Dissertation on

"RepRight - AI Based Gym Form Correction"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE20CS461A – Capstone Project Phase - 2

Submitted by:

Moulya Shetty	**PES2UG20CS204**
Nameeta Kuruwatti	**PES2UG20CS211**
Prajay V K	**PES2UG20CS245**
Nandita Pydikondala	**PES2UG20CS258**

Under the guidance of

Dr. Sandesh B J
Chairperson, Department of
Computer Science and Engineering
PES University

June - Nov 2023

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

RepRight - AI Based Gym Form Correction

is a bonafide work carried out by

Moulya Shetty	**PES2UG20CS204**
Nameeta Kuruwatti	**PES2UG20CS211**
Prajay V K	**PES2UG20CS245**
Nandita Pydikondala	**PES2UG20CS258**

In partial fulfilment for the completion of seventh semester Capstone Project Phase - 2 (UE20CS461A) in the Program of Study -Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period June 2023 – Nov. 2023. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th semester academic requirements in respect of project work.

Signature Dr. Sandesh B J Chairperson	Signature Dr. Sandesh B J Chairperson	Signature Dr. B K Keshavan Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. _____	_____
2. _____	_____

DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled **"RepRight - AI Assisted Gym Form Correction"** has been carried out by us under the guidance of Dr. Sandesh B J, Chairperson and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester June – Nov. 2023. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES2UG20CS204	**Moulya Shetty**
PES2UG20CS211	**Nameeta Kuruwatti**
PES2UG20CS245	**Prajay V K**
PES2UG20CS258	**Nandita Pydikondala**

ACKNOWLEDGEMENT

I would like to express my gratitude to Dr. Sandesh B J, Chairperson, Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE20CS461A - Capstone Project Phase – 2. I am grateful to the Capstone Project Coordinator, Dr. Sarasvathi V, Professor and Dr. Sudeepa Roy Dey, Associate Professor, for organizing, managing, and helping with the entire process. I take this opportunity to thank Dr. Sandesh B J, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support I have received from the department. I would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help. I am deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES University and Prof. Nagarjuna Sadineni, Pro-Vice Chancellor - PES University, for providing to me various opportunities and enlightenment every step of the way. Finally, this project could not have been completed without the continual support and encouragement I have received from my family and friends.

ABSTRACT

In response to the burgeoning interest in a healthier lifestyle, our project focuses on the growing gym culture, specifically in the realm of weight training. There is a prevalent issue of individuals risking injury due to improper form or failing to optimize their workout outcomes. Recognizing the constraints posed by the cost and availability of personal trainers, our app harnesses the power of AI to address this problem. By offering real-time feedback to users, our app helps guide them in correcting their exercise forms and derive maximum benefits.

The Gym Posture Recognition project aims to develop a machine learning model that can assess different postures during exercises in a gym or fitness setting. The model can be hosted on an app and a mobile phone camera can be used to capture video input of the user performing the exercise. The machine learning algorithm can then analyze the input and provide feedback on the user's form. The goal of this project is to help users improve their exercise techniques and reduce the risk of injury by providing accurate feedback on their posture based on a large dataset.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

It's 2023 and an increasing number of individuals have been migrating to a healthier lifestyle to counter the side effects of a hectic urban schedule. Gym culture which is at an infancy stage in India at present is growing at a swift pace as increasingly more people are joining the gym movement for its many benefits in leading a longer and healthier life.

The gym industry, especially weight training, is becoming extremely popular due to its many health benefits, i.e, weight loss, increased strength and endurance and building muscle mass. However many individuals either injure themselves due to incorrect form or are unable to maximize the benefits of each exercise.

Having a personal trainer is highly advised to overcome this but it comes at a heavy cost and is unavailable at your convenience. Hence leading to time constraints and a lack of individual attention.

Our app leverages AI to help users correct their form using real-time video as input. The immediate feedback helps users fix their form and allows them to maximize the exercise's benefits. It is highly advised for beginners since our application does it for you at a fraction of the cost as compared to a trainer, with a bonus of it being available at your convenience. An additional feature of rep counting will be implemented in the app to help users keep count of the number of times they perform the exercise

CHAPTER 2

PROBLEM STATEMENT

Gym culture and weight training have become increasingly popular, but can be intimidating for beginners. Personal trainers have traditionally helped people overcome these barriers, but an AI based gym form correction app would offer a new approach to personalized training and guidance.

An AI model will be designed to detect and correct popular exercises that are most likely to be done incorrectly hence acting as a real-time fitness coach. To provide additional convenience, the application uses only a camera to check the user's form and return audio feedback on which part of their form needs correction.

Our app not only helps users in the gym currently but also in bringing in more customers. The number one reason people avoid the gym or training with weights is the fear of injuring themselves. With the help of our app and a simple phone camera, users will be provided with the exact information on how to perform these exercises and instantly fix problems in their form. It is extremely cost-effective and can be available to anybody with a working phone camera.

It also helps with an extremely important concept known as the “mind-muscle connection” in the gym industry. It consists of feeling the target muscles activated while performing an exercise dedicated to that particular muscle group. This not only helps with maximizing the benefits of the exercise but also helps the user understand which muscle group is supposed to be targeted. This is only possible by performing exercises with the correct form.

CHAPTER 3

LITERATURE SURVEY

3.1 Monocular Human Pose Estimation: A survey of deep learning based methods [1]

3.1.1 Abstract

Vision-based monocular human pose estimation, as one of the most fundamental and challenging problems in computer vision, aims to obtain posture of the human body from input images or video sequences. The recent developments of deep learning techniques have brought significant progress and remarkable breakthroughs in the field of human pose estimation. This survey extensively reviews the recent deep learning-based 2D and 3D human pose estimation methods published since 2014. This paper summarizes the challenges, main frameworks, benchmark datasets, evaluation metrics, and performance comparison, and discusses some promising future research directions.

3.1.2 Problems addressed by this paper

Foreground Occlusion: Foreground occlusion in video processing occurs when an object or a person is temporarily blocked causing loss of information.

Solution provided:

Deep learning techniques, such as CNNs, can be trained to recognise and reconstruct occluded regions in videos. By learning to fill in the missing information based on contextual cues and visual patterns, these models can generate realistic and accurate predictions of the occluded regions.

Diverse body appearance including different clothing and self-similar parts

Solution provided:

Data Augmentation: To account for diverse clothing and varying body sizes you can augment the training data by artificially varying the clothes, size and texture of the person.

Multi-modal Fusion: To handle self-similar body parts, you can use multi-modal fusion techniques that combine information from different sensors, such as RGB cameras and depth sensors, to extract more informative features. This can help distinguish between similar-looking body parts and improve pose estimation accuracy.

3.1.3 Limitations

This survey contains various methods for both 2d and 3d HPE (Human Pose Estimation). Since RepRight only focuses on 2D pose estimation there are a few limitations or problems this paper doesn't address

Distance and the angle of the person standing in front of the camera: The survey fails to explain how this issue can be solved with respect to a 2D camera but solves this issue in case of 3d by utilizing depth sensors and multimodal fusion.

Lighting conditions: The quality of the lighting in the video can also affect the accuracy of pose estimation. Poor lighting conditions can create shadows, glare, or reflections that can obscure body parts, making it difficult to estimate their position accurately. The survey does not address any of the issues related to lighting conditions.

3.2 AI-based Workout Assistant and Fitness guide[2]

3.2.1 Abstract

An app called Fitcercise that guides users while performing gym exercises. Counts repetitions and diet recommendations for home workouts as well as in gyms to be used as smart trainers.

Implementation: Uses a combination of blaze pose and MediaPipe to get the 33 key points for pose estimation. This was made possible using a double step teacher pipeline. After this based on the

exercise, the angle formed is calculated and compared with a range to display the efficiency of the posture. For post-processing, the Non-Maximum Suppression algorithm is used.

3.2.2 Advantages

Uses a combination of blazepose and mediapipe. Blazepose is an algorithm meant to extract key points efficiently on a mobile phone GPU which makes it appropriate for an app.

3.2.3 Limitations

BlazePose has currently only implemented the upper half of the body.
The main point that we take away from this paper is to look into BlazePose since our project is also an app. Furthermore, the solution in the paper was inspired by the Stacked Hourglass solution which could be looked into for our solution as well.

3.3 Realtime Indoor Workout Analysis Using[Machine Learning & Computer Vision[3]

3.3.1 Abstract

Implements a four step pipeline

Pose estimation:
Trainer uploads a video of him performing an exercise and form coordinates are extracted using a Deep Learning CNN model. These are the optimal coordinates for the correct form of the exercise
Model consists of:

- First 10 layers of VGG19 net to obtain a fixed-size vector of given image
- Followed by two multi-step branches of CNNs
- First branch is used for predicting confidence maps for body point locations represented by dots

- Second branch predicts Vector Field maps which deduces the association between points of the body
- Optical Flow Tracking :
- Real time input of user performing exercise is taken
- Body points/coordinates are extracted for every 5th frame to reduce latency while giving immediate feedback
- Intermediate frames are obtained using optical flow tracking
- Lucas-Kanade algorithm implemented in OpenCV which ensures consistency between these intermediate frames
- Dynamic Time Warping:
- DTW is an algorithm for comparing or aligning a pair of time series sequences
- The user's video and the trainer's videos are passed through this algorithm
- DTW is used to sync user and trainer videos and obtain frame pair mapping to be compared

Affine Transformations:

To handle the following problems -

- User and trainer might have different body ratios
- Angle of user's camera might be different
- Distance between camera and user might vary

Thus, transform the user points to trainer reference frame and overlay the skeleton on the trainer's frame

3.3.2 Advantages

The system is able to detect minute errors in limb positions which could be critical in many exercises. It can handle users with different body ratios easily. It can also handle different camera angles.

3.3.3 Limitations

The CNN model used was trained on the COCO dataset for human pose estimation. Thus, it is not fine-tuned on a dataset targeted towards exercises.

3.4 Automatic recognition and assessment of physical exercises from RGB images[4]

3.4.1 Abstract

This paper implements Google MediaPipe and is used for pose estimation from RGB images. The DD-Net deep architecture is used for action recognition. Two frame-based and sequence-based scores are estimated. DTW algorithm is used to calculate these scores.

3.4.2 Advantages Built their own dataset that is catered towards physical exercise hence giving accurate results

3.4.2 Limitations

Fails to handle different camera angles and different distances between user and camera

3.5 Pose Estimation and Correcting Exercise Posture [5]

3.5.1 Abstract

The paper proposes a form estimation and correction application that analyzes a recorded video input and provides feedback accordingly.

This approach consists of two phases: posture detection and feedback generation.

It uses OpenPose for the classification and localisation of key joint points.OpenPose provides the output of key points in a frame-by-frame text in JSON format. Python uses pandas data frames to

read the JSON files. Further, the frame rate is used to calculate the velocity. For detecting correct or incorrect forms based on the calculated DTW distances, a nearest neighbour classifier is developed.

3.5.2 Advantages

The user can take the video from any angle or distance while using this approach. It also provides a graphical simulation to highlight the mistakes of the user.

3.5.3 Limitations

The user must record a video as the application is not real-time. After a comparative analysis of OpenPose and Mediapipe, Mediapipe is better suited to our project as it considers the key joint positions in three dimensions, which is vital to correctly assess the form of the user.

3.6 Real-Time Human Pose Recognition in Parts from Single Depth Images [6]

3.6.1 Abstract

The paper proposes using a depth camera for real time pose recognition.

It quickly and accurately predicts 3D positions of body joints from a single depth image, using no temporal information. The approach uses an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. A large and highly varied training dataset allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. Finally, confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes is generated.

The results show a high correlation between real and synthetic data and between the intermediate classification and the final joint proposal accuracy.

Key takeaways:

Mocap Data - discard many similar, redundant poses from the initial mocap data using ‘furthest neighbour’ clustering

Synthetic data -It uses a highly varied synthetic training data set to train very deep decision forests and yet avoid overfitting. A randomized rendering synthesis pipeline first randomly samples a set of parameters, and then uses standard computer graphics techniques to render depth and (see below) body part images. Further slight random variations in height and weight give extra coverage of body shapes. Other randomized parameters include the mocap frame, camera pose, camera noise, clothing and hairstyle.

Body part labelling - A key contribution of this work is the intermediate body part representation. The intermediate representation transforms the problem into one that can readily be solved by efficient classification algorithms at a low penalty.

3.6.2 Advantages

The results are highly accurate and have commercial applications(Kinect camera)

Uses a highly varied synthetic dataset to overcome overfitting of data.

3.6.3 Limitations

Must have depth cameras and can not use a phone (2D) camera. The classifier does not use temporal information

CHAPTER 4

PROJECT REQUIREMENTS SPECIFICATION DOCUMENT

4.1 Project Scope

The AI model will be designed to detect and correct popular exercises, acting as a real-time fitness coach. Immediate feedback will be provided for users to correct their form while performing the exercise. As an added feature, the number of times the user performs the exercise is kept track of.

Our app not only helps users in the gym currently but also in bringing in more customers. The number one reason people avoid the gym or training with weights is the fear of injuring themselves. With the help of our app and a simple phone camera, users will be provided with the exact information on how to perform these exercises and instantly fix problems in their form. It is extremely cost-effective and can be available to anybody with a working phone camera.
It also helps with an extremely important concept known as the “mind-muscle connection” in the gym industry. It consists of feeling the target muscles activated while performing an exercise dedicated to that particular muscle group. This not only helps with maximizing the benefits of the exercise but also helps the user understand which muscle group is supposed to be targeted which is only possible when the exercise is done correctly.

