A MINOR PROJECT REPORT

ON

HUMAN POSTURE RECOGNITION

Submitted by

GAURISHA(21CS19)
NITESH(21CS36)

Under the Guidance of

MS KOMAL MALSHA
Assistant Professor

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN
COMPUTER SCIENCE AND ENGINEERING

Department of Computer Science & Engineering
LINGAYAS VIDYAPEETH, FARIDABAD
December, 2023

Acknowledgement

We would like to express my sincere gratitude to my supervisor **“MS KOMAL MALSA”, “Assistant Professor ”, Department of Computer Science and Engineering,** for giving me the opportunity to work on this topic. It would never be possible for us to take this project to this level without their innovative ideas and their relentless support and encouragement.

We express my deepest thanks to **other faculties who have contributed** for taking part in useful decision & giving necessary advices and guidance and arranged all facilities to make life easier. I choose this moment to acknowledge his/her contribution gratefully.

We take immense pleasure in thanking **Dr. Ritu Sindhu**, Dean & Head of Department of Computer Science & Engineering. Her tolerances of having discussions, sacrificing her personal time, are extremely insightful and greatly appreciated.

Gaurisha (21CS19)

Nitesh(21CS36)

Declaration

We hereby declare that this project report entitled "**HUMAN POSTURE RECOGNISATION** " *by* **GAURISHA(21CS19), NITESH(21CS36)** being submitted in partial fulfillment of the requirements for the degree of Bachelor of Technology in **Computer Science & Engineering** under the Department of Computer Science and Engineering of Lingayas Vidyapeeth Faridabad, during the academic year 2023-2024 , is a bonafide record of our original work carried out under the guidance of **MS KOMAL MALSA, ASSITANT PROFESSOR , COMPUTER SCIENCE.**

We further declare that we have not submitted the matter presented in this Project for the award of any other Degree/Diploma of this University or any other University/Institute.

1. Gaurisha, 21CS19
2. Nitesh , 21CS36

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

Yoga, an ancient practice that promotes physical, mental, and spiritual well-being, has gained widespread popularity in modern society. With the advent of technology, there is an increasing interest in leveraging machine learning (ML) and deep learning (DL) techniques to automate the recognition of yoga postures, facilitating personalized guidance and feedback. This study explores the application of ML and DL algorithms for the automatic recognition of yoga postures from image and video data.

The proposed methodology involves the collection of a comprehensive dataset comprising diverse yoga poses performed by individuals of varying skill levels. Preprocessing techniques, including image normalization and augmentation, are employed to enhance the robustness of the model. Traditional ML algorithms such as Support Vector Machines (SVM) and Random Forests, along with state-of-the-art DL architectures like Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), are implemented and compared for their efficacy in accurately classifying yoga postures.

Additionally, the study investigates the impact of transfer learning, fine-tuning, and ensemble methods to improve model performance, especially in scenarios with limited training data. The evaluation metrics include accuracy, precision, recall, and F1-score, providing a comprehensive analysis of the model's classification capabilities.

Results indicate that DL models outperform traditional ML approaches, showcasing the potential of deep learning in capturing intricate features inherent in yoga poses. The study contributes to the burgeoning field of health tech by offering a reliable framework for real-time yoga posture recognition, laying the groundwork for future applications in virtual coaching, health monitoring, and the development of intelligent yoga assistance systems.

CHAPTER 1

INTRODUCTION

Human posture detection is now widely explored in many application contexts, ranging from content-based retrieval, surveillance, indoor and outdoor monitoring, virtual reality until animation and entertainment. In particular, we are interested in human posture detection in the framework of home-human interface.

1.1 Introduction

Introducing yoga postures through the integration of Machine Learning (ML) and Deep Learning (DL) technologies holds promise for revolutionizing the way individuals engage with their yoga practice. [5]Machine Learning, with its ability to recognize and categorize poses, can offer real-time feedback on alignment and suggest personalized modifications. This involves training ML algorithms on diverse datasets of yoga poses, enabling them to identify and assess user movements through cameras or video input. Additionally, ML models can adapt to individual needs, considering factors such as body proportions and flexibility, providing tailored recommendations for a more personalized yoga experience.

On the other hand, Deep Learning, particularly utilizing Convolutional Neural Networks (CNNs), can elevate the precision of pose estimation. These sophisticated models capture intricate details of body positions and joint angles, offering more accurate and nuanced feedback on posture. The integration of DL into interactive applications enables real-time pose correction, making it particularly beneficial for virtual classes or home practice. Data augmentation techniques enhance the diversity of training datasets, contributing to improved model generalization across various body types and environments.

However, challenges such as ensuring data quality and diversity, achieving low-latency real-time processing, addressing user privacy concerns, and promoting accessibility must be carefully considered in the development of ML and DL applications for yoga postures. Striking a balance between technological innovation and user-centric design will be crucial for creating a system that enhances the learning experience, respects user privacy, and accommodates individuals with varying abilities, ultimately fostering a more inclusive and personalized approach to yoga practice.

The fusion of Machine Learning (ML) and Deep Learning (DL) in the realm of yoga postures stands at the forefront of technological innovation, offering a transformative experience for practitioners. ML's prowess in pose recognition plays a pivotal role by training algorithms on diverse datasets, enabling them to discern and categorize an array of yoga poses. This capability facilitates real-time feedback during practice sessions, utilizing image or video input from cameras to analyse and guide users on aligning their bodies correctly.

Moreover, ML's adaptability shines through in its ability to personalize recommendations based on individual characteristics. This includes considerations for factors such as body proportions and flexibility, ensuring that users receive tailored suggestions to optimize their yoga practice. This personalization not only fosters a more engaging learning experience but also promotes adherence to correct form and alignment.

Complementing ML, Deep Learning, specifically through Convolutional Neural Networks (CNNs), takes pose estimation to a higher echelon. By delving into the intricate details of body positions and joint angles, DL models provide nuanced feedback, offering a more refined assessment of posture. The integration of DL into interactive applications allows for instantaneous pose corrections, a feature particularly beneficial for virtual yoga classes or self-guided sessions at home.

1.2 Problem Definition

The problem at hand revolves around the integration of Machine Learning (ML) and Deep Learning (DL) techniques to enhance the practice of yoga. At its core, the challenge is to create an intelligent system capable of real-time pose recognition, feedback, and personalized guidance for yoga practitioners. This involves developing algorithms that not only accurately identify and categorize diverse yoga poses but also provide instant feedback to users, guiding them on proper alignment and form. The goal extends further into the realm of personalization, where ML models must adapt to individual users' characteristics, such as body proportions and skill levels, tailoring recommendations to meet specific needs and goals.

