	Table of Contents	
Chapter	TITLE	Page No
	Abstract	1
1	Introduction	2
2	Literature Survey	8
3	System Requirements	9
	3.1 Software Requirements	10
	3.2 Hardware Requirements	11
	3.3 Data Set	11
4	Propose Approach , Modules Description, and UML Diagrams	24
	4.1 Modules	29
	4.2 UML Diagrams	30
5	Implementation, Experimental Results &Test Cases	36
6	Conclusion and Future Scope	48
7	References	49

	Appendix i) Full Paper-Publication Proof ii) Snapshot Of the Result iii)Optional (Like Software Installation /Dependencies/ pseudo code)	

ABSTRACT

Detecting spinal cord diseases in early stages is crucial for minimizing their impact on individuals' health, potentially reducing the severity of symptoms and complications. Creating a system to detect and classify spinal conditions like kyphosis, lordosis, or normal posture, and then recommending yoga poses accordingly.To detect the posture we use OpenPose library which is an open-source library that detects human body keypoints from images or video frames. It provides coordinates for key body parts such as the nose, eyes, ears, shoulders, elbows, wrists, hips , lumbar region, knees, and ankles.We utilize sensors or imaging technologies to capture the posture of the human body. This data can include angles of various body parts, curvature of the spine, and other relevant metrics.

If a kyphosis or lordosis is detected using the posture analysis system, we can integrate a Yoga recommendation system to suggest appropriate yoga poses for each condition. Yoga recommendation system will detect the yoga pose in real time by using Posenet/Openpose and Convolutional Neural Network(CNN) Algorithm. Based on the classification result, it recommends appropriate yoga poses. This could involve mapping each posture type to a set of yoga poses known to be beneficial for that condition.

For kyphosis - poses that open up the chest and strengthen the back muscles, such as Cobra Pose (Bhujangasana) or Bridge Pose (Setu Bandhasana) .For lordosis - poses that stretch the hip flexors and strengthen the abdominal muscles, such as Cat-Cow Pose (MarjaryasanaBitilasana) or Child's Pose (Balasana).Pose Matching Algorithm compares the detected key points of the person's posture in real-time with the key points of the recommended yoga pose and gives feedback accordingly. As a result we can try to improve the spinal health of a person.

KEYWORDS: Spinal cord diseases, Kyphosis, Lordosis, Posture analysis, OpenPose, Yoga recommendation system, Posenet, Convolutional Neural Network(CNN), Yoga poses for kyphosis, Yoga poses for lordosis, Pose Matching Algorithm, Spinal health

1.INTRODUCTION

In the modern era, where we spend at a third of our lives in front of digital equipment, we are prone to develop issues with the posture of the spinal cord. This is very common problem trailing in teens as much as with older generation. The idea of the project is to find if there is an underlying issue with the spinal cord and if yes, then what is the disease. A normal check-up would require us to go to a hospital and have our x-ray done. However, recent advancements in machine learning and computer vision have paved the way for automated and reliable techniques to address posture detection, including the utilization of Convolutional Neural Networks (CNNs).

CNNs are highly effective in image processing as they can learn intricate patterns and features from visual data. To detect posture abnormalities using CNNs, we use Open Pose library which is an open-source library that detects human body key points from images or video frames. It provides coordinates for key body parts such as the nose, eyes, ears, shoulders, elbows, wrists, hips, lumbar region, knees, and ankles. We utilize sensors or imaging technologies to capture the posture of the human body. This data can include angles of various body parts, curvature of the spine, and other relevant metrics.

A Convolutional Neural Network (CNN) is a type of deep learning model particularly well-suited for analyzing visual data. Its architecture is inspired by the visual cortex of animals and is characterized by its use of convolutional layers, which apply filters to the input data to detect various features such as edges, textures, and shapes. These features are progressively combined through multiple layers to form high-level representations, enabling the network to recognize complex patterns and objects within images.

CNNs also incorporate pooling layers, which reduce the spatial dimensions of the data, thereby lowering computational requirements and enhancing the model's robustness to spatial variations. Fully connected layers at the end of the network integrate these features to perform classification or regression tasks. Due to their ability to automatically and adaptively learn spatial hierarchies of features, CNNs have become the cornerstone of many modern computer vision applications, including image and video recognition, object detection, and even some natural language processing tasks.

Kyphosis is a spinal disorder characterized by an excessive outward curvature of the thoracic spine, leading to a hunched or rounded back appearance. It can occur due to various reasons, including degenerative diseases such as arthritis, osteoporosis-related fractures, developmental issues during childhood or adolescence, or poor posture over time.

Lordosis is a condition where there is an excessive inward curvature of the lumbar spine, often referred to as swayback. This exaggerated curve can lead to a noticeable arch in the lower back, causing the abdomen and buttocks to protrude. Causes of lordosis include poor posture, obesity, muscular imbalances, and certain medical conditions such as spondylolisthesis or osteoporosis.

What is CNN (Convolutional Neural Network) Algorithm?

A CNN Algorithm is a subset of Machine Learning Algorithm. It is one of the various types of Artificial Neural Networks which is used for different applications and data types. It is a kind of network architecture for Deep learning and is specifically used for Image recognition. It is suitable for application involving natural language Processing (NLP), Language translation and speech recognition.

CNN Layers: There are three kind of CNN Layers viz.
- Convolutional layer.
- Pooling layer.
- Fully Connected Layer.

In conclusion, posture detection using CNNs offers a promising approach to identify kyphosis, lordosis, or hunch back by leveraging deep learning and computer vision.

1.1 Existing System

Creating a system to detect and classify spinal conditions like kyphosis, lordosis, or normal

posture, and then recommending yoga poses accordingly. To detect the posture we use **Open Pose** library which is an open-source library that detects human body key points from images or video frames. It provides coordinates for key body parts such as the nose, eyes, ears, shoulders, elbows, wrists, hips, lumbar region, knees, and ankles.

We utilize sensors or imaging technologies to capture the posture of the human body. This data can include angles of various body parts, curvature of the spine, and other relevant metrics.

If a kyphosis or lordosis is detected using the posture analysis system, we can integrate a Yoga recommendation system to suggest appropriate yoga poses for each condition. Yoga recommendation system will detect the yoga pose in real time by using **Posenet/Openpose** and **CNN Algorithm**.

Based on the classification result, it recommends appropriate yoga poses. This could involve mapping each posture type to a set of yoga poses known to be beneficial for that condition. For kyphosis - poses that open up the chest and strengthen the back muscles, such as Cobra Pose (Bhujangasana) or Bridge Pose (Setu Bandhasana).

For lordosis - poses that stretch the hip flexors and strengthen the abdominal muscles, such as Cat-Cow Pose (MarjaryasanaBitilasana) or Child's Pose (Balasana). **Pose Matching Algorithm** compares the detected key points of the person's posture in real-time with the key points of the recommended yoga pose and gives feedback accordingly. As a result, we can try to improve the spinal health of a person.

1.1.1 Limitations in Existing System

1.Pose Matching Algorithm Robustness:

Robustness to Variations:

The pose matching algorithm must accurately detect and analyze postures despite variations in body types, clothing, lighting conditions, and backgrounds. It should be able to handle different angles, occlusions, and partial visibility of body parts.

Continuous improvement and training of the algorithm with diverse datasets can enhance its robustness. Incorporating machine learning techniques to adapt to new and unseen data can help the algorithm generalize better.

2.Ethical and Privacy Considerations:

Data Privacy:

Ensure that all data collected from users, especially video feeds and posture data, are encrypted and stored securely. Only authorized personnel should have access to this data.

Implement data anonymization techniques to protect user identities.

Consent and Transparency:

Clearly inform users about how their data will be used, stored, and processed.

Obtain explicit consent before collecting any data.

Provide users with options to delete their data or withdraw consent at any time.

Confidentiality:

Follow stringent protocols to ensure that sensitive health information remains confidential. Regularly audit data access and usage to prevent unauthorized access.

Compliance with legal and regulatory standards, such as GDPR or HIPAA, is essential.

3.Clinical Validation and Integration:

Validation Studies:

Collaborate with healthcare professionals to design and conduct clinical studies that validate the system's effectiveness in real-world settings.

Collect and analyze data from these studies to identify areas of improvement and to ensure that the system provides reliable and accurate recommendations.

Integration with Healthcare Systems:

Work on integrating the system with existing healthcare infrastructures, such as electronic health records (EHRs), to provide seamless and comprehensive care.

Develop protocols for healthcare professionals to review and interpret the system's recommendations, ensuring that they align with clinical guidelines.

4.Accessibility and Usability:

User Interface Design:

Design the user interface to be intuitive and easy to navigate, ensuring that users with varying levels of technological proficiency can use the system without difficulty.

Incorporate features such as voice commands, visual aids, and step-by-step tutorials to enhance usability.

Accessibility Features:

Ensure that the system is accessible to individuals with disabilities. This can include support for screen readers, adjustable text sizes, and high-contrast themes.

Conduct usability testing with diverse user groups to identify and address potential barriers to accessibility.

5.Accuracy of Posture Analysis:

Key Point Detection:

The system should accurately detect key body points (e.g., shoulders, hips, knees) in various conditions, including poor lighting or when parts of the body are obscured.

Enhance the underlying algorithm (such as OpenPose) through continuous training with high-quality, diverse datasets.

Handling Complex Poses:

Develop methods to accurately analyze complex body poses that may involve overlapping limbs or non-standard postures.

Use advanced image processing techniques to mitigate issues arising from occlusions or low-resolution images.

6.Classification Model Performance:

Model Training:

Train the classification model on a comprehensive dataset that includes a wide range of spinal conditions and normal postures. Ensure that the dataset is balanced to prevent bias.

Evaluation Metrics:

Use metrics such as precision, recall, F1-score, and confusion matrices to evaluate the model's performance in distinguishing between different spinal conditions. Conduct cross-validation and test the model on unseen data to ensure that it generalizes well.

Continuous Improvement:

Implement a feedback loop where user feedback and new data are continuously incorporated into the training process to improve model accuracy over time. Regularly update the model with the latest medical research and clinical guidelines to maintain its relevance and accuracy.