4.2 Project Perspective

RepRight is an innovative application designed to help individuals maintain proper form when exercising. With the growing trend towards a fit lifestyle, RepRight offers a unique solution to an often-overlooked aspect of fitness - correct posture and alignment during exercise.

The application leverages computer vision models to track the user's movements and provide real-time feedback on their form. This includes cues for adjusting posture, suggestions for correcting technique, and warnings for potential injury. By using RepRight, users can be confident

they are performing exercises safely and effectively, ultimately leading to better results and fewer injuries.

RepRight is particularly useful for individuals who are new to exercise or who are returning to fitness after a long break. As an Android application, RepRight offers accessibility and convenience to a large user base. Its user-friendly interface makes it a valuable addition to your fitness routine whether you are a beginner or a professional athlete.

4.3 Product Features

Real-time Feedback:

RepRight tracks users' movements and provides them with instant feedback on their exercise form. This includes cues for adjusting posture or suggestions for correcting technique. This can help users correct their exercise posture as they are performing the exercise. This leads to a better understanding of the mistakes and a greater chance of correcting them easily.

Personalized Experience:

Regardless of the user's goals or body type, RepRight can provide accurate and helpful feedback on their exercise form.

Exercise Tracking:

RepRight keeps track of the number of times the user performs the exercise at a time. This helps them keep track of the count as it will be displayed on the app's video screen.

User-friendly Interface:

RepRight's is a simple app with a user-friendly interface, making it accessible to users of all levels of experience.

4.4 User Classes And Characteristics

Fitness Enthusiast:

These users are dedicated to their fitness goals and exercise frequently. They are likely to use RepRight regularly to maintain proper form and prevent injury during workouts. They may be experienced with using fitness technology and are comfortable with advanced features and settings.

Beginners:

These users are new to exercise or returning to fitness after a long break. They may require more guidance and support in learning proper exercise techniques and form. RepRight's feedback makes it an ideal tool for helping beginners achieve their fitness goals safely and without harm of injury.

Casual Exercisers:

These users exercise infrequently and may be less experienced with fitness technology. They may use RepRight occasionally to ensure proper form during workouts and prevent injury. RepRight's user-friendly interface makes it accessible to casual exercisers.

Physical Therapy Patients:

These users may be recovering from an injury or illness and require specific exercises to aid in their recovery. RepRight's personalized feedback and customizable settings make it an ideal tool for physical therapy patients to ensure they are performing exercises safely and effectively.

Fitness Professionals:

These users may be personal trainers or fitness instructors who use RepRight to monitor their clients' form during workouts. They can also be used by fitness professionals online that may not be able to see their client’s form in person.

Elderly Users:

These users may require simpler interfaces and larger fonts due to age-related limitations. RepRight can be customized to meet the needs of elderly users to ensure they are performing exercises safely and effectively.

4.5 Operating Environment

RepRight is an Android app that uses Android OS version 5.0 and higher. Users require a mobile with a working camera. The app requires access to the user's camera to record and track the user's movements based on which real-time feedback will be displayed.

RepRight uses computer vision and machine learning models, to accurately track the user's movements and provide accurate feedback. The app can be used offline, allowing users to exercise without an internet connection.

4.6 General Constraints And Dependencies

4.6.1 Regulatory Policies

The app may need to comply with regulatory policies related to data privacy, security, and medical devices. These policies could limit the types of data the app can collect.

4.6.2 Hardware Limitations

The app's real-time feedback and detection of the person’s key body points require a mobile device with a camera. The users may need to consider the hardware limitations of their mobile device when using the app.

4.6.3 Safety And Security Considerations

The app protects and secures user information. No images or videos of the user will be recorded or stored.

4.6.4 Criticality Of The Application

The app's core functionality, which involves providing effective feedback on exercise form.

4.7 Risks

4.7.1 Resource Constraints

Developing a complex application like RepRight may require significant resources in terms of time, dataset availability, and expertise. The project may be at risk of delays or cost overruns if resource constraints are not tracked and managed. Datasets of videos could require high memory requirements and large GPU processing power.

4.7.2 Technical Constraints

Developing a real-time motion-tracking system that provides accurate feedback on exercise form requires sophisticated algorithms and machine learning models. Technical challenges may arise during development that require additional resources or expertise to overcome.

4.7.3 User Adoption

For RepRight to succeed, it hinges on users finding it worthwhile. It is crucial to concentrate on creating user-friendly interfaces that cater to the target users' needs in order to increase user adoption.

4.7.4 Security And Privacy

RepRight will require access to the user's cameras. We will need to ensure the app is secure and follows data privacy regulations to avoid legal liability.

4.7.5 Compatibility Issues

RepRight will need to be compatible with a range of Android devices and operating systems. Compatibility issues may arise during development that require additional resources or changes to the app's functionality

4.8 Functional Requirements

The main functionality of this product is to correct a user's exercise form in the gym using input from their phone camera. Feedback will be generated on how the user's form can be improved. Count of the number of times the user is performing the exercise will also be tracked and displayed. The flow of input to output:

- User opens the app and camera permissions are granted
- The camera is then opened and waits for the user to get in place
- The user can start performing his exercise
- Video input is processed automatically and frames are extracted
- The frames are passed through MediaPipe to extract key points
- The lowest point's frame key points are used to calculate required angles
- Angles are passed to our random forest model to predict form on the backend
- Output is obtained and displayed on the app's screen, i.e, "Too high", "Correct", etc
- The user can then use the feedback to fix his form
- The new exercise form is then taken and same steps are followed
- Count of the number of times user performs exercise is also kept track

Counting the number of times the user performs the exercise at a time is known as our rep counter. A rep is one movement of the exercise performed and a fixed number of reps is usually performed in a set. Providing a counter will help the user solely focus on their form and can stop once our counter reaches their desired count.

Possible errors:

- The camera might not be placed in an optimal position. Thus, the video input passed through the AI model will not be able to give accurate feedback.
- The amount of light or brightness could vary between users. Low lighting situations could lead to information being left out and thus leading to inaccurate results.

4.9 External Interface Requirements

4.9.1 User Interfaces

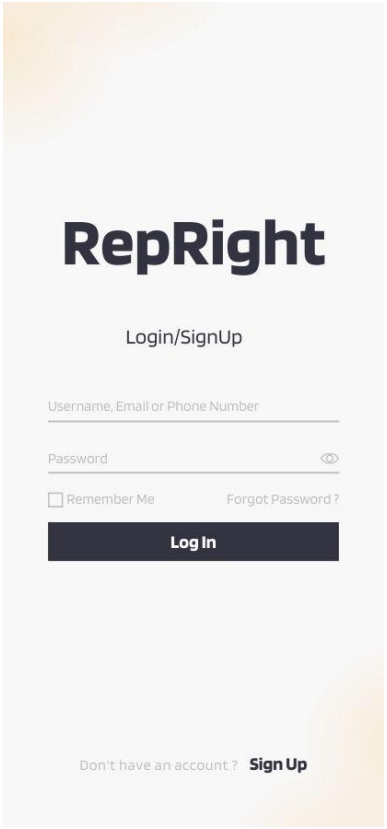


Fig 4.0: Login Page

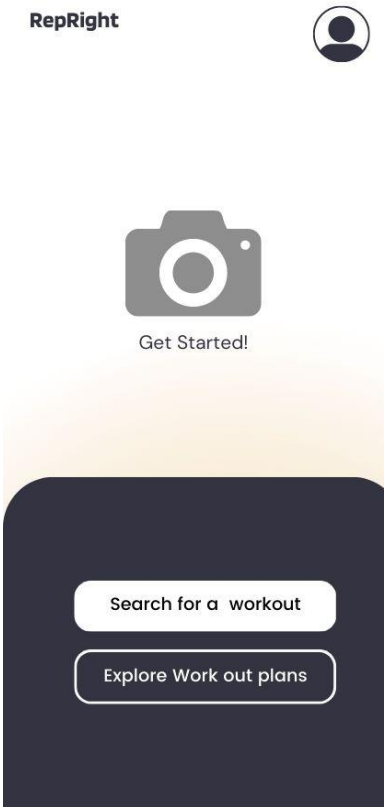


Fig 4.1: Home Page

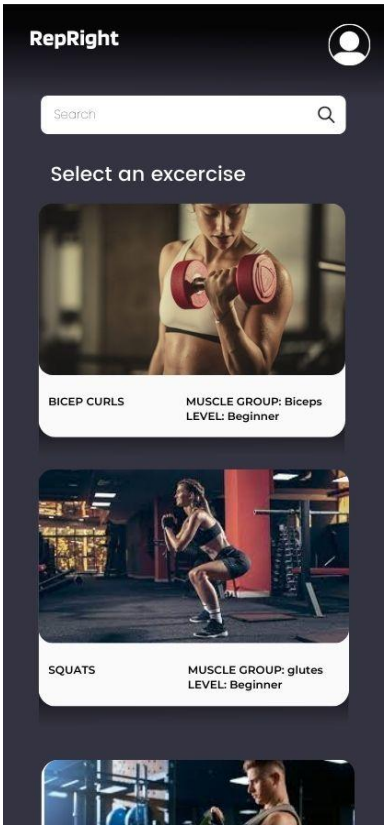


Fig 4.2: Exercise Catalog

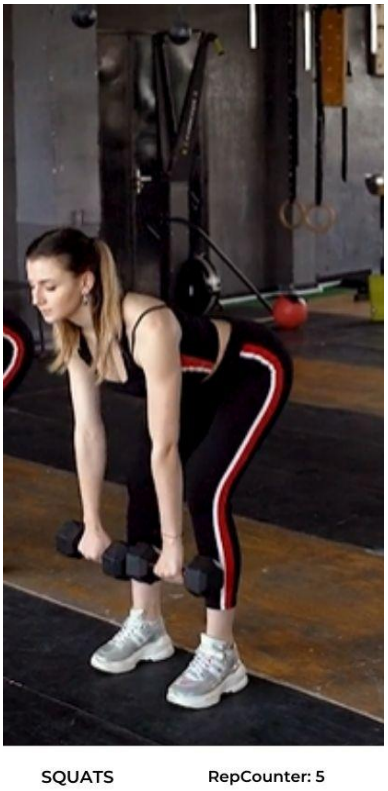


Fig 4.3: Form Checking and Rep Counting

As the app is meant to give real-time feedback there should be almost negligible delay between the video input and output displayed on the app.

4.9.2 Hardware Requirements

- Phone with a working camera
- Android version 5 or above

4.9.3 Software Requirements

- Python
- TensorFlow

- Keras
- Android Studio

4.9.4 Communication Interfaces

- Ngrok
- HTTPS

4.10 Non-Functional Requirements

4.10.1 Performance Requirements

Accuracy: The app should be able to give accurate and reliable feedback to the user's posture since the safety of the user depends on it. Wrong information would destroy the purpose of the app.

Real-time speeds: Since the app is meant to work in real time the system must be able to analyze the user's posture quickly and provide feedback immediately.

Adaptability and Usability: The app should be able to work accurately on any kind of user, irrespective of their build/height. It should be easy to use by all age demographics.

Availability: The app will be available at minimal cost.

Reliability: The instructions given by the app will be reliable. This will be made possible with testing and frequent customer reviews.

4.10.2 Safety Requirements

Privacy and security: The user's data should be protected which the app will abide by. User privacy should be maintained.

Limits: The app only helps you correct your form. The user will have to gauge what weight their body will be able to handle.

4.10.3 Security Requirements

The app should ensure the user's privacy at all times. Since the user is live streaming their video, they would expect maximum security of their data. The system should be designed with data protection and privacy features.

4.11 Other Requirements

Scalability: As the popularity of our app grows we expect more customers which could reduce the speed and efficiency of our results. Hence making it necessary to scale the app accordingly.

Compatibility: The app should be compatible with different devices and operating systems. It should work seamlessly on any platform irrespective of the device.

Compatibility with exercise equipment: RepRight will be compatible with most exercise equipment used by the user and should provide accurate feedback and correction.

Maintainability: Users should be able to provide feedback/ reviews which will be responded to in 24 hours.

CHAPTER 5

HIGH LEVEL DESIGN DOCUMENT

5.1 Current System

RepRight is a new development aimed at creating a gym posture recognition system from scratch. The project uses computer vision and machine learning techniques to analyze a user's exercise posture in real-time and provide feedback to the user. The user can use this feedback to correct their exercise posture.

We aim to improve upon existing methods of providing feedback on exercise posture. While some gyms and fitness centres offer personal trainers to provide guidance on posture, this can be expensive and may not be easily available to everyone. Other systems that exist to analyze posture are not as effective or affordable as RepRight.

5.2 Design Considerations

5.2.1 Design Goals

The primary goal of the Repright design is to create a gym posture recognition system that can analyze a user's exercise posture in real-time and provide corrective feedback to the user. To achieve this goal, the design of the system needs to follow:

Accuracy: The system must be highly accurate in the feedback given after detecting and analyzing the user's posture.
Performance: The system must process the user's input in real-time and provide immediate feedback.
User-friendly Interface: The system should have a user-friendly interface that is easy to use.

Portability: The system should be portable and accessible on mobile devices for any user to use.

Privacy: The system must ensure the privacy of the user's data and information.

To achieve these guidelines, the design principles of the RepRight system includes:
Machine Learning: The system uses computer vision and machine learning algorithms to analyze the user's posture accurately and in real-time.
Mobile Application: The system is designed as a mobile application, which is accessible and portable, allowing for easy use by the user.
User Experience Design: The system should provide a smooth user experience so that the user can receive feedback easily.

Current Implementation, their alternatives and their pros and cons-
When designing the Repright system, there were several alternate choices considered for each component. Here are some of the choices and the reasons for selecting the current architecture:

Computer Vision: The main alternative to using MediaPipe for the computer vision component was to use the YoloV7 pose estimation model, a new and upcoming body key point extraction technology. However, MediaPipe was chosen for its superior performance and accuracy in pose estimation, which is critical for accurately analyzing the user's posture.