Deep Learning, particularly through Convolutional Neural Networks (CNNs), is employed to elevate the precision of pose estimation. The challenge lies in capturing the intricate details of body positions and joint angles, enabling a more sophisticated analysis of user posture and movement. Moreover, the system must operate in real-time with minimal latency, ensuring that feedback is seamlessly integrated into the dynamic flow of a yoga session

Addressing data-related challenges is integral, requiring the curation of high-quality and diverse datasets for training. This includes various yoga poses, body types, ages, and skill levels to avoid biases and ensure the models generalize effectively. Privacy concerns add another layer of complexity, necessitating robust measures to protect user data, especially when utilizing cameras for pose recognition. Obtaining explicit consent and maintaining transparency about data usage are critical aspects of the solution.

Lastly, the development must prioritize accessibility, ensuring that the ML and DL applications cater to individuals with varying physical abilities. This involves thoughtful design considerations, usability features, and adaptive interfaces to make the technology inclusive and user-friendly across different experience levels and

capabilities. [2] In essence, the problem definition encompasses creating a holistic solution that not only harnesses the power of ML and DL for accurate pose recognition but also fosters a personalized, secure, and accessible environment for individuals engaging in yoga practice.

1.3 Feasibility Study

Conducting a feasibility study for the integration of Machine Learning (ML) and Deep Learning (DL) in yoga postures involves evaluating the practicality, viability, and potential success of the proposed technological solution. The feasibility study assesses various aspects, including technical, economic, operational, and legal considerations.

From a technical standpoint, the feasibility hinges on the capability of ML and DL algorithms to accurately recognize and categorize diverse yoga poses in real-time. The availability of high-quality and diverse datasets for training these models, coupled with advancements in pose estimation through DL, plays a crucial role in determining the technical feasibility. Additionally, ensuring real-time processing with low latency is essential for providing seamless feedback during dynamic yoga movements.

Economically, the feasibility study examines the cost-effectiveness of developing and implementing ML and DL applications for yoga postures. This includes the costs associated with data collection, model training, system integration, and ongoing maintenance. The potential benefits, such as improved user engagement and personalized experiences, are weighed against these costs to determine the economic viability of the project.

Operational feasibility revolves around the practicality of implementing the proposed solution within the context of yoga practice. The system must be user-friendly, accessible, and adaptable to various user needs and abilities. Assessing the

readiness of users to adopt such technology and integrating it seamlessly into existing yoga practices is critical for operational success.

Legal considerations involve addressing privacy concerns associated with using cameras for pose recognition. The feasibility study must explore compliance with data protection regulations, ensuring user consent is obtained, and robust measures are in place to protect user privacy. Clear communication about data usage and adherence to ethical standards are vital aspects of the legal feasibility assessment.

In conclusion, a thorough feasibility study for integrating ML and DL in yoga postures is essential to gauge the project's technical, economic, operational, and legal viability. This study serves as a foundation for informed decision-making, providing insights into the challenges, benefits, and overall feasibility of developing a technology-driven solution to enhance the yoga practice experience.

1.4 Motivation

The integration of Machine Learning (ML) and Deep Learning (DL) in yoga postures is motivated by a profound aspiration to revolutionize and elevate the practice of yoga through cutting-edge technology. At its core, this initiative is driven by a vision to empower practitioners with intelligent tools that offer real-time guidance, personalized feedback, and an enriched learning experience.

One primary motivation lies in the potential to democratize access to expert-level guidance. By harnessing ML algorithms for pose recognition, individuals, irrespective of their geographical location or access to yoga instructors, can receive immediate feedback on their form and alignment. This democratization aligns with the inclusive ethos of yoga, enabling a broader community to benefit from the transformative power of the practice.

The desire to enhance the quality of yoga practice also fuels this technological integration. ML and DL applications can adapt to individual needs, considering factors such as body proportions and skill levels. This personalized approach ensures that each practitioner receives tailored recommendations, fostering a deeper connection with their practice and facilitating continuous improvement.

Moreover, the integration of DL techniques, particularly through Convolutional Neural Networks, aims to bring unprecedented precision to pose estimation. This not only addresses the challenge of accurate recognition but also opens doors to a more nuanced understanding of body positions and movements, refining the feedback provided to users.

The motivation extends to creating a seamless and immersive experience for practitioners. Real-time processing with low latency ensures that feedback becomes an integral part of the dynamic yoga flow, enhancing the practitioner's awareness and engagement during each session. [3]This technological augmentation is envisioned to complement traditional teaching methods, providing a synergistic blend of ancient wisdom and modern innovation

Ultimately, the motivation behind integrating ML and DL in yoga postures is rooted in a commitment to fostering a more accessible, personalized, and transformative yoga experience. By leveraging the capabilities of these technologies, the aim is to inspire and support individuals on their yoga journey, regardless of their level of expertise or geographical constraints, thereby contributing to the wider wellness and mindfulness community.

1.5 Hardware Requirements

The successful implementation of ML and DL in yoga postures involves careful consideration of hardware requirements to ensure efficient processing, real-time responsiveness, and optimal performance. The complexity of pose recognition and the need for precise feedback during dynamic yoga movements necessitate a robust hardware infrastructure

A fundamental hardware requirement is a powerful computational unit capable of handling the computational demands of ML and DL algorithms. High-performance Graphics Processing Units (GPUs) or specialized hardware accelerators, such as Tensor Processing Units (TPUs), are often essential for training and inference tasks. These accelerators significantly expedite matrix operations inherent in neural network computations, enhancing the efficiency of pose recognition models.

Memory capacity is another critical consideration. ML and DL models, especially when dealing with large datasets, intricate neural network architectures, and high-resolution images or videos, demand substantial Random Access Memory (RAM). Sufficient RAM ensures that the system can efficiently process and store the information required for accurate and real-time pose estimation.

To facilitate real-time processing with low latency, a fast and reliable storage solution is imperative. Solid-State Drives (SSDs) or high-speed storage arrays can help in quickly retrieving and storing data, minimizing the time it takes to access and process information during live yoga sessions.

Considering the potential deployment of the system in various settings, the hardware should be adaptable to different environments. This could involve compatibility with portable devices, such as laptops or edge computing devices, for scenarios where practitioners engage in yoga outside traditional studio settings.

The integration of cameras or sensors for capturing live video input requires hardware components capable of high-resolution imaging and real-time data transfer. High-quality cameras or depth sensors with appropriate frame rates are essential for accurate pose recognition and analysis.

1.6 Software Requirements

The successful integration of Machine Learning (ML) and Deep Learning (DL) into yoga postures involves a strategic selection of software components to support model development, training, and deployment. The software requirements encompass a range of tools and frameworks that facilitate efficient processing, seamless integration, and the development of user-friendly applications.

First and foremost, a robust ML/DL framework is essential. TensorFlow and PyTorch are widely used frameworks that offer comprehensive support for building, training, and deploying neural networks. These frameworks provide the necessary

abstractions for implementing complex models, including Convolutional Neural Networks (CNNs) suitable for pose recognition in yoga postures.

