Chord Analyzer

Submitted in partial fulfillment of the requirements
for the degree of

B.E. Information Technology

By

Tanaya Khanvilkar	**211055**
Vibha Khetan	**211056**
Chris Crasto	**211015**
Sven Coelho	**211012**

Supervisor
Dr. Mrinmoyee Mukherjee
Assistant Professor

Department of Information Technology
St. Francis Institute of Technology
(Engineering College)

An Autonomous Institute , Affiliated to University of Mumbai

2024-202

SFIT LIBRARY 2024 - 2025

CERTIFICATE

This is to certify that the project entitled **"Chord Analyzer"** is a bonafide work of **Tanaya Khanvilkar (211055), Vibha Khetan (211056), Chris Crasto (211015) & Sven Coelho (211012)** submitted to the University of Mumbai in partial fulfillment of the requirement for the award of the degree of Bachelor of Engineering in Information Technology.

Dr. Mrinmoyee Mukherjee

(Project Guide)

Dr. Prachi Raut

(Head Of Department)

Dr. Sincy George

(Principal)

Project Approval for B.E.

This project report entitled *"Chord Analyzer"* by *Tanaya Khanvilkar (211055), Vibha Khetan (211056), Chris Crasto (211015)& Sven Coelho (211012)* is approved for the degree of *Bachelors of Engineering in Information Technology*.

1. H. Kurtheja

2. K.S. 22/4/25

Date: 22/04/2025

Place: Mumbai

SFIT

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, We have adequately cited and referenced the original sources. We also declare that We have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Tanaya Khanvilkar (211055)

Vibha Khetan (211056)

Chris Crasto (211015)

Sven Coelho (211012)

Date: 22/04/2025

SFIT

ABSTRACT

The guitar stands as one of the most beloved and widely played instruments worldwide. Its versatility, accessibility, and popularity make it a go-to choice for many aspiring musicians. As a result, there is an abundance of materials, both online and offline, aimed at helping individuals learn this instrument. However, despite the extensive resources, learning the guitar without the guidance of a professional can be both challenging and frustrating. Current systems available for learning and mastering the guitar often fall short in providing interactive and intuitive support, particularly in helping guitarists accurately identify chords.

Traditional music classes offer expert guidance but are often costly and inflexible in scheduling. Alternatively, online courses and mobile applications offer flexibility and affordability but gener- ally lack personalized feedback, which is crucial for real-time correction and progression. Thus, there is a growing demand for a comprehensive system that addresses these gaps and enhances the guitar-learning experience.

This is where the Chord Analyzer application comes into play. Our goal is to merge the most effective elements from various learning methods into a single, user-friendly platform that minimizes the drawbacks commonly associated with them. The Chord Analyzer application aims to cater to both novice and experienced guitarists by providing tools to identify chords, analyze finger placements, and track individual progress. By doing so, users can develop skills faster and more efficiently, without the need for a live instructor or an impersonal online course.

The Chord Analyzer is designed to be a one-stop solution for aspiring musicians eager to make significant strides in mastering the guitar. Equipped with advanced features powered by deep learning, computer vision (OpenCV), and a robust back-end (Firebase), this application will deliver real-time feedback and offer personalized insights, empowering users to improve consistently and effectively.

The application is built on the principles of deep learning and image recognition, utilizing Convolutional Neural Networks (CNNs) and the Keras framework to identify and assess guitar chords based on finger placements on the fretboard. Through this technology, we aim to provide an experience that is not only informative but also interactive and responsive to each user's learning journey.

Ultimately, Chord Analyzer seeks to be more than just a learning tool; it is envisioned as a musical companion that allows musicians to express themselves more freely and understand the language of music in a deeper, more intuitive way. As Victor Hugo aptly said, "When words fail, music speaks." Our application embodies this sentiment by breaking down the barriers to learning and making music accessible to all.

SFIT LIBRARY 2024 - 2025

Contents

1 Introduction **1**

1.1 Introduction 1
1.2 Problem formulation 1
1.3 Description 2
1.4 Motivation 2
1.5 Proposed solution 2
1.6 Scope of the project 2

2 Literature review **4**

2.1 Existing work 4
2.1.1 Literature review related to exiting system/methodology 4
2.1.2 Literature review related to existing algorithms 5
2.1.3 Literature review related to tools/technology/framework 6
2.2 Gap identified 6

3 System analysis **8**

3.1 Functional requirements 8
3.2 Non-functional requirements 9
3.3 Specific requirements 9
3.3.1 **Hardware** 9
3.3.2 **Software :** 9
3.4 Use case diagram 10

4 Model Analysis **11**

4.1 Data modeling 11
4.2 Activity diagrams 12
4.3 Functional modeling - DFD level 0 13
4.4 Functional modeling - DFD level 1 14
4.5 Functional modeling - DFD level 2 15
4.6 Timeline chart 16

5 Implementation **19**

5.1 Data collection 19

SIT LIBRARY 2024 - 2025

5.2 Algorithms and methodology....20
5.3 Chord Detection model....21
5.4 Training of the model....22
5.5 Algorithm....22
5.6 Working of the project....23
5.6.1 Dataset Creation....23
5.6.2 Training Dataset....25
5.6.3 Testing....28
5.6.4 Chord Detection....31
5.7 Testing and Results....32
5.7.1 Confusion Matrix....33
5.8 Accuracy....34
5.8.1 Model Accuracy....35
5.8.2 Model Loss....36

6 Design 37
6.1 Architecture design....37
6.2 User Interface design....38

7 Conclusion and future scope 45
7.1 Conclusion....45
7.2 Future scope....45

8 Literature cited 46

SFIT LIBRARY 2024 - 2025

List of Figures

3.1	Use case diagram	10
4.1	ER diagram	11
4.2	Activity diagram	13
4.3	Dataflow level 0	13
4.4	Dataflow level 1	14
4.5	Dataflow level 2	15
4.6	Timeline chart July 2024 - November 2024	16
4.7	Timeline chart January 2025 - March 2025	16
4.8	Timeline chart March 2025 - April 2025	16
5.1	Chord C Chord D on grayscale	20
5.2	Chord Em Chord G on grayscale	20
5.3	Representation of Keras model	21
5.4	Accuracy % of all chords	34
5.5	Graphs of Model Accuracy and Loss	35
6.1	Architecture of the system	37
6.2	Signup page	38
6.3	Signin page	39
6.4	Getstarted page	39
6.5	Practice chords	40
6.6	Music theory vizualization page	40
6.7	Chord C page	41
6.8	Chord D page	41
6.9	Chord Em page	42
6.10	Chord G page	42
6.11	Contact us page	43
6.12	User profile page	43

SITLIBRARY 2024 - 2025

List of Tables

2.1	Comparison of methodology	4
2.2	Comparison of algorithms	5
2.3	Comparison of tools/technology/framework	6
3.1	Use case diagram description	10
4.1	Week-wise Project Timeline	17
6.1	Overview of User Interface Pages	44

SFIT LIBRARY 2024 - 2025

Chapter 1

Introduction

Ever watched a video on YouTube and wondered what chords the artist is actually playing? Chord Analyzer is an application that will make learning and mastering the guitar easy for guitarists of all levels by reviewing video footage provided by applications such as YouTube. The system will generate live music theory based information by analyzing the finger positions of the guitarists. The system will also analyze the accuracy of the chords being played and give a score based feedback to the user in order to improve their performance

1.1 Introduction

The main function of Chord Analyzer the application is the finger position recognition system that is capable of analyzing finger patterns and converting the information to chords in real time. The Chord Analyzer application will also have a chord sheet generation system where a video footage of a person playing the instrument can be provided and a music sheet for the entire video will be generated by the system. The accuracy of the guitarist will also be measured by the system and a score based feedback will be provided to the user.

1.2 Problem formulation

- Many guitar players use video content such as YouTube to practice. If the content contains noise or background sounds then the player must watch the videos repeatedly, which is very troublesome
- Music lessons are very expensive By not spending money on a trainer and learning through our app one can save a lot of money.
- While there exist apps in the market that offer music training, there isn't one which can offer live feedback and identify your mistakes without having a real person as your trainer.
- The present applications have virtual instruments to practice and an actual guitar cannot be used.

SFIT LIBRARY 2024 - 2025

- There is a need for an app which makes free music training accessible to everyone and which helps give live feedback at any time or place

1.3 Description

The main function of the Chord Analyzer application is the finger position recognition system that is capable of analyzing finger patterns and converting the information to chords in real time. The Chord Analyzer application will also have a chord sheet generation system where a video footage of a person playing the instrument can be provided, and a music sheet for the entire video will be generated by the system. The accuracy of the guitarist will also be measured by the system, and a score-based feedback will be provided to the user.

1.4 Motivation

There's a large number of people who want to learn the guitar as well as trained musicians who want to perfect the art of playing an instrument. These people tend to lose their drive and motivation when they find out that learning the guitar can be a challenging and tedious process. Unless the musician has an external guide or attends classes, it's very easy for the individual to make mistakes that go unnoticed. There is no score-based system that can measure the accuracy of the guitarist and provide feedback about their performance. Finally, the end user would love to have an application like Chord Analyzer that would aid their learning experience while they used an actual instrument rather than a virtual one.

1.5 Proposed solution

The Chord Analyzer application is a user-friendly music assistant for guitarists who want to track their accuracy as well as interpret what other guitarists are playing. The primary feature of Chord Analyzer is a chord-based finger pattern analyzer that will help the user identify the chords being played. This feature will allow users to upload videos downloaded from various video streaming websites such as YouTube or guitar-playing videos of their friends and generate a chord sheet based on what's being played in the video. The user will then be able to store the generated chord sheet of various songs, covers, and videos and name them according to their preference. The Chord Analyzer application will also be able to monitor and analyze live video footage provided by the user. A score-based system will be used to track the progress that the user is making in their journey to master the instrument and become a talented guitarist.

1.6 Scope of the project

Chord Analyzer will be the go-to application for aspiring musicians and cover artists who want to perfect the art of playing the guitar. With our live chord pattern analyzing feature, we could bring a new revolution to the music and instrument learning industry. Every user will have the ability to use an actual instrument instead of an inbuilt virtual instrument. Since our application is capable of

SIT LIBRARY 2024 - 2025

measuring accuracy, certain exams such as the Trinity music exam can be taken via our application without the need for a music teacher. The user will be able to use the Chord Analyzer application to store their favorite songs along with their chord progressions and notation tabs.

SFIT LIBRARY 2024 - 2025

Chapter 2

Literature review

2.1 Existing work

2.1.1 Literature review related to exiting system/methodology

Table 2.1: Comparison of methodology

Sr. No.	Title of Paper	Review	Analysis/Limitations
[1]	Note Frequency Recognition and Finger Motion Cap- ture of a Guitarist: A Survey and Challenges Ahead	Survey approaches for beginner guitarists' challenges and propose a hybrid solution. Video processing, CNN for finger motion, DFT for pitch accuracy, Kinect sensors, gloves, neural networks	Limited finger motion detection, lighting issues, limited real-time feedback, confusion in detection
[2]	CNN Transfer Learning for Visual Guitar Chord Classification	Create a classifier for visually recognizing guitar chords using CNN models. Data collection, CNN (ResNet18, GoogLeNet) with transfer learning, data augmentation, ADAM optimizer	Low generalizability due to dataset limitations, lack of real-time application, insufficient robustness.
[3]	Finger-String Interaction Analysis in Guitar Playing With Optical Motion Capture.	Analyze the interaction between fingers and strings during gui- tar plucking using optical motion capture. Motion capture with high-speed cameras, hand models for occlusions, audio analysis, NARX machine learning	Occlusion of markers, low detection rates for fingertips, lack of generalization for plucking force estimation

SFIT LIBRARY 2024 - 2025

2.1.2 Literature review related to existing algorithms

Table 2.2: Comparison of algorithms

Sr. No.	Title of Paper	Review	Analysis/Limitations
[4]	A 3D Guitar Fingering Assessing System Based on CNN-Hand Pose Estimation and SVR-Assessment.	Evaluate guitar fingering using CNN-based hand pose estimation and SVR for fingering accuracy. CNN for hand pose estimation, SVR for assessment, DCT for fea- ture extraction, FCN for segmen- tation	Tools like color markers and fixed cameras are inconvenient, high sensitivity to lighting and back- ground.
[5]	Left and right-hand gui- tar playing techniques detection.	Develop algorithms to detect guitar techniques (e.g., hammer-ons, slides) using audio analysis. Audio analysis, onset detection, pitch profile analysis, STFT, and hexa- phonic piezo pickup for signals.	Limited to specific playing techniques, manual segmentation of recordings, lacks real-time integration.
[6]	Visual Methods for the Retrieval of Guitarist Fingering.	Develop a real-time visual method for detecting and recognizing gui- tarist fingering using video anal- ysis. Circular and linear Hough transform for fingertip and string detection, Sobel filter for gradient analysis.	Finger occlusion issues, challenges with non-real-time sound analysis, dependence on camera resolution.

SFIT LIBRARY 2024 - 2025

2.1.3 Literature review related to tools/technology/framework

Table 2.3: Comparison of tools/technology/framework

Sr. No.	Title of Paper	Review	Analysis/Limitations
[7]	Estimating the plucking point on a guitar string	Estimate plucking point on a guitar string using acoustic signals and detect left-hand fingering points.Fourier analysis for pitch detection, harmonic peak detec- tion, error minimization for pluck- ing point estimation	Inaccuracies in plucking point estimation for fretted strings, real-world playing variations affect re- sults.
[8]	Computer vision method for guitarist fingering retrieval	Develop a low-cost, real-time method for detecting guitarist fingerings using computer vision. Circular Hough Transform for fin- gertip detection, Canny edge de- tection, Sobel filter for string/fret detection	Finger occlusion problems, limited detection range to first five frets, challenges with finger intersections.
[9]	Finger Tracking Methods Using EyesWeb.	Compare algorithms for tracking finger positions in a 2D environment, focusing on HCI and gestural control. Circular Hough Trans- form, projection signatures, geo- metric properties, and color mark- ers for finger tracking.	High computational costs for real- time use, low precision for com- plex gestures, reliance on prepro- cessing.
[10]	A Segmentation-based Prototype to Compute String In- struments Fingering	Develop a computationally efficient algorithm for computing fin- gerings for string instruments with cognitive elements.	Fingering process complex- ity, lack of focus on musi- cal intentions, difficulty in hand repositioning along the neck.preprocessing.

SFIT LIBRARY 2024 - 2025

2.2 Gap identified

1. Limited support for videos with varying audio qualities or complex arrangements: The system may struggle to accurately process videos with inconsistent audio quality, such as those recorded in noisy environments or with varying background interference. This can make it difficult for the software to differentiate between music and background noise, leading to in-

correct or incomplete feedback. Complex arrangements involving multiple instruments or

SFIT LIBRARY 2024 - 2025

vocal harmonies may further confuse the system, making it harder to isolate individual elements and provide targeted analysis.

2. Potential latency issues affecting the accuracy of real-time feedback: Real-time feedback is essential for practice applications, especially when users rely on immediate responses to correct their mistakes as they play or sing. However, latency—delays in processing or delivering feedback—can interfere with this. Even slight delays can create a disjointed experience, mak- ing it challenging for users to align with the intended rhythm or correct their mistakes in the moment. Reducing latency while processing high-quality audio and video in real-time remains a technical challenge.

3. Dependence on the quality of the user's webcam and audio setup, leading to inconsistent results: The application's accuracy is highly dependent on the user's equipment. Lower-quality webcams or microphones may not capture nuances in sound or visuals accurately, affecting the feedback provided. For instance, if a webcam fails to capture finger placements or if a microphone picks up distorted audio, the system may produce inaccurate assessments. This reliance on equipment quality could lead to variable user experiences and less reliable feedback for users without high-end gear.

4. Lack of song variety, with a limited library of songs available for practice: Many users seek to practice with familiar songs or genres that resonate with them, but a limited song library restricts their options. Without a diverse selection of songs, users may find it difficult to stay engaged or progress, especially if they're unable to practice their favorite tunes. Expanding the library to include a variety of genres, skill levels, and styles could address this issue and increase user engagement.