Pros of using MediaPipe[7]:

- Real-time performance
- Integration with TensorFlow for machine learning
- Better performance on low quality images

Cons of using MediaPipe:

- Steep learning curve for implementation
- Occluded bodies were not as accurate as YoloV7

Pros of using YoloV7 pose estimation model[8]:

- Occluded bodies could be detected entirely
- Detect multiple people in a frame

- Low light conditions do not affect performance

Cons of using YoloV7 pose estimation model:

- Did not perform well on low quality images
- People in the background were detected which ruined our use case

Overall, the chosen architecture is using MediaPipe for body key point extraction, computer vision and TensorFlow for machine learning. This meets the system's requirements for real-time performance, accuracy, and scalability. While there were some cons to each choice, the benefits outweighed them for our system.

5.2.2 Architecture Choices

RepRight architecture's main focus is to achieve real-time performance and good accuracy. This has been achieved through the below components of the architecture

Mobile App: First and foremost is the interface used to interact with the user. This has been made possible by developing a mobile app. The mobile app records real-time video and displays feedback and rep count.

Computer Vision: Pose estimation is done using computer vision. Pose estimation involves an algorithm to extract key points of the user from the video and use these to identify correct and incorrect posture. This is done using MediaPipe or YOLOv7.

Machine Learning: The machine learning component is the main part of the backend where the user posture is analyzed and provides feedback. A neural network was trained for this with a dataset of correct and incorrect postures to classify appropriately. The machine learning component of the system is responsible for analyzing the user's posture and providing feedback. It uses a neural network trained on a dataset of correct and incorrect posture examples to classify the user's posture and provide feedback.

Feedback: The feedback system involves information on how to correct the posture and where the user is going wrong.

5.2.3 Constraints, Assumptions and Dependencies

Constraints and Limitations:

Maintenance: The application server should be updated with frequent model changes to ensure that there is an improvement in the model and should be trained on newer data frequently.

Scalability: If the number of users increases with time, the system may face issues. Therefore, the system must be designed to scale up as more users are added.

Availability: The system must be highly available and reliable. This includes failover mechanisms to ensure that the system is always available even if hardware or software fails.

Other requirements: The system should follow privacy and security regulations, and protect user data at all times.

Dependencies:

Hardware and software environment: The minimum hardware requirement for the Repright application to run is a mobile phone with a camera. It should also have sufficient computing power to run the process without any interruption. The software requirement is an Android operating system and the RepRight application.

Interoperability requirements: The Repright system uses standard communication protocols such as HTTP to transmit data between the client and server. The application assumes that these protocols are available and compatible with the user's mobile device and the server.

Performance related issues: The system requires real-time performance to provide instant feedback to the user. Therefore, the application assumes that the mobile device and server have enough computing power to handle the processing and perform video analysis in real-time.

End-user environment: The system assumes that the user will be in a well-lit environment with sufficient space to perform the exercises. The user should know the basic form of the exercise they are performing.

Availability of resources: The system assumes that there will be sufficient resources available to train and maintain the machine learning model. This includes large datasets of images and keypoint data to train and fine-tune the model.

5.3 Design Description

The app component will be the primary interface with which the user interacts. A real-time video input is taken of the user performing the exercise. The input is passed through MediaPipe to generate key body points which are then processed and passed to a model to predict if the form is correct or not. An output is generated and displayed on the app so the user can correct his mistakes on-spot.

5.3.1 Master Class Diagram

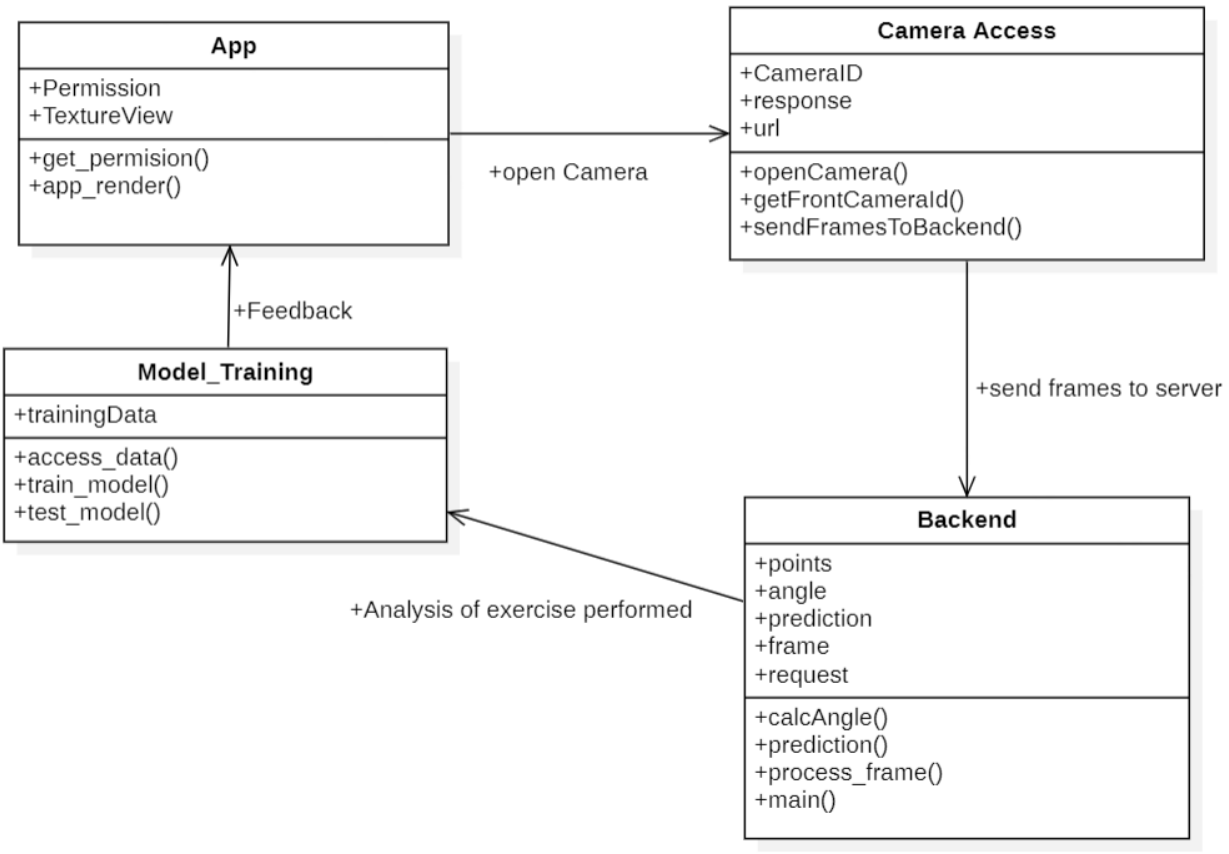


Fig 5.0: Master Class Diagram

5.3.2 Reusability Considerations

The machine learning model will be built to handle training on different exercises thus letting us scale the app to add multiple other exercises. Thus, the only change required would be the dataset of the exercises being passed through the model on training.

The widgets such as buttons and layouts built into the UI will be reused throughout the application.

5.4 Swimlane Diagram

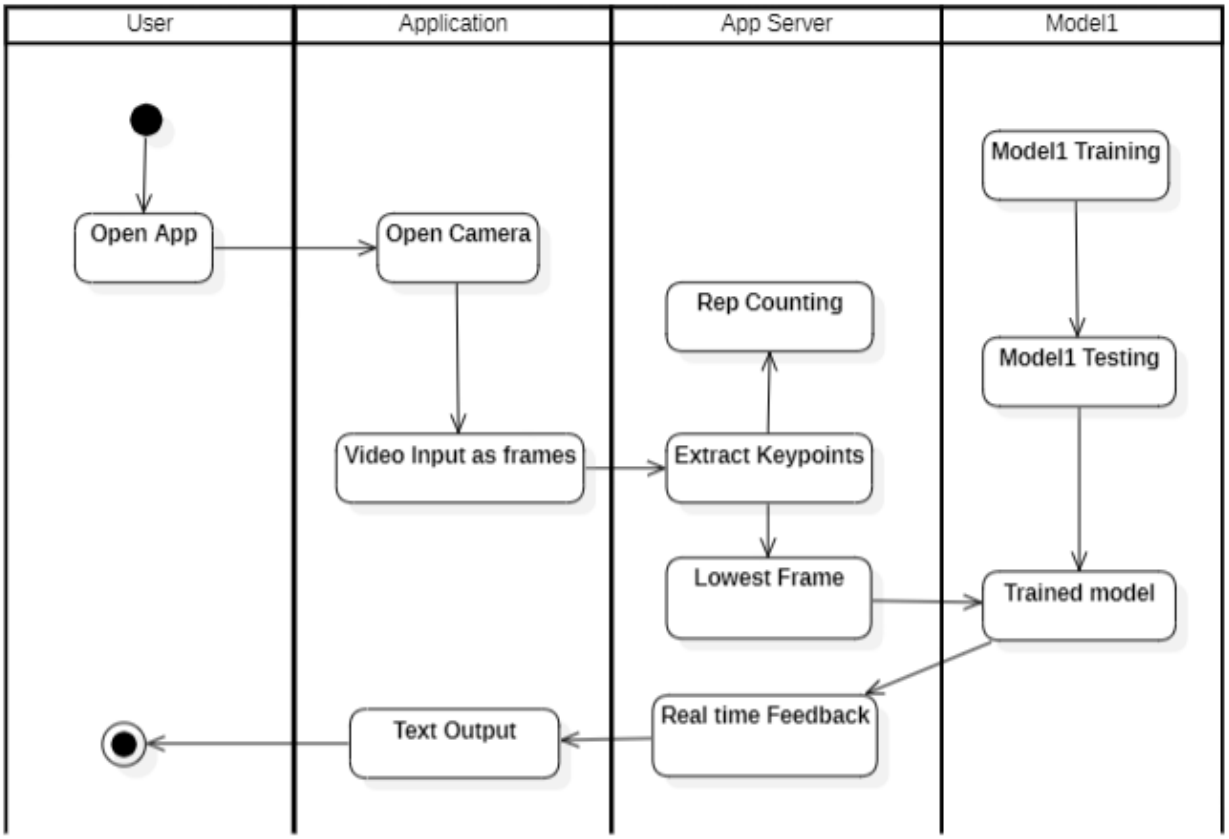


Fig 5.1: Swimlane Diagram

5.5 External Interfaces

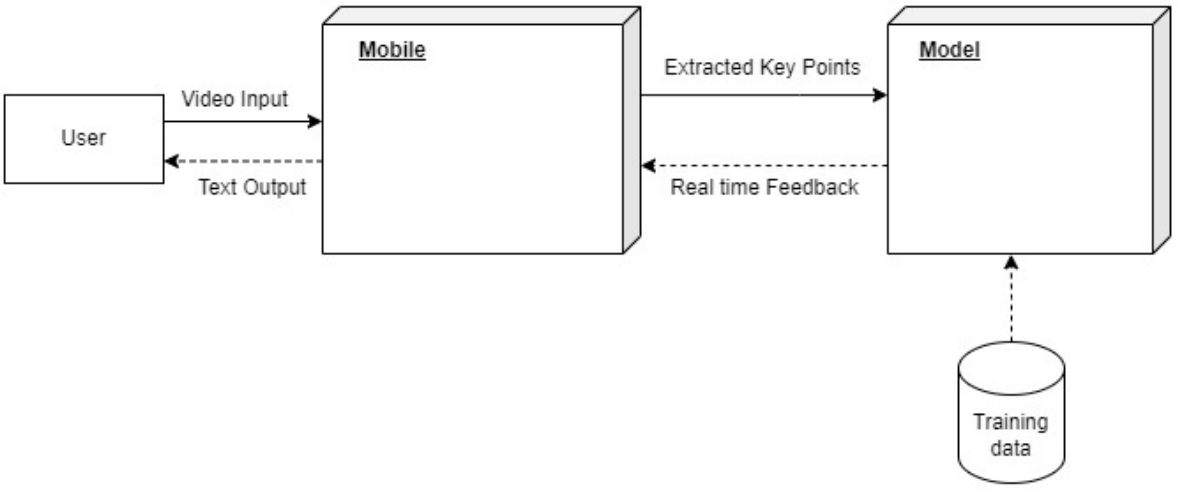


Fig 5.2: External Interfaces Diagram

Users access the app after authentication: The user accessed the app through their smartphone device.

User performs exercise: The user then performs the exercise while the app is capturing real time video of the user's movement through the camera of the device.

Video processing: The captured video would be processed by the app's AI module after extracting key points, which would use a computer vision model to analyze the user's movements and compare them to the ideal form for the exercise. These models are trained on data specific to the exercise.

Feedback generation: Based on the analysis of the user's movements, the model would generate feedback for the user in real-time.