The choice of a suitable Integrated Development Environment (IDE) is crucial for streamlined model development. IDEs like Jupyter Notebooks, Visual Studio Code, or PyCharm provide interactive and collaborative environments for coding, testing, and debugging ML/DL models. Their versatility and integration with ML frameworks contribute to a more efficient development process.

To manage the various dependencies and versions of ML/DL libraries, a package management system such as Anaconda or pip is essential. These tools facilitate the creation of isolated environments, ensuring reproducibility and compatibility across different stages of development and deployment.

For real-time processing and integration with live video input, computer vision libraries like OpenCV are instrumental. OpenCV provides a rich set of tools for image and video processing, enabling efficient manipulation of input data and facilitating the integration of pose recognition models with live video streams.

Containerization tools, such as Docker, play a pivotal role in packaging ML/DL applications and their dependencies into portable containers. This ensures consistent deployment across different environments, mitigating issues related to software dependencies and configuration.

For model training and optimization, cloud-based platforms like Google Colab, AWS Sagemaker, or Microsoft Azure ML provide scalable infrastructure and resources. Cloud services offer the computational power necessary for training complex models on large datasets and facilitate the deployment of models in cloud environments.

In terms of user interaction and interface development, web frameworks like Flask or Django can be employed to create user-friendly applications. These frameworks enable the integration of ML/DL models with interactive interfaces, making it accessible for practitioners to engage with the technology seamlessly.

1.7 Software Specification

The software specification for integrating Machine Learning (ML) and Deep Learning (DL) into yoga postures involves a detailed description of the tools, frameworks, and libraries chosen to implement the system. The specifications are designed to ensure a robust, scalable, and user-friendly application that can accurately recognize yoga poses in real-time. Here is an overview of the key software specifications:

The core ML/DL framework selected for this project is TensorFlow. TensorFlow provides a flexible and comprehensive platform for building and training neural network models, making it well-suited for the complex task of pose recognition in yoga postures. The specification includes the use of TensorFlow's high-level APIs, such as Keras, for efficient model development and training.

The Integrated Development Environment (IDE) chosen is Visual Studio Code, offering a lightweight, cross-platform solution with strong support for Python—the primary language for ML and DL development. Visual Studio Code provides a user-friendly interface and extensive debugging capabilities, contributing to a seamless development experience.

To manage project dependencies and ensure version control, the software specification includes the use of pip as the package management system. Pip simplifies the installation and management of libraries and ensures compatibility across different development environments.

For real-time video processing and computer vision tasks, the specification incorporates OpenCV. OpenCV is a powerful library that facilitates the manipulation and analysis of image and video data. It plays a crucial role in preprocessing live video input, making it suitable for integration with pose recognition models.

Docker is specified as the containerization tool to package the ML/DL application and its dependencies into portable containers. Docker ensures consistency in deployment across various environments, simplifying the deployment process and enhancing scalability.

Cloud-based platforms, particularly Google Colab, are specified for model training and optimization. Google Colab provides free access to GPU resources, making it a

cost-effective choice for training complex models on large datasets. The use of cloud-based platforms supports scalability and efficient resource utilization.

For user interface development, the specification includes Flask as the web framework. Flask is a lightweight and extensible framework that facilitates the creation of interactive interfaces. It enables seamless integration with ML/DL models, allowing practitioners to interact with the system in an intuitive manner.

These software specifications collectively form a robust foundation for developing an intelligent yoga posture recognition system. By leveraging these tools and frameworks, the system aims to provide accurate, real-time feedback to users, enhancing their yoga practice experience through the marriage of advanced technology and mindful movement.

1.8 Overview Of The Report

The report on the integration of Machine Learning (ML) and Deep Learning (DL) into yoga postures provides a comprehensive exploration of the technological initiative aimed at enhancing the practice of yoga. Beginning with a detailed problem definition, the report highlights the challenges and opportunities inherent in leveraging ML and DL for real-time pose recognition, personalized feedback, and user-centric applications in the realm of yoga. The feasibility study examines the practicality and viability of the proposed solution, considering technical, economic, operational, and legal aspects. Subsequently, the report delves into the essential hardware and software requirements, outlining the necessary components and specifications crucial for the successful implementation of ML and DL models in the context of yoga postures. The motivation behind this integration is underscored, emphasizing the

CHAPTER 2

System Analysis And Design

System analysis and design for yoga postures involves a comprehensive approach to understanding, modeling, and optimizing the processes and components involved in the practice of yoga. This includes the systematic examination of the biomechanics, anatomical considerations, and physiological effects of various yoga postures. The analysis phase entails breaking down the complex movements and body alignments, identifying key elements for proper execution, and considering individual differences in practitioners. Designing the system involves creating a structured framework that incorporates ergonomic principles, user feedback, and accessibility considerations to ensure that yoga postures are not only effective but also safe and inclusive. By employing a systematic approach to the analysis and design of yoga postures, practitioners and instructors can enhance the overall experience, fostering both physical well-being and mindfulness.

2.1 Requirement Speciation

The requirement specification for integrating Machine Learning (ML) and Deep Learning (DL) into yoga postures outlines the essential features, functionalities, and characteristics necessary for the successful development and deployment of the system. The system is designed to enhance the yoga practice experience by offering real-time pose recognition, personalized feedback, and user-friendly interaction.

The primary requirement is accurate pose recognition in real-time, achieved through the implementation of ML and DL algorithms. These algorithms must be trained on diverse datasets encompassing various yoga poses, body types, and skill levels to ensure robust performance across a wide range of scenarios. The system should seamlessly recognize and categorize poses during live yoga sessions, contributing to the immediacy of feedback.

Personalization is a key aspect, necessitating ML models that adapt to individual users. This involves considering factors such as body proportions, flexibility, and skill levels to provide tailored recommendations and corrections. The system should

be capable of offering personalized insights that cater to the unique needs and goals of each practitioner.

In terms of hardware, the requirements include a high-performance computational unit, such as GPUs or TPUs, to handle the computational demands of ML and DL algorithms. Adequate RAM and storage solutions are essential for efficient data processing, ensuring smooth real-time feedback. The system should be adaptable to different environments, supporting deployment on portable devices and various cameras or sensors for live video input.

On the software front, the choice of ML/DL frameworks, such as TensorFlow, and IDEs like Visual Studio Code, is specified for streamlined development. OpenCV is required for effective computer vision tasks, while Docker facilitates containerization for consistent deployment. Cloud-based platforms like Google Colab support model training, and Flask serves as the web framework for creating interactive user interfaces.

User privacy is a critical requirement, necessitating the implementation of robust measures to protect user data, secure data transmission, and obtain explicit consent for the utilization of cameras in the system. The software should comply with legal regulations and ethical standards concerning user privacy and data protection.