5. Insufficient guidance for beginners on how to improve based on feedback received: While feedback is valuable, beginners often need structured guidance on how to interpret it and improve their skills. If feedback is too technical or lacks step-by-step improvement tips, beginners may feel lost or frustrated. Clear, beginner-friendly instructions that break down common issues and suggest focused exercises or techniques would make the feedback more actionable, helping users grow and stay motivated in their learning journey.

SFIT LIBRARY 2024-2025

Chapter 3

System analysis

3.1 Functional requirements

- Chord Sheet Generator: The webapp Chord Analyzer has a number of features. One of them is the chord sheet generator. Usually musicians have issues in determining the chords played in a particular video they come across in social media. This problem has been solved by our website. A user can upload an mp4 video and get all the chords played in that video. This will be done using a deep learning algorithm. Apart from this if the video is in the form of a YouTube link they can just put in the link of the video and download the video from our site itself. This video can then be uploaded to get the corresponding chord sheet. If a user wants to get the chord sheet for their own song they can play it in front of the webcam in realtime and get the chord sheets. These sheets can then be stored in the database and can be accessed by the users through the profile section.
- Practice session: Once the users have a chord sheet for a particular song, they can practise it through our application itself. In this the user has to play the guitar in front of the camera and the system will analyse whether the chords played are correct or not. The accuracy rate can be determined by the model. They can choose any chord sheet saved in the database to practise or generate a new one. Since practising is very important for a musician this feature will be very helpful to many rising musicians who can analyse their skills through their instrument itself and not a simulator.
- Test Series: Testing your skills is equally important as practising. Our website has an inbuilt test series where users will get to play one chord at a time and they have to do it in front of the camera. The system will then analyse the accuracy of the particular chord being played. It would also have a timer to keep the participants on alert.
- Music Theory: The application will have a separate section for all the music related theory that will help the users in learning the basics of music if they are just beginning to learn music. It will include brief information about chords, scales, pitch etc

SFIT LIBRARY 2024 - 2025

3.2 Non-functional requirements

- Performance Requirements: The performance of the website would depend on the loading time of the pages. It would also depend on the time required by the deep learning model to make predictions and the time required for uploading the video . Less is the time more is the efficiency. The performance would also depend on the internet connection.
- Design Constraints: The major design constraint in our project is designing the website in such a way that every section is easily accessible to the user whenever they want to navigate. Also designing the practice section would be difficult since it would include live recording of the person along with the chord sheets.
- System attributes: Reliability: In order to ensure reliability, this system is being designed using a software that is established stable and easy to use. Availability: This system is designed to run 24/7 and be readily available to the user. Security: Access to the software is given only to valid operators. Users that are registered in the database can access major parts of the system, whereas un-registered users stay in View Only mode on the system.

3.3 Specific requirements

3.3.1 Hardware :

- Computer or Laptop
- Minimum 4GB required on the computer
- Internet connection

3.3.2 Software :

- Operating System: Windows
- Backend: Flask
- Machine learning model: Open CV, TensorFlow(Keras)
- Frontend: HTML [3], CSS [4], Javascript
- Database: Firebase
- Editor: Visual Studio Code

SFIT LIBRARY 2024 - 2025

Table 3.1: Use case diagram description

Actors	Description
User	-View chord recognition to identify chords played. - Access practice sections to improve guitar skills. - Take chord tests to assess chord knowledge - Access learning resources for additional guidance. - Manage account settings (e.g., update profile or preferences)..
Administrator	Manage application content, including adding, editing, or removing resources, practice sections, etc.

3.4 Use case diagram

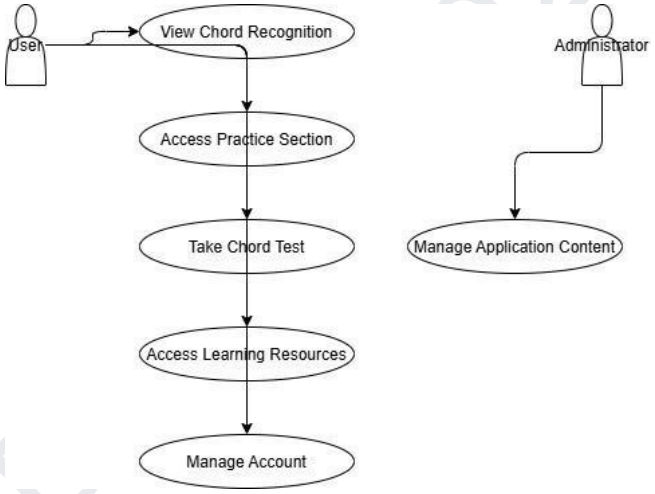


Figure 3.1: Use case diagram

The use case diagram provides a high-level overview of how different types of users interact with the Chord Analyzer system. It includes two main actors: the User and the Administrator. The user can perform several functions such as generating chord sheets, practicing chords, taking tests to assess chord knowledge, and accessing music theory resources. They can also manage their account settings through the profile section. On the other hand, the administrator manages application content, including adding or updating resources and maintaining practice and test modules. This diagram clearly illustrates the functional boundaries and the roles of each user type in the system.

Chapter 4

Model Analysis

4.1 Data modeling

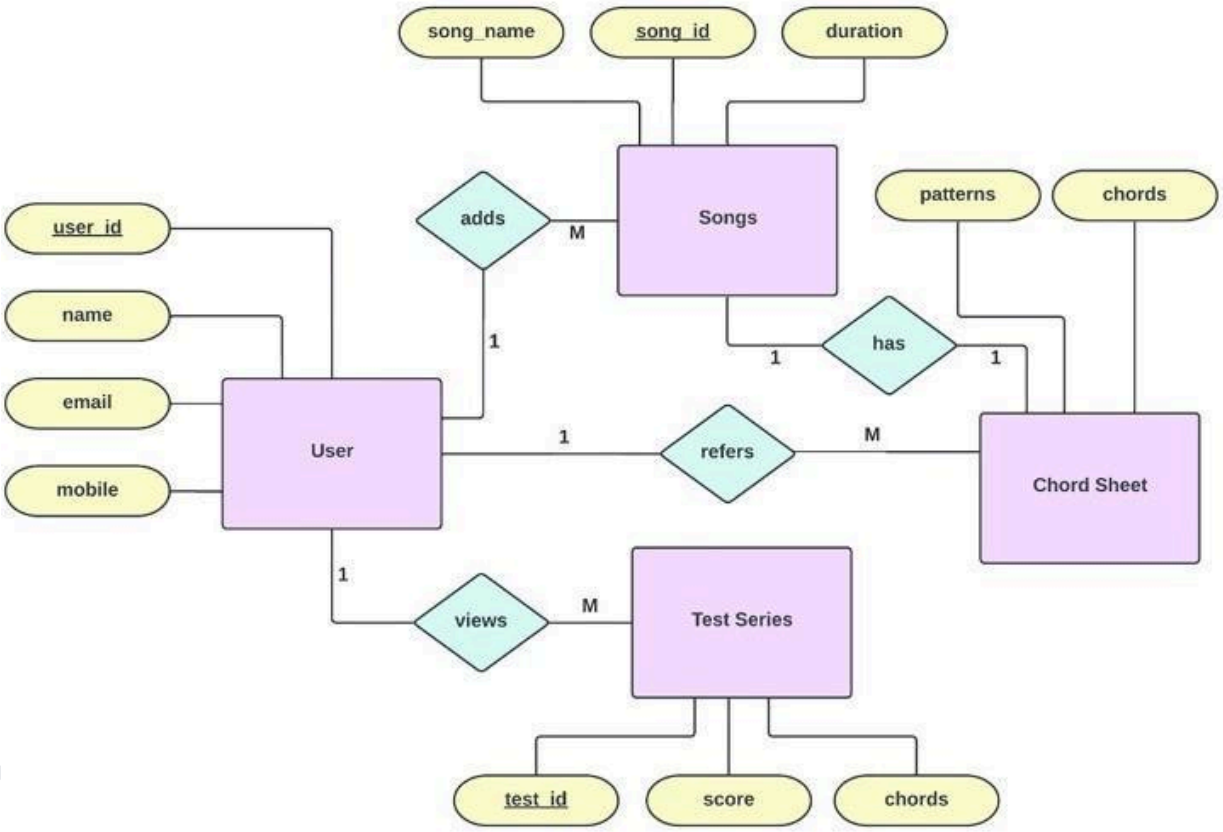


Figure 4.1: ER diagram

1. The User entity represents the individuals who interact with this music learning or sharing platform. Each user is uniquely identified by a userid. In addition to this unique identifier, the system stores the user's name, email address for communication and login purposes, and

their mobile phone number. A single user can contribute to the platform by adding multiple songs, as indicated by the one-to-many relationship with the Songs entity. Furthermore, each user can access and view various learning assessments organized as Test Series, establishing another one-to-many relationship. Finally, a user can refer to and study multiple Chord Sheets associated with different songs, again demonstrating a one-to-many relationship.

2. The Songs entity captures the musical content available within the platform. Each song is uniquely identified by a songid and has a songname and a specific duration. While many songs can be added to the system by different users, each song is associated with exactly one Chord Sheet, representing a one-to-one relationship. This suggests that for every song in the system, there is a primary chord arrangement available.

3. The Chord Sheet entity details the musical notation for playing songs. Each chord sheet contains the actual chords for a song and may also include patterns or guides for strumming or picking. Each chord sheet is uniquely linked to one specific song, forming a one-to-one relationship with the Songs entity. However, a single chord sheet can be referred to by multiple users who are learning or practicing that particular song, establishing a many-to-one relation- ship from the User entity.

4. The Test Series entity represents collections of musical assessments or learning modules. Each test series is uniquely identified by a testid and may have attributes like score (though this might be better associated with a user's attempt at a test) and potentially the types of chords being tested. A single test series can be viewed by multiple users who are engaging with the learning content, indicating a many-to-one relationship with the User entity.

4.2 Activity diagrams

The Activity Diagram is used to describe the dynamic aspects of the system. We have drawn Activity Diagram 4.2, which illustrates the main functionalities of the system: Generating the chord sheet, Practicing chords, and Testing chord knowledge.

When the user selects the Generate Chord Sheet option, they will be prompted to upload a video. Once the video is uploaded, the system will analyze the chords and generate a chord sheet, which will then be saved. After saving, the user can either practice the generated chord sheets or exit the feature.

If the Practice Songs feature is selected, the user will choose a chord sheet, and the system will use real-time detection to determine whether the chords played are correct. Once all chords from the chord sheet have been played, the user will be given the option to repeat the practice session or exit. If they choose to practice again, the process will restart; otherwise, they will exit the feature.

In the Test Your Knowledge feature, a random chord will be displayed on the screen, and the user will be asked to play it. The system will analyze the accuracy of the played chord and award points accordingly. After completing the test, the user will be asked whether they want to attempt the test again. If they agree, the process will be repeated; otherwise, they will exit the feature.

SIT LIBRARY-2024-2025

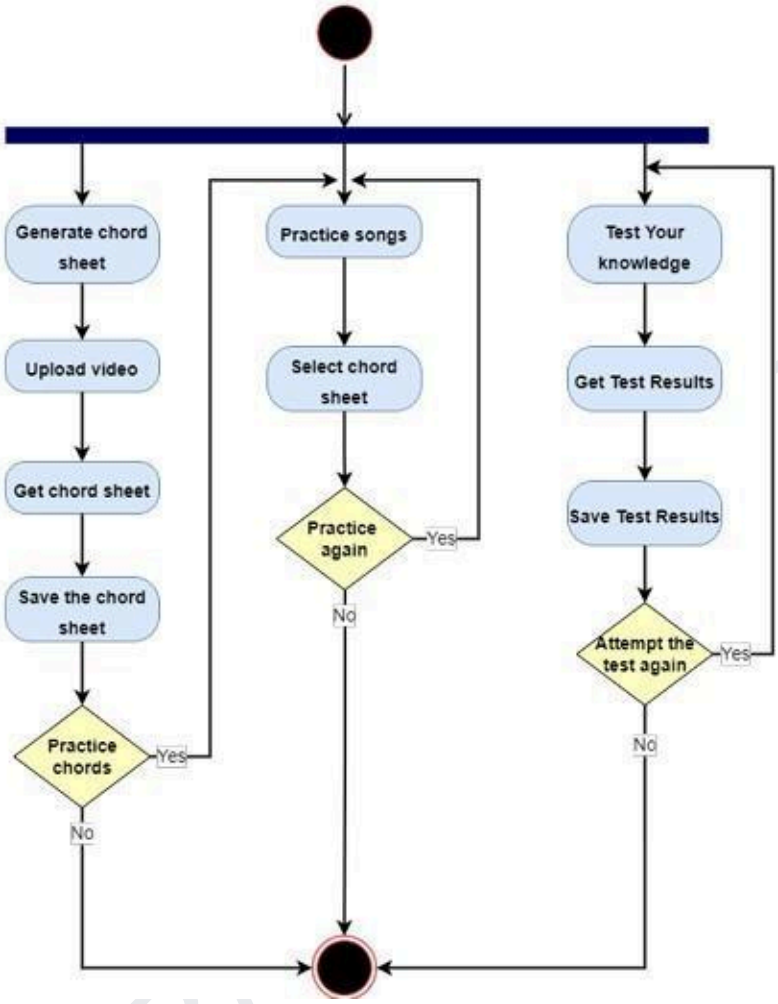


Figure 4.2: Activity diagram

4.3 Functional modeling - DFD level 0

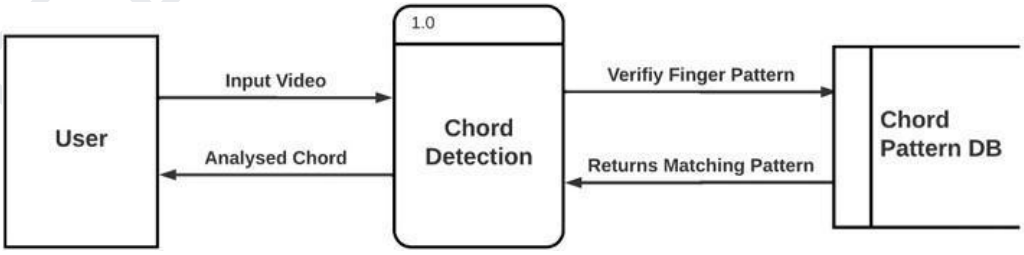


Figure 4.3: Dataflow level 0

The figure 4.3 represents the Data Flow Diagram (DFD), which is a visual representation of how data moves through the system. It illustrates the flow of data between various components and their interactions.

At DFD Level 0, the user inputs a video to detect the chords being played. The Chord Detection

Module then processes this input by analyzing the finger pattern used in the video. This information is then sent to the Chord Pattern Database, where it is verified against existing patterns.

Once the verification process is complete, the system retrieves the matching chord pattern from the database and sends it back to the Chord Detection Module. Finally, the module processes this data and delivers the analyzed chord information to the user, allowing them to see the detected chords in real time.

This structured data flow ensures an efficient and accurate chord detection process, helping musicians and learners interpret and practice chords effectively.

