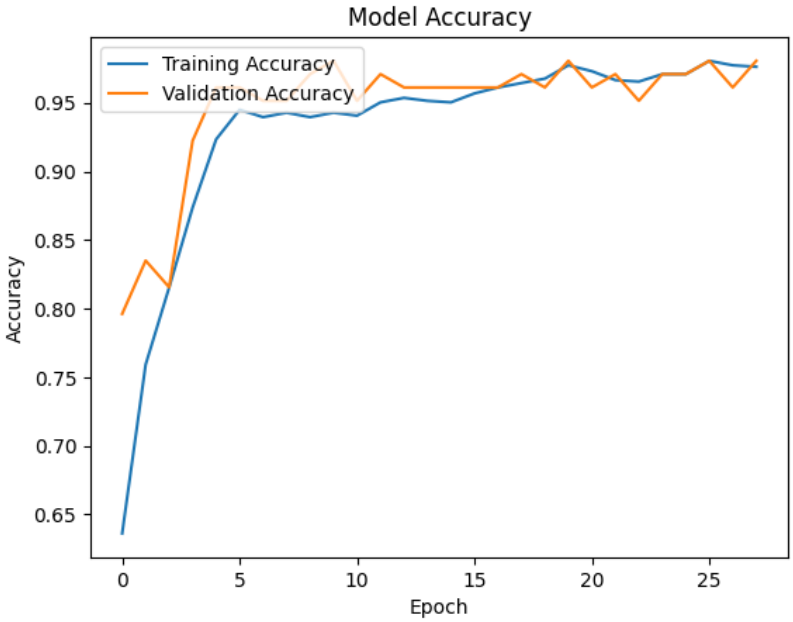
2-Ragam Model

Loss and Accuracy Curves

As described in the methods section, the 2-ragam LSTM model learned effectively from the data, with consistently decreasing loss and accuracy increasing all the way to 98%. This can be seen in the loss and accuracy curves below.



(a) Loss Curve for 2-ragam LSTM Model



(b) Accuracy Curves for 2-ragam LSTM Model

Figure 6.8: Loss and Accuracy Curves for 2-ragam LSTM Model

Confusion Matrix

The testing accuracy for the 2-ragam model distinguishing between *todi* and *nilambari* was an impressive 98%, meaning that this model was highly effective.

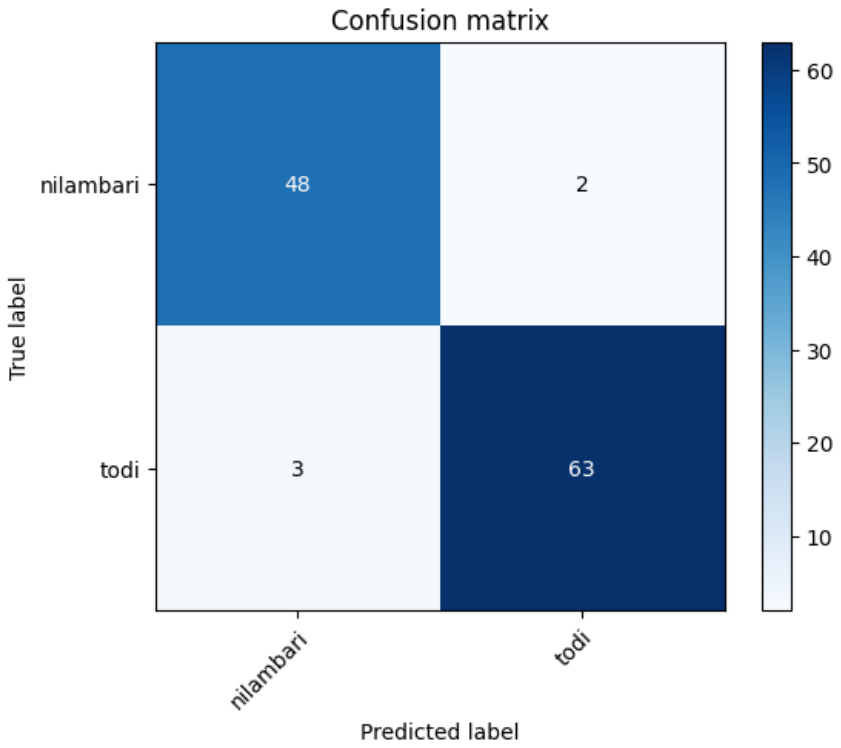


Figure 6.9: Confusion Matrix for 2-ragam LSTM Model

This impressive testing accuracy can be seen in the confusion matrix above, whose high diagonal counts demonstrate that this model hardly ever misclassified *nilambari* as *todi*, or vice versa.

10-Ragam Model

Confusion Matrix

As I described in the methods section, the LSTM models trained on five and ten ragams also learned effectively from the data, reaching testing accuracies of 90% and 92% respectively.

Below is a confusion matrix demonstrating the performance of the 10-ragam LSTM model on unseen, real-world testing data.