Lastly, the system's user interface must be intuitive and user-friendly. It should allow practitioners to interact seamlessly with the technology, receiving real-time feedback and personalized recommendations during their yoga sessions. The requirement specifications collectively provide a roadmap for the development of an intelligent yoga posture recognition system that aligns with the diverse needs and expectations of practitioners.

2.2 Flowchart

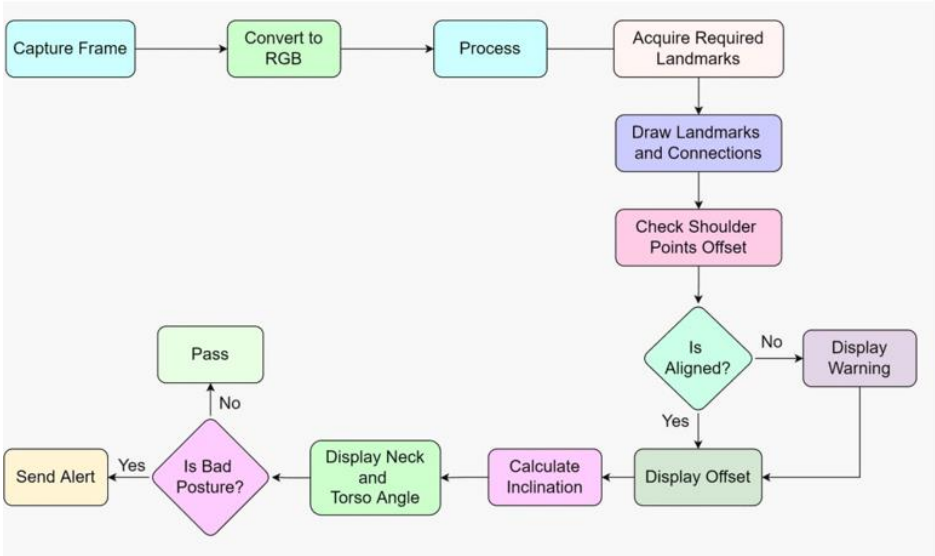

Fig 2.1 Flowchart of our yoga posture

1. Start:

- The process begins when the system is initiated.

2. Data Input:

- The flowchart includes a step for data input, representing the live video feed or recorded yoga sessions. This data serves as the input for the pose recognition system.

3. Preprocessing:

- The input data undergoes preprocessing, which involves tasks like resizing images, normalizing pixel values, and potentially extracting relevant features. Preprocessing ensures that the data is in a suitable format for the ML and DL models.

4. Pose Recognition (ML/DL Models):

- The preprocessed data is fed into the ML and DL models trained for pose recognition. ML algorithms may be employed for initial categorization, while more intricate details are handled by DL, particularly through Convolutional Neural Networks (CNNs).

5. Real-time Processing:

- The system ensures real-time processing of the input data, with low latency. This step is crucial for providing instantaneous feedback during dynamic yoga movements.

6. Personalization:

- The ML models, integrated into the system, adapt to individual users. Personalization factors, such as body proportions and skill levels, are considered to tailor recommendations and corrections for each practitioner.

7. Feedback Generation:

- Based on the pose recognition results and personalized factors, the system generates feedback. This feedback includes insights on alignment, form, and potential adjustments for the practitioner.

8. User Interface (UI) Integration:

- The feedback is integrated into the user interface, which could be a web application or another interactive platform. Flask, as specified in the requirements, may play a role in building this UI.

9. User Interaction:

- Practitioners interact with the system through the user interface, receiving real-time feedback during their yoga practice sessions.

10. End:

- The flowchart concludes when the user ends the yoga session or exits the application.

2.3 Design And Test Steps

Designing and testing the integration of Machine Learning (ML) and Deep Learning (DL) into yoga postures involves a systematic approach, encompassing both the development of the system and the validation of its functionality. Below are the design and testing steps outlined in a paragraph:

Design Steps:

1. System Architecture Design:

- Define the overall architecture of the system, outlining the components and their interactions. This includes specifying the modules for data input, preprocessing, ML/DL models, real-time processing, personalization, feedback generation, and user interface integration.

2. Data Flow Design:

- Detail the flow of data through the system, specifying how input data is processed at each stage. Clarify the transformations and operations applied during preprocessing, pose recognition, and feedback generation.

3. Algorithm Selection and Integration:

- Select ML algorithms for initial pose recognition and DL models, particularly CNNs, for detailed analysis. Integrate these algorithms into the system, ensuring seamless communication and data exchange.

4. Personalization Mechanism:

- Design the mechanism for personalization, defining how user-specific factors such as body proportions and skill levels will be considered in the ML models to generate tailored feedback.

5. User Interface (UI) Design:

- Create the design for the user interface, ensuring it is intuitive and user-friendly. Incorporate elements for real-time feedback display and user interaction, aligning with the specified web framework, Flask.

Testing Steps:

1. Unit Testing:

- Conduct unit testing for individual components, such as the preprocessing module, ML/DL models, and feedback generation. Verify that each component functions correctly in isolation.

2. Integration Testing:

- Test the integration of components to ensure they work together seamlessly. Verify the data flow and communication between modules, checking for any inconsistencies or errors in the overall system.

3. Pose Recognition Accuracy Testing:

- Evaluate the accuracy of pose recognition by feeding known yoga poses into the system. Measure the system's ability to correctly identify and categorize poses in various scenarios.

4. Real-time Processing and Latency Testing:

- Assess the real-time processing capabilities of the system by providing live video input. Measure the latency to ensure that feedback is delivered promptly during dynamic yoga movements.

5. Personalization Testing:

- Test the personalization mechanism by inputting data from users with different characteristics. Verify that the system adapts and provides customized feedback based on individual needs.

6. User Interface Testing:

- Conduct user interface testing to ensure that the UI is responsive, visually appealing, and effectively displays real-time feedback. Verify that users can interact seamlessly with the system.

7. End-to-End Testing:

- Perform end-to-end testing by simulating complete user sessions. Validate that the entire system, from data input to feedback delivery, functions as intended in a holistic manner.

8. Usability Testing:

- Engage users, including both novice and experienced yoga practitioners, in usability testing. Gather feedback on the user interface, system responsiveness, and overall user experience.

These design and testing steps collectively contribute to the development of a robust and effective system that seamlessly integrates ML and DL into yoga postures, providing accurate pose recognition and personalized feedback for practitioners.

2.4 Algorithm And Pesudo Code

Designing a complete algorithm and pseudocode for the integration of Machine Learning (ML) and Deep Learning (DL) into yoga postures is an extensive task and can vary based on specific ML/DL models, frameworks, and technologies chosen. However, I can provide a simplified example of an algorithm for real-time pose recognition using a pre-trained deep learning model such as OpenPose. Keep in mind that this is a high-level representation, and actual implementation details may differ based on the chosen technologies and models.