4.4 Functional modeling - DFD level 1

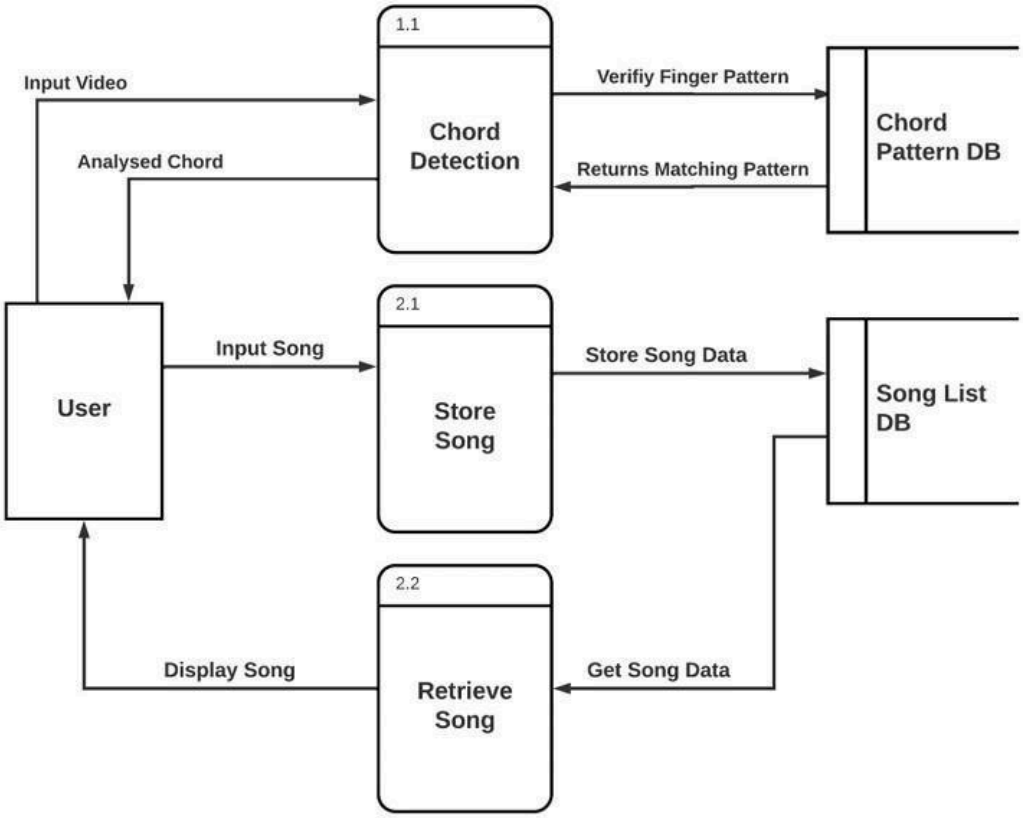


Figure 4.4: Dataflow level 1

The figure 4.4 represents the Data Flow Diagram (DFD) Level 1, which provides a more detailed breakdown of the chord detection and song storage process.

When the user uploads a video as input to the Chord Detector, the system analyzes the finger pattern and verifies it against existing chord patterns. If a match is found, the Chord Detector processes the analyzed chord information and sends it back to the user.

In another scenario, if the user inputs a song to be stored, the system saves it in the Song List Database. When the user requests the song data, the system retrieves the stored information from the database and displays it to the user.

This structured data flow ensures a smooth chord detection and song storage process, making it easier for users to analyze chords and access stored songs efficiently.

4.5 Functional modeling - DFD level 2

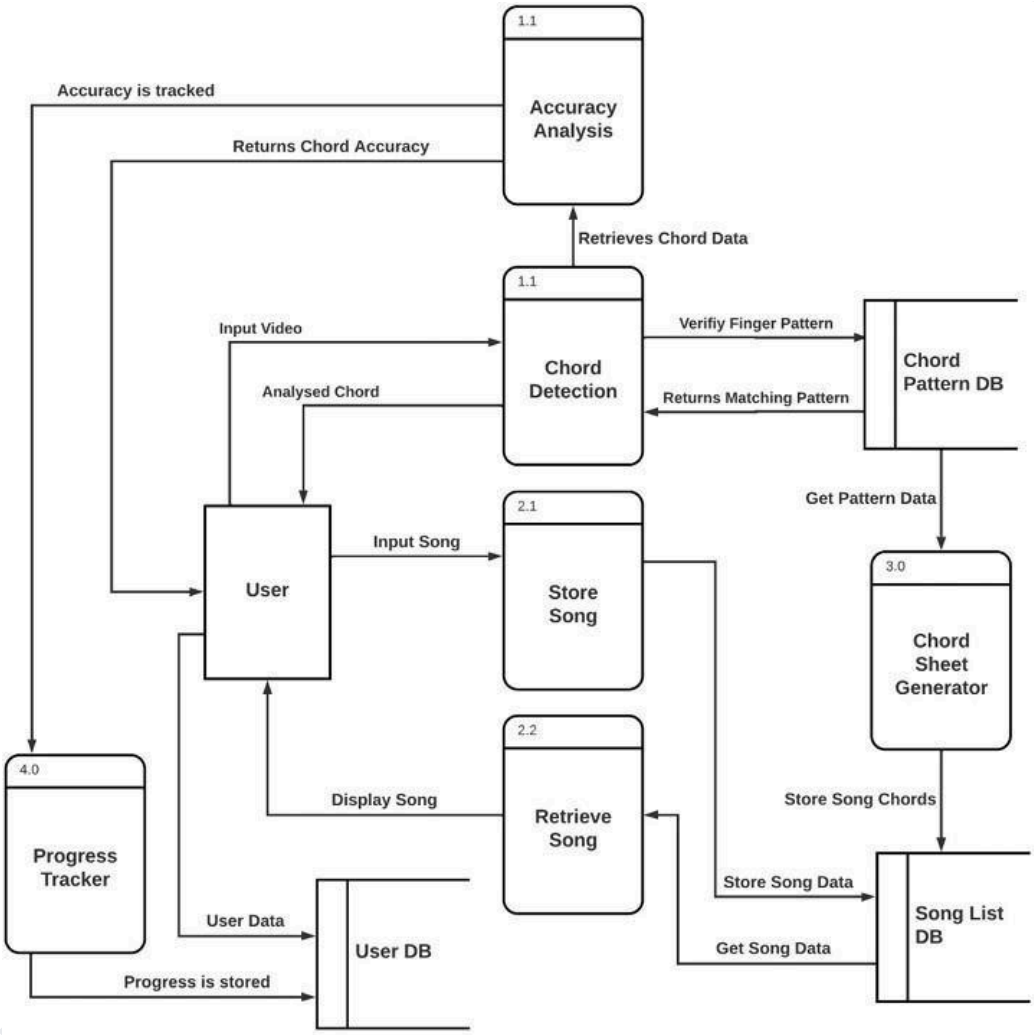


Figure 4.5: Dataflow level 2

The figure 4.5 is the level 2 data flow diagram. The User uploads a video to the chord detection module and verifies finger pattern and sends it to the chord pattern database and returns matching pattern and sends it to the chord detection module where it sends the analysed chord accuracy to the user as well as it retrieves the chord data and sends it for accuracy analysis where the accuracy is tracked in the progress tracker and the progress is stored in the user database

4.6 Timeline chart

		Name	Duration	Start	Finish	Predecessors
1		Major Project Proposal	5 days?	15/7/24 8:00 AM	19/7/24 5:00 PM	
2		Major Project Evaluation	5 days?	22/7/24 8:00 AM	26/7/24 5:00 PM	1
3		Major Project Presentation	5 days?	29/7/24 8:00 AM	2/8/24 5:00 PM	2
4		Research on Dataset	5 days?	5/8/24 8:00 AM	9/8/24 5:00 PM	3
5		Chord Selection Research	5 days?	12/8/24 8:00 AM	16/8/24 5:00 PM	4
6		Research on Papers	5 days?	19/8/24 8:00 AM	23/8/24 5:00 PM	5
7		Discussion about Literature Review	5 days?	26/8/24 8:00 AM	30/8/24 5:00 PM	6
8		Presentation preparation for MSE	5 days?	2/9/24 8:00 AM	6/9/24 5:00 PM	7
9		Documentation in Latex	5 days?	9/9/24 8:00 AM	13/9/24 5:00 PM	8
10		Presentation and Report finalization	5 days?	16/9/24 8:00 AM	20/9/24 5:00 PM	9
11		Mid Sem Evaluation	5 days?	23/9/24 8:00 AM	27/9/24 5:00 PM	10
12		Research on Algorithm	5 days?	30/9/24 8:00 AM	4/10/24 5:00 PM	11
13		Research on Dataset	5 days?	7/10/24 8:00 AM	11/10/24 5:00 PM	12
14		Dataset Creation	5 days?	14/10/24 8:00 AM	18/10/24 5:00 PM	13
15		End Sem Presentation	5 days?	21/10/24 8:00 AM	25/10/24 5:00 PM	14

Figure 4.6: Timeline chart July 2024 - November 2024

Name	Duration	Start	Finish	Predecessors
Creating the dataset	5 days?	14/1/25 9:00 AM	21/1/25 9:00 AM	
Working on the model	5 days?	21/1/25 9:00 AM	28/1/25 9:00 AM	1
Training	5 days?	28/1/25 9:00 AM	4/2/25 9:00 AM	2
Discussion of the platform for frontend	5 days?	4/2/25 9:00 AM	11/2/25 9:00 AM	3
Creating the User Interface	5 days?	11/2/25 9:00 AM	18/2/25 9:00 AM	4
Working on the first feature	5 days?	18/2/25 9:00 AM	25/2/25 9:00 AM	5
Workig on the second feature	4 days?	25/2/25 9:00 AM	3/3/25 9:00 AM	6
Working on the third feature	5 days?	3/3/25 9:00 AM	10/3/25 9:00 AM	7
Making of blackbook	5 days?	10/3/25 9:00 AM	17/3/25 9:00 AM	8

Figure 4.7: Timeline chart January 2025 - March 2025

		Name	Duration	Start	Finish	Predecessors	Resource Names
1		Integration of pages	5 days?	17/3/25 9:00 AM	24/3/25 9:00 AM		
2		Model integration	5 days?	24/3/25 9:00 AM	31/3/25 9:00 AM	1	
3		Colloquium Preparation	5 days?	31/3/25 9:00 AM	7/4/25 9:00 AM	2	
4		End Sem Evaluation	5 days?	7/4/25 9:00 AM	14/4/25 9:00 AM	3	

Figure 4.8: Timeline chart March 2025 - April 2025

Table 4.1: Week-wise Project Timeline

Week	Date Range	Activity Description
Week 1	1st July – 7th July	Project topic finalization and initial team setup. Defined project goals and scope.
Week 2	8th July – 14th July	Conducted literature review; studied existing systems and technologies.
Week 3	15th July – 21st July	Defined functional and non-functional requirements. Began ER diagram drafting.
Week 4	22nd July – 28th July	Completed ER diagram and initiated DFD Level 0 design.
Week 5	29th July – 4th Aug	Created DFD Level 1 and reviewed system flows.
Week 6	5th Aug – 11th Aug	Designed DFD Level 2 and finalized functional modeling.
Week 7	12th Aug – 18th Aug	Created activity diagrams for core features like practice, test, and chord sheet.
Week 8	19th Aug – 25th Aug	Finalized system architecture, started UI wireframing.
Week 9	26th Aug – 1st Sep	Designed UI screens for signup, signin, and get started pages.
Week 10	2nd Sep – 8th Sep	Completed UI designs for practice, test, and music theory pages.
Week 11	9th Sep – 15th Sep	Implemented initial layout using HTML, CSS, and Flask backend.
Week 12	16th Sep – 22nd Sep	Began dataset creation using OpenCV + Mediapipe for chords C, D, Em, G.
Week 13	23rd Sep – 29th Sep	Trained CNN model using Keras on chord dataset.
Week 14	30th Sep – 6th Oct	Integrated chord detection model into the Flask app.
Week 15	7th Oct – 13th Oct	Implemented real-time webcam input and chord prediction.
Week 16	14th Oct – 20th Oct	Linked practice and test sections with live detection and scoring.
Week 17	21st Oct – 27th Oct	Built music theory content pages and finalized UI flow.
Week 18	28th Oct – 3rd Nov	Performed unit testing for each module and debugged UI issues.

SET LIBRARY 2024 - 2025

Week	Date Range	Activity Description
Week 19	4th Nov – 10th Nov	Began integration testing and connected database (Firebase).
Week 20	11th Nov – 17th Nov	Finalized all backend APIs and database routes.
Week 21	18th Nov – 24th Nov	Uploaded all chord sheets, content pages, and verified flows.
Week 22	25th Nov – 1st Dec	Mid-submission documentation: diagrams, model summary.
Week 23	2nd Jan – 8th Jan	Resumed post-vacation: refined practice/testing UI accuracy.
Week 24	9th Jan – 15th Jan	Improved model performance based on new test data.
Week 25	16th Jan – 22nd Jan	User feedback collection and adjustments to UI.
Week 26	23rd Jan – 29th Jan	Added additional chords and tested on new users.
Week 27	30th Jan – 5th Feb	Performance metrics gathering and result graphs.
Week 28	6th Feb – 12th Feb	Final UI polish and mobile compatibility check.
Week 29	13th Feb – 19th Feb	Began project documentation and report writing.
Week 30	20th Feb – 26th Feb	Internal review and modifications based on feedback.
Week 31	27th Feb – 4th Mar	Final internal testing and system backup.
Week 32	5th Mar – 11th Mar	Preparation of viva materials and demonstration videos.
Week 33	12th Mar – 18th Mar	Project submission and internal viva.
Week 34	19th Mar – 25th Mar	Final debugging, fixes, polishing deployment build.
Week 35	26th Mar – 1st Apr	Final deployment and performance validation.
Week 36	2nd Apr – 8th Apr	Final report, citations, appendix formatting.
Week 37	9th Apr – 15th Apr	Print submission and final team sync-up.
Week 38	16th Apr – 20th Apr	External viva and project exhibition.

SPIT LIBRARY 2024 - 2025

Chapter 5

Implementation

A number of machine learning methods have been proposed for image classification and image processing. We will be using Deep learning methods for guitar chord detection and classification. For creating the model we need the dataset and algorithm which will suit our requirements. The Below sections give the complete idea about how the dataset is created and algorithms has been designed.

5.1 Data collection

Guitar chords are identified based on root notes and are expressed alphabetically as A, B, C, D, E, F, and G, which correspond to the seven musical notes ”la”, ”ti”, ”do”, ”re”, ”mi”, ”fa”, and ”sol”, respectively. A Deep learning model requires a large amount of Training data to work accurately. We have used OpenCV and mediaipe libraries of python to create the dataset. OpenCV
[6] is an open-source computer vision and machine learning library designed for real-time computer vision applications. It provides a wide range of tools and functions that allow developers to create and deploy computer vision applications quickly and easily. It helps in Image Processing, Object detection, Machine learning, and Performance Optimization. MediaPipe is an open-source framework developed by Google for building real-time, cross-platform computer vision and machine learning applications. It provides a wide range of customizable building blocks that allow developers to create complex pipelines for video and audio processing, machine learning inference, and more. We have used OpenCV for capturing, cropping, and converting the images to grey scale and Mediapipe to recognize the hand gestures. It is a built-in python framework which is used to build meachine learning algorithms which has in-built solutions for Hand tracking which is done with the help of 21 landmarks in 3D with multihand support. Mediapipe recognizes the gestures very accurately, even if there is a small change it will be captured. By using these we have created a dataset of 20000 images. 5000 images of each chord C, D, Em and G. All the images are 200 x 200 dimensions in grayscale.