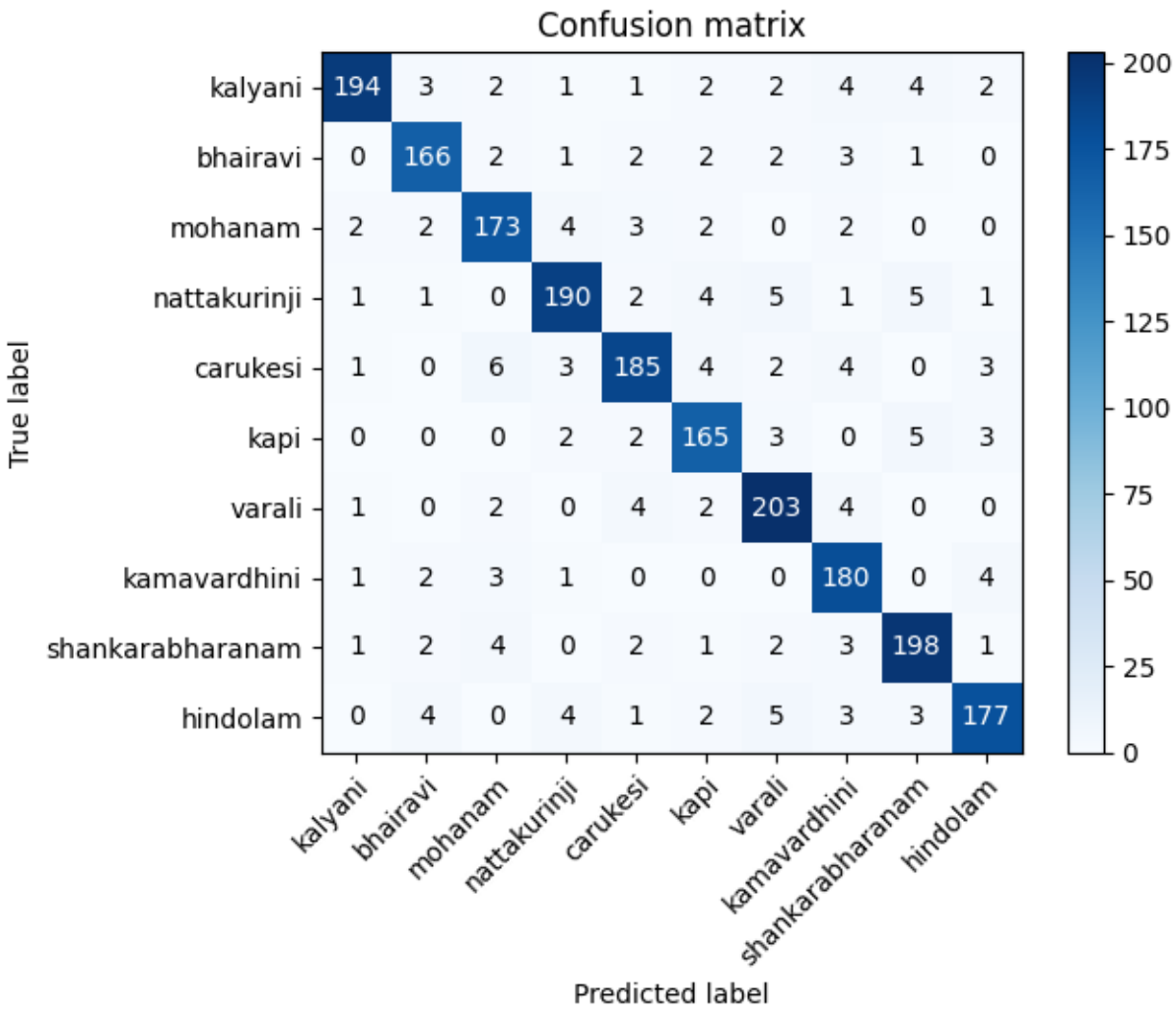


Figure 6.10: Confusion Matrix for 10-ragam LSTM Model

As you can see from the high diagonal counts in this visualization, the 10-ragam LSTM model was generally effective at classifying testing data audio splits into the correct ragam, though slightly less so than in the 10-ragam ANN case. Even so, based on my review of the literature, this is the largest pool of ragams on which LSTM models have shown high testing accuracy thus far. Furthermore, the 98% testing accuracy obtained in my 2-ragam model is one of the most accurate LSTM models developed for real-world Carnatic audio data thus far. [10]

6.4 Transformer (BERT) Model Results

2-Ragam Model

AUC/ROC Curve

The 2-ragam transformer model seemed to learn quite effectively from the data during training, as described in the methods chapter. Ultimately, during evaluation, the testing accuracy on unseen data was found to be 88.5%. While this absolutely shows evidence of the model learning and working effectively, this performance lags that of the ANN and LSTM models, perhaps because the transformer model, which is extremely data and computationally intensive, was not supplied with sufficient training data. The AUC score for this model was .86, which shows that the model has quite accurate predictive power. The ROC curve can be seen below.

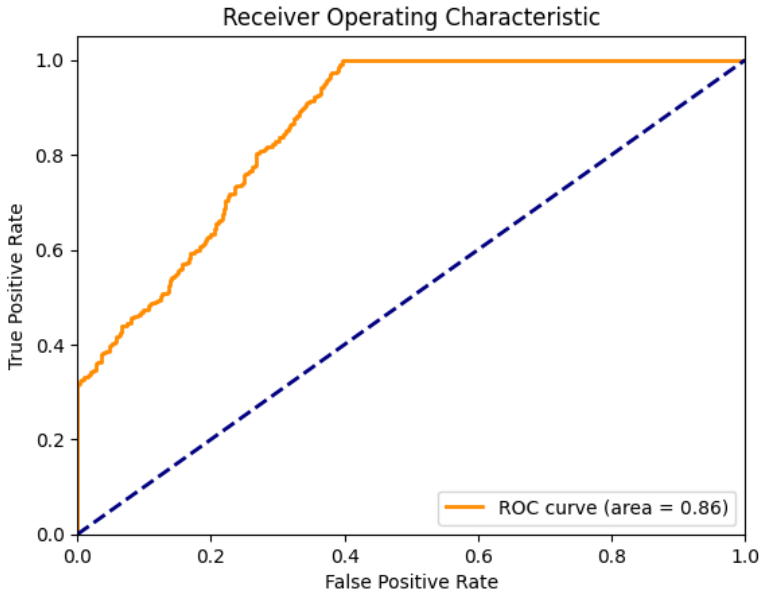


Figure 6.11: AUC/ROC Curve for 2-ragam Transformer Model

Confusion Matrix

The effective testing performance of the two-ragam transformer model can further be seen in the normalized confusion matrix below. Note that while the model almost always identifies

bhairavi correctly, it misclassifies *kalyani* nearly 20% of the time.