2.LITERATURE SURVEY

Previous Works:

S.No	Author	Year	Objective and Outcome
1.	Yalan li,jaiangquan huan	2021	Human Pose Detection Based on Multi-Scale and Multi-Stage Stucture Network
2.	Aruna Chittineni	2023	A Real-Time Virtual Yoga Assistant using Machine Learning
3.	Ching-Hang Chen Deva Ramanan	2017	3D Human Pose Estimation=2D Pose Estimation+Matching
4.	K.Mase T.Takahashi	2004	Spine Posture estimation method from human images using 3D spine model computation of the rough approximation of the physical forces working on vertebral bodies
5.	A Naresh Kumar, More Raju	2022	Yoga Pose Assessment for Self-Learning
6.	Adrian Paul Reyes Nooreen Bascallan	2023	A Yoga Pose Evaluation System for Enhanced Yoga Education
7.	Carlo Dindorf,Oliver Ludwig	2023	Machine Learning and Explainable Artificial Intelligence Using Counterfactual Explanations for Evaluating Posture Parameters
8.	Bhat Dittakavi Ayon Sharma	2022	Pose Tutor:An Explainable System for Pose Correction in the Wild-2022
9.	Xinagjie Huang,Da Pan	2021	Intelligent Yoga Coaching System Based on Posture Recognition

10.	Chris Cheng Zhang,Chenran Wang	2023	Camera-Based Analysis of Human Pose for Fall Detection

3.SYSTEM REQUIREMENTS

Introduction

To implement posture detection using Convolutional Neural Networks(CNN), certain software prerequisites are necessary:

- Python: Python is a widely adopted programming language known for its extensive libraries and frameworks used in machine learning and deep learning tasks. It provides a rich environment for image processing and neural network development.

- Deep Learning Framework: You must select a deep learning framework capable of supporting CNNs, such as TensorFlow, Keras, or PyTorch. These frameworks offer efficient tools and APIs for building and training neural networks.

- Image Processing Libraries: Libraries like OpenCV (Open Source Computer Vision Library) or PIL (Python Imaging Library) can be employed for preprocessing signature images. They facilitate tasks like converting images to grayscale or binary format, as well as applying operations such as resizing, normalization, and noise reduction.

- Data Manipulation Libraries: Libraries like NumPy and pandas are commonly utilized for efficient data manipulation and handling large datasets. They provide valuable data structures and functions for performing operations on input data.

- GPU Support: If access to a Graphics Processing Unit (GPU) is available, it can significantly accelerate the training and inference processes of CNN models. Deep learning frameworks like TensorFlow and PyTorch offer GPU acceleration support.

- Development Environment: An integrated development environment (IDE)like

PyCharm, Jupyter Notebook, or Anaconda offers a convenient coding environment with debugging tools and package management capabilities.

3.1 Software Requirements

Python: Python is the important programming language in the provided code, chosen for its simplicity, versatility, and extensive standard library. It supports various programming paradigms, aiding in the development of readable and efficient solutions. Python's wide adoption across platforms and its active community contribute to its robust ecosystem, enriched with libraries like NumPy, OpenCV, TensorFlow, and Mediapipe used in the code. To run Python programs, ensure it's installed on your system, available for download from python.org. Python's ongoing updates and community support ensure its suitability for diverse applications, including the real-time posture analysis and yoga pose recognition demonstrated in the code.

OpenCv: Open source computer vision library, commonly referred to as OpenCV is quite central in the performance of the code through the execution of image and video related operations. It aids in operations required for reading video frames form the webcam, resizing frames, converting from one color space to another, drawing annotations on the frames and displaying images.

NumPy: It is a mathematical package of Python, which is used for high-level, arrays and matrices calculations, supports various mathematical functions to work with increasing large, multi-dimensional arrays. It is elementary for managing or modifying data structures which are required in scientific computing and data processing operations.

TensorFlow: TensorFlow is widely used tensor computation framework with open source created by Google, suitable for building and training the machine learning models particularly deep learning neural networks. When it comes to the code, Tensor flow helps in loading as well as running a deep learning model that is already trained (yoga_pose_model. h5) for identifying yoga poses from the frames recorded by the webcam.

IDE or Text Editor: It is also recommended to choose an appropriate integrated development environment (IDE) or textual editor for an effective writing, editing and

executing of the examined Python programming language. Software that is often used includes PyCharm and Visual Studio Code as well as Jupyter Notebooks which add features such as syntax highlighters/debuggers and the ability to work in an environment where scripts can be executed. These are useful tools that make work easier and facilitates the execution and final inspection of the computer vision and machine learning aspects incorporated in the code.

Webcam: The code counts on utilizing the webcam as a source to get frames of the real-time videos. This is crucial since camera is commonly used and one must ensure proper connection to the computer and proper driver installed for it to work properly. It makes it possible for the code to operate in a loop as it simultaneously takes real-time video feed for determining the user's posture and for identifying different poses of a yoga session.

3.2 Hardware Requrirements

Computer with Webcam: A computer with a built-in or external webcam is essential. The webcam captures video frames in real-time for posture analysis and yoga pose recognition.

Sufficient Processing Power: The computer should have adequate processing power to handle real-time video processing and deep learning computations. A modern CPU (Central Processing Unit) with multiple cores (e.g., Intel Core i5 or higher) is recommended.

Sufficient RAM: Sufficient RAM (Random Access Memory) is crucial to ensure smooth operation. A minimum of 8 GB RAM is generally recommended for handling real-time video frames and deep learning model inference.

GPU (optional): While the code does not explicitly require a GPU (Graphics Processing Unit), having a compatible GPU (NVIDIA GPU with CUDA support) can significantly speed up deep learning model inference if TensorFlow is configured to use GPU acceleration.

Operating System: The code can run on various operating systems including Windows, macOS, and Linux distributions like Ubuntu. Ensure your chosen operating system is compatible with Python, OpenCV, TensorFlow, and other required libraries.

Storage: Ensure adequate storage space for storing the Python code files, pre-trained model files (like 'yoga_pose_model.h5'), and any captured video frames or logs generated during runtime.

3.3 DataSet Used

Custom Datasets: Documenting custom datasets for yoga pose recognition involves accurate steps to ensure the dataset meets specific application requirements. Researchers and developers begin by carefully selecting a varied range of yoga poses relevant to their project goals. These poses are captured using high-resolution cameras in controlled environments, often with consistent lighting and background settings to enhance image clarity. Each image or video frame is meticulously annotated, marking key body landmarks crucial for pose identification. Techniques like data augmentation, such as flipping and rotating images, are employed to enrich dataset variability and improve model robustness. It's essential to partition the dataset into training and validation sets, enabling effective model training and evaluation. Ethical considerations, including obtaining consent from participants and ensuring data privacy, are paramount throughout the documentation process. Detailed records are maintained, documenting pose definitions, annotation methodologies, and any challenges encountered, ensuring transparency and reproducibility in dataset creation for reliable yoga pose recognition models.

Purpose of The Requirements Documents

1.Introduction:

The requirements document is a fundamental element for any project, offering a blueprint that outlines the project's objectives, scope, and specifications to guide the development process. When utilizing Convolutional Neural Networks (CNN) for posture detection, this document plays a vital role in ensuring the project's clarity, coherence, and thoroughness.

A Convolutional Neural Network (CNN) is a deep learning model particularly effective for analyzing visual data. Inspired by the visual cortex of animals, CNNs use convolutional layers that apply filters to input data to identify features such as edges, textures, and shapes. These features are progressively integrated through multiple layers to create high-level representations, enabling the network to recognize complex patterns and objects within images.

This document aims to delineate the specific goals, functional requirements, and limitations of the CNN-based posture detection system. Its purpose is to establish the primary objective

of developing a posture detection system utilizing CNN technology.

Additionally, the document outlines any potential interdependencies, such as the necessity for a diverse dataset of authentic images to train the CNN model.

2.Functional Requirements:

The requirements document specifies the functional requirements of the posture detection system. Classifying posture into categories such as kyphosis, lordosis, or normal posture based on the collected data. We use machine learning techniques, such as CNN algorithm for classification.

3.Performance Metrics:

The document establishes the performance metrics that will be utilized to evaluate the effectiveness of the CNN-based posture detection system. Standard metrics like accuracy, precision, recall, and F1 score may be defined to assess the model's performance. These metrics offer quantifiable measures to assess the system's accuracy and efficiency.

4.Constraints and Considerations:

The requirements document identifies constraints or considerations that must be addressed during the development process. These may encompass limitations in computational resources, data privacy and security requirements, compliance with legal regulations, or specific technological constraints that could affect the implementation of the CNN-based posture detection system..

5.Conclusion:

The purpose of the requirements document for posture detection using CNN is to establish a clear understanding of the project's objectives, scope, functional requirements, performance metrics, and constraints. It provides a comprehensive road map for developing an accurate

and efficient posture detection system that leverages the power of Convolutional Neural Networks. By outlining these requirements, the document ensures a smooth development process, enabling the creation of a robust and reliable posture detection system.

Scope of the product

1.Posture Detection:

The main objective of the product is to identify defects in the spinal posture and ensure that the posture is correct, if not then recommend few yoga poses based on the condition the user is suffering from.

2.Convolutional Neural Networks (CNN):

The product will utilize CNN, a deep learning algorithm specifically designed for recognizing and analysing images. CNNs have demonstrated exceptional performance in various computer vision tasks, making them well-suited for detecting defects in the posture. CNNs also incorporate pooling layers, which reduce the spatial dimensions of the data, thereby lowering computational requirements and enhancing the model's robustness to spatial variations. Fully connected layers at the end of the network integrate these features to perform classification or regression tasks. Due to their ability to automatically and adaptively learn spatial hierarchies of features, CNNs have become the cornerstone of many modern computer vision applications, including image and video recognition, object detection, and even some natural language processing tasks.

3.Preprocessing and Feature Extraction:

The product will incorporate preprocessing steps to enhance the quality of the input images, including re-sizing, normalization, and noise reduction. Additionally, feature extraction techniques will be employed to capture pertinent patterns and details from the images, which will serve as inputs to the CNN.

4.Training and Model Development:

The product will undergo a training phase where a significant dataset of genuine images will be utilized to train the CNN model.