SIT LIBRARY 2024 - 2025

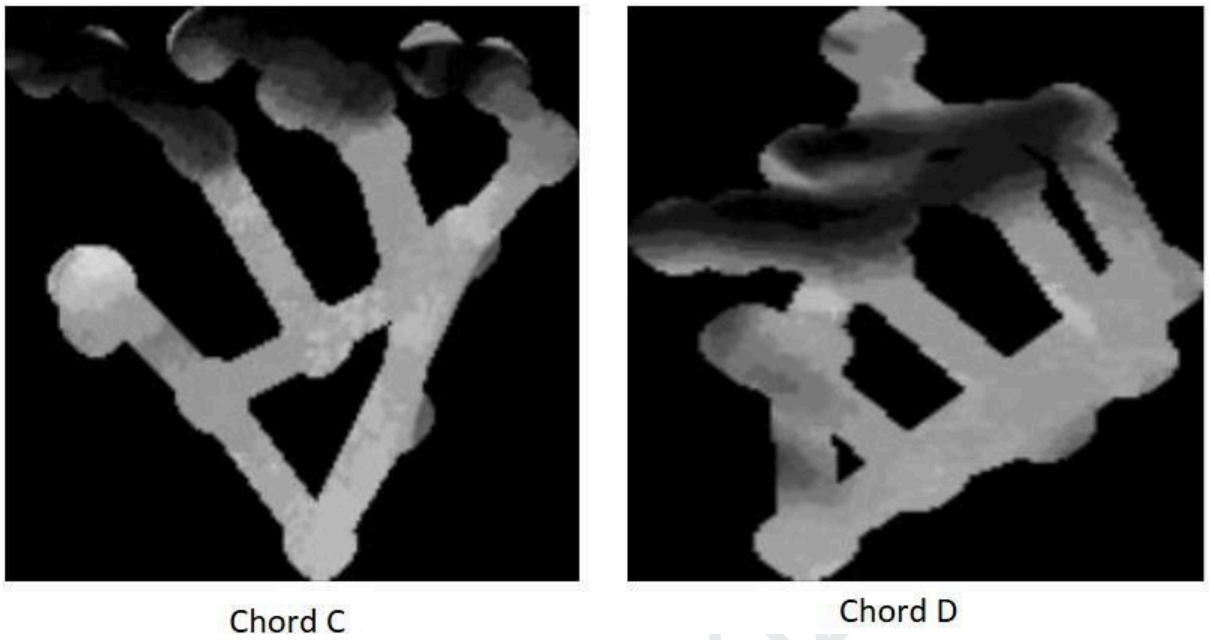


Figure 5.1: Chord C Chord D on grayscale

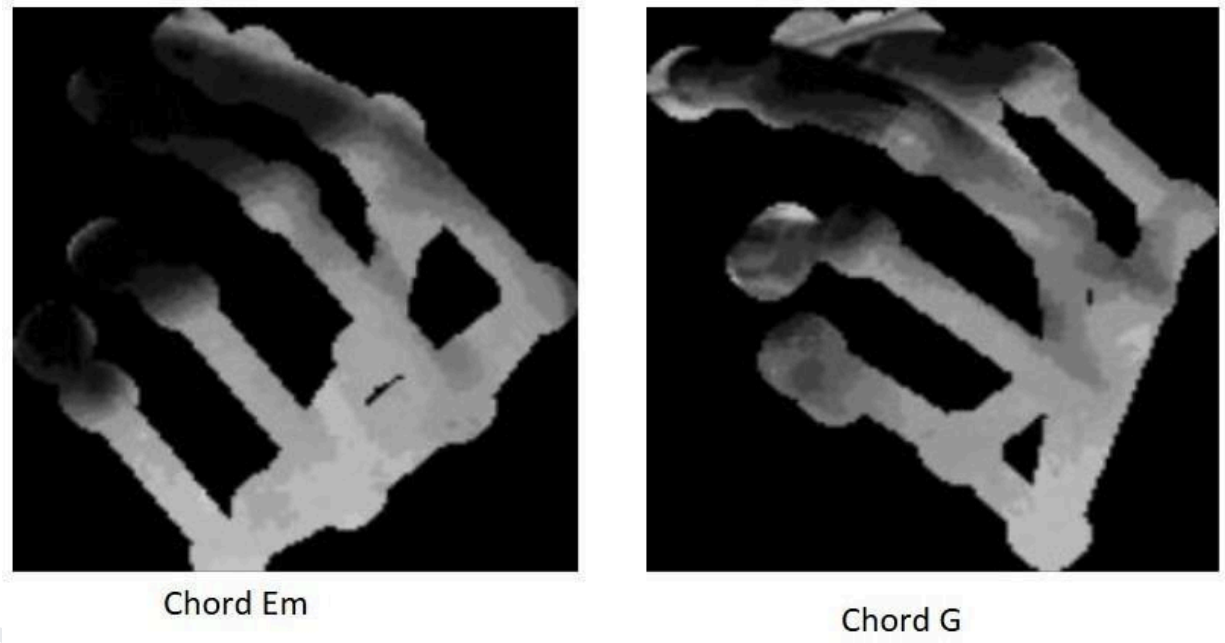


Figure 5.2: Chord Em Chord G on grayscale

5.2 Algorithms and methodology

For creating the deep learning model, we have used Keras. It is a powerful and flexible library that simplifies the process of building and training neural networks. Its modular architecture, high-level API, and pre-trained models make it an ideal choice for a wide range of applications,

from computer vision and natural language processing to reinforcement learning and time-series forecasting. With the help of Keras, we have designed our chord detection model.

5.3 Chord Detection model

The type of our model is Sequential (CNN) [7]. It is a feedforward neural network that is designed to automatically learn and extract features from images. It is of Sequential type as it has one input neuron and one output neuron. The input neuron will be the real-time hand gesture chord by the user, and the output neuron is the chord detected. The model consists of multiple layers of convolutional and pooling operations, followed by one or more fully connected layers. There are 8 Layers in our model. The 1st layer is 2D convolutional layer to the model with 32 filters, a kernel size of 5x5, and a ”ReLU” activation function. ReLu fucntion is used as it as better for time efficiency. The 2nd layer is Max pooling layer to the model with a pool size of 2x2 and a stride of 2 in both dimensions. The padding argument is set to ’same’ to ensure that the output feature maps have the same spatial dimensions as the input feature maps. The 3rd Layer is again 2D convolution layer and 4th is the Max pooling layer. The 5th layer is flattening layer to the model that flattens the previous layer’s output into a one-dimensional vector. The 6th layer is the Dense layer which adds a fully connected layer to the model with 1024 units and a ReLU activation function. The 7th layer is Dropout layer to the model with a rate of 0.6, which randomly drops 60prevent overfitting. The 8th layer is again the Dense layer adds the output layer to the model with number of class units which is 5 and a softmax activation function, which generates output probabilities that sum to 1. The figure below gives the Representation of our model for an example image.

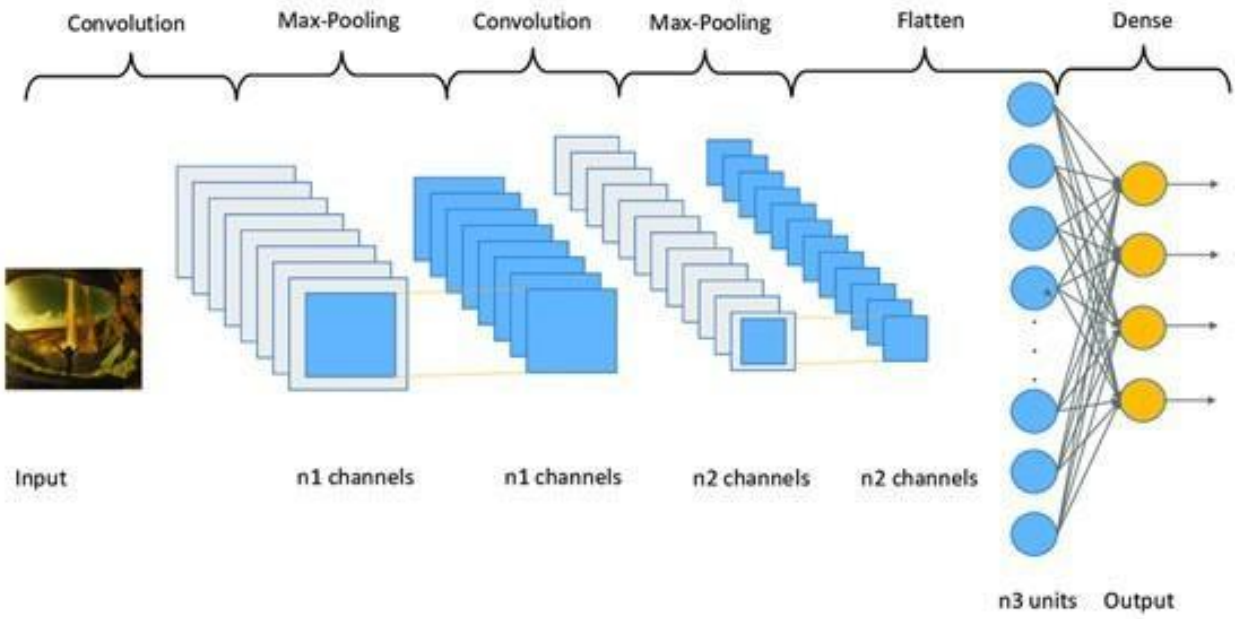


Figure 5.3: Representation of Keras model

5.4 Training of the model

The model is trained on the dataset which we have created. The dataset has been divided into 80% training and 20% testing. There are two important components while training a deep learning a model, ”Forward Pass” and ”Backward Pass”. In Forward Pass, The input data is fed forward through the network, and the output is compared to the true values. In Backward Pass, The er- ror between the predicted and true values is propagated backwards through the network, and the weights are updated using an optimization algorithm such as Adam. Another 2 importanat param- eteras are Batch size and epochs. Batch size refers to the number of training examples that are processed in a single forward/backward pass of the network during training and Epochs refer to the number of times the entire training dataset is processed during training. In each epoch, the entire dataset is fed into the network in batches, and the weights of the network are updated based on the error between the predicted and true values. The number of epochs is an important hyperparameter that determines how many times the network will see the training data and how long the training process will continue. The Batch size of our system is 64 and Epochs are 25. Both batch size and epochs are used in both the forward and backward passes during the training of a deep learning model. During the forward pass, the batch size determines the number of training examples that are processed in a single forward pass through the network, and the number of epochs determines how many times the entire training dataset is processed through the network. During the backward pass, the batch size is used to compute the gradient of the loss function with respect to the weights for each batch, and the number of epochs determines how many times the weights are updated based on the gradients computed from the entire training dataset.

5.5 Algorithm

An algorithm is a step-by-step procedure or set of instructions for solving a computational prob- lem. The steps for detecting the chord are as follows:

Step 1: Upload a video or capture through webcam. For the feature ”Generate Chord Sheet”, video input will be taken and for the features ”Practice Chords” and ”Test Series”, real-time web-cam captures will be taken as input.

Step 2: The input is in the form of video. The real-time captures are also video but continuous. The video will be converted into frames. Frames are images. Each frame is replicated and stored in two variables: one will be used for detecting the chord, and the other will be used for displaying the output to the user.

Step 3: Preprocess both frames to extract the hand gesture. Hand gesture detection is done using the MediaPipe library. Gesture landmarks are drawn on the frame.

Step 4: One of the frames, with the hand gesture drawn, is further preprocessed and converted into a grayscale image.

Step 5: The grayscale image is cropped to isolate only the hand gesture. The cropped image size is 200x200 pixels.

Step 6: The cropped image is fed as input to the model.

Step 7: The output of the model is the *detected chord* and the probability associated with the chord.

SIT LIBRARY 2024-2025

Step 8: The detected chord is written on the other frame that had the hand gesture drawn. This results in an output image containing the frame, detected gesture, and detected chord.

Step 9: The output image is displayed to the user in video format. The processed images are continuously shown to recreate the video.

5.6 Working of the project

5.6.1 Dataset Creation

```
import cv2
import os
import mediapipe as mp


''' This code is for creating dataset '''
# cv2 is the Open CV library used for image and video processing
# mediapipe is a library used for getting the hand gestures


# initializing the mediapipe variables for drawing the landmarks and connections on hands detected in the video stream
mp_drawing = mp. solutions. drawing_utils
mp_hands = mp. solutions. hands


# the size of the image
image_x , image_y = 200 , 200


# function for creaking a folder to save the images of dataset
def create_folder ( folder_name ):
    if  not  os. path . exists( folder_name ):
    os. mkdir( folder_name )


# main function
def main ( c_id ):
    total_pics = 5000 # setting the total number of images

        # hands is an instance of the mp_hands. Hands class , which is used for detecting hand landmarks in the video stream .
        hand_landmark_drawing_spec and hand_connection_drawing_spec define the drawing style for the landmarks and
        connections respectively .
    hands = mp_hands. Hands(
    min_detection_confidence =0.5 ,  min_tracking_confidence =0.5)
    hand_landmark_drawing_spec  =  mp_drawing . Drawing Spec ( thickness =5 ,  circle_radius =5)
    hand_connection_drawing_spec  =  mp_drawing . Drawing Spec ( thickness =10 ,  circle_radius =10)

    cap = cv2 . Video Capture (0) # initializing a variable for capturing images
    create_folder (" chords/" + str( c_id )) # folder for where the images will be saved
    pic_no = 0 # to keep a count on number of images , initially set to zero
```

```
39      flag_start_capturing = False # initialling setting as Flase later set to True when capturing started
40      frames = 0
41
42      # when the camera is opened the loop will start
43      while  cap . isOpened ():
44          ret , image = cap . read () # captures the image .
45          image = cv2 . flip ( image , 1) # flips the image horizontally .
46          image_orig = cv2 . flip ( image , 1)
47          image = cv2 . cvtColor( cv2 . flip ( image , 1), cv2 . COLOR_BGR 2 RGB ) # converting the iamge to RGB format.
48          results_hand = hands. process( image ) # passing it to hands object to detect the hand landmarks and connections .
49          image . flags. writeable = True
50          image = cv2 . cvtColor( image , cv2 . COLOR_RGB 2 BGR ) # converting the image frame to BGR and making it writable
51          # checking if there are any detected hand landmarks and connections in the current image frame
52          # if yes the draw the landmark on the frame
53          if results_hand . multi_hand_landmarks :
54              for  hand_landmarks  in  results_hand . multi_hand_landmarks :
55                  mp_drawing . draw_landmarks (
56                      image =image ,
57                      landmark_list = hand_landmarks ,
58                      connections = mp_hands. HAND_CONNECTIONS ,
59                      landmark_drawing_spec = hand_landmark_drawing_spec ,
60                      connection_drawing_spec = hand_connection_drawing_spec )
61          # applying a bitwise AND between the original frame and changes image
62          res  =  cv2 . bitwise_and ( image ,  cv2 . bitwise_not ( image_orig ))
63
64          # converting the image to garyscale and applying a threshold to create a binary image
65          gray = cv2 . cvtColor( res , cv2 . COLOR_BGR 2 GRAY )
66          ret , th1 = cv2 . threshold ( gray , 25 , 255 , cv2 . THRESH_BINARY )
67          contours , hierarchy = cv2 . find Contours ( th1 , cv2 . RETR_EXTERNAL , cv2 . CHAIN_APPROX_SIMPLE )
68
69          # if there is a contour in image then save the image in bounding rectangel
70          if len ( contours) > 0:
71              contour = max( contours , key= cv2 . contourArea )
72              # if cv2 . contourArea ( contour) > 10000 and frames > 50:
73              x1 , y1 , w1 , h1 = cv2 . bounding Rect ( contour)
74              pic_no  +=  1
75              cv2 . rectangle ( image , ( x1 , y1 ), ( x1 + w1 , y1 + h1 ), (0 , 255 , 0), 2)
76              save_img = gray[ y1 : y1 + h1 , x1 : x1 + w1 ]
77              save_img = cv2 . resize ( save_img , ( image_x , image_y ))
78              cv2 . putText( image , " Capturing ...", (30 , 60) , cv2 . FONT_HERSHEY_TRIPLEX , 2 ,
                    (127 , 255 , 255))
79              cv2 . imwrite (" chords/" + str( c_id ) + "/" + str( pic_no ) + ". jpg", save_img )
80
81              # cv2 . rectangle ( image , (x, y), ( x + w, y + h), (0 , 255 , 0), 2)
82              cv2 . putText( image ,  str( pic_no ), (30 ,  400) , cv2 . FONT_HERSHEY_TRIPLEX ,  1.5 ,
                    (127 , 127 , 255))
83              keypress = cv2 . waitKey (1)
```

```
84
85                      # it will continue to capture until the c is pressed and will stop when
                            captured 5000 images if
86                      keypress == ord ('c'):
87                          if flag_start_capturing == False : flag_start_capturing = True
88                          else :
89                              flag_start_capturing = False frames = 0
90                      if flag_start_capturing == True : frames += 1
91                      if pic_no == total_pics : break
92              cv2 . imshow (" Capturing gesture ", image )
93              cv2 . imshow (" Res", res)
94
95
96
97
98
99 # entering the name of the chord so to images in that specific folder
100 c_id = input(" Enter chord : ")
101 main ( c_id )
```