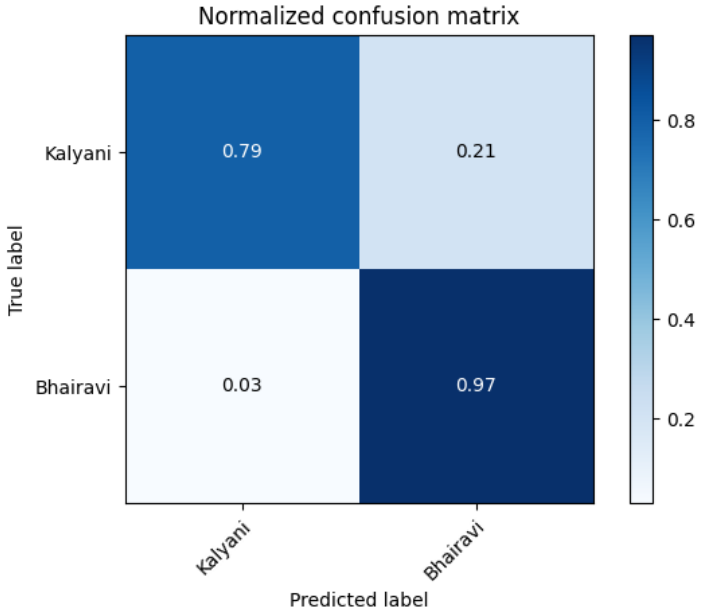


Figure 6.12: Confusion Matrix for 2-ragam Transformer Model

3-Ragam Model

Confusion Matrix

However, the testing accuracy began to break down as I scaled the model to identify more ragams. The testing accuracy for the 3-ragam case was measured to be 60%, which, while much better than a random guess score of 33%, is still not sufficiently high. In the five-ragam case, the model apparently did no learning at all and produced a discouraging 20% test accuracy, likely for the reasons described in the transformer section of the methods chapter. The confusion matrix below demonstrates this degradation in performance for the three-ragam case.

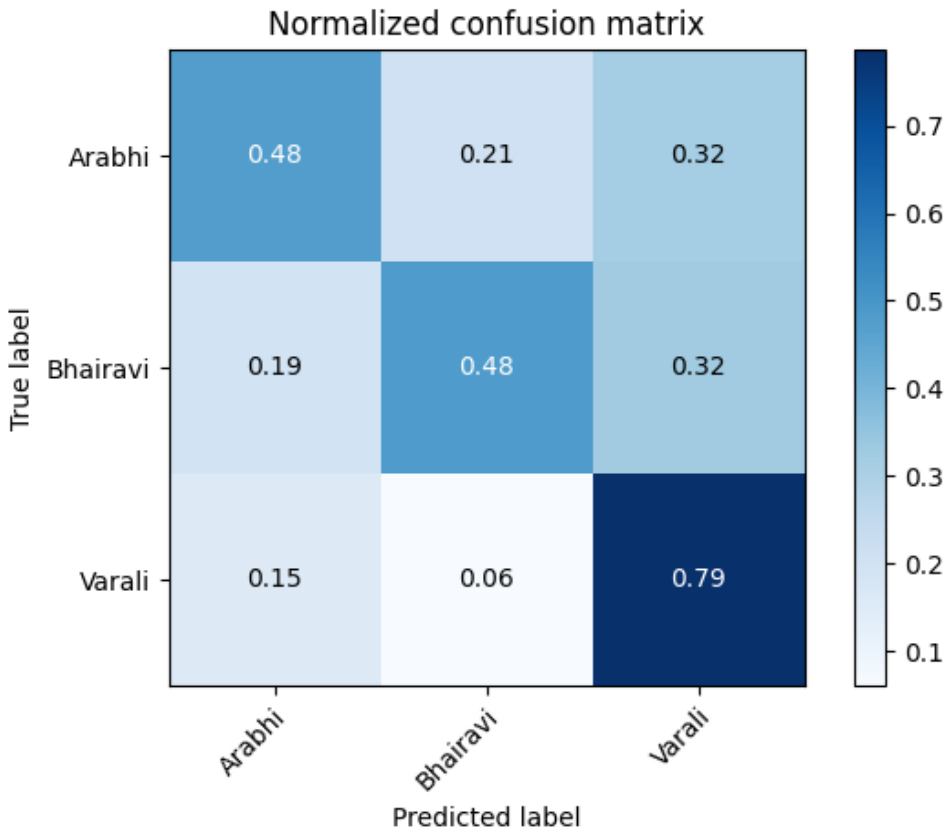


Figure 6.13: Confusion Matrix for 3-ragam Transformer Model

While these test accuracies are not comparable to those of the LSTM and ANN networks, this is still a promising result because it shows the potential of transformer models to learn to identify ragams, which has not yet been demonstrated in the literature. As they are today's cutting-edge model framework, transformers, and their attention mechanisms will continue to be applied in music classification tasks for the foreseeable future. [5]

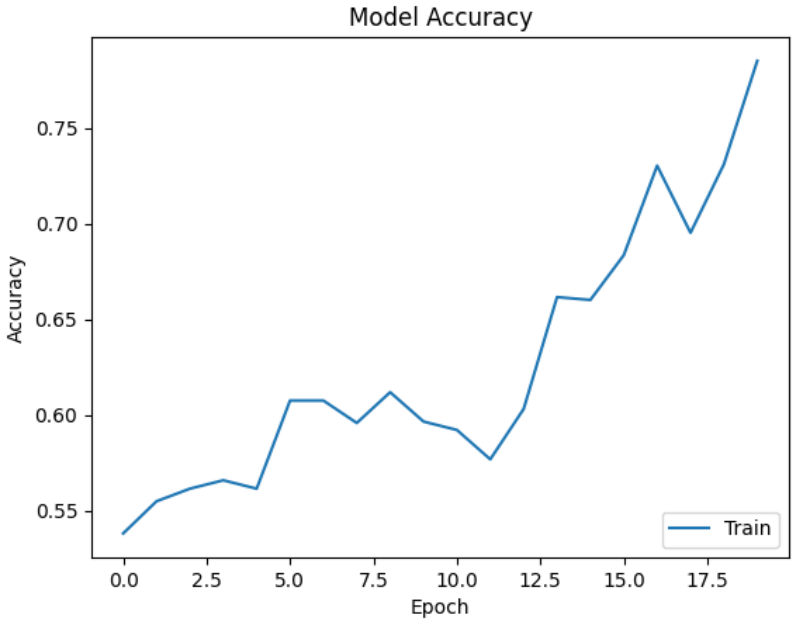
6.5 Convolutional Neural Network (CNN) Model Results