5.Evaluation and Testing:

Following the training phase, the product will undergo thorough testing to assess its efficacy in detecting issues with the posture. This testing will involve an independent dataset containing images that were not used during training. The accuracy, precision, recall, and other relevant metrics will be measured to evaluate the product's performance.

6.User Interface:

The product may incorporate a user-friendly interface that enables users to upload images for analysis. The interface should provide clear instructions and deliver feedback on the results, indicating whether the posture is normal of not.

7.Plagiarism Prevention:

To uphold the product's integrity, measures should be implemented to prevent plagiarism. This can involve utilizing proprietary algorithms, employing obfuscation techniques, or incorporating other security measures.

Definitions, Acronyms, And Abbreviations

1.Machine Learning (ML)

Machine learning is a subset of artificial intelligence that involves developing algorithms and models that learn from data to make predictions or decisions, without being explicitly programmed to perform the task.

2.Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a type of deep learning algorithm tailored for image recognition tasks. It utilizes interconnected layers of nodes to extract features from input images automatically. Convolutional layers apply filters to identify spatial patterns, and pooling layers reduce the dimensionality of the features.

CNN: Convolutional Neural Network

ML: Machine Learning

References

1. https://www.computer.org/csdl/proceedings-article/icaa/2021/373000a700/1zL1XsxKlGw
2. https://www.geeksforgeeks.org/machine-learning/
3. https://www.javatpoint.com/machine-learning
4. https://www.geeksforgeeks.org/machine-learning/

Overview of The Remainder of The Document

- Creating a system to detect and classify spinal conditions like kyphosis, lordosis, or normal posture, and then recommending yoga poses accordingly. To detect the posture, we use OpenPose library which is an open-source library that detects human body key points from images or video frames. It provides coordinates for key body parts such as the nose, eyes, ears, shoulders, elbows, wrists, hips, lumbar region, knees, and ankles. We utilize sensors or imaging technologies to capture the posture of the human body. This data can include angles of various body parts, curvature of the spine, and other relevant metrics.
- Yoga recommendation system will detect the yoga pose in real time by using Posenet/Openpose and CNN Algorithm. Based on the classification result, it recommends appropriate yoga poses. This could involve mapping each posture type to a set of yoga poses known to be beneficial for that condition. Pose Matching

Algorithm compares the detected key points of the person's posture in real-time with the key points of the recommended yoga pose and gives feedback accordingly. As a result, we can try to improve the spinal health of a person.

General Description

The system combines two main features: real-time posture analysis and yoga pose recognition using a webcam. Firstly, it utilizes the MediaPipe Pose model to identify key landmarks on a person's body, such as ears, shoulders, elbows, and hips. These landmarks are used to calculate angles that assess posture metrics like hunchback, good posture, and lordosis angles. Visual feedback is provided on-screen in real-time, highlighting whether hunchback, lordosis, or good posture is detected, with specific recommendations for corrective yoga poses if needed. This part of the application utilizes OpenCV for webcam input and visualization, alongside MediaPipe's pose estimation capabilities.

Simultaneously, the code employs a pre-trained TensorFlow/Keras model to recognize various yoga poses from live webcam footage. It continuously captures and processes frames, displaying the recognized pose name on the screen if the prediction confidence exceeds a specified threshold. This functionality enhances the application's utility by offering real-time feedback on yoga posture correctness, contributing to improved practice and alignment awareness. Overall, this application leverages computer vision and machine learning to promote better posture awareness and yoga practice in real-time, offering users practical insights and guidance for improving their physical well-being through visual feedback.

Product Perspective

The product described by the above code is an innovative solution that combines computer vision, machine learning, and fitness technology. It aims to enhance posture awareness and refine yoga practice in real-time using a webcam interface. The system seamlessly integrates OpenCV to handle webcam input, enabling real-time processing of frames and

visualization of pose estimation results. This ensures a smooth user experience with interactive visual feedback directly on the webcam feed. By continuously analyzing frames, the system identifies the current yoga pose being performed and provides immediate feedback by displaying the recognized pose name. This enhances user awareness and accuracy in yoga practice, making it ideal for individuals aiming to monitor and improve their posture during fitness activities and yoga sessions, thereby promoting overall health and reducing the risk of musculoskeletal issues.

Product Functions

This integrated application combines real-time posture detection and yoga pose recognition using a webcam. It uses two main components: MediaPipe Pose analyzes each webcam frame to detect landmarks like ears, shoulders, elbows, and hips. By calculating angles between these landmarks, the app identifies poor posture such as hunchback or lordosis. It displays visual cues and recommends yoga poses for correction based on these calculations.

Simultaneously, a TensorFlow/Keras model processes the same webcam frames resized to fit its input dimensions. Trained to recognize various yoga poses, the model predicts the user's current pose. When confident in its prediction, the recognized yoga pose is superimposed on the webcam feed. This instant feedback helps users monitor their posture and encourages corrective actions through suggested yoga poses.

The application runs continuously, seamlessly integrating posture assessment and yoga pose recognition. Users receive immediate visual feedback on their posture status and recommended yoga poses, facilitating real-time adjustments to enhance posture alignment and overall physical well-being.

User Characteristics

The documentation is designed for developers and AI specialists with a good understanding of Python, TensorFlow, and OpenCV. It explains how to develop a system that can recognize yoga poses in real-time using a pre-trained TensorFlow model ('yoga_pose_model.h5'). The system captures video from a webcam, adjusts the size of each frame for processing, and uses the model to predict which yoga pose is being performed. It displays the recognized pose on the screen in real-time using OpenCV.

Users have the flexibility to replace the model with their own trained versions and tweak settings like how confident the system needs to be before identifying a pose, or adjust the size of the frames processed for better performance on different devices. Users can modify the confidence threshold (0.5 in this case) to control when a pose is recognized or when no pose is detected. This documentation not only teaches users how to implement the system but also provides guidance on deploying it effectively, troubleshooting any issues that may arise, and integrating it into other applications or devices.

General Constraints

When setting up the posture analysis and yoga pose recognition system, it's crucial to consider several key constraints to ensure reliable functionality. Firstly, verify that your webcam supports the specified frame size of 1280x720 pixels, as resizing can affect performance and image quality. Adjust parameters like min_detection_confidence and min_tracking_confidence cautiously to balance accuracy and real-time processing speed, depending on your hardware and environment.Pay close attention to handling landmark indices (mp_pose.PoseLandmark) to accurately calculate angles between joints such as ears, shoulders, elbows, and hips. These angles are critical for evaluating posture issues like hunchback or lordosis. Ensure angle detection thresholds, such as 130 degrees for hunchback, align with ergonomic guidelines for reliable posture assessment.

For yoga pose recognition, ensure the TensorFlow model (yoga_pose_model.h5) is available and compatible. Load the model correctly, verifying file paths and compatibility with TensorFlow and Keras versions used. Resize frames to 224x224 pixels to maintain consistent input dimensions for accurate yoga pose predictions.The overall performance relies heavily on sufficient hardware resources, including CPU and GPU capabilities.

These resources must handle tasks like video capture, pose estimation, and deep learning inference simultaneously without significant delays or frame rate drops.

To improve user experience, implement clear text overlays (cv2.putText) and intuitive controls in the user interface. This helps users interpret detected posture issues or recognized yoga poses easily in real-time. Additionally, incorporate robust error handling to manage potential issues during webcam setup, model loading, or frame processing, ensuring a stable and dependable application across different platforms and environments.

Assumptions and Dependencies:

To effectively use the system several assumptions and dependencies should be considered. Firstly, it assumes the presence of essential Python libraries such as OpenCV (cv2), MediaPipe (mediapipe), NumPy (numpy), and TensorFlow (tensorflow). These libraries are important for tasks like capturing real-time video from a webcam, performing pose estimation using MediaPipe's Pose model, and conducting operations like array manipulations and model predictions.OpenCV (cv2) enables the capture of real-time video from a webcam, a critical function for both the posture assessment and yoga pose recognition scripts.MediaPipe (mediapipe) provides the Pose model, enabling accurate estimation of human poses from the webcam feed. This includes detecting key landmarks such as ears, shoulders, and hips, which are pivotal for calculating angles to assess posture.In terms of hardware requirements, the setup assumes access to a functional webcam to capture live video input, which is crucial for the operation of both scripts. This capability allows the scripts to continuously receive and process video frames from the webcam feed.

Additionally, the code relies on specific models and files, such as yoga_pose_model.h5, which must be present in the working directory or a specified path. This file contains a trained TensorFlow/Keras model designed to accurately recognize various yoga poses from the video input.Furthermore, the setup necessitates access to a functioning webcam connected to the computer. This hardware requirement is essential for capturing real-time video input, which is integral to both the posture assessment and yoga pose recognition functionalities implemented in the scripts. This usage enhances users' awareness and

ability to correct body alignment during activities like yoga practice, thereby promoting better posture and overall physical health.

Specific Requirements

Specific requirements for a posture perfect system can vary depending on the specific context and goals of the system. However, here are some examples of functional, non-functional, and interface requirements that may be relevant to such a system:

1. Functional Requirements:

The system provides real-time feedback on posture quality by visually annotating angles like hunchback or lordosis directly on the webcam feed. When posture issues are detected, such as hunchback or lordosis, corresponding corrective yoga poses are recommended and displayed on-screen. For yoga pose recognition, the system employs a CNN model trained on various poses. It resizes and preprocesses frames from the webcam before feeding them into the model for prediction. Recognized yoga poses are superimposed on the webcam feed if their confidence surpasses a preset threshold, allowing users to promptly assess their performance. Users can exit the application by pressing a key, and the system ensures reliable performance across different environmental conditions, aiming to enhance usability and user experience.

Additionally, the system's posture analysis uses MediaPipe Pose to track landmarks like shoulders, ears, elbows, and hips, calculating angles critical for posture assessment. This includes angles indicative of good posture, hunchback, and lordosis, enhancing the accuracy of feedback provided to users. The yoga pose recognition component utilizes a CNN model trained on a diverse range of yoga poses. Frames captured from the webcam undergo resizing and preprocessing before being inputted for inference, ensuring efficient and accurate identification of yoga poses. This dual functionality aims to promote better posture awareness and correct pose execution through intuitive visual feedback and recommendations.