5.6.2 Training Dataset

```
1       import matplotlib . pyplot as plt
2 from keras. callbacks import ModelCheckpoint
3 from keras. layers import Dense , Flatten , Conv2D
4 from keras. layers import MaxPooling2D , Dropout
5 from keras. models import Sequential
6 from keras. preprocessing . image import Image Data Generator
7 from keras import callbacks
8
9
10 ''' This code is for creating the model '''
11
12
13 image_x , image_y = 200 , 200
14 batch_size = 64 # he number of samples processed during training
15 train_dir = " chords"
16
17
18 # creating CNN model architechture
19 def keras_model ( image_x , image_y ):
20      num_of_classes  =  5
21      model = Sequential ()
22          model. add ( Conv2D (32 , (5 , 5), input_shape =( image_x , image_y , 1), activation =' relu ')) # performing convolution on the
            input image
23          model. add ( MaxPooling2 D ( pool_size =(2 , 2), strides =(2 , 2), padding =' same ')) # reducing the size of the feature maps
            obtained from previous layer
24            model. add ( Conv2D (64 , (5 , 5), activation =' relu ')) # adding another convolution layer to extract more complex features
25          model. add ( MaxPooling2 D ( pool_size =(5 , 5), strides =(5 , 5), padding =' same ')) # adding another pooling layer to
            reduce the size of feature maps
26      model. add ( Flatten ()) # flattening the previous layer into a one - dimensional vector
```

```
27          model. add ( Dense (1024 , activation =' relu ')) # a fully connected layer with 1024 units
28          model. add ( Dropout (0.6)) # randomly drops some of the neurons in previous layer to prevent overfitting
29          model. add ( Dense ( num_of_classes , activation =' softmax ')) # output layer with softmax activation which genrates output
                probabilities
30
31          # compiling the modal and saving the best model
32          model. compile ( loss=' categorical_crossentropy ', optimizer=' adam ', metrics =[' accuracy ']) 
33          model. summary ()
34          filepath = " guitar_learner_final . h5 "
35          checkpoint = ModelCheckpoint ( filepath , monitor=' val_acc ', verbose =1 , save_best_only = True , mode =' max ')
36          callbacks_list = [ checkpoint ]
37
38          return model , callbacks_list
39
40
41  def     main ():
42          # setting up the dataset for creating training and testing                                    sets
43          train_datagen = Image Data Generator (
44                  rescale =1./255 ,
45                  width_shift_range =0.2 ,
46                  height_shift_range =0.2 ,
47                  shear_range =0.2 ,
48                  rotation_range =15 ,
49                  zoom_range =0.2 ,
50                  horizontal_flip =False ,
51                  validation_split =0.2 ,
52                  fill_mode =' nearest ')
53
54
55          # Training dataset 80%
56          train_generator  =  train_datagen . flow_from_directory (
57                  train_dir ,
58                  target_size =( image_x , image_y ),
59                  color_mode =" grayscale ",
60                  batch_size = batch_size ,
61                  seed =42 ,
62                  class_mode =' categorical ',
63                  subset=" training ")
64
65          class_indices = train_generator . class_indices
66          print( class_indices )
67
68          # Testing  dataset  20%
69          validation_generator  =  train_datagen . flow_from_directory (
70                  train_dir ,
71                  target_size =( image_x , image_y ),
72                  color_mode =" grayscale ",
73                  batch_size = batch_size ,
74                  seed =42 ,
75                  class_mode =' categorical ',
76                  subset=" validation ")
```

```
77
78      model , callbacks_list = keras_model ( image_x , image_y )
79
80      print( callbacks_list )
81      # model. fit_generator ( train_generator ,  epochs =5 ,  validation_data = validation_generator )
82            # early_stopping = callbacks. EarlyStopping ( monitor =" val_accuracy ", mode = " max", patience =5 , verbose =0 ,
              restore_best_weights = True )
83
84      # starts  training  with  25  epochs
85            history = model. fit_generator ( train_generator , epochs =25 , validation_data = validation_generator )
86      # scores  =  model. evaluate_generator ( generator= validation_generator , steps =64)
87
88      # printing results
89      plt. plot( history . history [' accuracy ']) 
90      plt. plot( history . history [' val_accuracy ']) 
91      plt. title (' model accuracy ') 
92      plt. ylabel(' accuracy ') 
93      plt. xlabel(' epoch ') 
94      plt. legend ([' Train ', ' Validation '], loc=' upper left ') 
95      plt. show () 
96      # summarize history for loss
97      plt. plot( history . history [' loss ']) 
98      plt. plot( history . history [' val_loss ']) 
99      plt. title (' model loss ') 
100     plt. ylabel(' loss ') 
101     plt. xlabel(' epoch ') 
102     plt. legend ([' Train ', ' Validation '], loc=' upper left ') 
103     plt. show () 
104
105     scores  =  model. evaluate_generator ( generator= validation_generator ,  steps =64)
106
107     print(" CNN Error: %.2 f%%" % (100 - scores [1] * 100)) 
108     print(" Accuracy : %.2 f%%" % ( scores [1] * 100)) 
109
110     model_json = model. to_json () 
111     with open (" model_final . json ", " w") as json_file : 
112     json_file . write ( model_json ) 
113
114     model. save (' guitar_learner_final . h5 ') 
115
116
117
118
119 main ()
```

5.6.3 Testing

```
import cv2
import mediapipe as mp
from keras. models import load_model
import numpy as np
import tempfile
import pyrebase
import time

config = {
        ' apiKey ': " AIza Sy CZi E 4 vt37 ux A_s Jrl03 KAhk 0 j YKOtl5 s U ",
        ' auth Domain ':  " guitarguru -484 be. firebaseapp . com ",
        ' projectId ': " guitarguru -484 be",
        ' storage Bucket ':  " guitarguru -484 be. appspot. com ",
        ' messagingSenderId ': " 467197811011 ",
        ' appId ':  " 1 :467197811011 : web :95286 f40 bdc0 c718 c0 f7 f6 ",
        ' measurementId ':  "G- Y 74 QRV 7 FFZ ",
        ' databaseURL ': " https :// guitarguru -484 be - default - rtdb . asia - southeast1 . firebasedatabase . app "
}

def rescale_frame ( frame , percent =100):
        width = int( frame . shape [1] * percent / 100)
        height = int( frame . shape [0] * percent / 100)
        dim = ( width , height)
        return  cv2 . resize ( frame ,  dim ,  interpolation = cv2 . INTER_AREA )

def keras_predict ( model , image ):
        processed = keras_process_image ( image )
        pred_probab  =  model. predict( processed )[0]
        pred_class = list( pred_probab ). index( max( pred_probab ))
        return max( pred_probab ), pred_class


def keras_process_image ( img):
        image_x , image_y = 200 , 200
        img = cv2 . resize ( img , ( image_x , image_y ))
        img = np. array( img , dtype =np. float32 )
        img = np. reshape ( img , (-1 , image_x , image_y , 1))
        return  img


def  generate_chord_sheet ( chords):
        count = 0
        prev_item  =  None
        for chord in chords:
                if chord == prev_item :
                        count = count + 1
                else :
                        if prev_item is not None :
                                print( f" Play Chord { prev_item } for { count} secs \ n")
                                count = 1
                        else :
```

```
                        count =1

                prev_item = chord
        print( f" Play Chord { prev_item } for { count} secs \ n")

# setting up the test camera
def process_video_test ():
        global cap
        global chord_sheet
        global hands
        image_x , image_y = 200 , 200
        mp_drawing = mp. solutions. drawing_utils
        mp_hands = mp. solutions. hands
        model = load_model (' guitar_learner_final . h5 ')
        chord_dict = {0: 'A', 1: 'C', 2: 'F', 3: 'G ',4: 'NA '}
        chord_sheet = []

        hands = mp_hands. Hands(
                min_detection_confidence =0.7 , min_tracking_confidence =0.7)
        hand_landmark_drawing_spec = mp_drawing . Drawing Spec ( thickness =5 , circle_radius =5)
        hand_connection_drawing_spec = mp_drawing . Drawing Spec ( thickness =10 , circle_radius =10)
        pic_no = 0
        flag_start_capturing = False
        frames = 0
        cap = cv2 . Video Capture (0)

        countdown = 40 # seconds
        start_time = time . time ()
        # your code to process the video goes here
        while True :
                ret , image = cap . read ()
                if not ret:
                        break
                else :
                        image = cv2 . flip ( image ,1)
                        image_orig = cv2 . flip ( image ,1)
                        image = cv2 . cvtColor( cv2 . flip ( image ,1) , cv2 . COLOR_BGR 2 RGB )
                        results_hand = hands. process( image )
                        # image . flags. writable = True
                        image = cv2 . cvtColor( image , cv2 . COLOR_RGB 2 BGR )
                        if results_hand . multi_hand_landmarks :
                                for hand_landmarks in results_hand . multi_hand_landmarks :
                                        mp_drawing . draw_landmarks (
                                                image = image ,
                                                landmark_list = hand_landmarks ,
                                                connections = mp_hands. HAND_CONNECTIONS ,
                                                landmark_drawing_spec = hand_landmark_drawing_spec ,
                                                connection_drawing_spec = hand_connection_drawing_spec )
                        res = cv2 . bitwise_and ( image , cv2 . bitwise_not ( image_orig ))

                        gray = cv2 . cvtColor( res , cv2 . COLOR_BGR 2 GRAY )
                        ret , th1 = cv2 . threshold ( gray , 25 , 225 , cv2 . THRESH_BINARY )
                        contours , heirarchy = cv2 . find Contours ( th1 , cv2 . RETR_EXTERNAL , cv2 . CHAIN_APPROX_SIMPLE )
```

```
105
106                     if len ( contours) >0:
107                         contours = sorted ( contours , key= cv2 . contourArea )
108                         contour = contours [-1]
109                         x1 , y1 , w1 , h1 = cv2 . bounding Rect ( contour)
110                         save_img = gray[ y1 : y1 + h1 , x1 : x1 + w1 ]
111                         save_img = cv2 . resize ( save_img , ( image_x , image_y ))
112                         pred_probab , pred_class = keras_predict ( model , save_img )
113                         if pred_class == 0:
114                             print(" Chord : A")
115                             chord_sheet . append (" A")
116                         elif pred_class == 1:
117                             print(" Chord : C")
118                             chord_sheet . append (" C")
119                         elif pred_class == 2:
120                             print(" Chord : F")
121                             chord_sheet . append (" F")
122                         elif pred_class == 3:
123                             print(" Chord : G")
124                             chord_sheet . append (" G")
125                         elif pred_class == 4:
126                             print(" Chord : No Chord ")
127                             chord_sheet . append (" No  Chord ")
128                         else :
129                             print( pred_class , pred_probab )
130
131                         # cv2 . putText( image , str( chord_dict [ pred_class ]), ( x1 + 50 , y1 - 50) , cv2 . FONT_HERSHEY_SIMPLEX , 6 , (255
                                , 0 , 0), 9)
132                         ret , buffer = cv2 . imencode (’. jpg ’, image )
133                         image = buffer. tobytes ()
134                         yield (b’-- frame \ r\n’ b’ Content - Type : image / jpeg\ r\ n\ r\n’                    +   image    +   b ’\ r\n
                                ’)
135                         elapsed_time = time . time () - start_time
136                         if elapsed_time >= countdown :
137                             cap . release ()
138                             break
139                     else :
140                         print(" OOPS !!! ISUUE ")
141                         chord_sheet . append (" NO  Chord ")
142         cap . release ()
143         cv2 . destry AllWindows ()
144
145
146 # Calculating the score and accuracy of chord played
147 def generate_score ( chord ):
148         cap . release ()
149         score  =  100
150         total_chords  =  len ( chord_sheet )
151         count = chord_sheet . count( chord )
152         correct  =  total_chords  -  count
153         score = 100 - correct
154         chord_correct = ( count/ len ( chord_sheet ))*100
155         chord_correct = round ( chord_correct ,2)
156         chord_correctness = str( chord_correct )
```

```
157         score_str = str( score )
158
159         final_str = " Your Accuracy of Chord " + chord +" is " + chord_correctness + " and Score
                is " + score_str
160
161          return final_str , chord_correctness , score
```

5.6.4 Chord Detection

```
1            import pyrebase
2
3 ''' For Retrieving the chordhseets '''
4
5 def get_chordsheet ():
6         global config
7         config  = { ' apiKey
8         ': "",
9       *
            auth Domain ': "",
10
11      *
            projectId ': "",
12      *
            storage Bucket ':   "", '
13
            messagingSenderId ': " 1 ", ' appId
14          ': "",
15      *
            measurementId ': "",
16
17      *
            databaseURL ': ""
18        }
19        firebase = pyrebase . initialize_app ( config ) db = firebase .
20        database ()
21        items = db. child (' Chord Sheets '). order_by_key (). limit_to_last (5). get (). val () item_homepage = list( reversed (
          list( items. values ())))
          return  item_homepage
22
23 def  get_all_chordsheet ():
24          global config
25          config = {
26          ' apiKey ': "",
27          ' auth Domain ': "",
28          ' projectId ': "",
29          ' storage Bucket ': "",
30          ' messagingSenderId ': "",
31          ' appId ': "",
32          ' measurementId ': "",
33          ' databaseURL ': ""
34          }
35          firebase = pyrebase . initialize_app ( config )
36          db = firebase . database ()
37          items = db. child (' Chord Sheets '). order_by_key (). get (). val ()
38          item_homepage = list( reversed ( list( items. values ())))
39          return  item_homepage
42          global config
40
43          config = {
41 def get_realtime ():
```

```
44        *     apiKey ': "",
45        *     auth Domain ': "",
46        *     projectId ': " e",
47        *     storage Bucket ':  "", '
48              messagingSenderId ': " 1 ", ' appId
49              ': "",
50        *     measurementId ': "",
51        *     databaseURL ': ""
52          }
53          firebase = pyrebase . initialize_app ( config ) db = firebase . database
54          ()
55          items = db. child (' Realtime_Chordsheets '). order_by_key (). get (). val () item_homepage = list(
56          reversed ( list( items. values ())))
57          return  item_homepage
58            global config
59            config = {
60  def practice_chordsheet ( link):
61            ' apiKey ': "",
62            ' auth Domain ': "",
63            ' projectId ': "",
64            ' storage Bucket ': "",
65            ' messagingSenderId ': "",
66            ' appId ': "",
67            ' measurementId ': "",
68            ' databaseURL ': ""
69            }
70            firebase = pyrebase . initialize_app ( config )
71            db = firebase . database ()
72            items  =  db. child (' Chord Sheets '). order_by_key (). limit_to_last (5). get (). val ()
73            item_homepage = list( reversed ( list( items. values ())))
74            return  item_homepage
75
```

5.7 Testing and Results

Before integrating the model to the system, we tested it using the real-time inputs. The video feed was captured from the webcam, and algorithm gave the following output. To Display this on the web app we have used Flask. Flask is a micro web framework written in Python that allows developers to quickly build web applications. To check whether the Output that we got is correct or not, we have also performed the model using performance evaluation metrics. The metrics are Accuracy, Precision, Recall, F1 score and Confusion matrix.

Accuracy:

This measures the percentage of correctly classified images out of the total number of images in the test set.

$$Accuracy = \frac{number\ of\ correctly\ classified\ images}{total\ number\ of\ images}$$

Precision:

This measures the proportion of true positive predictions among all positive predictions.

$$Precision = \frac{true\text{-}positives}{true\text{-}positives + false\text{-}positives}$$

Recall:
This measures the proportion of true positive predictions among all actual positive samples.