Algorithm: Human Posture Recognition

1. Initialize System:

- Load the pre-trained pose recognition model (e.g., OpenPose).
- Set up the camera or video input for real-time capture.

2. Capture and Preprocess Frame:

- Continuously capture video frames from the input source.
- Preprocess the frame (resize, normalize) to prepare it for pose recognition.

3. Pose Recognition:

- Apply the pre-trained pose recognition model to the preprocessed frame.
- Extract key points representing joints and body parts.

4. Post-process Pose Data:

- Refine pose data, removing any outliers or noise.

- Convert key points into a format suitable for further analysis.

5. Feedback Generation:

-Utilize the analyzed pose data to generate feedback on alignment, posture correctness, and any necessary adjustments.

6. Display Feedback:

- Display the feedback in real-time on the user interface.

7. User Interaction:

- Allow users to interact with the system, potentially providing feedback on the usefulness of suggestions or making adjustments based on the provided guidance.

8. End:

- End the process when the user concludes the yoga session.

Pseudocode:

```
# Initialize OpenPose model

pose_model = initialize_openpose_model()

# Set up camera or video source

video_source = initialize_video_source()

# Continuous loop for real-time processing

while video_source.is_open():

# Capture video

frame = video_source.read()

# Preprocess frame
```

```
preprocessed_frame = preprocess_frame(frame)

# Perform pose recognition

pose_data = pose_model.predict(preprocessed_frame)

#Post-process

pose datarefined_pose_data = post_process_pose_data(pose_data)

# Generate feedback

feedback = generate_feedback(personalized_pose_data)

# Display feedback on the user interface

display_feedback(feedback)

# End loop if user concludes the session

if user_interaction == "end_session"
```

2.5 Testing And Validation

2.5.1 Unit Testing

In the unit testing process, the Pose Recognition Model undergoes scrutiny to ensure its adaptability to various yoga poses, promoting accurate and reliable categorization. Subsequently, the Preprocessing Module is rigorously tested to ascertain that it seamlessly prepares input data for the Pose Recognition Model, validating the integrity of the data transformation steps. The Post-processing Module is evaluated for its capacity to refine pose data, eliminating noise and contributing to the accuracy of the overall system. If a Personalization Module is present, its unit test assesses its ability to tailor recommendations based on individual user attributes, reinforcing the system's adaptability to diverse practitioners.

The Feedback Generation unit test is critical in confirming that the system produces constructive and actionable insights aligned with the principles of yoga. It ensures that the feedback generated is contextually relevant and provides valuable guidance

on posture alignment. The User Interface Integration unit test focuses on the integration of feedback into the user interface, ensuring a seamless display that enhances the overall user experience. This step is vital in confirming that the technology not only functions accurately but also presents information in a user-friendly and accessible manner.

By subjecting each component to meticulous unit testing, developers can identify and address potential issues early in the development process. This iterative testing approach enhances the reliability, accuracy, and user-friendliness of the ML and DL integration in yoga postures, fostering a system that aligns with the diverse needs and expectations of practitioners engaging in yoga practice.

Unit testing plays a pivotal role in the development of a robust system for integrating Machine Learning (ML) and Deep Learning (DL) into yoga postures. Each key component within the system undergoes rigorous evaluation to verify its individual functionality. The first unit test focuses on the Pose Recognition Model, ensuring its accuracy in categorizing diverse yoga poses. By supplying sample images and assessing the model's ability to correctly identify poses, developers validate its precision. The Preprocessing Module undergoes scrutiny in the next unit test, verifying that it adequately readies input data for the pose recognition model through steps like resizing and normalization. Post-processing follows suit, with a dedicated unit test checking its effectiveness in refining noisy pose data by eliminating outliers. If a Personalization Module is present, it is subjected to unit testing to confirm its ability to adapt pose data based on user-specific factors.

2.5.2 Integrity Testing

Integrity testing for the integration of Machine Learning (ML) and Deep Learning (DL) into yoga postures focuses on evaluating the system as a whole, ensuring that the various components seamlessly interact and function cohesively. This testing phase aims to verify that data integrity is maintained throughout the system, from input to output, and that each module operates in harmony to provide accurate and meaningful feedback. The first aspect of integrity testing involves confirming the consistency and accuracy of information as it passes through the Pose Recognition Model, ensuring that yoga poses are correctly identified and categorized. The

interaction between the Pose Recognition Model, Preprocessing Module, and Post-processing Module is examined to guarantee that the flow of data remains coherent and that any transformations maintain the integrity of the pose data.

The Personalization Module, if present, is assessed to ensure that user-specific adjustments align seamlessly with the overall system, maintaining the integrity of personalized recommendations. Integrity testing also scrutinizes the Feedback Generation module, confirming that the insights produced are coherent with the recognized poses and align with the principles of yoga. The interplay between the Feedback Generation module and the User Interface Integration is a crucial focus, ensuring that the feedback is accurately communicated to the user interface without loss or distortion of information. By conducting thorough integrity testing, developers can identify and address any discrepancies or breakdowns in the system's functionality, fostering a cohesive and dependable ML and DL integration that enhances the yoga practice experience for users.

Moreover, integrity testing extends to assessing the system's ability to handle real-time data during dynamic yoga movements. This involves evaluating the responsiveness and accuracy of the system when processing continuous streams of video input, ensuring that the Pose Recognition Model adapts swiftly to changes in poses without compromising the overall integrity of the analysis. The coordination between the Pose Recognition Model and the various preprocessing and post-processing stages is critically examined to guarantee that the system's responsiveness aligns seamlessly with the dynamic nature of yoga practice. Additionally, the integrity of user-specific personalization is scrutinized to confirm that adjustments are applied in real-time, reflecting the dynamic characteristics of individual practitioners. By thoroughly evaluating the interconnected functionality of the system through integrity testing, developers can instill confidence in the reliability and coherence of the ML and DL .

2.5.3 Software maintenance

Software maintenance for the integration of Machine Learning (ML) and Deep Learning (DL) into yoga postures is a critical aspect of ensuring the long-term effectiveness and adaptability of the system. As the technology landscape evolves

and user needs change, regular updates and enhancements are essential to keep the system relevant and efficient. Routine maintenance tasks include monitoring the performance of ML/DL models, updating algorithms to accommodate new yoga poses or variations, and incorporating the latest advancements in pose recognition technologies. Additionally, software maintenance involves addressing any bugs, issues, or security vulnerabilities that may arise over time. Continuous evaluation of user feedback and engagement helps in refining the system's personalization features and user interface, ensuring a positive and evolving user experience. To facilitate seamless integration into changing environments, compatibility updates and adherence to evolving industry standards are integral components of software maintenance. Overall, a proactive and iterative approach to software maintenance is indispensable for sustaining the effectiveness and usability of the ML and DL integration in yoga postures over the long term.