Feedback is given to the user: The feedback generated by the model would be displayed on the app screen

Rep Counting: The number of times the user performs the exercise is also kept track of

5.6 Packaging and Deployment Diagram

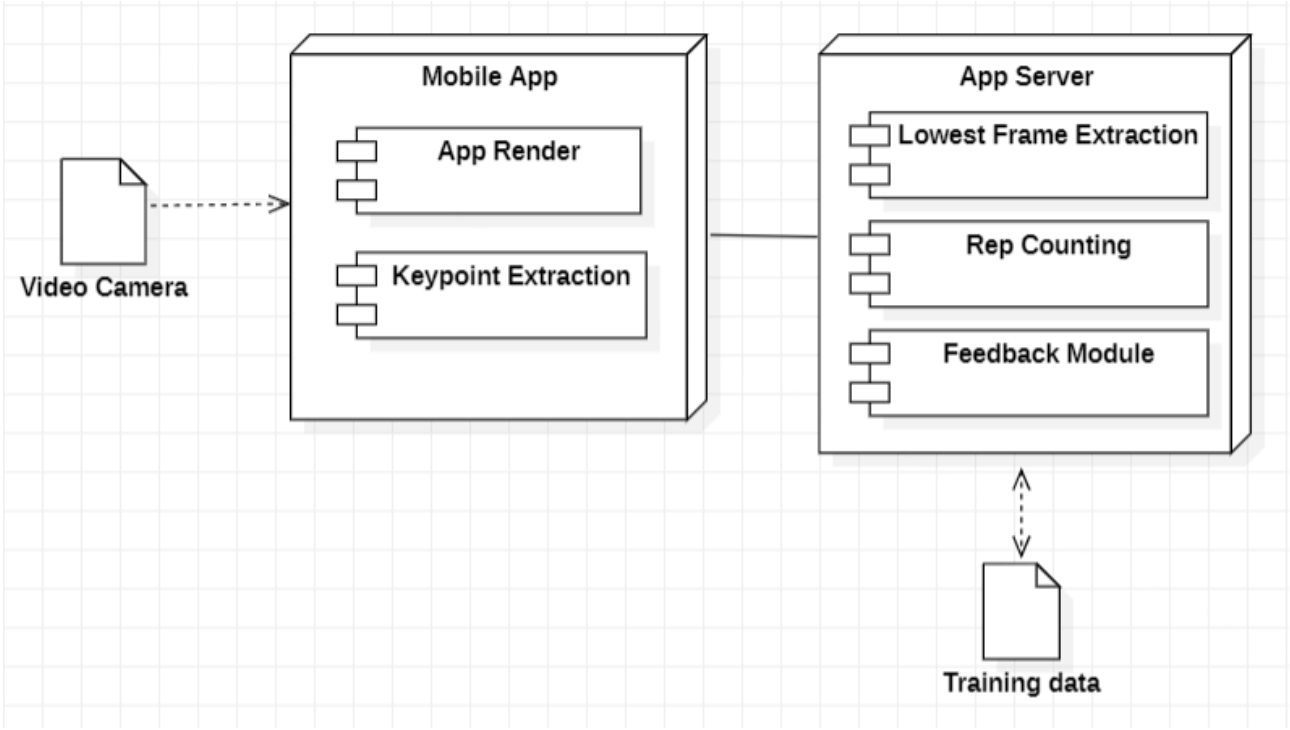


Fig 5.3: Packaging and Deployment Diagram

The mobile device block consists of an app. The video camera provides the app with real-time video info. From this video frames are extracted. The frames are passed through MediaPipe and the lowest frame is identified. The frame's extracted key points are used to calculate angles. These angles are sent to the trained model and an output is obtained. This output is sent to the user and

displayed on the app as feedback. A rep counter is also displayed to keep track of the number of times the user performs the exercise.

5.7 Help

A video tutorial of the app will be provided during the release of the product which will consist of specific instructions on how to make the most of our product.

5.8 Design Details

Novelty : The product would be novel in the fitness industry by leveraging advanced technologies. These technologies involve computer vision and machine learning to provide real-time feedback. Hence improving fitness outcomes and reducing the risk of injury. The product also involves counting the number of reps hence adding to its innovativeness.

Interoperability: The app is built to be interoperable. It should work seamlessly with a range of devices and platforms.

Performance: Since the app is designed to be real-time, it is expected to deliver high performance. The algorithms, processing and model feedback procedure also need to be optimized to ensure instant feedback.

Security: User data security is a key point to keep in mind to ensure privacy. This involves secure transmission protocols and complying with data regulations.

Reliability: Rigorous testing should be done to ensure the reliability of the app since the user blindly trusts the feedback. Testing should be done across various devices and user types.
Maintainability: RepRight development should ensure that it can be updated and improved over time. This can be taken care of with a well organized code database and systematic developers.

Portability: RepRight is to be portable across different platforms for more reach and accessibility.

Reusability: Every piece of code should be designed with reusability in mind. RepRight also involves designing the app in a modular and flexible way to ensure that it can be adapted and repurposed.

Resource utilization: RepRight would contain optimized resource utilization to minimize its impact on device battery life and processing power.

CHAPTER 6

LOW LEVEL DESIGN DOCUMENT

6.1 Introduction

6.1.1 Overview

RepRight is a cutting-edge application poised to revolutionize the fitness industry, especially in the realm of weight training. As gym culture takes a dominant presence in urban lifestyles, RepRight offers a solution to an emerging challenge - ensuring the correct form during exercises to prevent injuries and maximize benefits. By leveraging AI, RepRight provides real-time video analysis to guide users in correcting their exercise form through instantaneous audio feedback

6.1.2 Purpose

This Low-Level Design (LLD) document provides a detailed blueprint for the implementation of the AI assisted gym form correction system. It bridges the gap between the High-Level Design (HLD) and the actual coding phase and offers a comprehensive view of the system's internal architecture, components, and their interactions.

6.1.3 Scope

This document encompasses the design considerations for RepRight, focusing on:

- System architecture and modular structure.
- Data flow and data design.
- Interfaces, external and internal.
- Algorithms and procedures essential for real-time video analysis and feedback generation.

6.2 Design Constraints, Assumptions, and Dependencies

Design Considerations:

User-centric Design: The application should be intuitive, accommodating users ranging from fitness enthusiasts to beginners.

Real-time Processing: Minimizing latency between video input and feedback is crucial for user experience and safety.

Scalability: The application should efficiently handle an increasing user base without compromising performance.

Compatibility: RepRight must be adaptable across various Android devices and versions.

Assumptions:

Users will have mobile devices with cameras and the necessary processing power for the application.

An internet connection will be available when necessary to process the user's posture in real time.

Users will correctly position their devices to capture their exercises effectively since there is no additional function to verify this. The user's entire body should be visible hence allowing all possible key points to be marked

Dependencies:

- The application's real-time feedback system heavily depends on fast communication between client and server as well as fast prediction machine learning models.
- RepRight's functionality relies on certain software components including Python, TensorFlow, Keras, Android Studio, and Kotlin
- External interfaces, especially ngrok, play a vital role in the communication interface.

- Regulatory policies related to data privacy and security must be adhered to, influencing design considerations around user data handling.

6.3 Design Description

6.3.1 Use Case Diagram

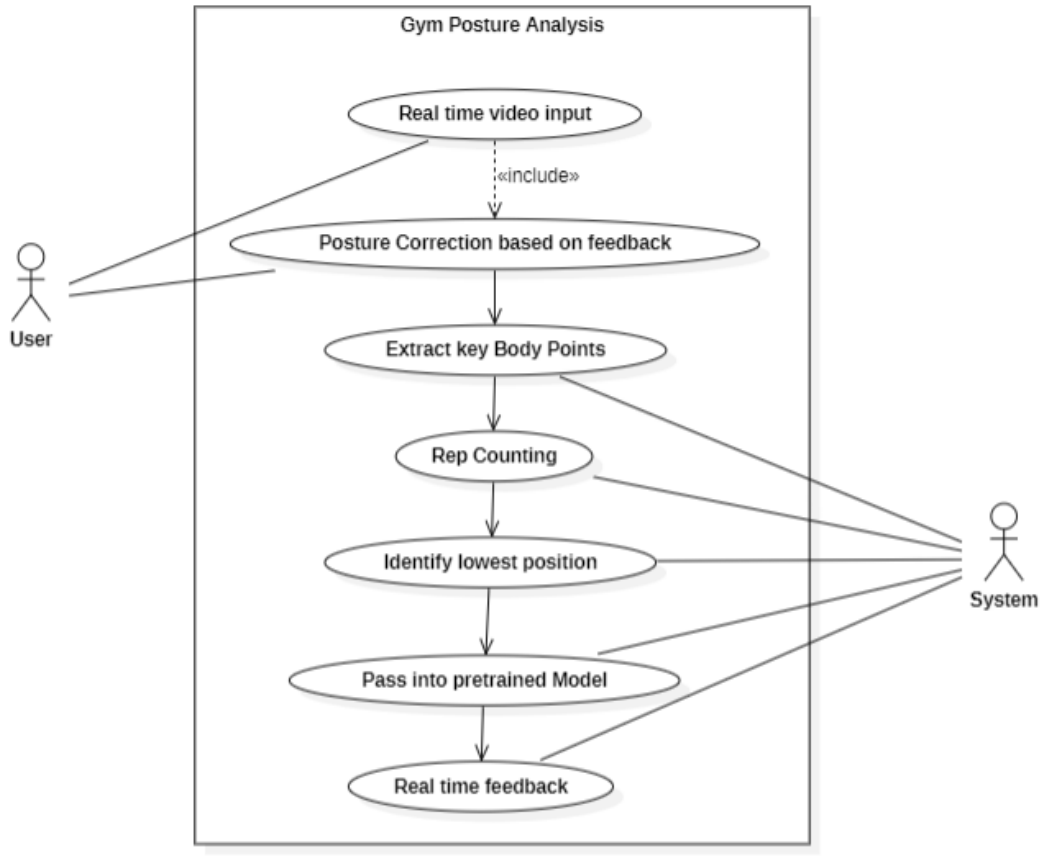


Fig: 6.0 Use Case Diagram

Use Case Item	Description
Real time video input	The camera opens up, and the app receives a real-time video input of the user performing the exercise
Posture correction based on feedback	The user can correct their form based on the feedback given for the rep which will be displayed on the screen.
Extract key body points	The video input is then passed through mediapipe and the key body points are extracted at each frame
Rep Counting	A global counter variable is updated for every repetition of the squat
Identify lowest point	The lowest point of the squat is extracted to test how deep the squat reached
Pass to pretrained model	Deep learning model is loaded and the lowest point passed through it to get feedback
Real time feedback	Feedback displayed on screen of the mobile device

Table 6.0: Use Case Descriptions

6.3.2 Class Diagram

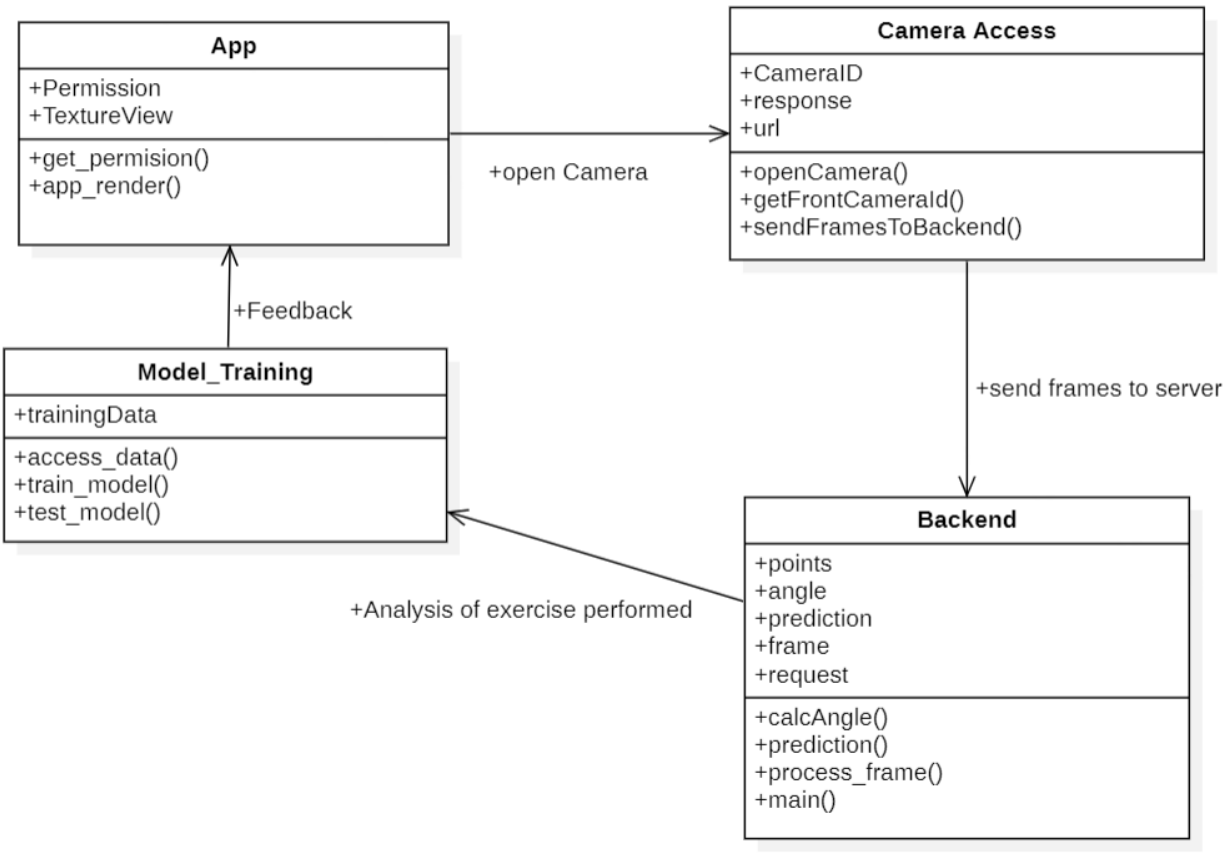


Fig: 6.1 Class Diagram

6.3.2.1 Android App

Description: The main method where the data flow begins. This is the user's first interaction with the product. It is essentially an Android app interface which is the link to take user video input for processing. Coded in Kotlin takes permission to access the front camera and passes the video frame by frame to the backend server to give necessary feedback.