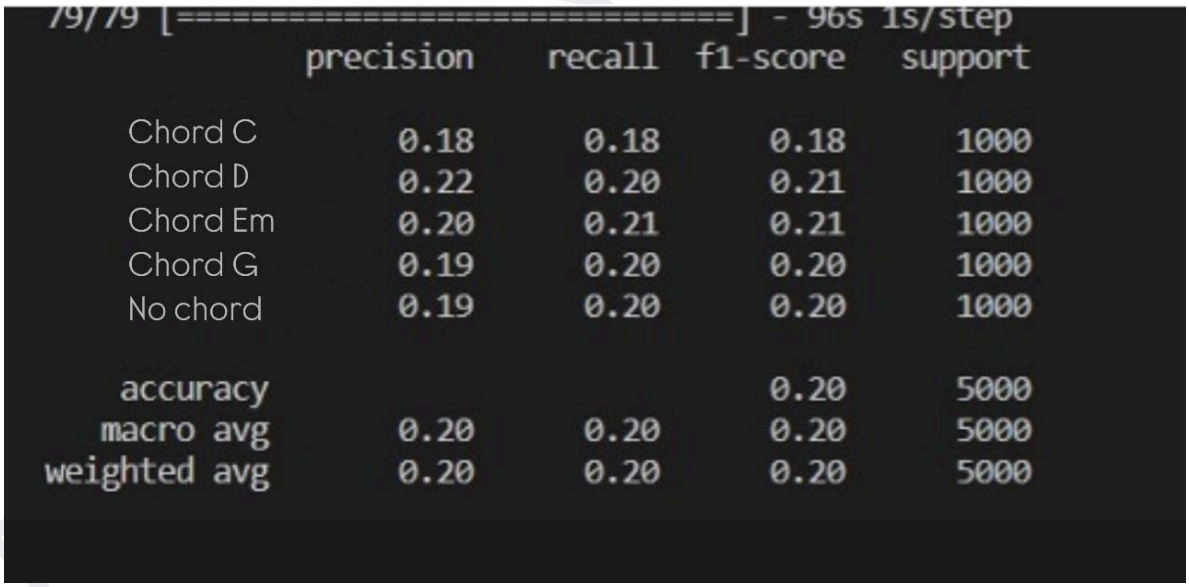
$$Recall = \frac{true\text{-}positives}{true\text{-}positives + false\text{-}negatives}$$

F1 Score:
This is a weighted average of precision and recall, which gives equal importance to both metrics.

$$F1Score = \frac{2 \times (precision \times recall)}{precision + recall}$$

Recall:
This is a table that summarizes the performance of a multiclass classification model by showing the number of correct and incorrect predictions for each class

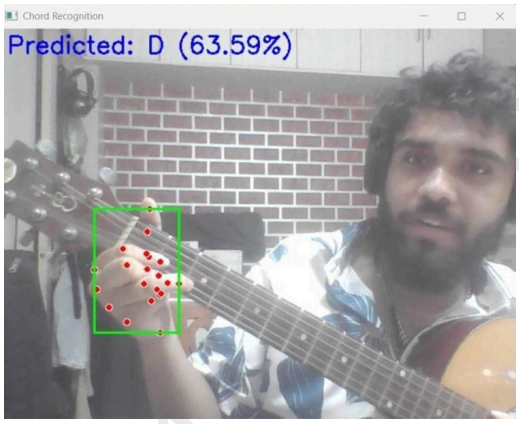
5.7.1 Confusion Matrix



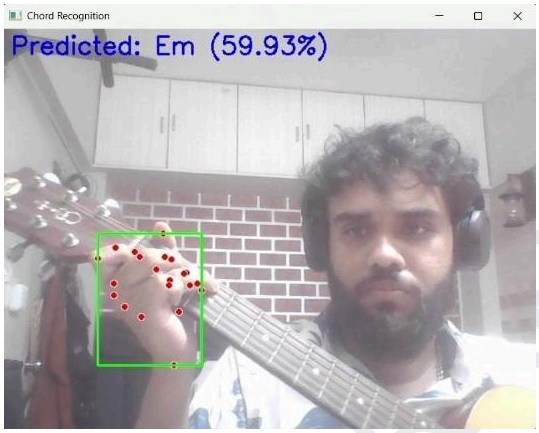
SE Academy 2024 - 2025

5.8 Accuracy

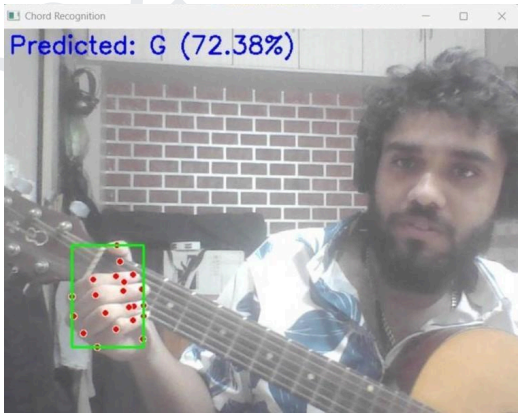
Chord C Accuracy



Chord D Accuracy



Chord Em Accuracy



Chord G Accuracy

Figure 5.4: Accuracy % of all chords

SFIT LIBRARY 2024

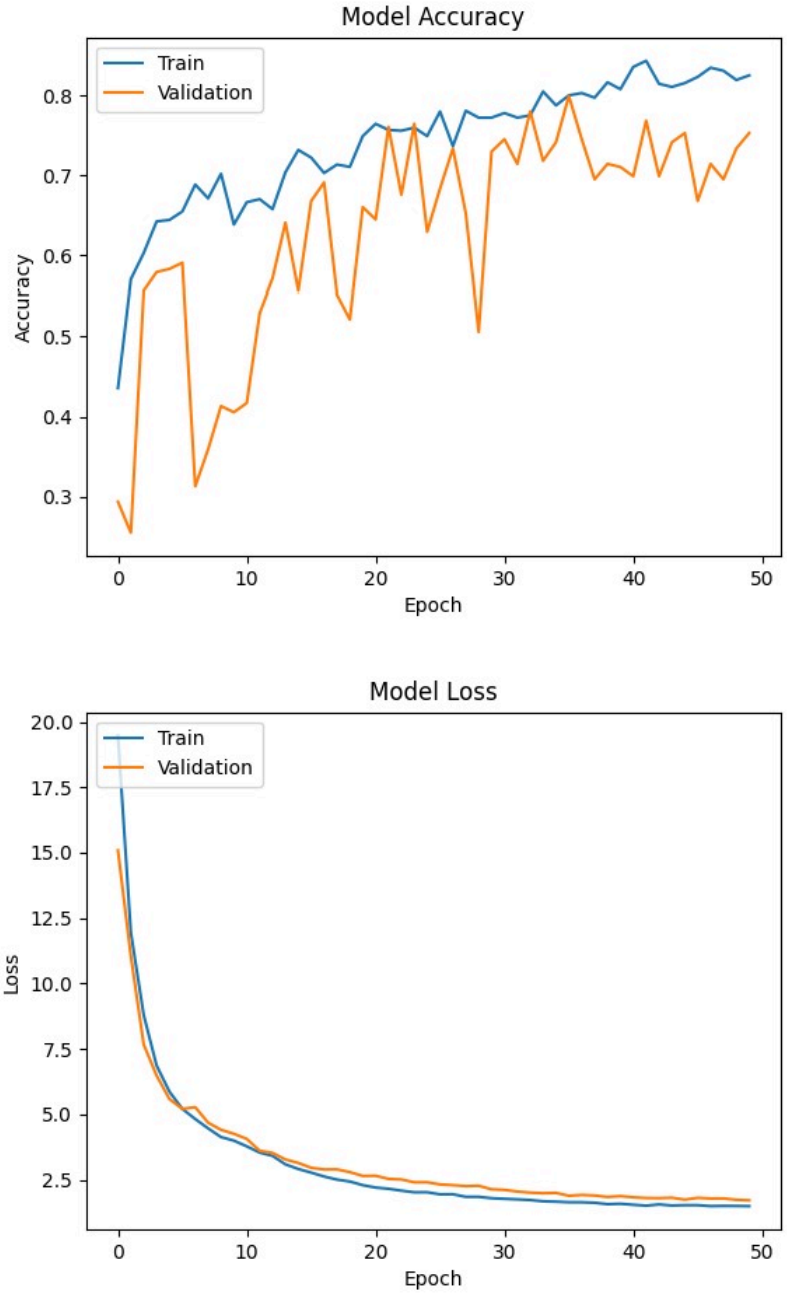


Figure 5.5: Graphs of Model Accuracy and Loss

5.8.1 Model Accuracy

1. X-axis: The horizontal axis represents the Epoch.
    - What is an Epoch? An epoch is one complete pass through the entire training dataset. During one epoch, the model sees every training example once.
    - Unit: The unit for the x-axis is simply the number of epochs, which is a count of the

complete passes through the training data.

2. Y-axis: The vertical axis represents the Accuracy.
    - What is Accuracy? Accuracy is a metric used to evaluate the performance of a classification model. It represents the proportion of correctly classified instances out of the total number of instances. In simpler terms, it tells you how often the model's predictions match the actual correct labels.
    - Unit: Accuracy is typically expressed as a percentage (from 0% to 100%) or a decimal (from 0.0 to 1.0). In this graph, it's shown as a decimal.
3. Lines:
    - Blue Line (Train): This line shows the accuracy of the model on the training dataset as the training progresses through the epochs. You can observe how well the model is learning the patterns in the data it's being trained on.
    - Orange Line (Validation): This line shows the accuracy of the model on a separate valida- tion dataset that the model does not see during training. The validation accuracy provides an estimate of how well the model is likely to generalize to new, unseen data.

5.8.2 Model Loss

1. X-axis: Similar to the top graph, the horizontal axis represents the Epoch. The definition and unit remain the same.
2. Y-axis: The vertical axis represents the Loss
    - What is Loss? Loss is a measure of how poorly the model's predictions deviate from the actual target values. It's a value that the model tries to minimize during the train- ing process. Different machine learning tasks and model architectures use different loss functions (e.g., binary cross-entropy for binary classification, categorical cross-entropy for multi-class classification, mean squared error for regression).
    - Unit: The unit of loss depends on the specific loss function being used. Without knowing the exact problem and model type, we cannot definitively say the unit of the loss in this graph. It could be a value without a specific real-world unit, representing a calculated error magnitude.
3. Lines:
    - Blue Line (Train): This line shows how the loss on the training dataset decreases as the model learns over the epochs. Ideally, the training loss should decrease.
    - Orange Line (Validation): This line shows how the loss on the validation dataset changes over the epochs. The validation loss is crucial for detecting overfitting. If the validation loss starts to increase while the training loss continues to decrease, it might indicate that the model is starting to memorize the training data rather than learning generalizable patterns.

SFIT LIBRARY 2024 - 2025

Chapter 6

Design

6.1 Architecture design

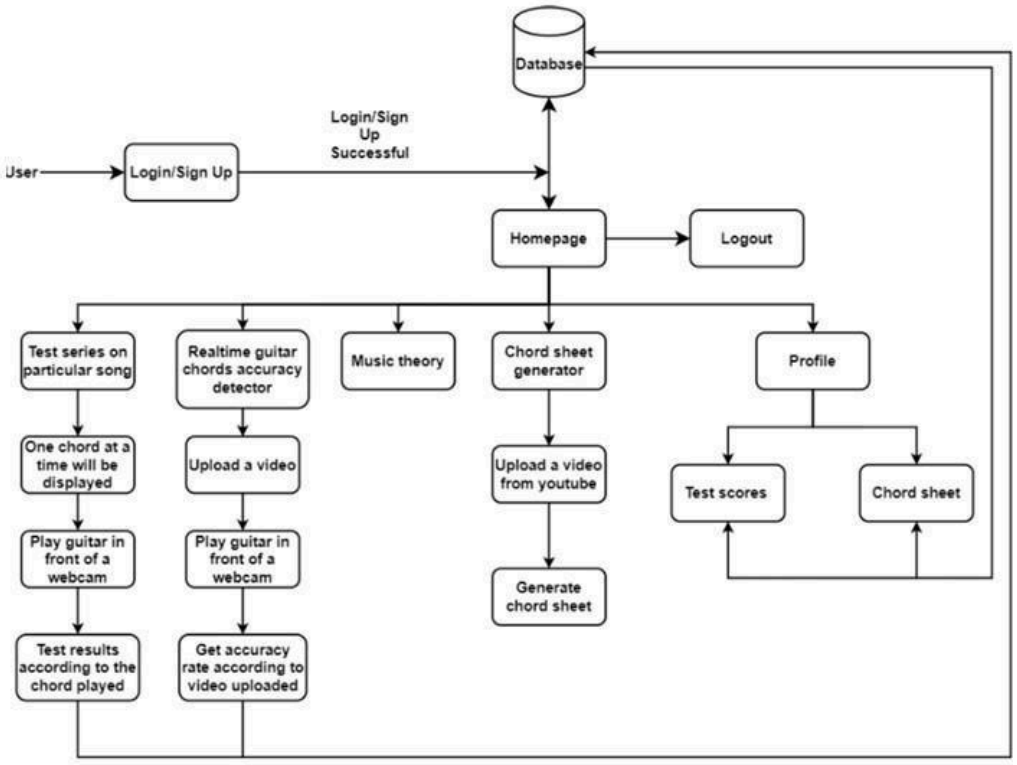


Figure 6.1: Architecture of the system

This is the block diagram of the system, which includes user authentication, chord detection, practice modules, and test series. The system begins with user authentication, where users sign up and their details are stored in the database. When logging in, the credentials are verified, and upon successful authentication, access is granted. The homepage provides navigation to various modules, including the Chord Sheet Generator, Practice Section, Test Series, Music Theory, and Profile Section. The Chord Sheet Generator allows users to upload

a video file or record a real-time video to generate a chord sheet, which can be saved for future reference and stored in the database. The Practice Section enables real-time chord detection, where users can select a pre-saved chord sheet or generate a new one, then play the song in front of the webcam to receive an accuracy score based on the reference chord sheet. The Test Series module presents one chord at a time, prompting the user to play it in real-time, with the system analyzing accuracy and storing scores in the database. The Music Theory Section provides fundamental knowledge on chords, scales, and other basics for beginners. In the Pro- file Section, users can access their saved chord sheets and test scores, allowing them to track progress. This structured workflow ensures a seamless experience for learning, practicing, and improving guitar-playing skills with Chord Analyzer.