2. Non-Functional Requirements:

The non-functional requirements ensure the overall performance, usability and reliability of the system.

Performance: The system must efficiently process real-time data, minimizing delays from capturing webcam frames to analyzing poses and providing feedback. This involves optimizing code for speed and responsiveness to ensure smooth operation across different types of hardware setups.

Reliability: The application needs to reliably detect and recognize poses despite changes in lighting and background conditions. It should maintain accuracy and consistency in pose detection across different environmental settings, ensuring dependable performance for users.

Usability: The user interface should be easy to use, providing clear visual cues for posture feedback and recognized yoga poses. This design enhances the user experience by ensuring straightforward navigation and understanding of the information displayed.

Security: The system must protect user privacy by avoiding the storage or transmission of sensitive data captured from the webcam feed. It should follow best practices for securely handling data to ensure confidentiality and maintain user trust.

These non-functional requirements collectively contribute to a robust, user-friendly, and reliable application for posture analysis and yoga pose recognition.

3. Interface Requirements

Use Interface:The user interface displays a live webcam feed in the center, flanked by sections showing real-time posture feedback. "Good Posture" appears in green for correct posture, while "Hunchback Detected" or "Lordosis Detected" shows in red with recommended yoga poses like "Cobra Pose" or "Child's Pose" for issues. Below, the recognized yoga pose updates based on the model's highest confidence prediction, displaying names such as "Cobra" or "Child". Users can quit by pressing 'q', ensuring straightforward interaction and clear feedback on their posture and recognized poses from the webcam feed.

External Interface: The system interacts with the webcam to capture live video for posture analysis using MediaPipe Pose. It also utilizes a pre-trained TensorFlow model

stored externally to recognize yoga poses from the video feed. These interfaces enable real-time feedback on posture and recognized poses directly from the webcam input.

API: The code utilizes MediaPipe Pose for real-time posture analysis from the webcam, and TensorFlow to recognize yoga poses based on the detected postures. This combination provides instant feedback on posture quality and identifies specific yoga poses directly from the live video feed.

Compatibility: The code is compatible with systems that have a webcam and Python installed with necessary libraries like OpenCV, MediaPipe, NumPy, and TensorFlow. It analyzes posture in real-time using MediaPipe Pose and identifies yoga poses using a pre-trained TensorFlow model, delivering immediate feedback on posture and recognized poses

4.DESIGN

Proposed System

1. **Data Collection and Posture Analysis:**
    i **Imaging Technologies:** Utilize imaging technologies such as cameras or sensors to capture posture data in real-time. These technologies can capture angles of various body parts (e.g., shoulders, hips, spine curvature) and provide precise coordinates for key body points.
    ii **OpenPose Library:** Implement the OpenPose library, which is a popular tool for detecting human body key points from images or video frames. It provides accurate coordinates for body parts like joints and limbs, enabling detailed posture analysis.

2. **Posture Classification:**
    i **Machine Learning Techniques:** Use machine learning algorithms, specifically Convolutional Neural Networks (CNNs), for posture classification. CNNs are well-suited for image-based classification tasks and can effectively distinguish between different posture types such as kyphosis (excessive outward curvature of the spine), lordosis (excessive inward curvature), or normal posture.
    ii **Training the Model:** Train the CNN model on a labeled dataset that includes examples of various postures. The dataset should be diverse and representative of different body types, ages, and conditions to ensure the model's robustness.

3. **Yoga Pose Feedback and Recommendation:**
    i **Pose Matching Algorithm:** Develop a pose matching algorithm that compares the detected posture with ideal yoga poses. This algorithm evaluates how closely the user's posture matches the desired pose.
    ii **Feedback Mechanism:** Provide real-time feedback to the user based on the comparison results. This feedback informs the user whether they are performing the yoga pose correctly or need adjustments.

iii **Yoga Pose Recommendation:** Based on the classification of the user's posture (e.g., kyphosis, lordosis), recommend appropriate yoga poses. For instance, recommend Cobra Pose to correct kyphosis or Child's Pose to alleviate lordosis.

4. **Integration and User Interface:**
   i **Unified System Integration:** Integrate the posture analysis, yoga recommendation, and pose matching components into a cohesive system. This integration ensures seamless interaction and data flow between the different modules.
   ii **User-Friendly Interface:** Design a user interface that is intuitive and easy to use. The interface should allow individuals to interact with the system, visualize posture analysis results (such as graphical representations of posture angles), receive personalized yoga pose recommendations, and view real-time feedback on their pose performance.

5. **Testing and Validation:**
   i **Real-World Data Testing:** Test the system using real-world data collected from individuals with various spinal conditions. This testing phase validates the accuracy and reliability of the posture classification, yoga pose detection, and pose matching algorithms.
   ii **Validation of Algorithms:** Conduct validation studies to assess how well the posture classification algorithm identifies different spinal conditions and how effectively the pose matching algorithm provides accurate feedback and recommendations.

4.1.1 Advantages over Existing System

1. **Immediate Feedback using OpenPose and PoseNet**:

- **OpenPose and PoseNet**: OpenPose and PoseNet are advanced computer vision models that provide real-time feedback on posture detection. They analyze body

keypoints from images or video frames, allowing for immediate assessment of posture quality.

- **Response Time**: These technologies enable the system to detect posture deviations promptly and suggest corrective measures in real-time. For example, if a user is performing a yoga pose incorrectly, the system can identify specific areas where adjustments are needed, such as correcting alignment or adjusting limb positions.
- **Enhanced Learning Experience**: Immediate feedback enhances the learning experience by enabling users to make adjustments during their practice sessions. This helps in achieving correct posture alignment and improving overall technique efficiently.

2. **Personalized Recommendations based on Detected Conditions**:

- **Customized Yoga Poses**: Upon detecting specific spinal conditions such as kyphosis or lordosis, the system can provide personalized yoga pose recommendations. These recommendations are tailored to address the individual's unique condition and health goals.
- **Effectiveness for Spinal Health**: Personalization ensures that users receive targeted interventions that are most beneficial for their spinal health. For instance, individuals with kyphosis may receive recommendations for poses that focus on spinal extension and strengthening, while those with lordosis may benefit from poses that promote spinal flexion and stretching.
- **Progress Monitoring**: The system can track users' progress over time by recording performance metrics and monitoring improvements in posture and flexibility. This continuous monitoring allows for adjustments in recommendations as the user's condition evolves or improves.

3. **Community Support through Open-Source Projects**:

- **Open-Source Community**: Leveraging open-source projects like OpenPose fosters collaboration and continuous improvement. The community contributes to refining algorithms, addressing bugs, and adding new features based on user feedback and technological advancements.
- **Continuous Updates**: Regular updates and improvements from the community ensure that the posture detection and recommendation system remains up-to-date with

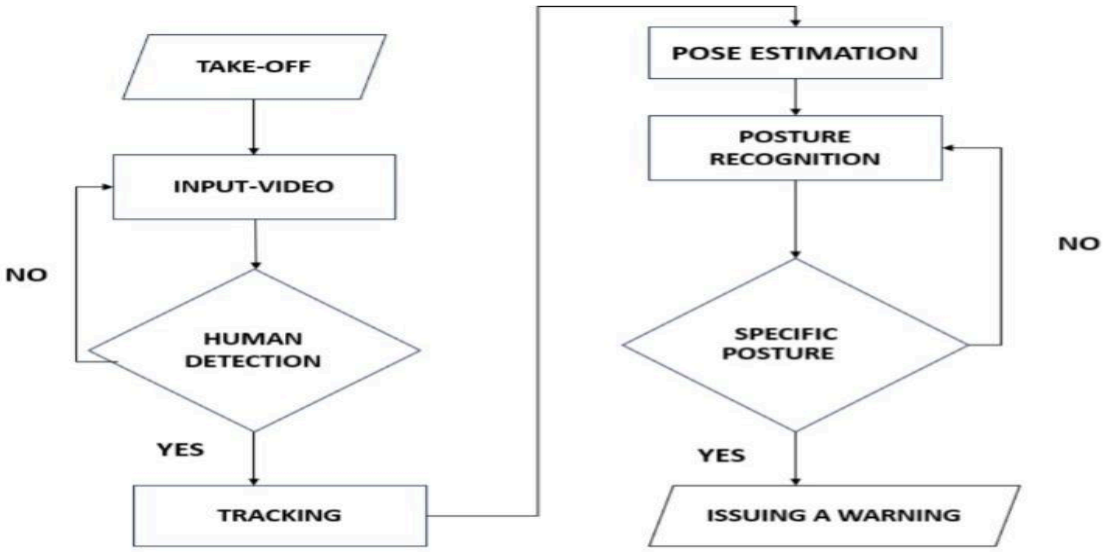
the latest advancements in computer vision and machine learning. This ensures reliability, accuracy, and adaptability to evolving user needs and technological standards.