Method 1: openCamera()- The function after accessing the camera to iteratively take user input and start a capture session

Algorithm 1: open_camera()

```
// Get front camera ID
cameraID ← get_front_camera_ID();
// Open the camera with a CameraDevice.StateCallback
open the camera with a CameraDevice.StateCallback::;
    // onOpened callback
    onOpened::;
        // Set cameraDevice variable
        cameraDevice ← getCameraDevice();
        // Get surfaceTexture from textureView
        surfaceTexture ← getSurfaceTextureFromTextureView();
        // Create surface from surfaceTexture
        surface ← createSurfaceFromTexture(surfaceTexture);
        // Create captureRequest for TEMPLATE_PREVIEW
        captureRequest ←
         createCaptureRequest(TEMPLATE_PREVIEW);
        // Add target to captureRequest
        addTargetToCaptureRequest(captureRequest, surface);
        // Create a capture session with the surface and
           CameraCaptureSession.StateCallback
        createCaptureSession(surface,
         CameraCaptureSession.StateCallback)::;
           // onConfigured callback
           onConfigured::;
              // Set repeating request on the capture session
              setRepeatingRequestOnCaptureSession(captureRequest);
           // onConfigureFailed callback
           onConfigureFailed::;
              // Do nothing
              doNothing();
```

Method 2: To get Id for frontCamera since most devices have multiple cameras, finding the appropriate front facing camera is made into a function.

Algorithm 2: getFrontCameraId()

```
// Get cameraIdList from cameraManager
cameraIdList ← getCameraIdListFromCameraManager();
// Iterate over cameraIdList
for each cameraId in cameraIdList do
    // Get characteristics for each cameraId
    characteristics ← getCharacteristicsForCameraId(cameraId);
    // Get facing value from characteristics
    facing ← getFacingValueFromCharacteristics(characteristics);
    if facing is front then
        // Return cameraId
        return cameraId;
// Throw IllegalStateException if front camera not found
IllegalStateException("Front camera not found");
```

Method 3: get_permission()- camera access needs permission in most devices

Algorithm 3: get_permission()

```
// Check if CAMERA permission is not granted
if CAMERA permission is not granted then
    // Request CAMERA permission with requestCode 101
    requestCameraPermission(101);
```

Method 4: onCreate()- The main function that takes user video and displays it on the Device using a videoThread handler.

Algorithm 4: onCreate()

```
// Call super.onCreate() and
   setContentView(R.layout.activity_main)
super.onCreate();
setContentView(R.layout.activity_main);
// Get permission for CAMERA
get_permission();
// Initialize imageProcessor with ResizeOp(300, 300,
   ResizeOp.ResizeMethod.BILINEAR)
imageProcessor ← initializeImageProcessor(300, 300,
```

```
    ResizeOp.ResizeMethod.BILINEAR);
   // Start handlerThread named "videoThread"
5  startHandlerThread("videoThread");
   // Initialize handler with handlerThread's looper
6  handler ← initializeHandler(handlerThread.getLooper());
   // Find imageView by ID and assign to imageView variable
7  imageView ← findViewById(R.id.imageView);
   // Find textureView by ID and set up SurfaceTextureListener
8  textureView ← findViewById(R.id.textureView);
9  setSurfaceTextureListener(textureView, SurfaceTextureListener):;
       // onSurfaceTextureAvailable callback
10     onSurfaceTextureAvailable:;
           // Call open_camera()
11         open_camera();
       // onSurfaceTextureSizeChanged callback
12     onSurfaceTextureSizeChanged:;
       // Do nothing
       // onSurfaceTextureDestroyed callback
13     onSurfaceTextureDestroyed:;
           // Return false
14         return false;
       // onSurfaceTextureUpdated callback
15     onSurfaceTextureUpdated:;
           // Increment frameCounter
16         frameCounter← frameCounter+1;
           // If frameCounter is a multiple of 3
17         if frameCounter is a multiple of 3 then
               // Get bitmap from textureView
18             bitmap ← getBitmapFromTextureView();
               // Convert bitmap to black and white using
                  convertToBlackAndWhite function
19             blackAndWhiteBitmap ←
                convertToBlackAndWhite(bitmap);
               // Create a resized bitmap (224x224)
20             resizedBitmap ← resizeBitmap(blackAndWhiteBitmap, 224,
                224);
               // Create a TensorImage and load the resized bitmap
21             tensorImage ← createTensorImage();
22             tensorImage.load(resizedBitmap);
               // Process the image using imageProcessor
23             processedFrame ← processImage(imageProcessor,
                tensorImage);
               // Send the processed frame to the backend using
                  sendFrameToBackend
24             sendFrameToBackend(processedFrame);
```

```
    // Initialize cameraManager as CAMERA_SERVICE
25  cameraManager ← initializeCameraManager(CAMERA_SERVICE);
```

Method 5: sendFrames()- The function that connects to the backend through ngrok and sends frames as an http request to the backend server. Receives prediction from backend and processes it accordingly before displaying it on the app surface texture view.

Algorithm 5: sendFrameToBackend(frame)

```
  // Set URL to the backend endpoint
1 URL ← setBackendEndpoint();
  // Convert frame to byte array
2 byteArray ← convertFrameToByteArray(frame);
  // Encode byte array to base64
3 base64Data ← encodeToBase64(byteArray);
  // Create a JSON object with the base64 frame data
4 jsonObject ← createJSONObject(base64Data);
  // Create a request body with the JSON data
5 requestBody ← createRequestBody(jsonObject);
  // Build a POST request with the URL and request body
6 request ← buildPOSTRequest(URL, requestBody);
  // Enqueue the request with a callback
7 enqueueRequest(request, Callback):;
    // onFailure callback
8   onFailure:;
      // Print error
9     printError("Request failed");
    // onResponse callback
10  onResponse:;
      // Parse the response JSON
11    responseJSON ← parseResponseJSON(response);
      // Extract prediction and rep
12    prediction ← extractPrediction(responseJSON);
13    rep ← extractRep(responseJSON);
      // Display prediction and rep using text on the UI
14    displayOnUI(prediction, rep);
      // Remove the processed frame from the map
15    removeFrameFromMap(frame);
```

6.3.2.2 Backend Server

Description: Frames are processed as a part of the backend. This involves keypoint extraction and model processing.

Method 1: calcAngle()-The key points extracted give no information unless processed to get an angle from them.

Algorithm 6: calculate_angle(a, b, c)

```
   // Calculate angle using three points saved as numpy array
   // Assuming points a, b, and c are numpy arrays
1  a ← numpy array of point a;
2  b ← numpy array of point b;
3  c ← numpy array of point c;
   // Calculate angle using arctan2
4  radians ← arctan2(c[1] - b[1], c[0] - b[0]) - arctan2(a[1] - b[1], a[0] - b[0]);
   // Convert radians to degrees
5  angle ← absolute value of (radians * 180.0 / pi);
   // Ensure the angle is in the range [0, 180]
6  if angle ¿ 180.0 then
7      angle ← 360 - angle;
   // Return the calculated angle
8  return angle;
```

Method 2: prediction()- Pass angle value through loaded machine learning model and return classification as prediction label.

Algorithm 7: get_prediction(angle_knee)

```
   // Create a new_data DataFrame with a single column 'angle'
      containing angle_knee
1  new_data ← createDataFrame(angle_knee);
   // Use the forest_model to predict on new_data
2  prediction ← forest_model.predict(new_data);
   // Create a labels dictionary mapping prediction labels to
      human-readable strings
3  labels ← createLabelsDictionary();
   // Return the human-readable prediction label corresponding
      to the predicted class
4  return labels[prediction[0]];
```

Method 3: process_frame()- Frame returned from the app is not passed unchanged through the model. The model uses angle value to give the prediction on the user's form. The frame is hence processed with MediaPipe /YoloV7. The same function also sends results from prediction() as a json response to the front-end of the app. This function also includes a logic to count reps

Algorithm 8: process_frame()

```
// Retrieve the frame data from the POST request
frameData ← getFrameDataFromPOSTRequest();
// Decode the base64-encoded image data
imageData ← decodeBase64(frameData);
// Convert image data to a NumPy array
frame ← convertToNumPyArray(imageData);
// Convert color space from BGR to RGB
frame ← convertColorSpace(frame, BGR2RGB);
// Use MediaPipe Pose to detect landmarks in the frame
results ← detectLandmarksWithMediaPipePose(frame);
// If pose landmarks are detected
if pose landmarks are detected then
    // Extract relevant landmarks for shoulder, hip, knee,
       and ankle
    landmarks ← extractLandmarks(results);
    // Calculate knee joint and hip angles using
       calculate_angle function
    angleKnee ← calculate_angle(landmarks[hip], landmarks[knee],
     landmarks[ankle]);
    angleHip ← calculate_angle(landmarks[shoulder], landmarks[hip],
     landmarks[knee]);
    // Update global stage based on knee angle
    updateGlobalStage(angleKnee);
    // If knee angle is within valid range
    if knee angle is within valid range then
        // Get prediction using get_prediction function
        prediction ← get_prediction(angleKnee);
        // Update global counter if necessary based on the
           stage and knee angle
        updateGlobalCounter(prediction, angleKnee);
    // Set prev to current knee angle for the next iteration
    prev ← angleKnee;
```

```
15 else
       // Set default results indicating the need to get into
          position
16     results ← getDefaultResults();
   // Return results as JSON
17 return results;
```

Method 4: main() hosts the flask app on local host as well as includes initializing global variables function main():

Algorithm 9: main()

```
   // Initialize global variables for previous knee angle,
      counter, and stage
 1 prev ← initializeGlobalVariable();
 2 counter ← initializeGlobalVariable();
 3 stage ← initializeGlobalVariable();
   // Initialize MediaPipe Pose
 4 pose ← initializeMediaPipePose();
   // Load RandomForest model and relevant CSV files
 5 forest_model ← loadRandomForestModel();
 6 correct_df ← loadCSVFile('correct.csv');
 7 too_high_df ← loadCSVFile('too_high.csv');
 8 too_low_df ← loadCSVFile('too_low.csv');
   // Prepare data for training the model
 9 data_df ← prepareDataForModel(correct_df, too_high_df, too_low_df);
   // Train the RandomForest model
10 trainRandomForestModel(forest_model, data_df);
   // Start Flask app on host 0.0.0.0, port 5000, in threaded
      mode
11 startFlaskApp(host='0.0.0.0', port=5000, threaded=True);
```