6.2 User Interface design

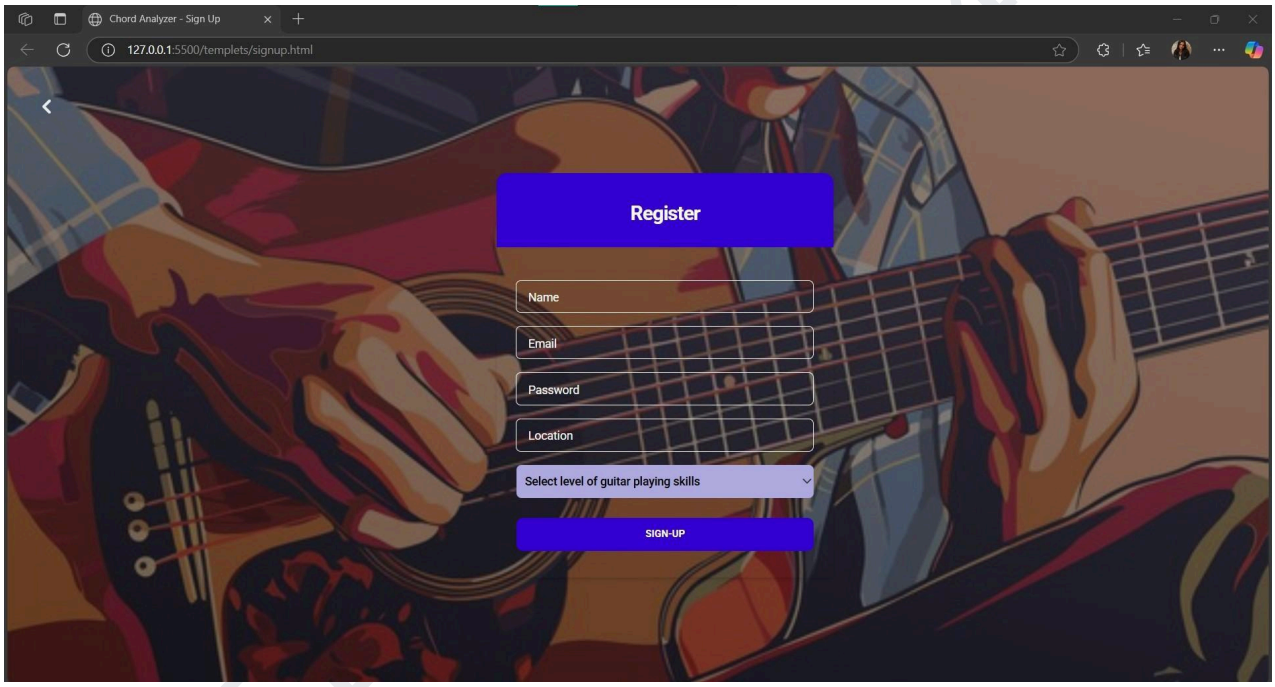


Figure 6.2: Signup page

SFIT - 2025

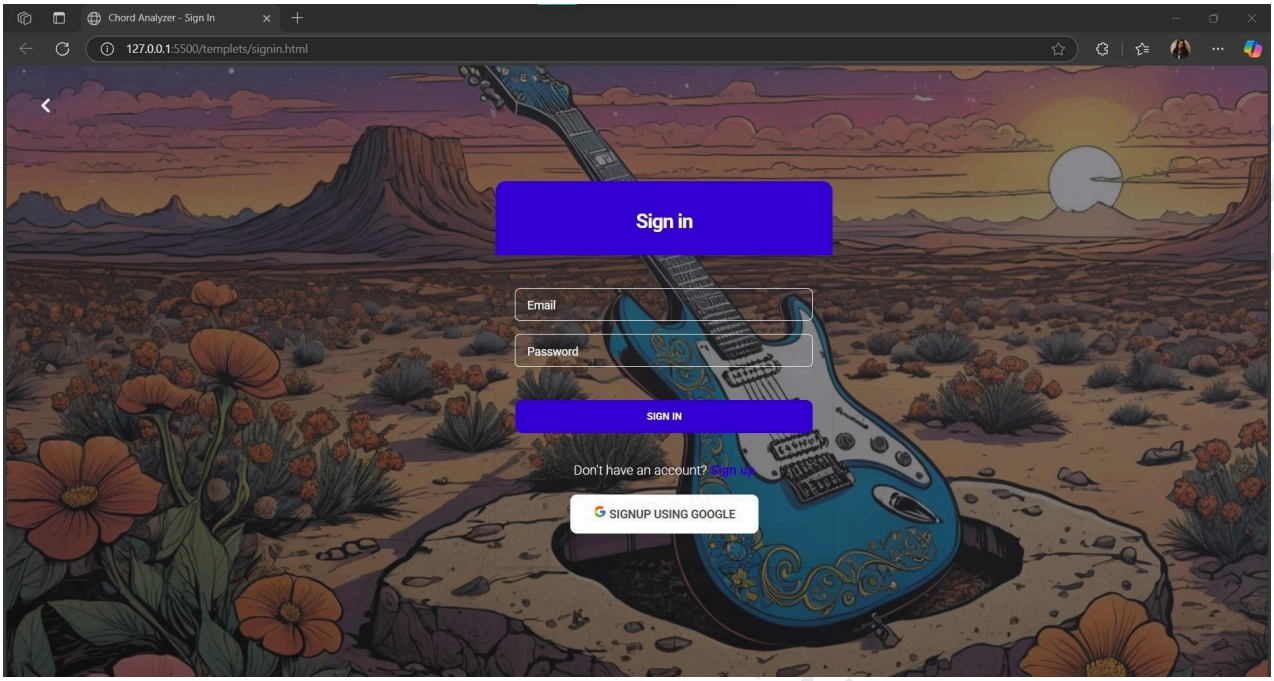


Figure 6.3: Signin page

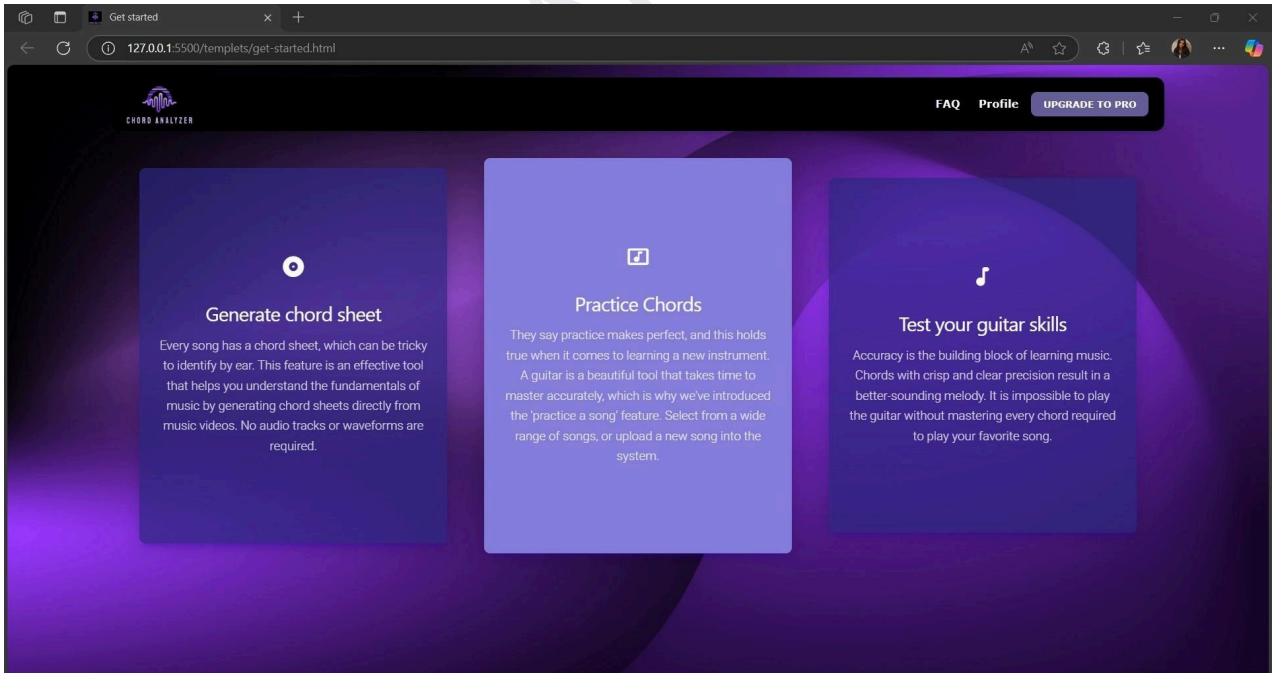


Figure 6.4: Getstarted page

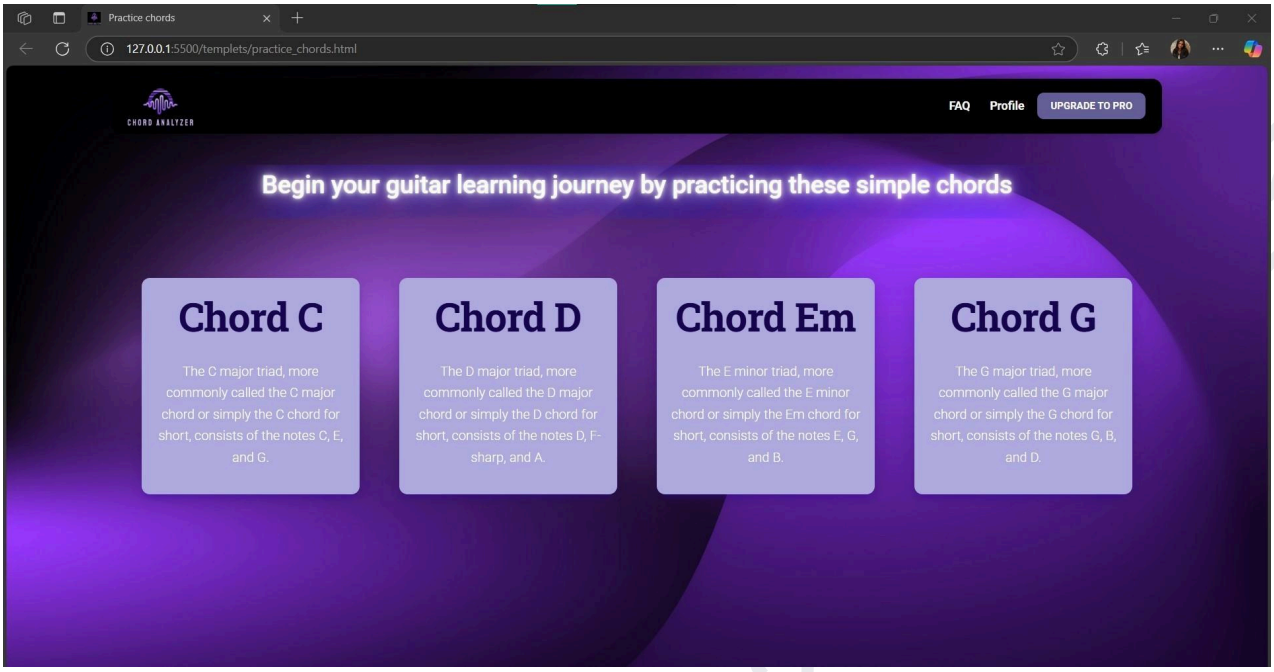


Figure 6.5: Practice chords

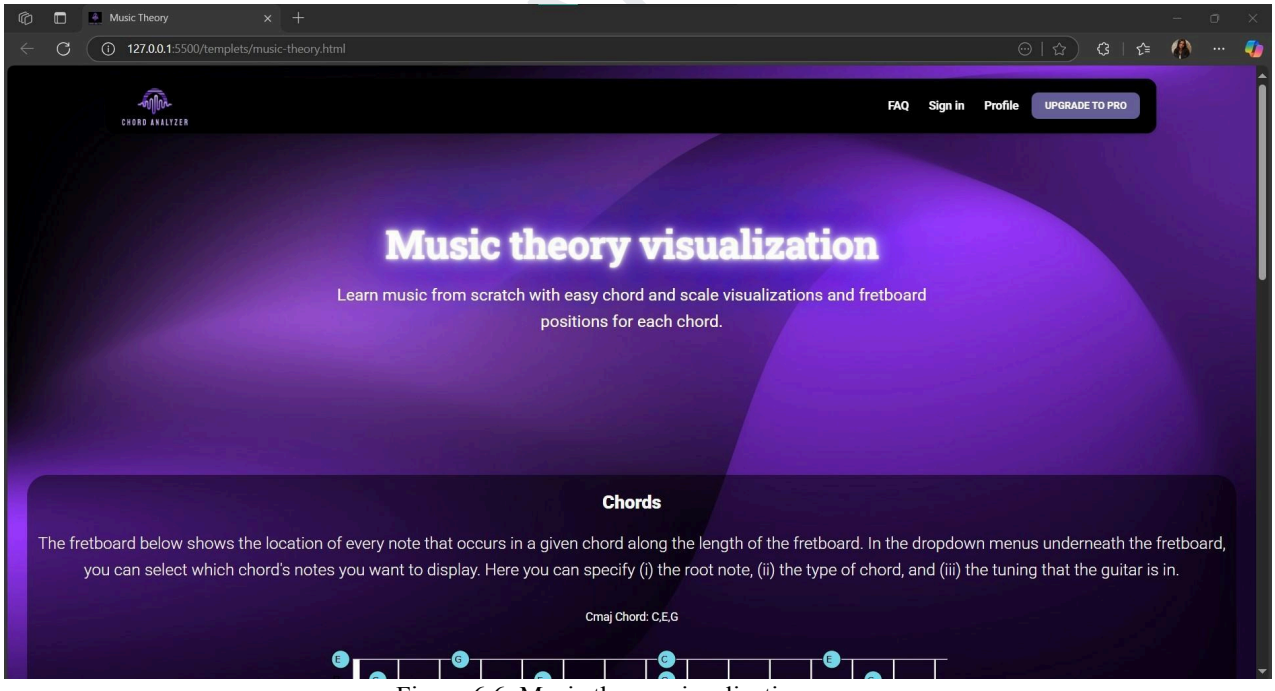


Figure 6.6: Music theory vizualization page

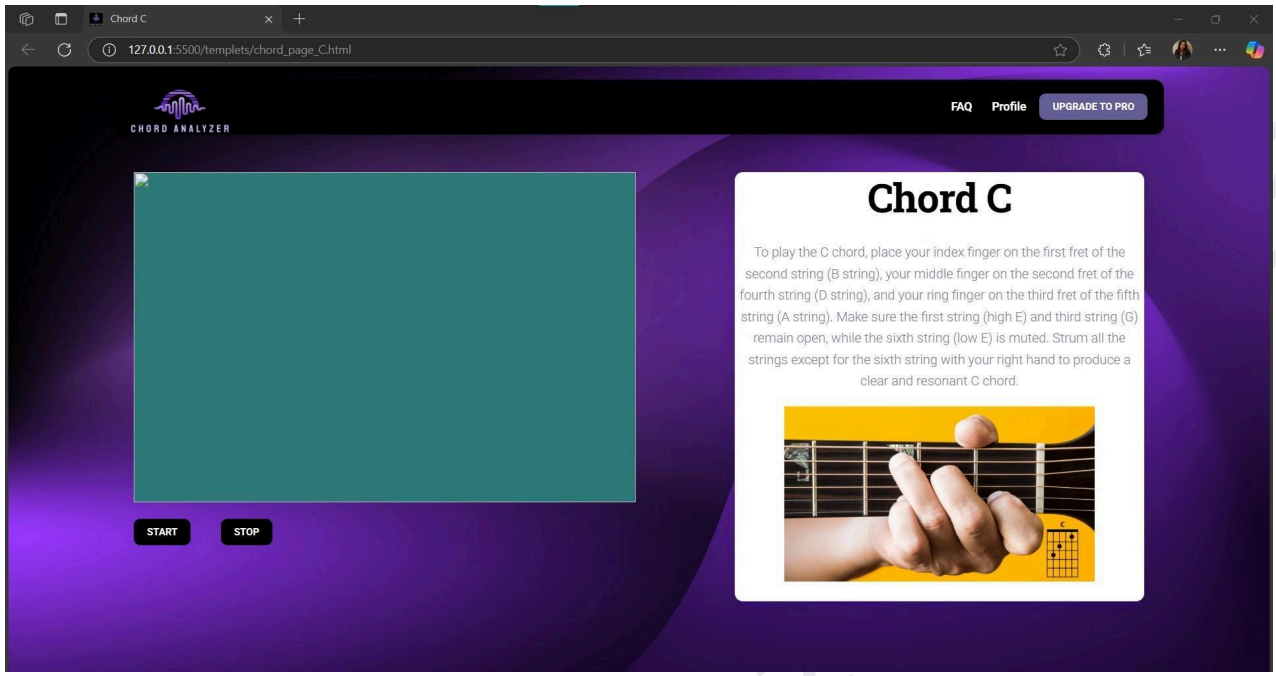


Figure 6.7: Chord C page

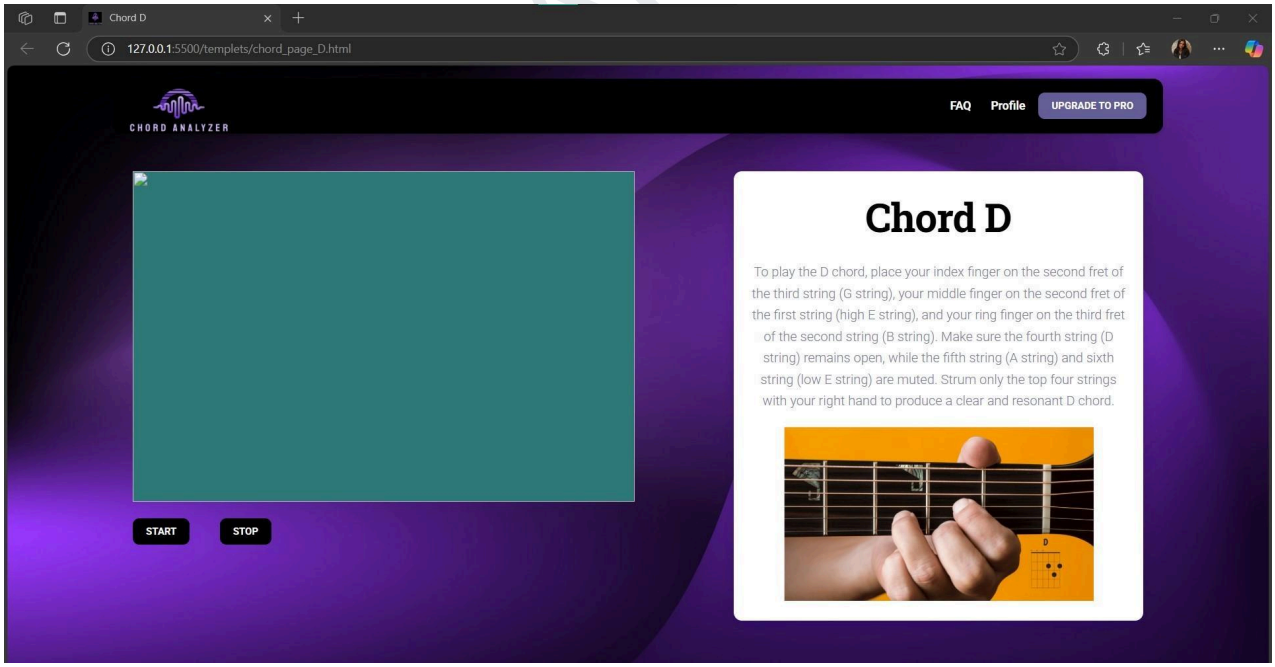


Figure 6.8: Chord D page

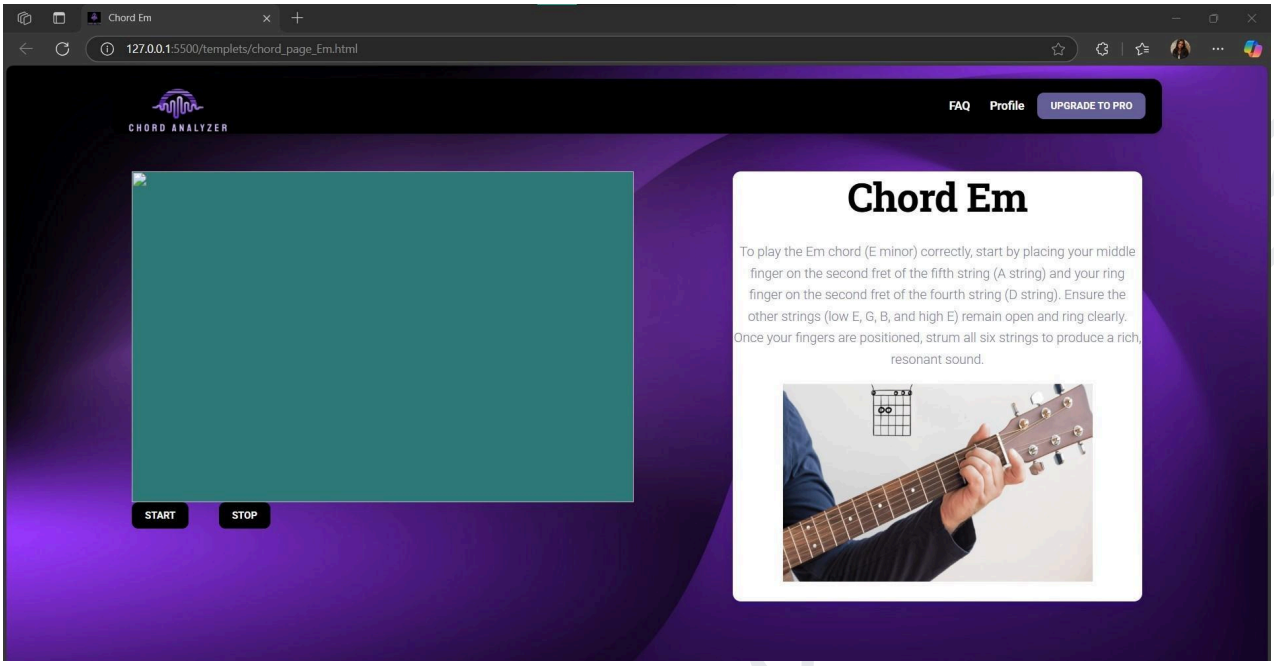


Figure 6.9: Chord Em page

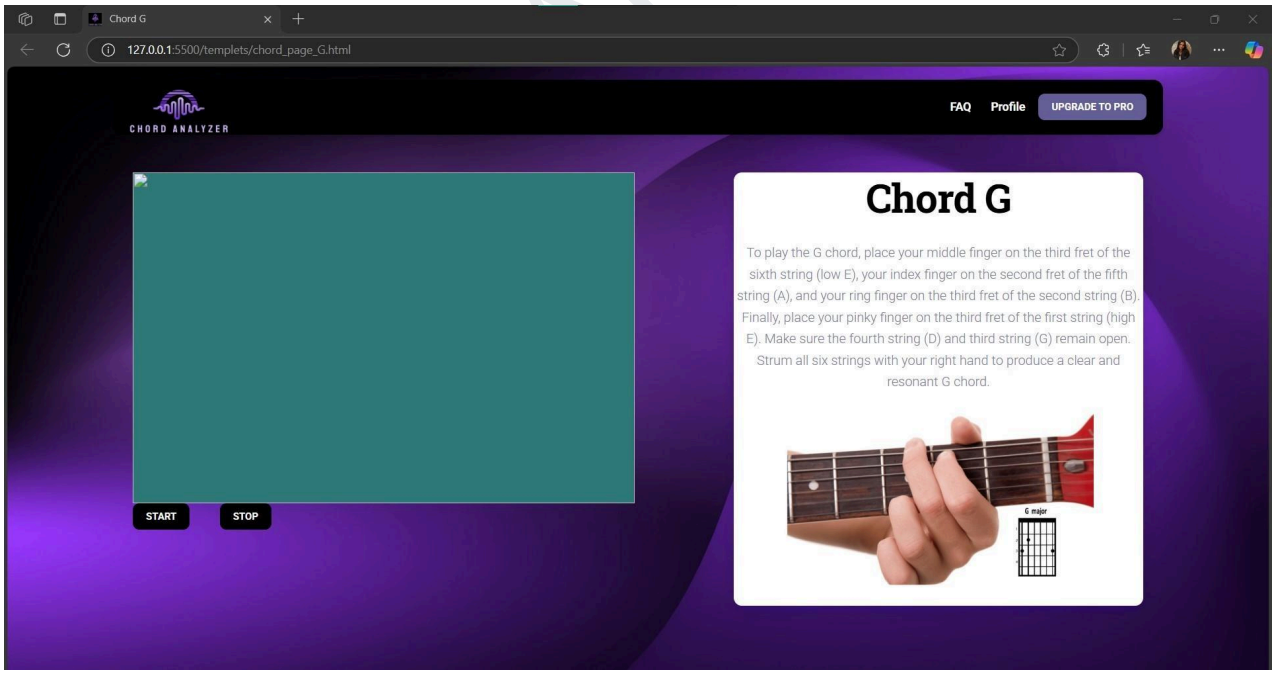


Figure 6.10: Chord G page

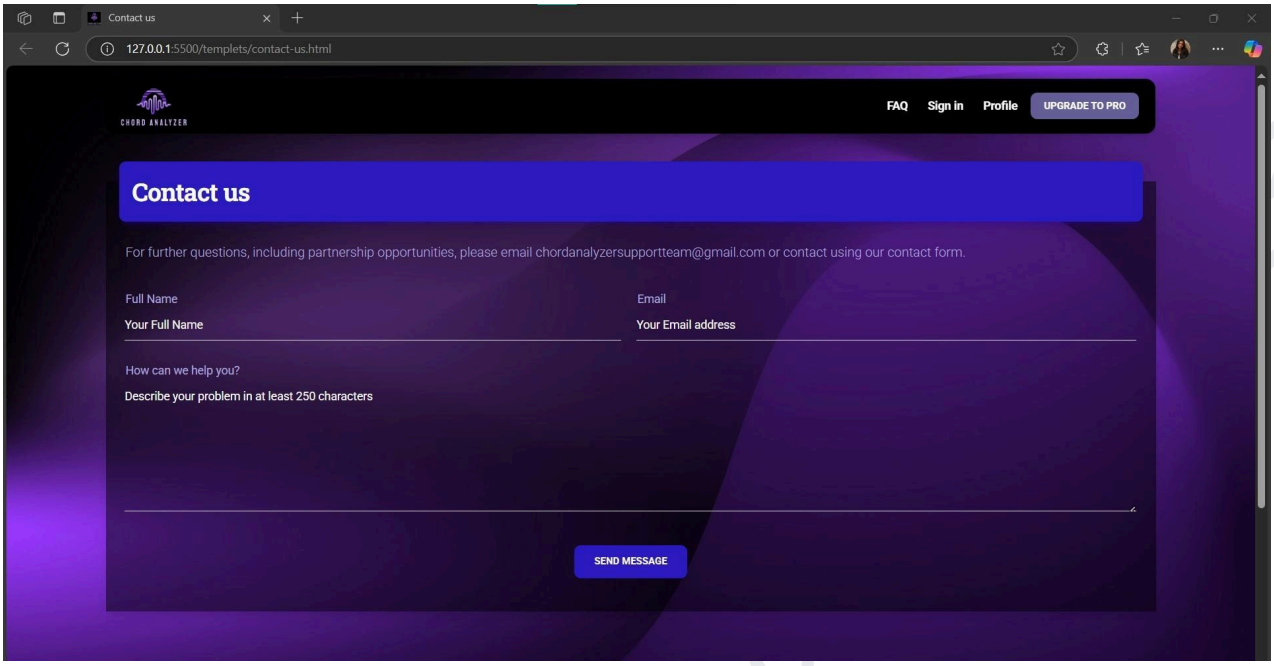


Figure 6.11: Contact us page

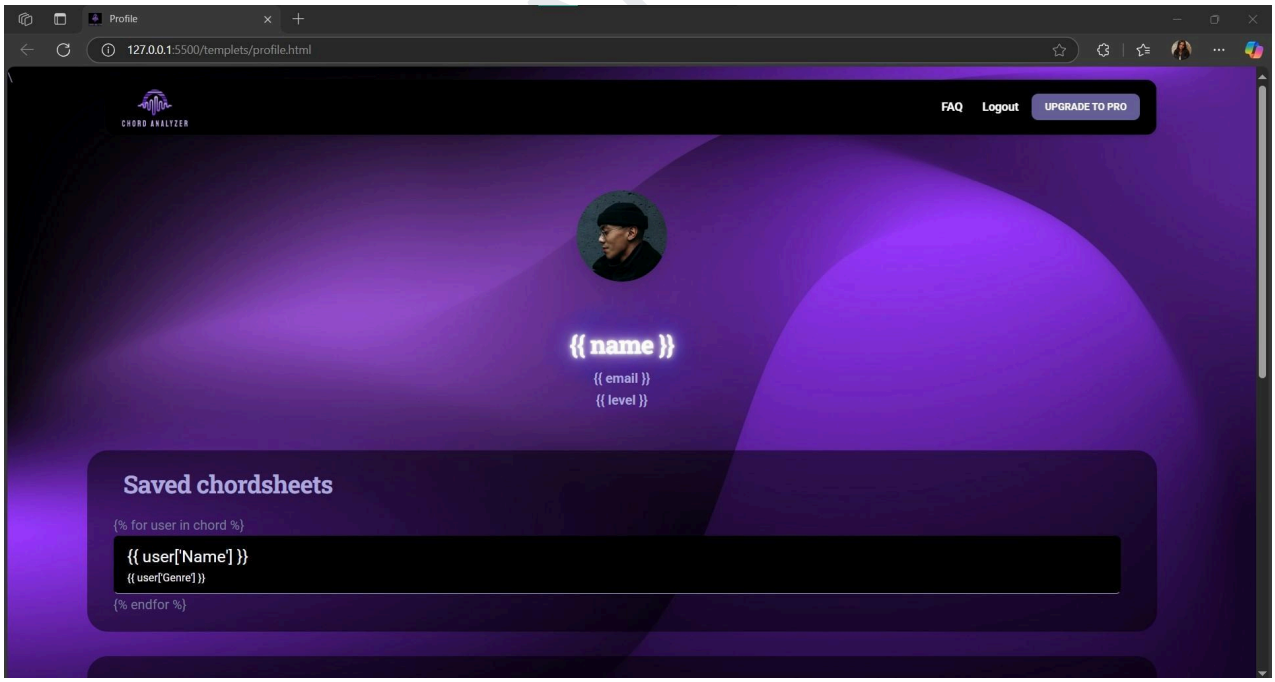


Figure 6.12: User profile page

Table 6.1: Overview of User Interface Pages

Fig	Title	Description
6.1	Architecture of the System	Block diagram showing system components like user authentication, chord detection, practice modules, and database interactions.
6.2	Signup Page	UI for user registration where new users input their details to create an account.
6.3	Signin Page	Login page for existing users to access the app by entering credentials.
6.4	Getstarted Page	A main navigation page offering access to chord sheets, practice, tests, and music theory.
6.5	Practice Chords	Real-time chord practice page where users play guitar in front of a webcam and get accuracy feedback.
6.6	Music Theory Visualization Page	Educational page displaying basics of music theory like chords and scales for beginners.
6.7	Chord C Page	Displays guidance and live detection for playing the C chord with correct finger placement.
6.8	Chord D Page	Similar to the C chord page but focused on learning and detecting the D chord.
6.9	Chord Em Page	Offers visual and detection-based help for practicing the E minor (Em) chord.
6.10	Chord G Page	Helps users identify and play the G chord using webcam-based chord recognition.
6.11	Contact Us Page	A communication page where users can reach out to the developers for support or feedback.
6.12	User Profile Page	Displays user data including saved chord sheets and test performance for tracking progress.

SFIT LIBRARY 2024 - 2025

Chapter 7

Conclusion and future scope

7.1 Conclusion

The guitar is a highly popular instrument, and many people aspire to learn it. However, learning to play can be challenging, and there are limited platforms available to assist beginners. To address this issue, we have developed Chord Analyzer, a system designed to help aspiring musicians. Chord pattern recognition has always been a challenge, especially for those who want to play the correct chords for a song from an online music video or cover. This is where Chord Analyzer proves to be a valuable tool, making the learning process easier without requiring formal classes, online courses, or personal coaching. The system encourages users to select and learn songs of their choice, eliminating the need to go through unnecessary information. While working on this project, we have explored various technologies, including OpenCV and Keras, to implement chord pattern recognition and machine learning models. Our application is designed to be user-friendly and convenient, offering real-time assistance in every possible way. The system is capable of detecting four primary chords: C, D, Em, and G, making it an effective tool for beginners and intermediate players alike.