2-Ragam Model

Loss and Accuracy Curves

As can be seen in the graphs below, the model saw promising training and validation performance, decreasing from over .8 model loss after the first epoch and decreasing to .3 model loss by the 20th epoch. Model training accuracy also improved from just about the 50% random guess level to well over 90% by the 20th epoch. On the out-of-sample testing data, the model performed with 95.8% testing accuracy.

(a) Loss Curve for 2-ragam CNN Model



(b) Accuracy Curves for 2-ragam CNN Model

Figure 6.14: Loss and Accuracy Curves for 2-ragam CNN Model

Furthermore, I was able to index the data points and retain the indices for which points had been sorted into train and which into test. Using these indices, I was able to determine for which points the ragam label corresponding to the maximum predicted probability index

was not equal to the true label. This allowed me to inspect the misclassified spectrograms specifically. Using a `librosa` inverse function going from a mel-spectrogram to audio data, applying the Griffin-Lim algorithm (GLA), an algorithm used for audio signal processing, specifically for reconstructing a signal from its short-time Fourier transform computed during mel-spectrogram construction, it was possible to examine the audio for the misclassified tracks.

Introduction to Griffin-Lim Algorithm

The Griffin-Lim algorithm is a method used for phase reconstruction in audio signal processing. It essentially tries to estimate the phase of the time-domain signal from its magnitude spectrogram representation. Given a magnitude spectrogram $|X|$, where X is the Short-Time Fourier Transform (STFT) of the original audio signal x, the Griffin-Lim algorithm iteratively updates the phase of X to minimize the discrepancy between the reconstructed signal and the original signal. [13]

Let $X = |X| \cdot e^{i\theta}$ denote the complex STFT of the original audio signal x, where θ represents the phase information. The Griffin-Lim algorithm estimates the phase θ by iteratively updating it to minimize the discrepancy between the reconstructed signal $\hat{x}$ and the original signal x:

$$\theta^{(t+1)} = \arg\min_{\theta} \|x - \text{ISTFT}(|X| \cdot e^{i\theta^{(t)}})\|_2^2$$

where ISTFT denotes the Inverse Short-Time Fourier Transform. This optimization problem is typically solved iteratively using gradient descent methods. This example is on a magnitude spectrogram $|X|$, but it might just as well apply to a mel-spectrogram $|M|$. [13]

Implementation in Python

I then implemented the Griffin-Lim algorithm in Python, using `NumPy` for numerical computations and `librosa` for audio processing. The implementation involves initializing the phase of the time-domain signal, iteratively updating the phase using the Griffin-Lim algorithm, and converting the reconstructed signal back to the time domain, where it can be listened to as a `librosa` audio object.

By inspection, despite my efforts during the data preprocessing stage to remove such tracks, it became clear that many of the misclassified tracks corresponded to applause, percussion sections, or other such clips where a human listener may not have been able to determine the ragam of the song being played either, due to the absence of melody. This suggested that the true testing accuracy of the model for real-world applications was likely greater than 96% in this binary classification case.

Further inspection of the misclassifications yielded the following plots, and an ROC curve with an area of .99, which was an extremely promising result. The graph of this curve can be seen below.

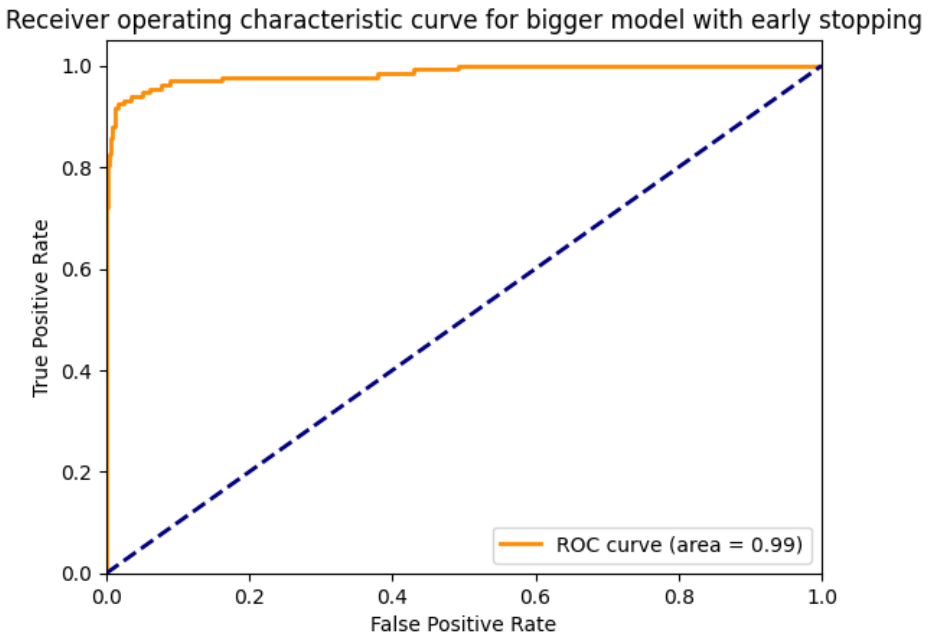
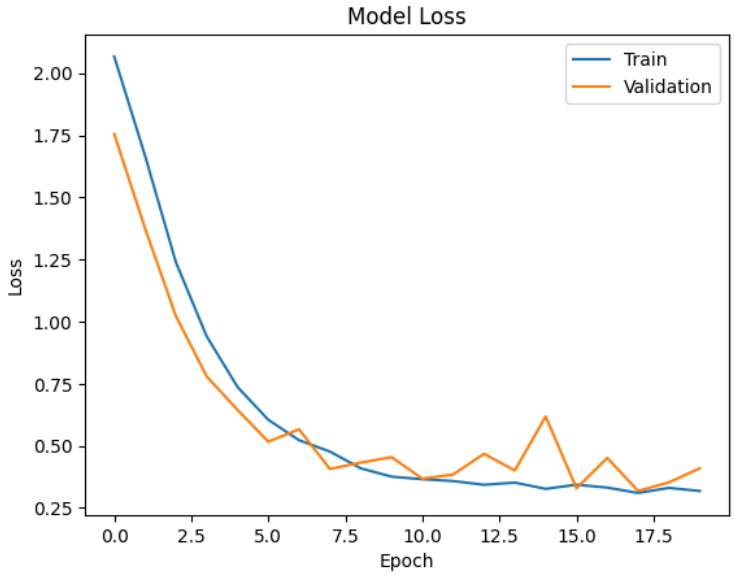