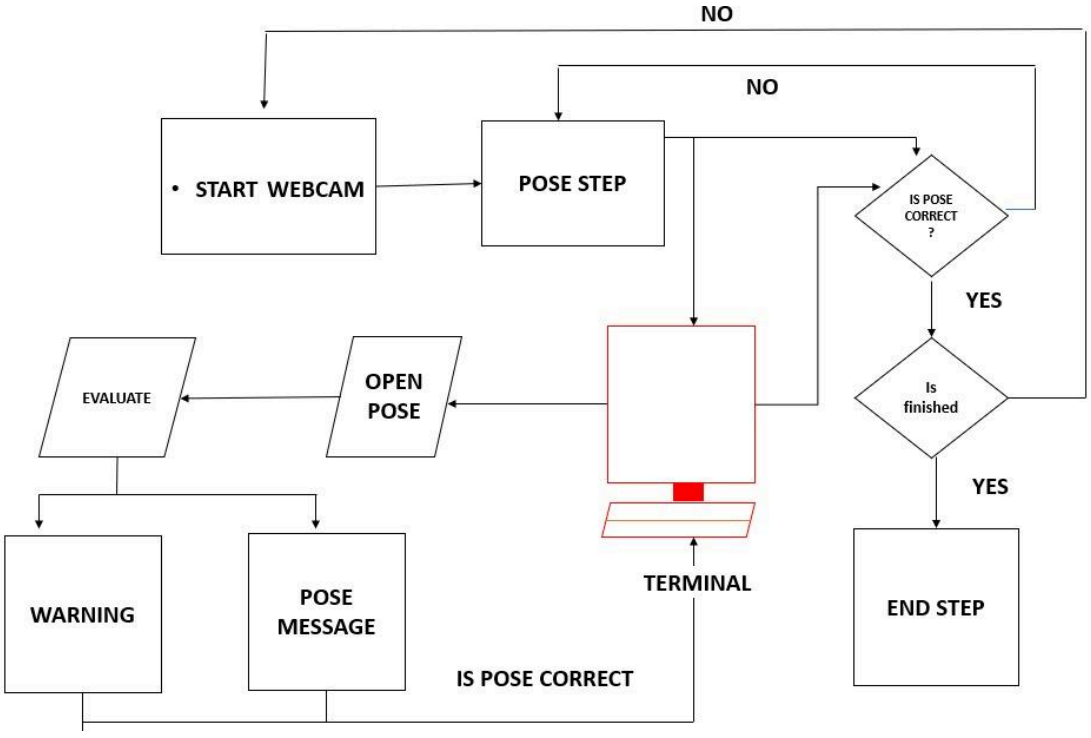
4. **Affordable Solution compared to Traditional Methods**:

- **Cost-Effectiveness**: Automated posture detection systems are more cost-effective compared to traditional methods that involve multiple visits to specialists or physical therapists. Users can access the system from the comfort of their homes, reducing travel costs and time spent on appointments.
- **Accessibility**: The affordability of the system increases accessibility to quality posture correction and yoga recommendations for a broader population. This democratizes access to healthcare-related services, particularly beneficial for individuals in remote or underserved areas.
- **Long-Term Savings**: Over time, the automated system proves cost-effective as it reduces the need for ongoing professional consultations. Users can maintain their spinal health independently with periodic guidance and monitoring from the system.

Block Diagram:

(Overall System Architecture)





4.1 Modules

OpenCv: OpenCV is one of the fundamental requirement in the field of computer vision in term of providing faculty for most of the computer vision related tasks pertaining to images and videos. Functions it is comprised of are basic functions such as VideoCapture for reading video frames, resize for resizing the frames . OpenCV is used extensively for processing video frames collected from a webcam where the life cycle involves tasks such as capture, processing, and display of frames. This functionality is crucial facilitating real-time posture analysis through MediaPipe Pose as well as identifying yoga poses by using Tensorflow model.

MediaPipe: MediaPipe plays an essential role in processing video frames using such things as the poser. It is a process that allows detection and tracking of the pose landmarks during operation and real-time recognition of objects such as the results. pose_landmarks. Mp_drawing is then used to display these landmarks on the layer of the video frame. draw_landmarks, which helped to provide a more or less accurate observation of body posture dynamics, where the webcam feed was acting as the assessment source.

Numpy: The arrow functionality of NumPy comes in handy in the calculate_angle function that takes coordinate data of pose landmarks obtained from the MediaPipe pose estimation. Using NumPy's np. array , it controls and performs angles in actions such as np. arctan2 and np. degrees. These calculations are fundamental to other posture angles like the hunchback and lordosis to make correct evaluation and analysis of body posture based the distinguished landmark information obtained from the video frame.

Tensorflow: Tensorflow comes in handy when using keras integration where the tensorflow model (‘tf.keras.models.load_model’) is used to import a deep learning model (‘yoga_pose_model.h5’) that has been trained in advance. This model is particularly beneficial for identifying any of the various yoga poses from specific frames that have been scaled down and preprocessed. helping them, the application utilizes TensorFlow means to predict classes of yoga poses. These predictions are then finally shown in what is titled ‘real-time’ so as to give an almost instant feedback on the posed yoga asanas and therefore the user is also able to receive further instructions or even corrections across their postures.

4.2 UML Diagrams

4.2.1 System Use-Case Diagrams

Actors:

User: The individual who uses the system to detect spinal conditions and receive yoga pose recommendations.

Posture Detection and Analysis System:

This part of the system is responsible for detecting and analyzing the user's posture to identify any spinal conditions such as kyphosis or lordosis.

Capture Video Frames:

User --> (Capture Video Frames): The user initiates the process by capturing video frames using a webcam or camera.

(Capture Video Frames) --> (Process Frame as Image): The captured video frames are processed and converted into image format.

Analyze Posture:

(Process Frame as Image) --> (Analyze Posture): The processed images are analyzed to identify key points and angles related to the user's posture.

Classify Condition:

(Analyze Posture) --> (Classify Condition): Based on the analyzed data, the system classifies the user's posture as normal, kyphosis, or lordosis.

Recommend Yoga Poses:

(Classify Condition) --> (Recommend Yoga Poses): Depending on the classified condition, the system recommends appropriate yoga poses.

Yoga Recommendation System:

This part of the system uses video input to recognize yoga poses performed by the user and provides feedback or further recommendations.

Capture Pose Video:

User --> (Capture Pose Video): The user captures video while performing yoga poses.

(Capture Pose Video) --> (Extract Key Frames): Key frames are extracted from the captured video to identify crucial moments in the user's pose.

Analyze Key Frames:

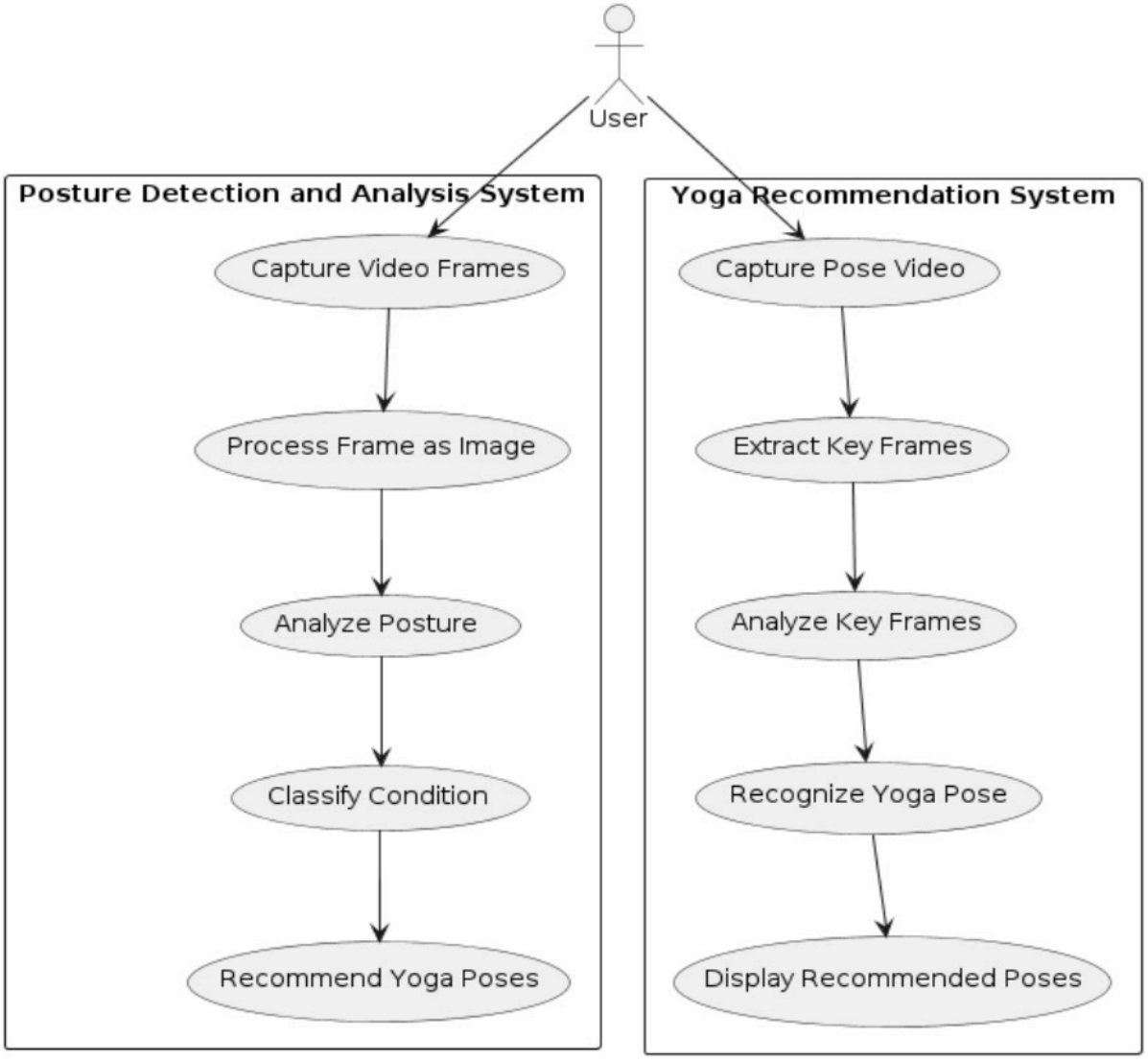
(Extract Key Frames) --> (Analyze Key Frames): The extracted key frames are analyzed to detect the user's body key points.

Recognize Yoga Pose:

(Analyze Key Frames) --> (Recognize Yoga Pose): The system recognizes the yoga pose being performed by comparing the key points with predefined pose models.

Display Recommended Poses:

(Recognize Yoga Pose) --> (Display Recommended Poses): The system displays the recognized pose and may provide additional recommendations for improvement.



4.2.2 CLASS DIAGRAM

Posture Detection System

Methods:

+ **capture_video_frames():** Captures real-time video frames using a webcam or camera.

+ **process_frame_as_image():** Processes each video frame and converts it into an image format for further analysis.

+ **analyze_posture():** Analyzes the processed image to detect key points and angles related to the user's posture.

+ **classify_condition():** Classifies the analyzed posture into specific spinal conditions (e.g., normal, kyphosis, lordosis).

+ **recommend_yoga_poses():** Recommends appropriate yoga poses based on the classified spinal condition.

Relationships:

Uses Yoga Recommendation System: The Posture Detection System interacts with the Yoga Recommendation System to recommend yoga poses.

Identifies Spinal Condition: The Posture Detection System identifies specific spinal conditions based on posture analysis.