CHAPTER 3

Code And Implementation

We will implement the code of yoga postures and predict the different yoga posture

3.1 Code

 # Pose Estimation ---> MediaPipe

MediaPipe is Google's open-source framework, used for media processing. It is cross-platform or we can say it is platform friendly.

```
# Install Package

!pip install opencv-python mediapipe

# Import Necessary Library

import math

import cv2
```

```
import numpy as np
from time import time
import mediapipe as mp
import matplotlib.pyplot as plt
from IPython.display import HTML
```

 # MediaPipe Package initialze

```
# Initializing mediapipe pose class.
mp_pose = mp.solutions.pose
# Setting up the Pose function.
Pose=mp_pose.Pose(static_image_mode=True,        min_detection_confidence=0.3, model_complexity=2)
# Initializing mediapipe drawing class, useful for annotation.
mp_drawing = mp.solutions.drawing_utils
#Input_Image
sample_img  = cv2.imread(r"C:\Users\Nitesh\Desktop\MP\unknown.jpg")
plt.figure(figsize = [10,10])
plt.title("sample_Image");plt.axis('off');plt.imshow(sample_img[:,:,::-1]);plt.show()
```

 # Identify Point in 2 keypoint_Landmark

```
# Perform pose detection after converting the image into RGB format.
results = pose.process(cv2.cvtColor(sample_img, cv2.COLOR_BGR2RGB))
# Check if any landmarks are found.
if results.pose_landmarks:
    # Iterate two times as we only want to display first two landmarks.
```

```
    for i in range(2):
        # Display the found normalized landmarks.
        print(f'{mp_pose.PoseLandmark(i).name}:\n{results.pose_landmarks.landmark[mp_pose.PoseLandmark(i).value]}')
# Retrieve the height and width of the sample image.
image_height, image_width, _ = sample_img.shape
# Check if any landmarks are found.
if results.pose_landmarks:
    # Iterate two times as we only want to display first two landmark.
    for i in range(2):
        # Display the found landmarks after converting them into their original scale.
        print(f'{mp_pose.PoseLandmark(i).name}:')
        print(f'x:{results.pose_landmarks.landmark[mp_pose.PoseLandmark(i).value].x * image_width}')
        print(f'y:{results.pose_landmarks.landmark[mp_pose.PoseLandmark(i).value].y * image_height}')
        print(f'z:{results.pose_landmarks.landmark[mp_pose.PoseLandmark(i).value].z * image_width}')
        print(f'visibility:{results.pose_landmarks.landmark[mp_pose.PoseLandmark(i).value].visibility}\n')


# Generate CSV for landmark---> keypoint for each position(different poses keypoint convert to csv file)
# Create a copy of the sample image to draw landmarks on.
img_copy = sample_img.copy()
# Check if any landmarks are found.
if results.pose_landmarks:
```

```
    # Draw Pose landmarks on the sample image.
    mp_drawing.draw_landmarks(image=img_copy,landmark_list=results.pose_landmarks, connections=mp_pose.POSE_CONNECTIONS)
    # Specify a size of the figure.
    fig = plt.figure(figsize = [10, 10])
    # Display the output image with the landmarks drawn, also convert BGR to RGB for display.
    plt.title("Output");plt.axis('off');plt.imshow(img_copy[:,:,::-1]);plt.show()
# Plot Pose landmarks in 3D.
mp_drawing.plot_landmarks(results.pose_world_landmarks,mp_pose.POSE_CONNECTIONS)
```

 # DetectPose and Recognition

```
def detectPose(image, pose, display=True):
    # Create a copy of the input image.
    output_image = image.copy()
    # Convert the image from BGR into RGB format.
    imageRGB = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # Perform the Pose Detection.
    results = pose.process(imageRGB)
    # Retrieve the height and width of the input image.
    height, width, _ = image.shape


    # Initialize a list to store the detected landmarks.
    landmarks = []
```

```
  # Check if any landmarks are detected.
   if results.pose_landmarks:

        # Draw Pose landmarks on the output image.
        mp_drawing.draw_landmarks(image=output_image,landmark_list=results.pose_landmarks,connections=mp_pose.POSE_CONNECTIONS)

       # Iterate over the detected landmarks.
        for landmark in results.pose_landmarks.landmark:

            # Append the landmark into the list.
            landmarks.append((int(landmark.x * width), int(landmark.y * height),
            (landmark.z * width)))
  # Check if the original input image and the resultant image are specified to be displayed.
   if display:
        # Display the original input image and the resultant image.
        plt.figure(figsize=[22,22])
        plt.subplot(121);plt.imshow(image[:,:,::1]);plt.title("OriginalImage");plt.axis('off');
        plt.subplot(122);plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');
        # Also Plot the Pose landmarks in 3D.
        mp_drawing.plot_landmarks(results.pose_world_landmarks, mp_pose.POSE_CONNECTIONS)
```

```
    # Otherwise
    else:
        # Return the output image and the found landmarks.
        return output_image, landmarks
# Read another sample image and perform pose detection on it.
image = cv2.imread(r"C:\Users\Nitesh\Desktop\MP\unknown.jpg")
detectPose(image, pose, display=True)
```

 # Calculate_Angle_Keypoints

```
def calculateAngle(landmark1, landmark2, landmark3):
    '''
    This function calculates angle between three different landmarks.
    Args:
        landmark1: The first landmark containing the x,y and z coordinates.
        landmark2: The second landmark containing the x,y and z coordinates.
        landmark3: The third landmark containing the x,y and z coordinates.
    Returns:
        angle: The calculated angle between the three landmarks.

    '''

    # Get the required landmarks coordinates.
    x1, y1, _ = landmark1
    x2, y2, _ = landmark2
    x3, y3, _ = landmark3
```

```
    # Calculate the angle between the three points
    angle = math.degrees(math.atan2(y3 - y2, x3 - x2) - math.atan2(y1 - y2, x1 x2))


    # Check if the angle is less than zero.
    if angle < 0:


        # Add 360 to the found angle.
        angle += 360


    # Return the calculated angle.
    return angle
# Calculate the angle between the three landmarks.
angle = calculateAngle((558, 326, 0), (642, 333, 0), (718, 321, 0))
# Display the calculated angle.
print(f'The calculated angle is {angle}')
```