6.3.3 Sequence Diagram

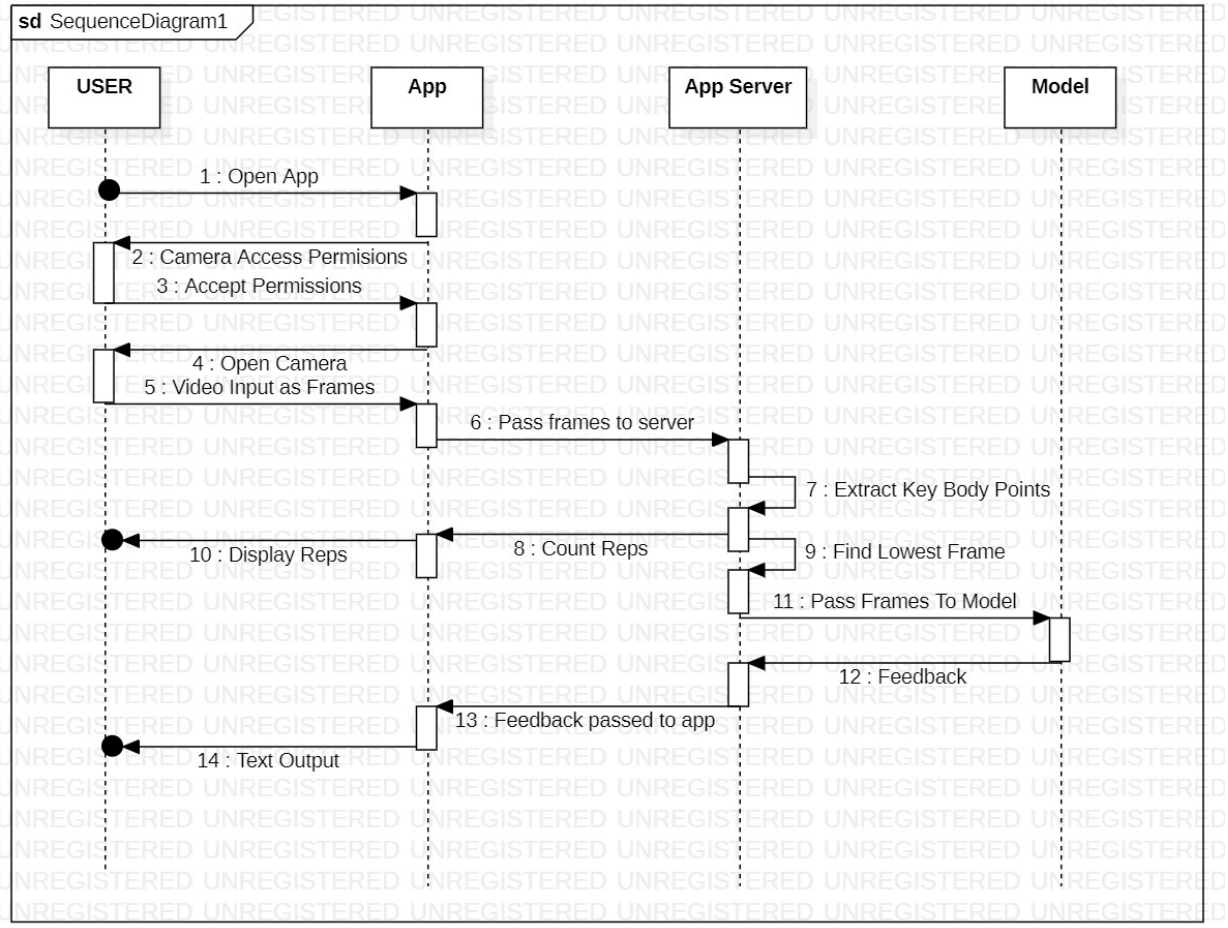


Fig : 6.2 Sequence Diagram

6.3.4 Packaging and Deployment Diagrams

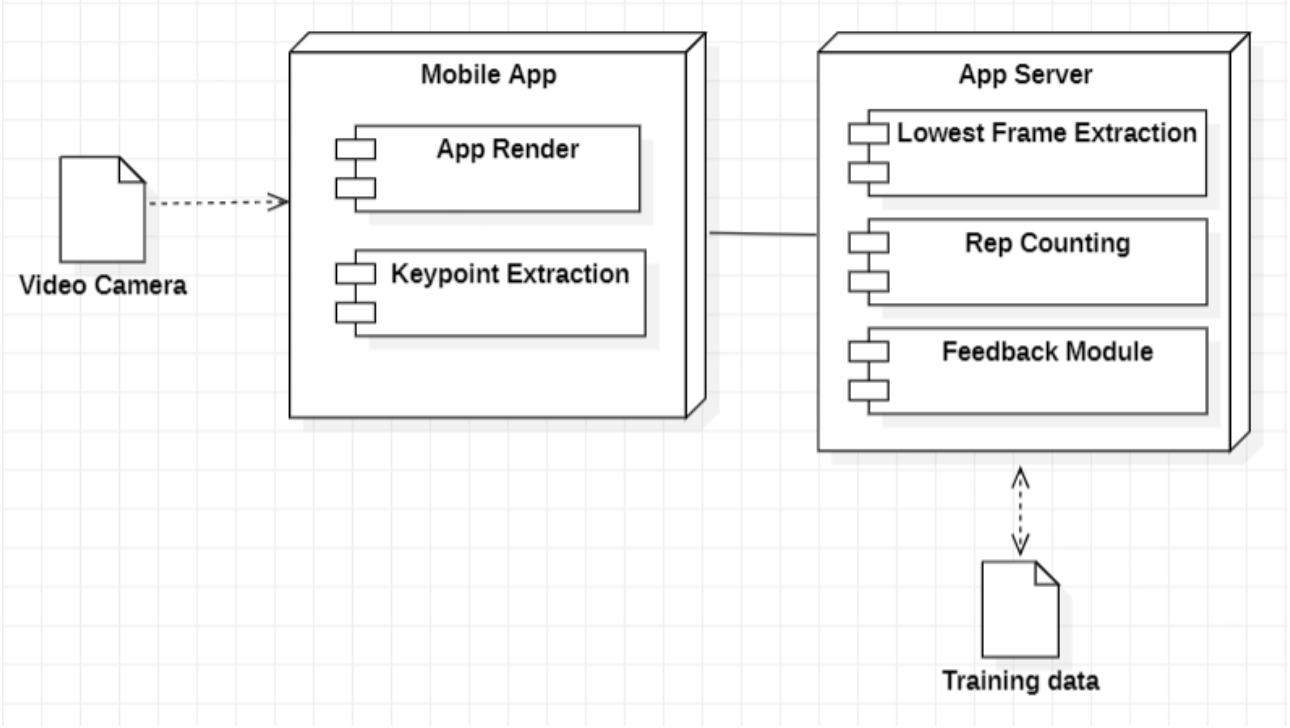


Fig : 6.3 Packaging and Deployment Diagram

6.4 Proposed Methodology / Approach

6.4.1 Algorithm and Pseudocode

i) Squat

Step 1: Input module uses camera access and records real-time video as input

Step 2: MediaPipe is utilized to find the angle at the knee for each frame.

Step 3:Random Classifier model is trained on the dataset of squat images consisting of 3 classes- high, low and correct.

Step 4: The model is converted into a joblib file and then to h5 for the Android application

Step 5: ML model predicts the form of exercise by checking relevant features and parameters

Step 6: All the frames are passed into the model and a real time prediction is displayed onto the live video with the angle.

6.4.2 Implementation and Results

Input Module:

Camera input starts once the user begins performing the exercise. On performing the exercise, video input is sent to Mediapipe frame by frame. The angle is calculated for each and every frame and the angle, the image class is displayed on the video.

Model Training:

Squat: -

Preprocessing data:

Images were passed through by MediaPipe and the angle between the thigh, knee and foot are calculated for each and every image in the dataset for all 3 different classes and their angles are stored in a CSV. There are 3 CSV files high, low and correct with the angles of all the images of that class. These CSV files are used in the training of a random forest model which is then converted into a joblib file and then an h5 file that can be directly uploaded onto the app for prediction.

Model:

RandomForestClassifier: This is a class from scikit-learn that implements a random forest classifier. A random forest is an ensemble learning method that operates by constructing a multitude of decision trees at training time and outputs the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

n_estimators=100: This parameter specifies the number of trees in the forest. In this case, the model is using 100 decision trees.

Pseudo Code:

1) Input: The algorithm takes a dataset represented by features (X) and corresponding labels (y)
2) Splitting the Dataset: Divide the dataset into training and testing sets using an 80-20 split, ensuring reproducibility with a random seed of 42. Assign the training and testing sets to X_train, X_test, y_train, and y_test, respectively.
3) Initializing the Model: Create a Random Forest Classifier with 100 decision trees. Store the classifier in a variable named model.
4) Training the Model: Train the model using the training dataset (X_train and y_train).
5) Saving the Model: Save the trained model to a file named "new_model.h5" using the joblib library
6) Making Predictions: Predict the labels for the testing dataset (X_test) using the trained model. Store the predictions in a variable named y_pred
7) Output: The algorithm returns the trained model (model) and the predicted labels for the testing set (y_pred).

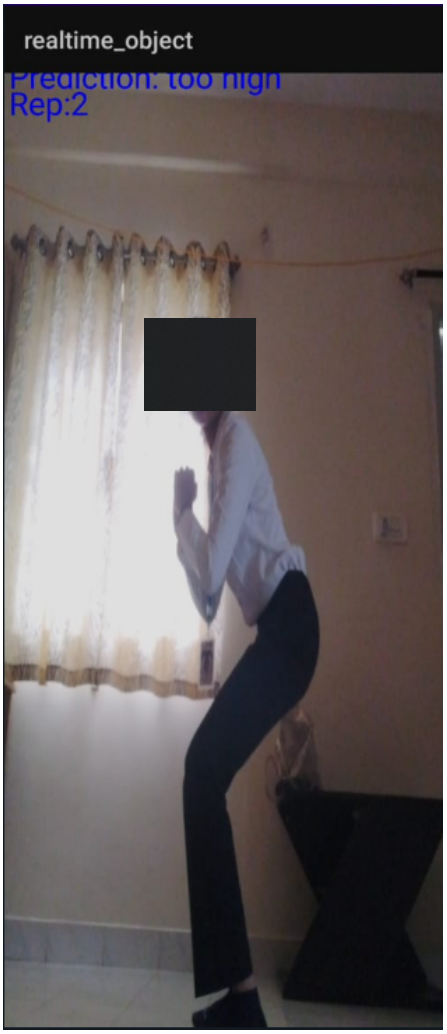


Fig: 6.4 App Screenshot

Barbell Row

Data Preprocessing: The dataset comprises images with json labels and splits. There are 2 types of error: lumbar error and torso angle error. Two splits of training , testing and validating images and json labels with 1 and 0 indicating there is an error, present or not respectively.

A two model approach is taken to check for 2 separate labels independent of one another. The first model is used to check the torso error, i.e, if the user performing the exercise is bending far enough.

The second model checks for a lumbar error, i.e, if the user's back is rounded or not. If rounded, it reports it as an error.

Thus, the image is passed to 2 models simultaneously and the user's form is checked against the two mentioned criteria.

The same two model approach was also tested using VGG16 and methods to handle class imbalance

Torso Angle Error

```
Epoch 9/10
437/437 [==============================] - 74s 169ms/step - loss: 0.2019 - accuracy: 0.9308 - val_loss: 0.2663 - val_accuracy: 0.8907
Epoch 10/10
437/437 [==============================] - 73s 167ms/step - loss: 0.1994 - accuracy: 0.9397 - val_loss: 0.2928 - val_accuracy: 0.8821
```

Lumbar Error

```
Epoch 9/10
375/375 [==============================] - 64s 171ms/step - loss: 0.2767 - accuracy: 0.9196 - val_loss: 0.3734 - val_accuracy: 0.8501
Epoch 10/10
375/375 [==============================] - 65s 172ms/step - loss: 0.2992 - accuracy: 0.9141 - val_loss: 0.3892 - val_accuracy: 0.8437
```

On handling the class imbalance, the loss improved significantly by reducing from 0.4 to 0.2

CHAPTER 7

SYSTEM DESIGN

7.1 Data Flow Diagram

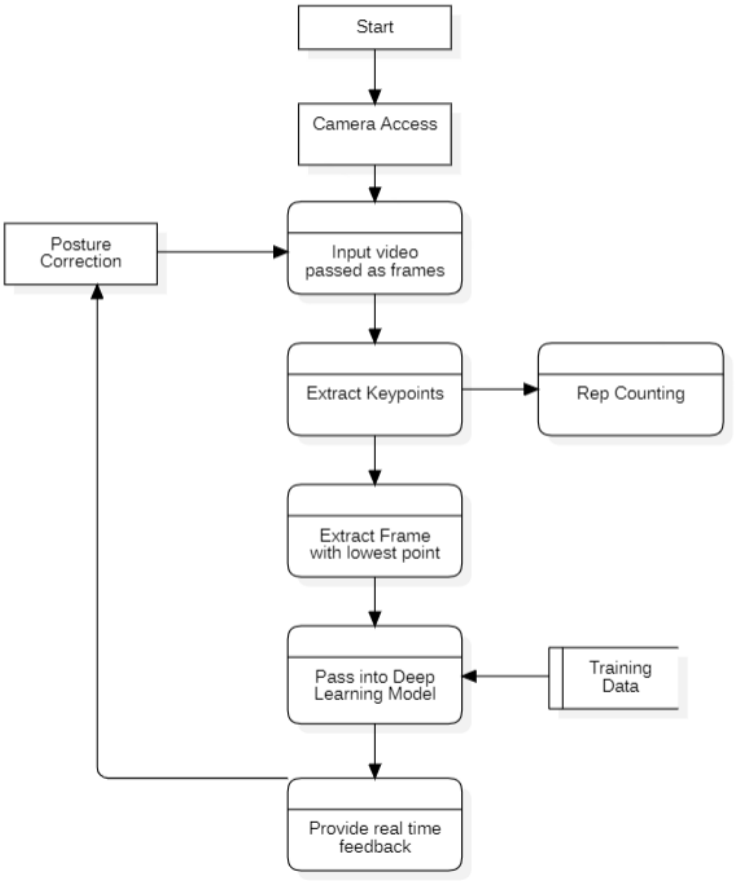


Fig 7.0 : Data Flow Diagram

7.2 UML Interaction Diagram

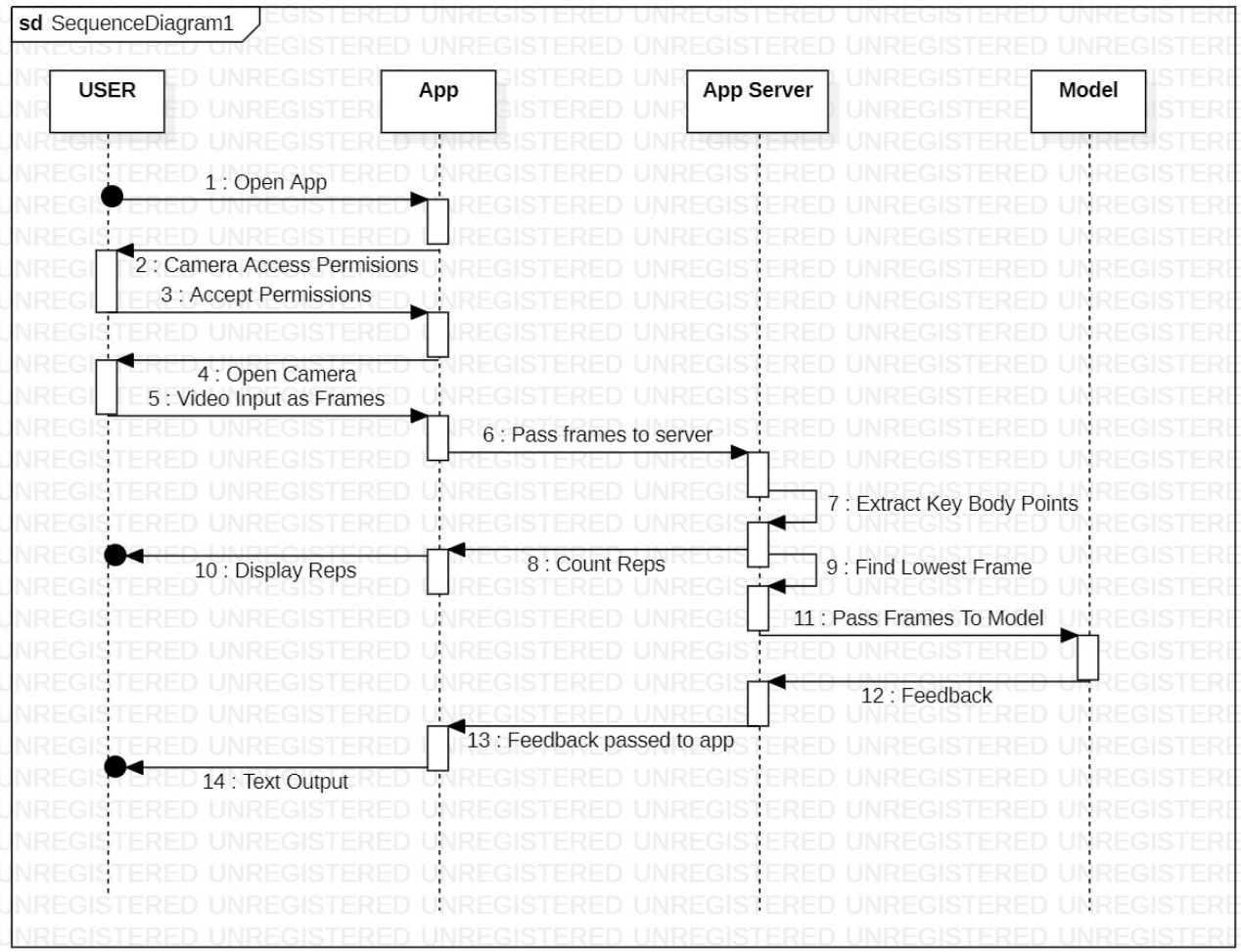


Fig 7.1 : UML - Interaction Diagram

7.3 Modules

The following modules will be used to develop the app:

App render : App rendering refers to the process of displaying the results of the app's analysis on the user's device in real-time. This is typically achieved through the app's user interface.