Yoga Recommendation System

Methods:

+ **capture_pose_video():** Captures video of the user performing yoga poses.

+ **extract_key_frames():** Extracts key frames from the captured video to focus on important moments of the yoga pose.

+ **analyze_key_frames():** Analyzes the key frames to detect the user's body key points and posture during the yoga pose.

+ **recognize_yoga_pose():** Recognizes the yoga pose being performed by comparing detected key points with predefined pose models.

+ **display_recommended_poses():** Displays the recognized pose and provides additional pose recommendations for improvement.

Relationships:

Used by PostureDetectionSystem: The YogaRecommendationSystem is utilized by the PostureDetectionSystem to provide yoga pose recommendations.

Recommends SpinalCondition: Based on the recognized yoga pose, it provides

recommendations for improving or correcting spinal conditions.

User

Attributes:

- username: string: The username of the user interacting with the system.

- age: int: The age of the user.

SpinalCondition

Attributes:

- name: string: The name of the spinal condition (e.g., kyphosis, lordosis).

- postureRecommendations: string[]: A list of yoga pose recommendations tailored to the specific spinal condition.

Relationships:

Identified by PostureDetectionSystem: The system identifies specific spinal conditions based on posture analysis.

Provides recommendations to YogaRecommendationSystem: The system uses the condition information to recommend appropriate yoga poses.

4.2.3 ACTIVITY DIAGRAM

User Actions

Provide video feed: The user starts the process by providing a real-time video feed to the system. This could be through a webcam or any other camera device.

Posture Detection System (PDS) Actions

Capture and preprocess frames: The Posture Detection System (PDS) captures video frames from the user's feed.

These frames are then preprocessed, which may include resizing, normalizing, or converting to a different color space suitable for further analysis.

Analyze posture: The system analyzes the preprocessed frames to detect key points of the body and calculate angles between joints. This step involves identifying the user's posture based on the positions of body landmarks.

Decision Point

Posture anomaly detected?: The system evaluates whether there is a posture anomaly by comparing the calculated angles and positions against predefined criteria for normal posture.There are two possible paths from this decision point: "yes" (anomaly detected) and "no" (no anomaly detected).

If Anomaly Detected (Yes Path)

Identify spinal condition: If an anomaly is detected, the system identifies the specific spinal condition (e.g., kyphosis, lordosis) based on the nature of the detected anomaly.

Recommend yoga poses: The identified spinal condition is used to determine suitable yoga poses that can help correct or improve the detected anomaly. This recommendation is provided by the Yoga Recommendation System (YRS).

Display recommended poses:

The recommended yoga poses are displayed to the user. This provides actionable feedback to the user on how to correct their posture.

If No Anomaly Detected (No Path)

Display good posture message: If no posture anomaly is detected, the system informs the user that their posture is good. This positive feedback reassures the user that their posture is correct.

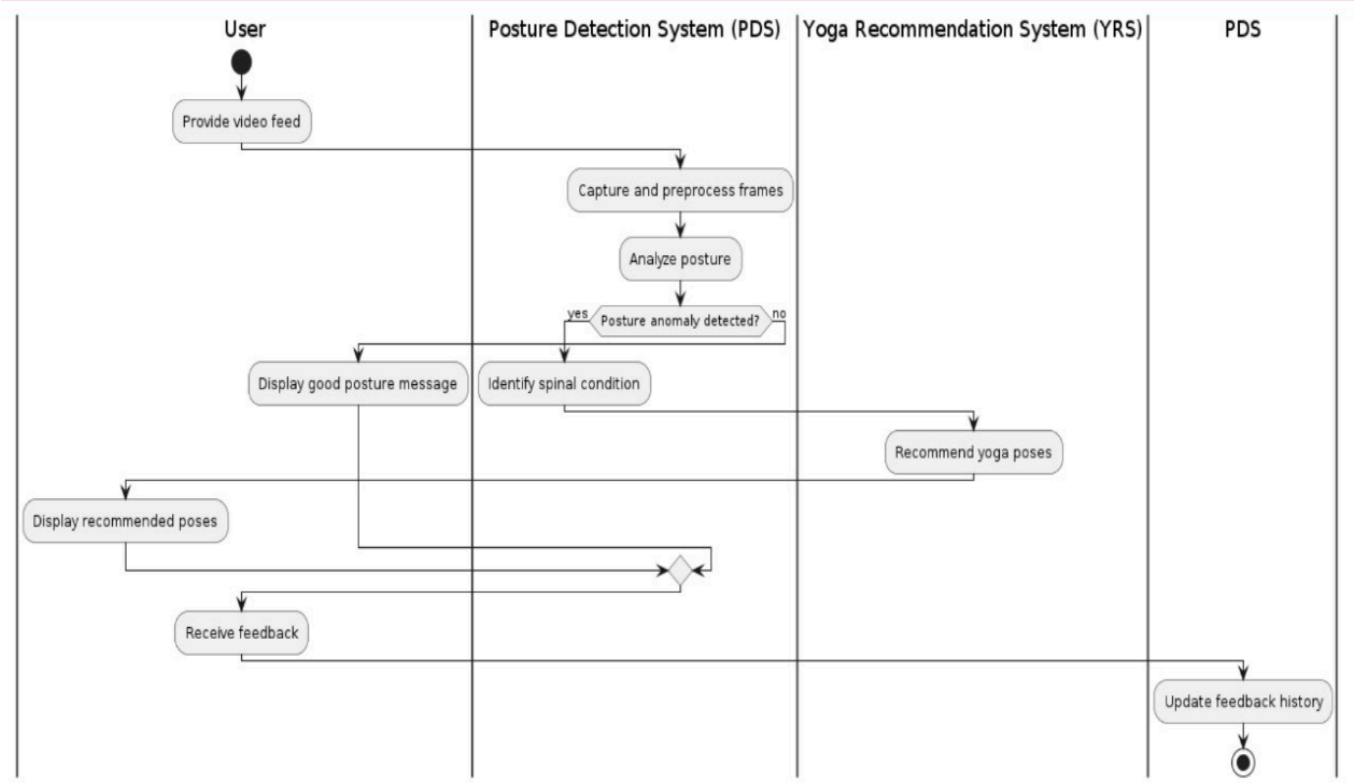
Feedback Loop

Receive feedback: The user has the option to provide feedback on the system's

recommendations and analysis. This feedback can include the effectiveness of the recommended poses or any other relevant user input.

Update feedback history:

The PDS updates its feedback history with the user's feedback. This can help improve the system's accuracy and recommendations over time by learning from user input.



5. IMPLEMENTATION

POSTURE DETECTION:

```python
import cv2
import mediapipe as mp
import numpy as np

# Initialize MediaPipe Pose
mp_pose = mp.solutions.pose
pose = mp_pose.Pose(static_image_mode=False, min_detection_confidence=0.5, min_tracking_confidence=0.5)

# Initialize MediaPipe drawing utils
mp_drawing = mp.solutions.drawing_utils

# Function to calculate the angle between three points
4 usages  new *
def calculate_angle(a, b, c):
    a = np.array(a)
    b = np.array(b)
    c = np.array(c)
    radians = np.arctan2(c[1] - b[1], c[0] - b[0]) - np.arctan2(a[1] - b[1], a[0] - b[0])
    angle = np.degrees(radians)
    if angle < 0:
        angle += 360
    return angle

# Open the webcam
cap = cv2.VideoCapture(0)
```

```
25
26  # Set the frame width and height
27  cap.set(cv2.CAP_PROP_FRAME_WIDTH, value: 1280)
28  cap.set(cv2.CAP_PROP_FRAME_HEIGHT, value: 720)
29
30  while cap.isOpened():
31      ret, frame = cap.read()
32      if not ret:
33          break
34
35      # Convert the frame to RGB
36      image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
37
38      # Process the frame with MediaPipe Pose
39      results = pose.process(image_rgb)
40
41      # Draw the pose annotation on the frame
42      if results.pose_landmarks:
43          mp_drawing.draw_landmarks(frame, results.pose_landmarks, mp_pose.POSE_CONNECTIONS)
44
45          # Extract landmarks
46          landmarks = results.pose_landmarks.landmark
47
48          # Get the coordinates of ear, shoulder, and hip joints
49          left_ear = [landmarks[mp_pose.PoseLandmark.LEFT_EAR.value].x * frame.shape[1],
```

```
49          left_ear = [landmarks[mp_pose.PoseLandmark.LEFT_EAR.value].x * frame.shape[1],
50                      landmarks[mp_pose.PoseLandmark.LEFT_EAR.value].y * frame.shape[0]]
51          right_ear = [landmarks[mp_pose.PoseLandmark.RIGHT_EAR.value].x * frame.shape[1],
52                       landmarks[mp_pose.PoseLandmark.RIGHT_EAR.value].y * frame.shape[0]]
53          left_shoulder = [landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].x * frame.shape[1],
54                           landmarks[mp_pose.PoseLandmark.LEFT_SHOULDER.value].y * frame.shape[0]]
55          right_shoulder = [landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].x * frame.shape[1],
56                            landmarks[mp_pose.PoseLandmark.RIGHT_SHOULDER.value].y * frame.shape[0]]
57          left_hip = [landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].x * frame.shape[1],
58                      landmarks[mp_pose.PoseLandmark.LEFT_HIP.value].y * frame.shape[0]]
59          right_hip = [landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].x * frame.shape[1],
60                       landmarks[mp_pose.PoseLandmark.RIGHT_HIP.value].y * frame.shape[0]]
61
62          # Calculate angles
63          left_hunchback_angle = calculate_angle(left_ear, left_shoulder, left_shoulder)
64          right_hunchback_angle = calculate_angle(right_ear, right_shoulder, right_shoulder)
65          left_lordosis_angle = calculate_angle(left_shoulder, left_hip, c: [left_hip[0], left_hip[1] + 1])
66          right_lordosis_angle = calculate_angle(right_shoulder, right_hip, c: [right_hip[0], right_hip[1] + 1])
67
68          # Display the angles
69          cv2.putText(frame, text: f'L_Hunch: {int(left_hunchback_angle)}', org: (int(left_shoulder[0]), int(left_shoulder[1] - 50)),
70                      cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (255, 255, 255), thickness: 2, cv2.LINE_AA)
71          cv2.putText(frame, text: f'R_Hunch: {int(right_hunchback_angle)}', org: (int(right_shoulder[0]), int(right_shoulder[1] - 50)),
72                      cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (255, 255, 255), thickness: 2, cv2.LINE_AA)
73          cv2.putText(frame, text: f'L_Lord: {int(left_lordosis_angle)}', org: (int(left_hip[0]), int(left_hip[1] - 50)),
```

```
        cv2.putText(frame, text: f'L_Lord: {int(left_lordosis_angle)}', org: (int(left_hip[0]), int(left_hip[1] - 50)),
                    cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (255, 255, 255), thickness: 2, cv2.LINE_AA)
        cv2.putText(frame, text: f'R_Lord: {int(right_lordosis_angle)}', org: (int(right_hip[0]), int(right_hip[1] - 50)),
                    cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (255, 255, 255), thickness: 2, cv2.LINE_AA)

        # Check for hunchback posture
        if left_hunchback_angle > 100 or right_hunchback_angle > 130:
            cv2.putText(frame, text: 'Hunchback Detected', org: (50, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 0, 255), thickness: 2, cv2.LINE_AA)
            cv2.putText(frame, text: 'Recommended Yoga Pose: Cobra Pose', org: (50, 100),
                        cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 0, 255), thickness: 2, cv2.LINE_AA)
        # Check for lordosis posture
        elif left_lordosis_angle > 190 or right_lordosis_angle > 190:
            cv2.putText(frame, text: 'Lordosis Detected', org: (50, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 0, 255), thickness: 2, cv2.LINE_AA)
            cv2.putText(frame, text: 'Recommended Yoga Pose: Child\'s Pose', org: (50, 100),
                        cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 0, 255), thickness: 2, cv2.LINE_AA)
        else:
            cv2.putText(frame, text: 'Good Posture', org: (50, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 255, 0), thickness: 2, cv2.LINE_AA)

    # Display the resulting frame
    cv2.imshow( winname: 'MediaPipe Pose', frame)

    if cv2.waitKey(10) & 0xFF == ord('q'):
```

```
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

# Release the webcam and close windows
cap.release()
cv2.destroyAllWindows()
```