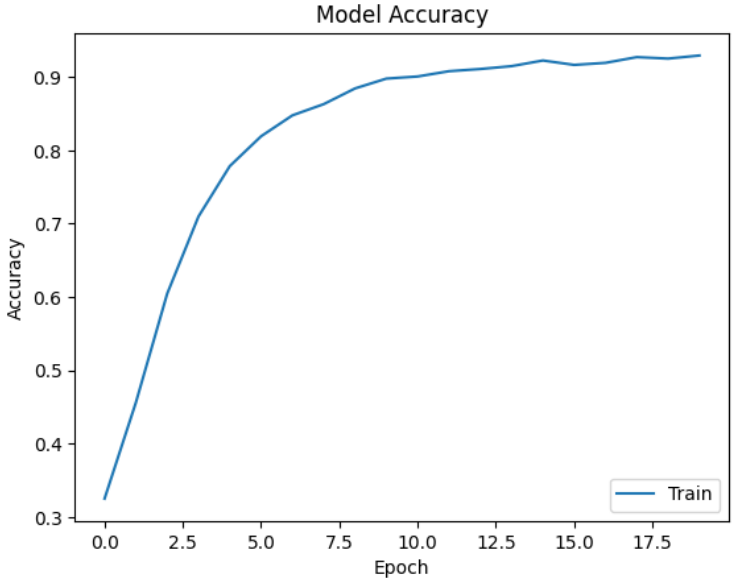
Figure 6.15: Confusion Matrix for 3-ragam Transformer Model

10-Ragam Model

The 10-ragam model showed clear evidence of learning from the data, including training and validation loss decreasing from 1.9 to .3, a training accuracy of 93%, and validation accuracy eclipsing 90%. Below are graphs of the training results, showing clear improvement in model loss and accuracy over the twenty epochs:



(a) Loss Curve for 10-ragam CNN Model



(b) Accuracy Curves for 10-ragam CNN Model

Figure 6.16: Loss and Accuracy Curves for 10-ragam CNN Model

Testing accuracy for out-of-sample unseen data was measured at 90%. Once again, misclassified data points were found to skew towards unclassifiable audio clips such as applause, speeches, and percussion sections, but there was clearly an increase in genuine error rate that accompanied the expanded pool of ten ragams as well.

A confusion matrix, demonstrating the frequency at which samples of each ragam were classified as each of the ten ragams under consideration, including the true ragam, is found below. The fact that some ragams, such as mohanam, have greater numbers of correct classifications along the diagonal, and elsewhere, is reflective of some ragams being far more common than others, in my dataset and in the real world. As mentioned before, the intuition here was that the CNN model would pick up on and learn from this class imbalance.