Send frames to backend : A major part of the app is to process the image and send it to the backend as an HTTP request and hence connection and sending data is a part of this module

Key point extraction module : This module is used to extract the key body points and derive the necessary features used for exercise form analysis

Feedback module: Implemented using the deep learning model to return relevant feedback to correct the posture of the user

Rep counting module: This module is used to count repetitions and print it on the app as an additional functionality.

7.4 Security

The application will request permission to access the camera before recording the video.

Data is sent back and forth across the client and server on a secure HTTPS channel.

All the data collected are kept private in accordance with the Privacy laws.

CHAPTER 8

IMPLEMENTATION AND PSEUDOCODE

8.1 Angle finding using Mediapipe

Algorithm 1: Angle Finding using Mediapipe

```
if results.pose_landmarks then
    landmarks ← results.pose_landmarks.landmark;
    // Try to extract key points
    hip ←
     np.array([landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].x,
     landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].y]);
    knee ←
     np.array([landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value].x,
     landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value].y]);
    ankle ←
     np.array([landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value].x,
     landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value].y]);
    angle ← np.arccos(np.dot((hip - knee), (ankle - knee)) /
     (‖hip − knee‖ · ‖ankle − knee‖)) angle← np.degrees(angle);
    new_data ← pd.DataFrame('angle': [angle]);
    prediction ← model.predict(new_data);
    squat_label ← label_dict[prediction[0]];
    print(f'Squat label: squat_label');
    cv2.putText(image, f'Squat: squat_label); // Handle exceptions
    Exception as e print(e);
    // If there is any error in the landmark detection,
       ignore it
    pass;
    image ← cv2.cvtColor(image, cv2.COLOR_RGB2BGR);
    cv2.imshow('Squat Posture Evaluation', image);
```

Output:

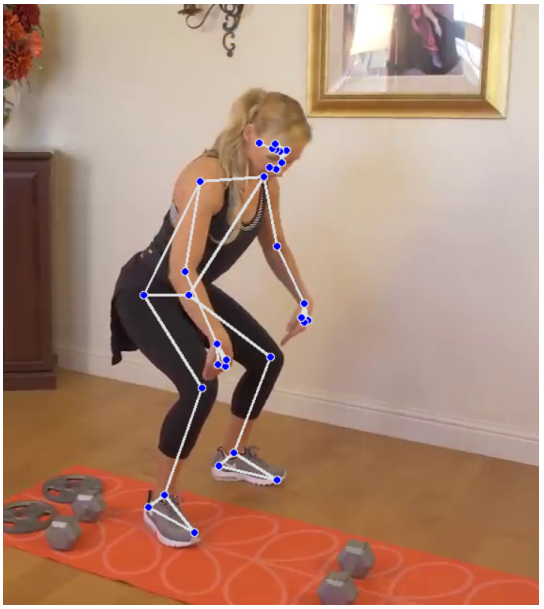


Fig 8.0 : Keypoint extraction with MediaPipe

8.2 Pseudo Code for Real-Time Video Capture

Algorithm 1: Real-Time Video Capture

```
// Initialize video capture
cap ← initialize_video_capture();
// Create video writer object
writer ← create_video_writer();
// Start recording
while True do
    // Read a frame from video capture
    frame ← read_frame(cap);
    // Write the frame to the video writer object
    write_frame(writer, frame);
    // Check if the user has stopped recording
    if user_has_stopped_recording() then
        break;
// Release video capture and writer resources
release_video_capture(cap);
release_video_writer(writer);
```

CHAPTER 9

EXPERIMENTATION RESULTS AND DISCUSSION

The goal of Phase 1 was to establish a well defined problem statement and propose a detailed methodology on how to solve it. A feasibility study and requirement analysis was performed along with research on existing solutions and papers. Our chosen problem statement is to build an app that leverages AI to help users correct their form using real-time video as input.

The goal of Phase 2 was to find or make a reliable dataset and build our project entirely. On conducting extensive research on this topic, we found a large scarcity of data available online. On scouring various sources such as GitHub we found datasets for two exercises, squat and barbell rows.

The Squat dataset consisted of 900 images split between three classes: “correct”, “too high” and “too low”. The following images are from the dataset:

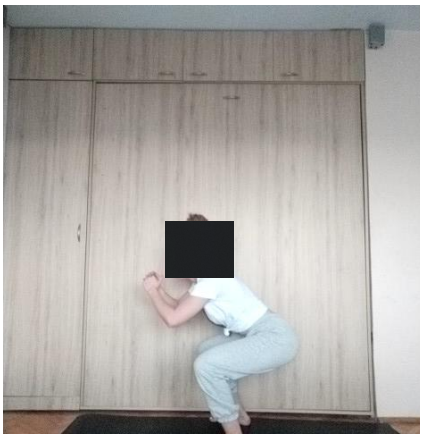


Fig 9.0: Correct

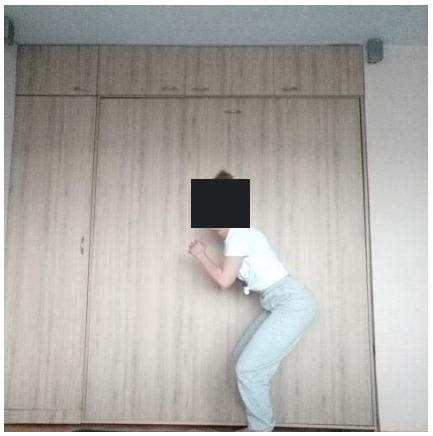


Fig 9.1: Too High

Fig 9.2 \: Too Low

The Barbell row dataset consisted of images and json files. The json files consisted of image ID as keys and label of “0” for no error or “1” for error. Two errors were checked for: torso error which checked how far the user is bending and lumbar error which checked if the user’s back is straight or not.

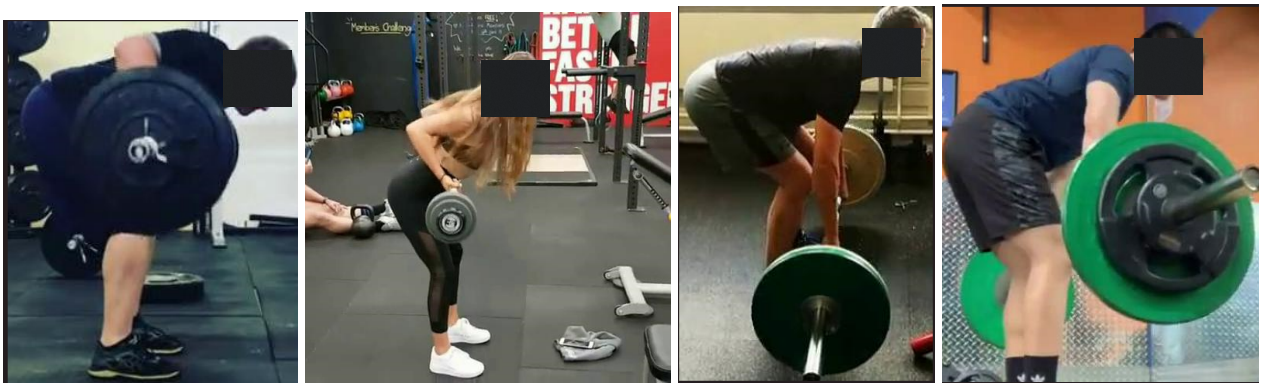


Fig 9.3: Barbell Row dataset images

Thus models were built for the respective exercises.

The literature papers also had common patterns that could be leveraged for our use case. Google's MediaPipe was used to extract the key body points. These points were then utilized to calculate angles and determine the form of the user. An alternative was found which was the YoloV7 pose estimation model. Based upon our use cases MediaPipe was implemented to extract key points for our squat model and find relevant angles. MediaPipe was however not implemented for our Barbell Row model.

Each image was passed through a mediapipe model and the key points were predicted but the predicted keypoints had lots of errors. The testing images on mediapipe are added below.

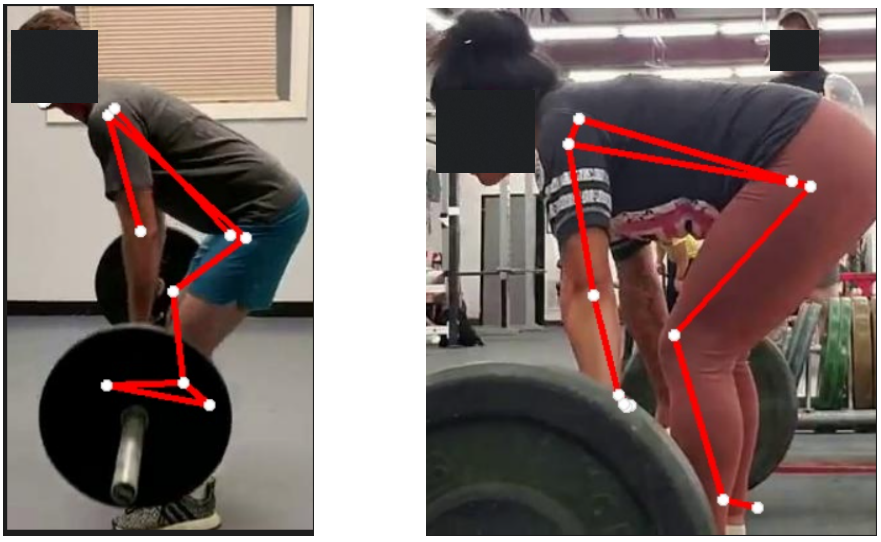


Fig 9.4 : Key Points extracted by mediapipe overlapped on images

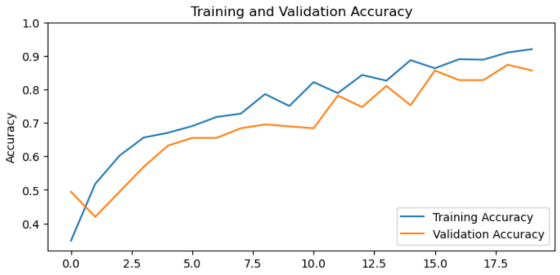
Hence, MediaPipe preprocessing technique was removed and barbell row model was trained on original dataset images:

Training Models-

Squat Model Training:

1) VGG16:

Even though validation accuracy is good we observed high loss




```
Epoch 19/20
11/11 [==============================] - 14s 1s/step - loss: 0.3693 - accuracy: 0.9101 - val_loss: 0.4365 - val_accuracy: 0.8736
Epoch 20/20
11/11 [==============================] - 16s 2s/step - loss: 0.3598 - accuracy: 0.9201 - val_loss: 0.4073 - val_accuracy: 0.8563
```

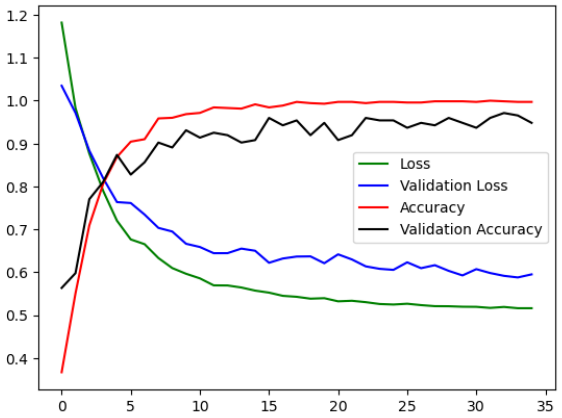
Fig 9.5: VGG16 model results for squat dataset

Epoch	Validation Loss	Validation Accuracy
19	0.4365	0.8736
20	0.4073	0.8563

Table 9.0: VGG16 model results for squat dataset

2) MobileNET:

Performed worse than VGG16 showing a higher loss




```
Epoch 35/35
11/11 [==============================] - 23s 2s/step - loss: 0.5159 - categorical_accuracy: 0.9971 - val_loss: 0.5949 - val_cate
gorical_accuracy: 0.9483
```