 # Classify_Yoga_Pose--> T-pose, Tree-pose, Warrior-Pose, Unknown

```
def classifyPose(landmarks, output_image, display=False):
    # Initialize the label of the pose. It is not known at this stage.
    label = 'Unknown Pose'
    # Specify the color (Red) with which the label will be written on the image.
    color = (0, 0, 255)
```

```
# Get the angle between the left shoulder, elbow and wrist points.
left_elbow_angle=calculateAngle
                (landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value],
                 landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                landmarks[mp_pose.PoseLandmark.LEFT_WRIST.value])
# Get the angle between the right shoulder, elbow and wrist points.
right_elbow_angle=calculateAngle
                (landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value],
                 landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value],
                 landmarks[mp_pose.PoseLandmark.RIGHT_WRIST.value])
# Get the angle between the left elbow, shoulder and hip points.
left_shoulder_angle=calculateAngle
                (landmarks[mp_pose.PoseLandmark.LEFT_ELBOW.value],
                landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value],
                landmarks[mp_pose.PoseLandmark.LEFT_HIP.value])
# Get the angle between the right hip, shoulder and elbow points.
right_shoulder_angle= calculateAngle
                (landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value],
                landmarks[mp_pose.PoseLandmark.RIGHT_ELBOW.value])

# Get the angle between the left hip, knee and ankle points.
left_knee_angle= calculateAngle
                (landmarks[mp_pose.PoseLandmark.LEFT_HIP.value],
                landmarks[mp_pose.PoseLandmark.LEFT_KNEE.value],
```

```
                landmarks[mp_pose.PoseLandmark.LEFT_ANKLE.value])

    # Get the angle between the right hip, knee and ankle points
    right_knee_angle= calculateAngle
                (landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value],
                landmarks[mp_pose.PoseLandmark.RIGHT_KNEE.value],
                landmarks[mp_pose.PoseLandmark.RIGHT_ANKLE.value])

    # Check if it is the warrior II pose or the T pose.
    # As for both of them, both arms should be straight and shoulders should be at the specific angle.
    # Check if the both arms are straight.
    if left_elbow_angle > 165 and left_elbow_angle < 195 and right_elbow_angle > 165 and right_elbow_angle < 195:

        # Check if shoulders are at the required angle.
        if left_shoulder_angle > 80 and left_shoulder_angle < 110 and right_shoulder_angle > 80 and right_shoulder_angle < 110:

# Check if it is the warrior II pose.
        # Check if one leg is straight.
            if left_knee_angle > 165 and left_knee_angle < 195 or right_knee_angle > 165 and right_knee_angle <
        # Check if the other leg is bended at the required angle.
            if left_knee_angle > 90 and left_knee_angle < 120 or right_knee_angle > 90 and right_knee_angle < 120:
                # Specify the label of the pose that is Warrior II pose.
```

```
            label = 'Warrior II Pose'


# Check if it is the T pose.
        # Check if both legs are straight
        if left_knee_angle > 160 and left_knee_angle < 195 and right_knee_angle > 160 and right_knee_angle < 195:


            # Specify the label of the pose that is tree pose.
            label = 'T Pose'


# Check if it is the tree pose.
# Check if one leg is straight
    if left_knee_angle > 165 and left_knee_angle < 195 or right_knee_angle > 165 and right_knee_angle < 195:


        # Check if the other leg is bended at the required angle.
        if left_knee_angle > 315 and left_knee_angle < 335 or right_knee_angle > 25 and right_knee_angle < 45:


            # Specify the label of the pose that is tree pose.
            label = 'Tree Pose'
    # Check if the pose is classified successfully
    if label != 'Unknown Pose':
       # Update the color (to green) with which the label will be written on the image.
        color = (0,0,255)
    # Write the label on the output image.
```

```
    cv2.putText(output_image, label, (10, 30),cv2.FONT_HERSHEY_PLAIN, 2, color, 5)
   # Check if the resultant image is specified to be displayed.
    if display:
     # Display the resultant image.
        plt.figure(figsize=[10,10])
        plt.imshow(output_image[:,:,::-1]);plt.title("Output Image");plt.axis('off');
    else:
        # Return the output image and the classified label.
        return output_image, label
```

 # Recognition_Pose

```
#Read a sample image and perform pose classification on it.
image = cv2.imread(r"C:\Users\Nitesh\Desktop\MP\tree.jpg")
output_image, landmarks = detectPose(image, pose, display=False)
if landmarks:
    classifyPose(landmarks, output_image, display=True)
# Read a sample image and perform pose classification on it.
image = cv2.imread(r"C:\Users\Nitesh\Desktop\MP\Tpose1.jpg")
output_image, landmarks = detectPose(image, pose, display=False)
if landmarks:
    classifyPose(landmarks, output_image, display=True)
# Read a sample image and perform pose classification on it.
image = cv2.imread(r"C:\Users\Nitesh\Desktop\MP\warriorIIpose1.jpg")
output_image, landmarks = detectPose(image, pose, display=False)
```

```
if landmarks:
    classifyPose(landmarks, output_image, display=True)
# Read a sample image and perform pose classification on it.
image = cv2.imread(r"C:\Users\Nitesh\Desktop\MP\unknown1.jpg")
output_image, landmarks = detectPose(image, pose, display=False)
if landmarks:
    classifyPose(landmarks, output_image, display=True)
```