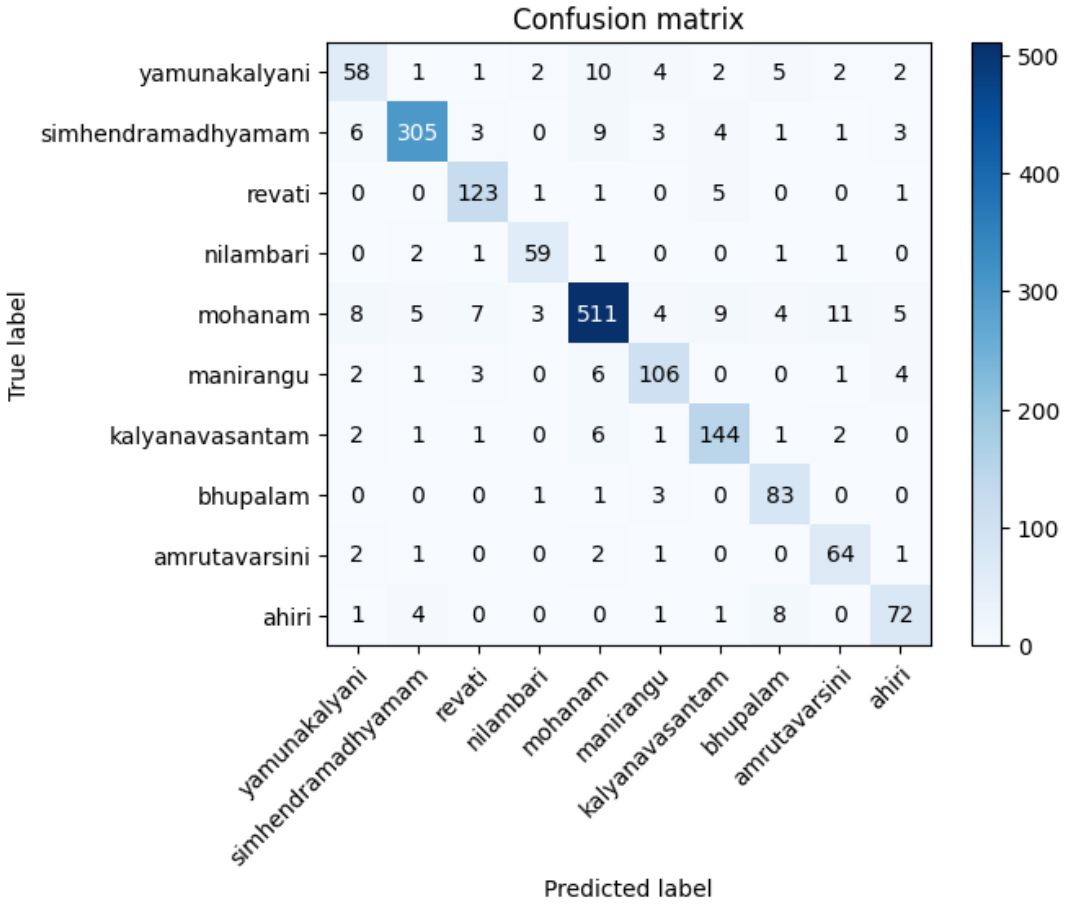


Figure 6.17: Confusion Matrix for 10-ragam CNN Model

Chapter 7

Biological Basis for Methods (Aside)

7.1 Overview

As my applied math concentration application area is biology, I wanted to briefly outline some of the biological inspirations for the methods applied in this project, drawing on the concepts I have encountered in my life sciences and neuroscience coursework at Harvard with Dr. Kreiman. I especially want to illustrate how biological vision and hearing mechanisms inform the models and feature representations I use in this project.

This section of the report is not directly related to my results in this project, which appear in the previous chapter. This chapter can be thought of as an informational aside. [1]

7.2 Biological Basis for Convolutional Neural Networks

The inception of Convolutional Neural Networks (CNNs) is deeply rooted in the understanding of the visual cortex's structure and function in animal brains. The seminal work by Hubel and Wiesel in the mid-20th century revealed that neurons in the visual cortex in cats are sensitive to specific regions of the visual field, known as receptive fields. [11] These neurons act as localized filters, responding to particular spatial patterns like edges or bars

[1] Note that I used ChatGPT to help me write and edit some of the language in this chapter.

of light, laying the groundwork for hierarchical and spatial processing of visual information. As I describe below, CNNs take their inspiration for their convolution operations and filters from these receptive fields.

7.2.1 Hierarchical Structure of the Visual Cortex

The visual cortex processes visual stimuli through a hierarchical organization, where simple cells in the primary visual cortex (V1) respond to elementary features like edges and orientations. In contrast, complex cells in higher cortical areas respond to more intricate patterns formed by combinations of lower-level features. This hierarchical processing facilitates the decomposition of visual stimuli into progressively abstract feature representations. [11]

Mathematical Model of Receptive Fields

A Gabor filter provides a simplified mathematical model to describe the receptive field of a simple cell in the visual cortex. The Gabor filter is defined as:

$$G(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right) \tag{7.1}$$

with $x' = x\cos\theta + y\sin\theta$ and $y' = -x\sin\theta + y\cos\theta$, where λ is the wavelength, θ is the orientation, ψ is the phase offset, σ is the Gaussian envelope's standard deviation, and γ is the spatial aspect ratio. [4]

These parameters correspond to various characteristics of the receptive field, such as the preferred orientation (θ), spatial frequency (λ), and spatial extent (σ). The exponential term represents the spatial envelope of the receptive field, encoding information about the spatial spread and sensitivity of the neuron's response. The cosine function captures the sinusoidal modulation of the neuron's response to the stimulus, reflecting its sensitivity to the phase and orientation of visual features. [11]

7.2.2 Convolution in CNNs: A Biological Analogy

CNNs employ convolution operations with learned filters, akin to the activation of receptive fields in the visual cortex described above, where each filter detects a specific feature in its input. This mechanism is reflective of how neurons in the visual cortex respond to particular patterns within their receptive fields.

Feature Hierarchy in CNNs

CNNs learn a hierarchy of features from input images, like mel-spectrograms, analogous to the visual cortex's hierarchical processing. Early layers in a CNN capture basic features such as edges, while deeper layers synthesize these features into more complex and abstract representations.

7.2.3 Pooling: Mimicking Complex Cells

Pooling layers in CNNs, especially max pooling, abstract the function of complex cells in the visual cortex, which are able to maintain their response to a stimulus even if the stimulus is slightly shifted in position within their receptive field. In other words, the neurons continue to fire in response to the stimulus regardless of its precise location within their spatial domain. Similarly in CNNs, max pooling, by down-sampling feature maps, introduces robustness to small changes and reduces spatial resolution, paralleling the behavior of complex cells. [4]

Integration into CNN Architecture

The architecture of CNNs integrates these biologically inspired principles—localized receptive fields shared weights for feature detection across different locations, and hierarchical feature representation—into a trainable model framework. This design allows CNNs to learn complex visual representations, making them effective for a wide range of image processing

tasks, akin to the capabilities of biological vision systems.

7.3 Biological Basis for Mel Scale

As I described in the data preprocessing chapter, the mel scale is a perceptual scale of pitches that mimics the human ear's response more closely than linearly-spaced frequency bands. Once again, the formula for converting frequency f to the mel scale, denoted as $\text{Mel}(f)$, is given by:

$$\text{Mel}(f) = 2595 \log_{10} \left(1 + \frac{f}{700}\right)$$

Biologically, the Mel scale is based on research into the frequency discrimination capabilities of the human auditory system. It is inspired by the behavior of the cochlea, the spiral-shaped organ in the inner ear responsible for converting sound vibrations into neural signals. The cochlea contains hair cells that are sensitive to different frequencies along its length. However, the distribution of these frequencies is not linear but rather follows a pattern similar to the logarithmic mel scale, where equal intervals on the scale correspond to perceptually equal differences in pitch.