Fig 9.6: MobileNET model results for squat dataset

Epoch	Validation Loss	Validation Accuracy
11	0.5949	0.9483

Table 9.1: MobileNET model results for squat dataset

3) Random forest model and CSV file of angles

A CSV file of angles and labels was built. Angles were calculated by passing training images through MediaPipe and extracting key body points. The angle at the knee was

calculated and stored in a CSV file with its label. This CSV file was used as training data for the random forest model which learnt optimal range angles for each category of label

```
              precision    recall  f1-score   support

           0       0.96      1.00      0.98        52
           1       1.00      0.96      0.98        55
           2       1.00      1.00      1.00        68

    accuracy                           0.99       175
   macro avg       0.99      0.99      0.99       175
weighted avg       0.99      0.99      0.99       175
```

```
Epoch 7/8
22/22 [==============================] - 3s 149ms/step - loss: 0.0070 - accuracy:
0.9986 - val_loss: 0.1882 - val_accuracy: 0.9829
Epoch 8/8
22/22 [==============================] - 3s 148ms/step - loss: 0.0019 - accuracy:
0.9986 - val_loss: 0.1760 - val_accuracy: 0.9943
```

Fig 9.7: RandomForest model results for squat dataset

Epoch	Validation Loss	Validation Accuracy
7	0.1882	0.9829
8	0.1760	0.9943

Table 9.2: RandomForest model results for squat dataset

Hence, the third model, i.e, Random Forest Model trained on CSV file, performed the best and was used to integrate into our app RepRight

Barbell Model training:

1) VGG16

Checking torso error

```
Epoch 9/10
108/108 [==============================] - 10s 89ms/step - loss: 0.0284 - accurac
y: 0.9907 - val_loss: 0.4882 - val_accuracy: 0.9132
Epoch 10/10
108/108 [==============================] - 10s 95ms/step - loss: 0.0216 - accurac
y: 0.9916 - val_loss: 0.4227 - val_accuracy: 0.9045
```

Checking lumbar error

```
Epoch 9/10
91/91 [==============================] - 9s 95ms/step - loss: 0.0317 - accuracy:
0.9918 - val_loss: 0.4744 - val_accuracy: 0.8432
Epoch 10/10
91/91 [==============================] - 9s 95ms/step - loss: 0.0353 - accuracy:
0.9900 - val_loss: 0.4309 - val_accuracy: 0.8333
```

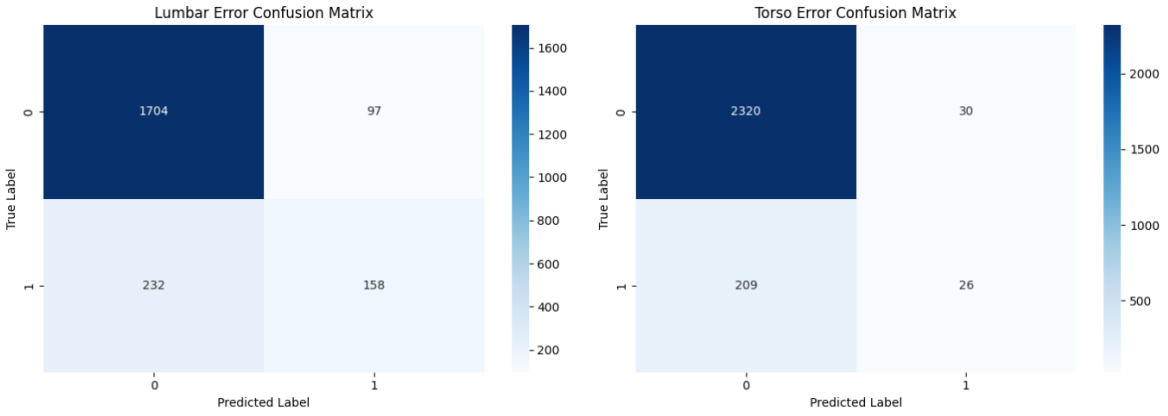


Fig 9.8: VGG16 model results for barbell row

Epoch	Validation Loss	Validation Accuracy
9 for torso error	0.4882	0.9123

Epoch	Validation Loss	Validation Accuracy
10 for torso error	0.4227	0.9045
9 for lumbar error	0.4744	0.8432
10 for lumbar error	0.4309	0.8333

Table 9.3: VGG16 model results for barbell row

Good accuracy was found in both models but a high validation loss was seen

2) VGG16 with class balancing

More correct images were present than images with errors. Hence class imbalancing methodology was used to try and balance the ratio by giving more weightage to the images with an error. While loss improved for Torso error, it got worse for lumbar error

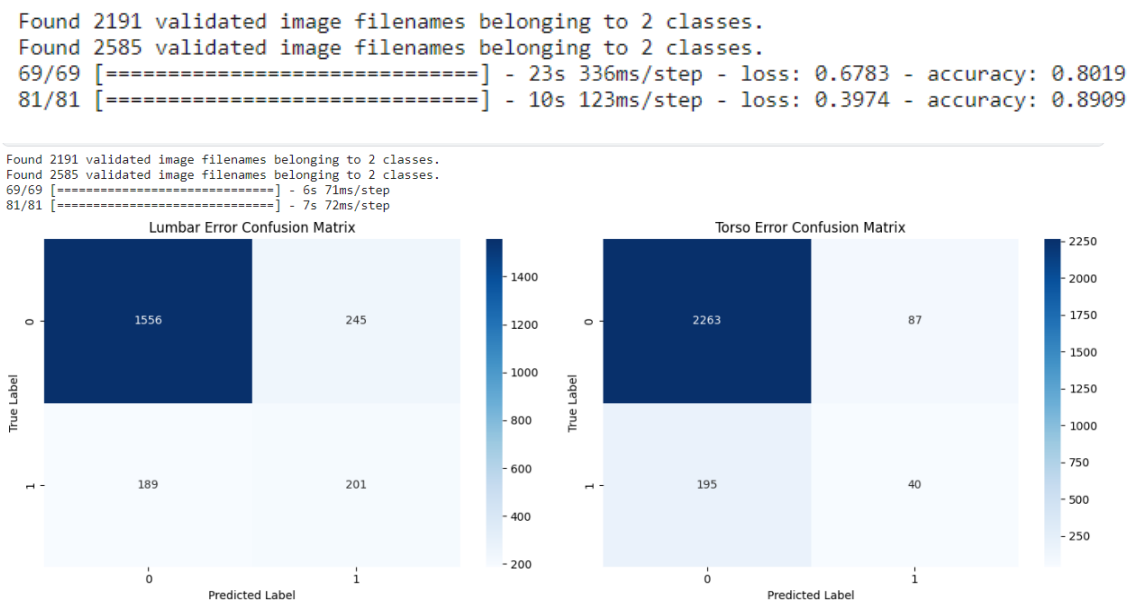


Fig 9.9: VGG16 with class balancing model results for barbell row

Error	Loss	Accuracy
Lumbar	0.6783	0.8019
Torso Angle	0.3974	0.8909

Table 9.4: VGG16 with class balancing model results for barbell row

3) RESNet 50

```
Found 2191 validated image filenames belonging to 2 classes.
Found 2585 validated image filenames belonging to 2 classes.
69/69 [==============================] - 19s 274ms/step - loss: 0.4826 - accuracy: 0.8498
81/81 [==============================] - 10s 125ms/step - loss: 0.3044 - accuracy: 0.9075
```

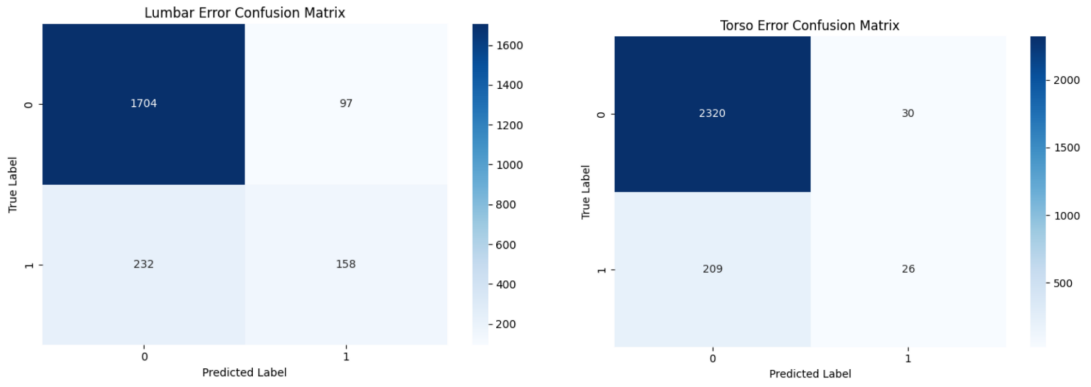


Fig 9.10: RESNet 50 model results for barbell row

Error	Loss	Accuracy
Lumbar	0.4826	0.8499
Torso Angle	0.3044	0.9075

Table 9.5: RESNet 50 model results for barbell row

Hence, RESNet 50 was used as our final model due to lower loss.

Testing the models-

Squat Model:

Fig 9.11: Test results for squat model

Barbell Model:

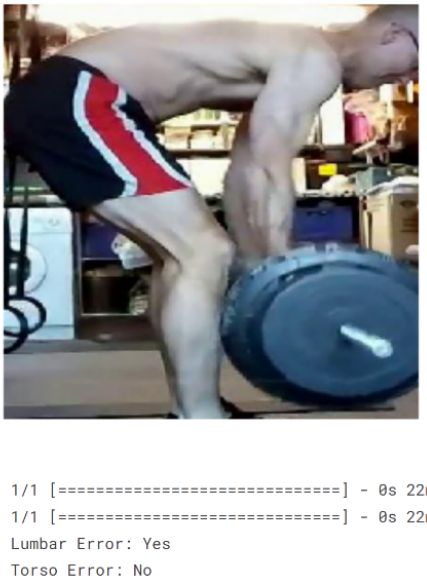


Fig 9.12: Test results for barbel row mode

CHAPTER 10

CONCLUSION AND FUTURE WORK

RepRight was built successfully with the help of Android Studio. Two exercises were focused on, Squat and Barbell Row. Models were built and trained on specific datasets. The Squat Model was built using a random forest model where it learnt the range of angles for an optimal squat and a wrong squat. Google's MediaPipe was used to extract key body points and calculate relevant angles. The Barbell Row Model was built using a pre-trained image classification model, VGG16, and used input images to check for two errors: torso error and lumbar error. The squat model was integrated into the app RepRight along with a feature to count the number of times the user performs the exercise.

Future work includes adding the Barbell model to the app. To optimize the model to enable real-time processing Future enhancements in model architecture and efficient backend processing to ensure swift data processing at real-time speeds. Furthermore, additional exercises can be implemented and integrated into the app by building relevant models. A user database and login credentials could be added for users to track progress and store additional health data.

REFERENCES AND BIBLIOGRAPHY

[1] Yucheng Chen, Yingli Tian, Mingyi He,Monocular human pose estimation: A survey of deep learning-based methods,Computer Vision and Image Understanding,Volume 192, 2020, 102897, ISSN 1077-3142, https://doi.org/10.1016/j.cviu.2019.102897.

[2] G. Taware, R. Kharat, P. Dhende, P. Jondhalekar and R. Agrawal, "AI-Based Workout Assistant and Fitness Guide," 2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA, Pune, India, 2022, pp. 1-4, doi: 10.1109/ICCUBEA54992.2022.10010733.

[3] A. Nagarkoti, R. Teotia, A. K. Mahale and P. K. Das, "Realtime Indoor Workout Analysis Using Machine Learning & Computer Vision," 2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Berlin, Germany, 2019, pp. 1440-1443, doi: 10.1109/EMBC.2019.8856547.

[4] Q. -T. Pham et al., "Automatic recognition and assessment of physical exercises from RGB images," 2022 IEEE Ninth International Conference on Communications and Electronics (ICCE), Nha Trang, Vietnam, 2022, pp. 349-354, doi: 10.1109/ICCE55644.2022.9852094.

[5] Pose Estimation and Correcting Exercise Posture: Rahul Ravikant Kanase, Akash Narayan Kumavat, Rohit Datta Sinalkar, Sakshi Somani ITM Web Conf. 40 03031 (2021) DOI: 10.1051/itmconf/20214003031

[6] J. Shotton et al., "Real-time human pose recognition in parts from single depth images," CVPR 2011, Colorado Springs, CO, USA, 2011, pp. 1297-1304, doi: 10.1109/CVPR.2011.5995316.

[7] H. O. Velesaca, J. Vulgarin and B. X. Vintimilla, "Deep Learning-based Human Height Estimation from a Stereo Vision System," 2023 IEEE 13th International Conference on Pattern Recognition Systems (ICPRS), Guayaquil, Ecuador, 2023, pp. 1-7, doi: 10.1109/ICPRS58416.2023.10179079.

[8]S. -H. Nguyen et al., "Segmentation and observation of hand rehabilitation exercises by supporting of acceleration signals," 2023 Asia Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), Taipei, Taiwan, 2023, pp. 1291-1295, doi: 10.1109/APSIPAASC58517.2023.1031730

APPENDIX A: DEFINITIONS, ACRONYMS AND ABBREVIATIONS

i. Form: refers to the posture of an individual while performing an exercise

ii. Rep: refers to a repetition or cycle of an exercise

iii. AI: Artificial Intelligence refers to the development of computer systems that can perform tasks that typically require human intelligence, such as learning, reasoning, problem-solving, and understanding natural language.

iv. ML: Machine Learning is a field of artificial intelligence that enables systems to automatically learn and improve from experience without being explicitly programmed.

v. MediaPipe: an open-source framework developed by Google used for building real-time multimedia applications, particularly focusing on computer vision and machine learning pipelines. It facilitates tasks such as object detection, pose estimation, and facial recognition.

vi. YOLOv7: a real-time object detection algorithm that efficiently detects and classifies objects in images or videos, notable for its speed and accuracy.

vii. Ngrok: Used to securely expose local servers online for the app.

viii. Https: It is used for secure data transmission across the internet.