DATA PREPROCESSING

```python
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os

# Get the absolute path of the dataset directory
base_dir = os.path.abspath(r"C:\Users\sowmy\Desktop\yoga\dataset")

# Print directory contents for debugging
print("Base directory content:", os.listdir(base_dir))
print("Train directory content:", os.listdir(os.path.join(base_dir, 'train')))

# Define image data generator for data augmentation
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,  # Use 20% of data for validation
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

```

```
# Load dataset
train_generator = datagen.flow_from_directory(
    os.path.join(base_dir, 'train'),
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    os.path.join(base_dir, 'validation'),
    target_size=(224, 224),
    batch_size=16,
    class_mode='categorical',
    subset='validation'
)
```

TRAINING A MODEL

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define the directory paths
train_dir = r"C:\Users\sowmy\Desktop\yoga\dataset\train"
validation_dir = r"C:\Users\sowmy\Desktop\yoga\dataset\validation"

# Define image data generators with augmentation for training and validation
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Define the batch size
batch_size = 16

```

```
# Generate batches of augmented data for training and validation
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical'
)

validation_generator = test_datagen.flow_from_directory(
    validation_dir,
    target_size=(224, 224),
    batch_size=batch_size,
    class_mode='categorical'
)

# Define the model
model = tf.keras.Sequential([
    tf.keras.applications.MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3)),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense( units: 256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(train_generator.class_indices), activation='softmax')
])

```

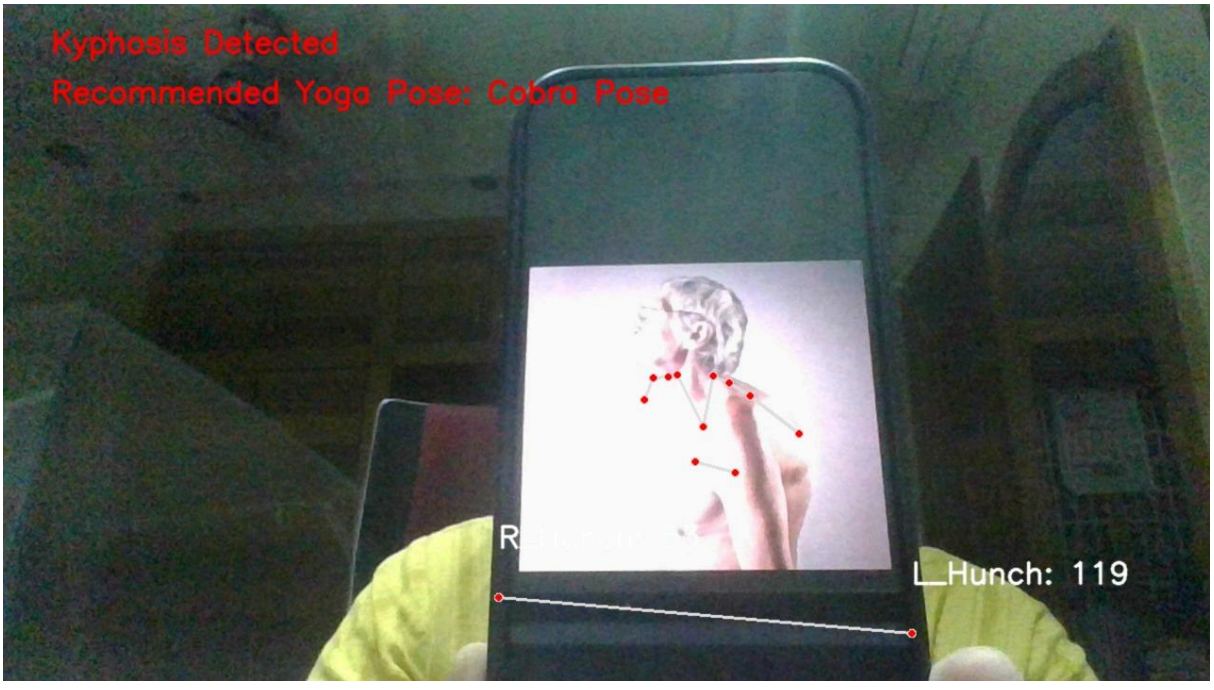
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size
)

# Save the model
model.save('yoga_pose_model.h5')
```