This non-linear, logarithmic compression reflects the non-linear nature of human auditory perception, where the perceived difference between two frequencies becomes smaller as the frequency increases. This allows us to accurately mimic how humans perceive pitch, making mel-scaled spectrograms especially useful as training data for ragam identification models, where I am essentially trying to replicate an audio classification task that human ears and brains are demonstrably effective, with computational methods. Additionally, the logarithmic compression inherent in the mel scale enhances the discriminative power of the spectrograms, making them ideal inputs for machine learning models.

Chapter 8

Live Demonstration of my Model in Action

8.1 Overview

Having developed the biological basis for mel-spectrograms and Convolutional Neural Networks in the previous section, I now provide a link to a video demonstration of my 2-D Convolutional Neural Network model working to identify the ragams of songs in real-world Carnatic music audio data using mel-spectrogram representations. I demonstrate the model's efficacy not only on concert audio data but also on a tune I hum into my phone in real time. In both cases, the model correctly identifies the ragam of the song or melody being played, a promising result.

Hyperlink: Live Model Demonstration

URL in Plain Text: https://youtu.be/1fyhf3C-mfQ?si=6ubEb9uUtgfsrM7H

Chapter 9

Discussion

9.1 Overview

In this section, I will first review the significance and implications of my results, which are described in chapter 6 of this report, before describing some of the potential extensions of my work to future research. [1] [2]

Summary of Testing Accuracy Results:

Table 9.1: Testing Accuracy Scores Across Models

	ANN	**LSTM**	**BERT**	**CNN**
Small Pool	97%	98%	88%	96%
Medium Pool	94%	90%	20%	92%
Big Pool	86%	N/A	N/A	N/A

[1] Note that I used ChatGPT to help me write and edit some of the language in this chapter.

[2] Some of the language in this chapter comes from my Neuro 240 final report, which I produced for Dr. Kreiman in 2023, and which is cited in my bibliography below.

9.2 Models Trained on Numerical Feature Data

ANN Models

Obtaining 93.6% and 86% testing accuracy on a pool of ten and fifteen ragams respectively is an extremely encouraging result for the ANN models. Not only is the fifteen ragam model more expansive than any models described in previously published research, but the ten ragam model is also as accurate on testing data as many models of lesser scope from previous studies, such as the MIT World Peace University study, which demonstrated similar accuracy on a pool of seven ragams. [10] This suggests that the numerical feature vectors I constructed, as well as the hyperparameter tuning and model engineering I conducted, yielded a more robust and powerful ANN model than has previously been applied to this task. Furthermore, my analysis of the relative importance of the features extracted for the vector representation allowed me to understand what the model considers most when making its ragam prediction, and to think about the "computational essence" of ragams. This allowed me to see how important Chroma and CENS features were to ragam detection, as well as RMSE. I was also able to learn that the Zero-crossing rate is typically much less relevant to the ragam prediction. This is a novel result, as previous applications of neural networks to ragam identification have typically involved black-box models where the underlying weights are hidden.

LSTM Models

The 98% and 90% testing accuracies recorded for the two-ragam and the ten-ragam models respectively were also encouraging results for similar reasons. The combination of my numerical feature representations and parameter tuning on the LSTM models appears to have yielded a more accurate LSTM model to classify within small pools of ragams than any in the literature. While there is a clear degradation in performance as we scale to ten ragams, I was not able to find a study demonstrating LSTM model efficacy on a pool of

any more than seven ragams, as I was conducting my research. Thus, these results on the LSTM models and ANN models constitute meaningful strides toward bigger, more accurate models capable of classifying the most commonly occurring Carnatic ragams.

Transformer Models

While the results for the transformer models were more limited, especially when compared to the other architectures, the 88% testing accuracy obtained from the two-ragam model was a promising result because transformer models have not previously been successfully applied to the task of identifying ragams. Moreover, the application of a pre-trained BERT model to numerical feature data representing music audio is certainly novel, and it is exciting to demonstrate that the model is actually learning from the numerical feature data.

9.3 Models Trained on Image Data

The testing accuracy of over 90% of the ten-ragam CNN model and performance on unseen, informally recorded music is an encouraging preliminary result for a few reasons. Firstly, it expands upon the results achieved in the Jagtap study by demonstrating an effective accuracy on real-world concert data, which is not of standardized quality and has not been curated for ML classification, in the same way that the dataset in that study was. There, the audio was specially selected to have been recorded under studio conditions and represented 10 highly dissimilar ragams that human listeners could easily distinguish. By demonstrating model efficacy, even if slightly lower than the 96% testing accuracy achieved in the Jagtap study, my ten-ragam CNN model makes advancements towards the goal of having a Shazam-like application for instant and reliable ragam recognition under a range of real-world circumstances including recordings of informal home practices, live concerts, and audio playing from speakers. This could make a big difference to everyone from first-time listeners to seasoned Carnatic music connoisseurs.

One of the additional advantages of using massive amounts of real-world data is that my model is implicitly trained to incorporate biases of loudness, scale, and pitch that may exist under certain recording conditions or in certain ragams. This is because with over 3000 tracks in the Sangeethapriya database alone, and over 70,000 overall in my dataset, differences in recording conditions balance out across ragam categories. This makes it unlikely that the model is classifying ragams based on such confounding characteristics that could be associated with certain ragams or recording conditions. Secondly, previous studies have mainly demonstrated efficacy on common ragams whereas my ten ragam CNN model was able to successfully classify rare ragams such as *ahiri* and *revati* which may have not been included in previous ragam classification projects. [10] Thirdly, my CNN model was able to make progress towards addressing the issue of declining classification accuracy as the number of ragams under consideration increased, showing that implementing larger 2D CNN architectures, early stopping, and dynamically adjusting learning rates are all effective levers to pull to ensure persistent categorical accuracy even when the size of the dataset and the number of possible ragams increases.