7.2 Future scope

The aim is to make Chord Analyzer a universal application for all guitarists and musicians to learn and master the art of playing the guitar. To achieve this, we plan to increase the number of identifiable chords, allowing musicians to play a wider range of genres, including Jazz and Rock. Expanding the system's capabilities will make it more versatile and applicable to diverse musical styles. Once this goal is accomplished, we aim to further enhance the application by incorporating support for other instruments such as the keyboard, drums, and more. This add-on feature will provide a comprehensive learning experience, making Chord Analyzer a complete music education platform for aspiring musicians.

SFIT LIBRARY 2024 - 2025

Chapter 8

Literature cited

[1] John Emad, Ahmed Seraga, Karim Khaleda, Ahmed Yehia, Karim Mohamed, Hager Sobeaha, and Walaa Hassana,"Note Frequency Recognition and Finger Motion Capture of a Guitarist: A Survey and Challenges Ahead" International Journal of Computing and Communication, Cairo, Egypt 2022.

[2] Tran, L., Zhang, S., Zhou, E. (2019). "CNN Transfer Learning for Visual Guitar Chord Classification." 30th Conference on Neural Information Processing Systems, Barcelona, Spain 2016.

[3] Ooaku, T., Linh, T. D., Arai, M., Maekawa, T., Mizutani, K. (2018). "Guitar chord recognition based on finger patterns with deep learning." 4th International Conference on Communication and Information Processing 2018.

[4] Kavana KM and Suma NR, "RECOGNIZATION OF HAND GESTURES USING MEDIAPIPE HANDS" International Research Journal of Modernization in Engineering Technology and Science 2022.

[5] Ketan Gomase, Akshata Dhanawade, Prasad Gurav, Sandesh Lokare, and Dr. Jyoti Dange, "Hand Gesture Identification using Mediapipe" International Research Journal of Engineering and Technology (IRJET) 2022.

[6] Niranjani, V., Keerthana, R., Mohana Priya, B., Nekalya, K., Padmanabhan, A. K. "System application control based on Hand gesture using Deep learning" 7th International Conference on Advanced Computing and Communication 2021.

[7] Castillo, J. R., Flores, M. J. (2021), "Web-Based Music Genre Classification for Timeline Song Visualisation and Analysis." IEEE Access 2021.

[8] B., Abhishek Krishi, Kanya M., Meghana Daaniyaal, Mohammed S., Anupama. "Hand gesture recognition using machine learning algorithm" Computer Science and Information Technologies 2020.

SET LIBRARY 2024-2025

Acknowledgment

We are extremely grateful to our college St. Francis Institute of Technology for the confidence bestowed in us and entrusting out project entitled ”Chord Analyzer: Real time chord detection sys- tem”. We express our sincere gratitude to our respected director Bro. Shantilal Kujur, our principal Dr. Sincy George and our HOD Dr. Prachi Raut for encouragement and facilities provided to us. We owe our profound gratitude to our project guide Dr. Mrinmoyee Mukherjee, who introduced us to the methodology of work and timely completion of various stages of our project. Many people especially all team members and classmates have made valuable comments on this proposal which gave us inspiration to improve our assignment.

SFIT LIBRARY 2024 - 2025