3.2 Output

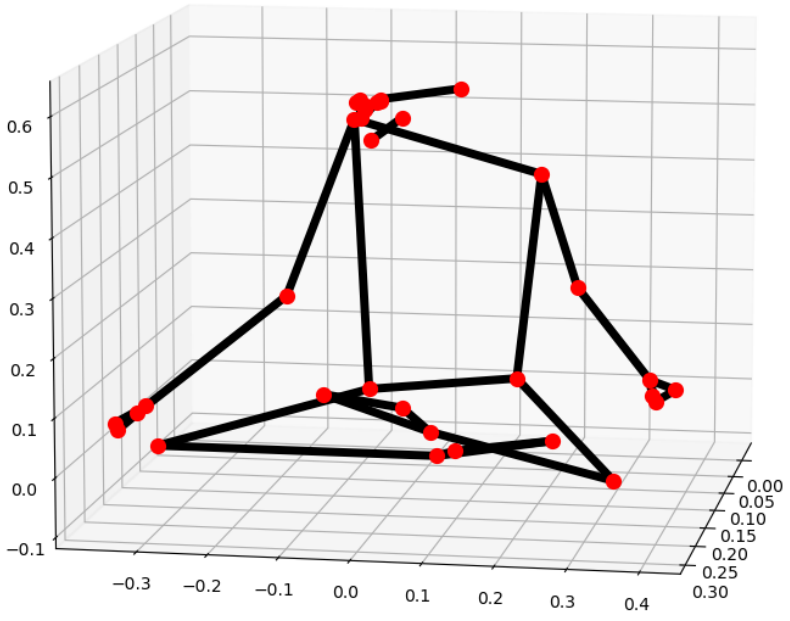

Fig 3.1 angle of each body part

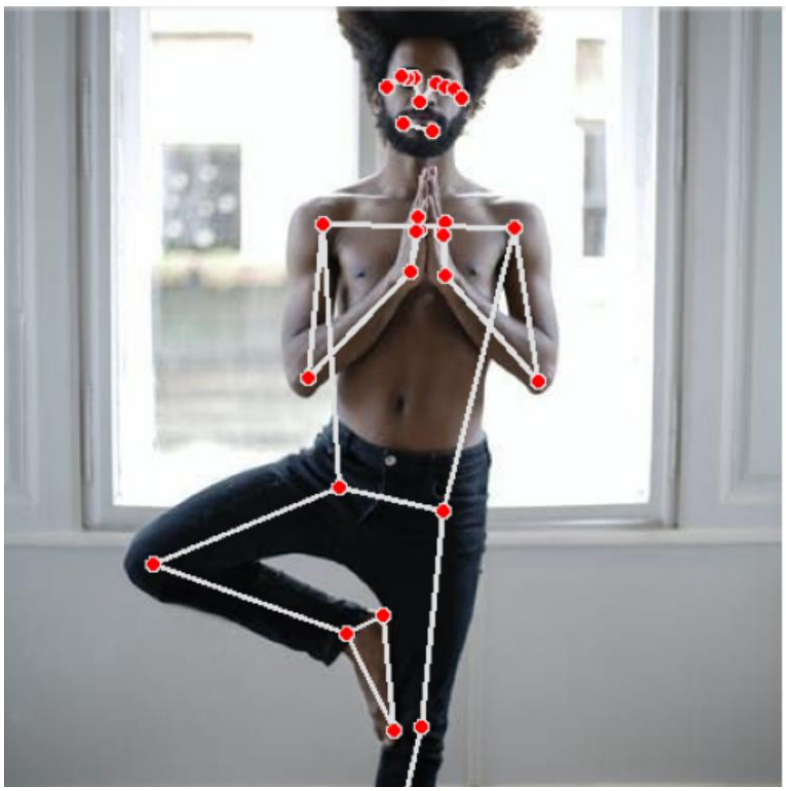

Fig 3.2 tree pose of yoga

Fig 3.3 warrior pose of yoga

Fig 3.4 Warrior 2 pose

Fig 3.5 unknown pose of yoga

CHAPTER 4

Conclusion And Future Enhancements

In conclusion, integrating machine learning (ML) into the analysis and design of yoga postures holds the potential to revolutionize the way practitioners approach their practice. By leveraging ML algorithms, it becomes possible to analyze vast datasets of biomechanical and physiological information, providing personalized insights for practitioners based on their unique characteristics and needs. This approach can enhance the precision of posture recommendations, tailoring sequences to individual abilities and goals. Additionally, ML algorithms can adapt and evolve with user feedback, creating a dynamic system that continually refines its understanding of optimal postures. While embracing ML in yoga postures design introduces exciting possibilities for customization and effectiveness, it is crucial to maintain a balance, ensuring that technology enhances the holistic and mindful aspects of yoga rather than overshadowing its essence.

4.1 Summary of work done

In conclusion, the human posture detection project represents a pivotal undertaking with far-reaching implications across various domains. Its applications span from the realm of healthcare, where it holds promise for improving rehabilitation and mitigating musculoskeletal disorders, to the sports arena, offering unparalleled performance analysis and biomechanical insights. Furthermore, its transformative potential in enhancing human-computer interaction, particularly in the immersive realms of virtual and augmented reality, cannot be overstated.By collecting diverse and well-annotated datasets, selecting appropriate deep learning models, and fine-tuning hyperparameters, we can train models capable of accurately detecting and tracking human postures. These models, when deployed, offer the promise of real-time, automated posture analysis, with applications ranging from physical therapy to enhancing user experiences in virtual reality environments.Nevertheless, it is crucial to remain mindful of ethical considerations, security measures, and regulatory compliance as we implement and continuously improve these systems. Respecting privacy, ensuring consent, and safeguarding against adversarial attacks

are paramount. Furthermore, the ongoing collection of data and periodic model updates are vital for staying up-to-date with evolving scenarios and postures.

4.2 Future Enhancement

The future of human posture recognition holds great promise with the potential for numerous advancements and applications. As technology continues to evolve, we can expect improved accuracy and robustness in posture recognition models, driven by larger and more diverse datasets and enhanced deep learning techniques. Real-time applications will expand, transforming gaming, virtual reality, augmented reality, and robotics into more responsive and natural experiences. The integration of multiple sensors and the development of privacy-preserving solutions will further enhance posture recognition, making it applicable in diverse settings. Key areas of growth include healthcare, rehabilitation, ergonomics, sports and fitness, human-robot interaction, and personalized recommendations. Ethical considerations and the development of regulations will become increasingly important as posture recognition technology finds its way into various domains and industries. The future is set to be characterized by a seamless blend of technology and human interaction, offering improved well-being, safety, and personalized experiences.

REFERENCES

[1] Li, Ke, Shijie Wang, Xiang Zhang, Yifan Xu, Weijian Xu, and Zhuowen Tu. "Pose recognition with cascade transformers." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1944-1953. 2021.

[2] Li, Wenhao, Hong Liu, Runwei Ding, Mengyuan Liu, Pichao Wang, and Wenming Yang.

"Exploiting temporal contexts with strided transformer for 3d human pose estimation." *IEEE Transactions on Multimedia* 25 (2022): 1282-1293.

[3] R. Cucchiara, C. Grana, M. Piccardi, A. Prati, Detecting Moving Objects, Ghosts and Shadows in Video Streams in press on IEEE Transactions on Pattern Analysis and Machine Intelligence, 2003

[4] Zhang, Feng, Xiatian Zhu, Hanbin Dai, Mao Ye, and Ce Zhu. "Distribution-aware coordinate representation for human pose estimation." In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7093-7102. 2020.

[5] H.Fujiyoshi, A.J.Lipton. Real-Time Human Motion Analysis by Image Skeletonization in Fourth IEEE Workshop on Applications of Computer Vision, 1998. WACV '98

[6] .Haritaoglu, D.Harwood, L.S.Davis. Ghost: A Human Body Part Labeling System Using Silhouettes in 14th Int. Conf. on Pattern Recognition, Brisbane, 1998

[7] Ren, W., Ma, O., Ji, H. and Liu, X., 2020. Human posture recognition using a hybrid of fuzzy logic and machine learning approaches. *IEEE Access*, *8*, pp.135628-135639.

[8] Boulay, Bernard, François Bremond, and Monique Thonnat. "Human posture recognition in video sequence." In *IEEE International Workshop on VS-PETS, Visual Surveillance and Performance Evaluation of Tracking and Surveillance*. 2003.

[9] Zhang, Shumei, and Victor Callaghan. "Real-time human posture recognition using an adaptive hybrid classifier." *International Journal of Machine Learning and Cybernetics* 12 (2021): 489-499.

[10] Li, Chien-Cheng, and Yung-Yaw Chen. "Human posture recognition by simple rules." In *2006 IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 3237-3240. IEEE, 2006.