9.4 Immediate Implications of my Models' Results

Human accuracy in ragam identification varies with individual expertise, but most Carnatic music educators and appreciators with more than a few years of training can reliably classify songs into one of 150 of the more common ragams, consistently and independently. If most concerts involve the presentation of 10 pieces, a seasoned listener might go to four or five concerts before encountering a song whose ragam they struggle to identify. While my model likely underperforms relative to such experts, it could still benefit early students for whom identification of any song's ragam is likely a daunting challenge, as mentioned in my abstract. Learning to identify ragams is an iterative process of attempting guesses and internalizing mistakes, and this tool could serve students by instantly providing them

with the right answer after they make an identification attempt, abbreviating the feedback cycle, and improving human learning speeds. As I demonstrated in my video walkthrough in Chapter 8, my model is ready to serve this use case now, which is a very exciting result.

9.5 Possible Extensions and Next Steps

In the longer term, however, there are a few extensions I might consider for future study of this problem, starting from data collection. Firstly, depending on who is labeling a track before uploading it to Sangeethapriya.com, certain tracks may be incorrectly labeled, which could skew models like mine, raising the issue of multi-annotator reliability for ground truth labels. Separate Carnatic music experts would be expected to nearly always agree on a classification for a song's ragam given a 10-second recording, and further studies could explore the effects of crowdsourcing and consensus annotations for ground truth labels on the ultimate performance of the model, as opposed to the combination of string parsing and individual hand-annotations I used to establish ground truth labels in this project. Also on the point of data and labeling, it would be exciting to repeat these experiments using the full scope of Harvard's Rubin collection, as I alluded to in the data sources chapter of this thesis. Not only would this collection introduce thousands of tracks to the dataset, but also the fact that they are not available anywhere else would guarantee they are new to the model, preventing overfitting and improving robustness.

More generally, repeating these experiments with significantly more data is likely to yield more robust ANN models than can classify over greater pools of ragams without experiencing degradation in testing accuracy. It is not inconceivable that had I been able to label my entire set of 72,000 songs, a slightly larger ANN model than the one I designed here might be able to classify up to 50 of the most commonly occurring ragams, instead of just the top 15.

Additionally, now that we have some sense from the feature importance maps for what

the model considers most when making a prediction, it would also be interesting to design a new vector with only the most salient features, such as Chroma and CENS coefficients. It could be possible that this simpler representation would enable future ANN models to train more efficiently and to learn patterns in the data more rapidly.

For the LSTM and Transformer models, it would be very interesting to train them on sequential data in future studies that are perhaps less storage intensive than a mel-spectrogram. My intuition is still that these models are best equipped to learn patterns across time-series data, making them ideal for learning the harmonic patterns associated with ragams in Carnatic music.

For the CNN models, it might be interesting to explore the relationships between the constructed mel-spectrogram representations for similar ragams. This would allow for features such as retrieval of a list of similar ragams when classifying a track, and accompanying explanations of the often-subtle differences between the correct ragam and such adjacent ones. Lastly, explorations of changing the recognition window time from 10 seconds could be fruitful, as many human listeners can reliably classify in 2-5 seconds, especially when using a ragam's "tells", or characteristic melodic phrases within certain ragams that I have mentioned above. Teaching a model to internalize these tells would require more data and modified architectures, but could lead to vastly reduced prediction times in general for ragams that have these "tells," when one such is perceived.

Bibliography

[1] James rubin collection of indian classical music. AWM Spec Coll 94, Eda Kuhn Loeb Music Library.

[2] Indian classical music for all. `www.sangeethapriya.org/`, 2008.

[3] List of janya ragas. `en.wikipedia.org/wiki/List_of_Janya_ragas`, 2024.

[4] Neural network (machine learning). `en.wikipedia.org/wiki/Neural_network_(machine_learning)`, 2024.

[5] Transformer (deep learning architecture). `en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)`, 2024.

[6] Ankit Anand. Raga identification using convolutional neural network. In *2019 Second International Conference on Advanced Computational and Communication Paradigms (ICACCP)*, pages 1–6, 2019.

[7] K. Choi, G. Fazekas, and M. Sandler. Automatic tagging using deep convolutional neural networks. *arXiv preprint arXiv:1606.00298*, 2016.

[8] YMG Costa, LS Oliveira, JCN Silla, and CN Jr Silla. An evaluation of convolutional neural networks for music classification using spectrograms. *Applied Soft Computing*, 52:28–38, 2017.

[9] Editor. Swaras in carnatic classical music. `musikclass.com/content/swaras-in-carnatic-classical-music`, 2023.

[10] Devayani Hebbar and Vandana Jagtap. A comparison of audio preprocessing techniques and deep learning algorithms for raga recognition. MIT World Peace University.

[11] DH Hubel and TN Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J Physiol*, 160:106–154, 1962.

[12] Jingxian Li, Lixin Han, Xiaoshuang Li, Jun Zhu, Baohua Yuan, and Zhinan Gou. An evaluation of deep neural network models for music classification using spectrograms. *Springer Multimedia Tools and Applications*, 2021.

[13] Meinard Müller. *Fundamentals of Music Processing: Audio, Analysis, Algorithms, Applications*. Springer, 2016.

[14] Hari Narayanan. Classifying ragams in carnatic music with convolutional neural networks. Neuro 240, Kreiman Lab, Harvard University, 2023.

[15] P.K. Srimani and Y.G. Parimala. A comparative study of carnatic and hindustani raga systems by neural network approach. *International Journal of Neural Networks*, 2(1):35–38, 2012.