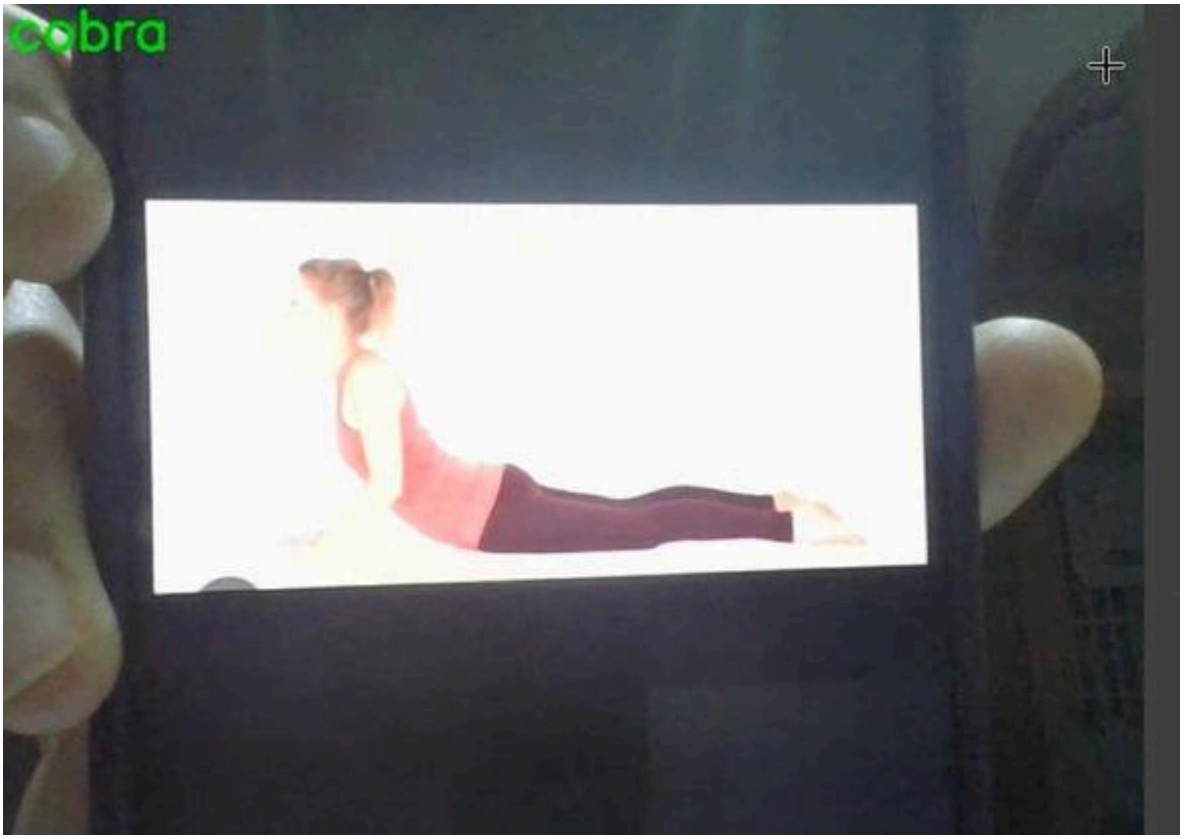
YOGA RECOMMENDATION SYSTEM

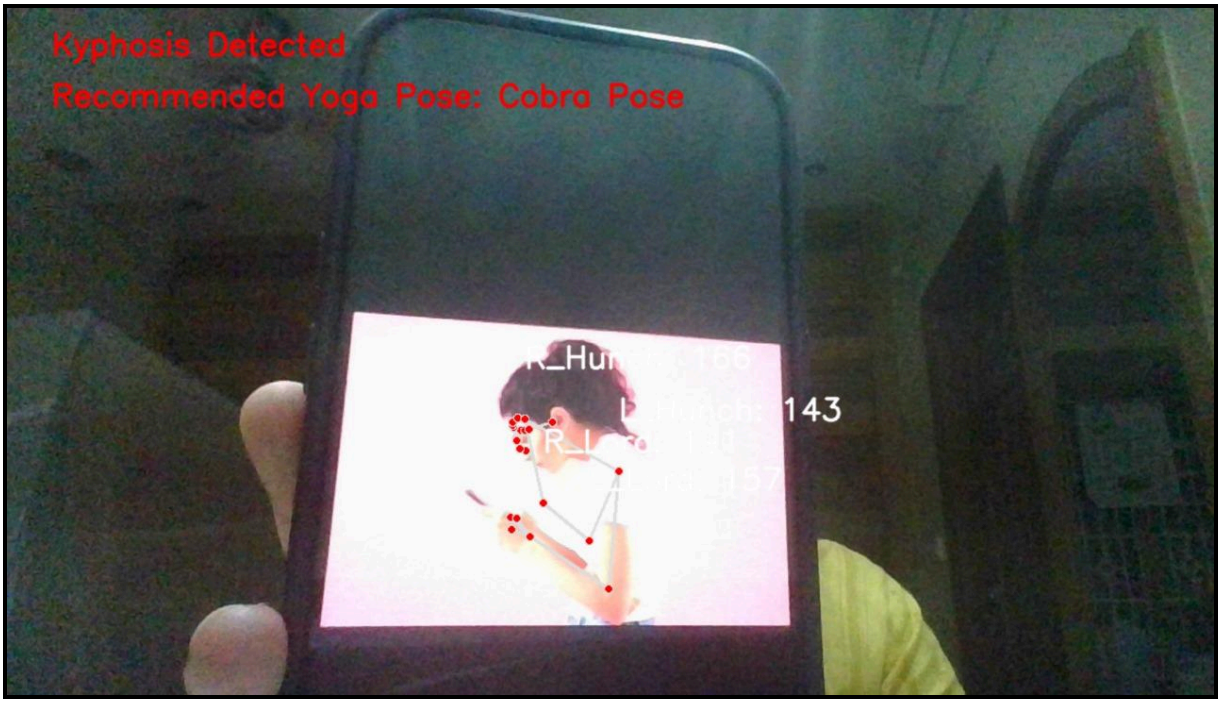
```
import cv2
import numpy as np
import tensorflow as tf

# Load the trained model
model = tf.keras.models.load_model('yoga_pose_model.h5')

# Dictionary to map class indices to class names
class_names = {0: 'bridge', 1: 'cat-cow', 2: 'child', 3: 'cobra'}

# Initialize the webcam
cap = cv2.VideoCapture(0)

# Define the new dimensions for resizing
new_width = 224
new_height = 224

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Resize the frame to the new dimensions
    resized_frame = cv2.resize(frame,  dsize: (new_width, new_height))

```

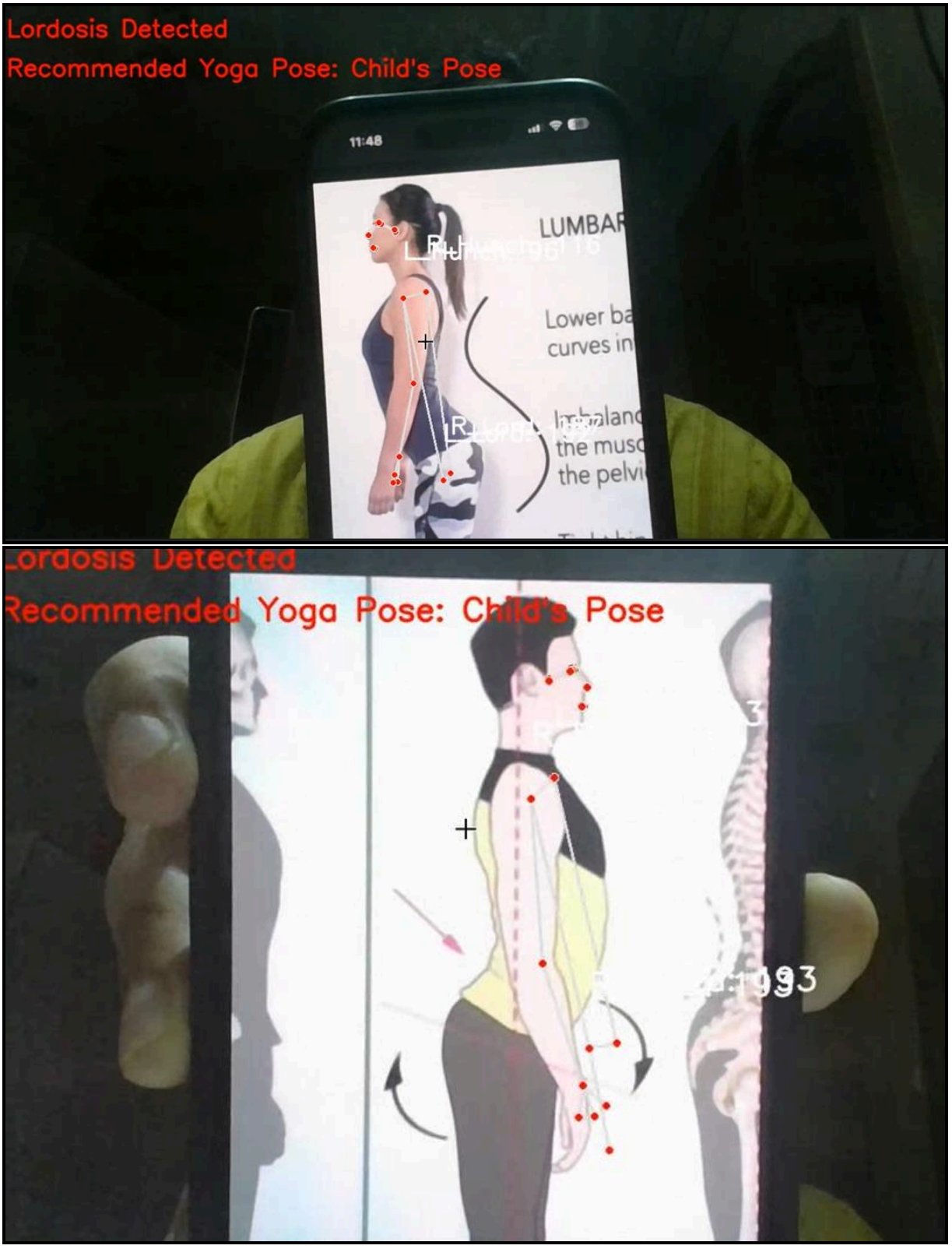
```
    # Preprocess the resized frame
    img = np.expand_dims(resized_frame, axis=0)
    img = img / 255.0

    # Predict the pose
    preds = model.predict(img)
    max_confidence = np.max(preds)
    class_idx = np.argmax(preds)
    class_name = class_names[class_idx]

    # Display the prediction if confidence is above a certain threshold
    if max_confidence > 0.5:
        cv2.putText(frame, class_name, org: (10, 30), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 255, 0), thickness: 2,
                    cv2.LINE_AA)
    else:
        cv2.putText(frame, text: "No pose detected", org: (10, 30), cv2.FONT_HERSHEY_SIMPLEX, fontScale: 1, color: (0, 0, 255),
                    thickness: 2, cv2.LINE_AA)

    cv2.imshow( winname: 'Yoga Pose Recognition', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
```

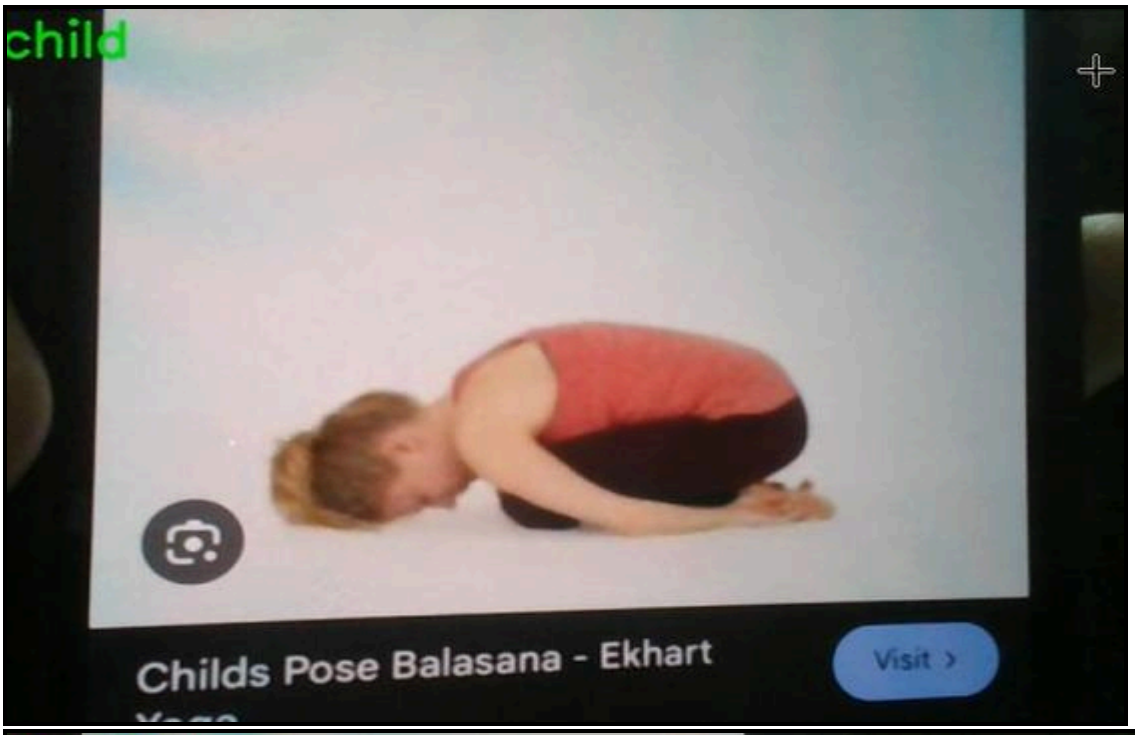
5.2 RESULTS

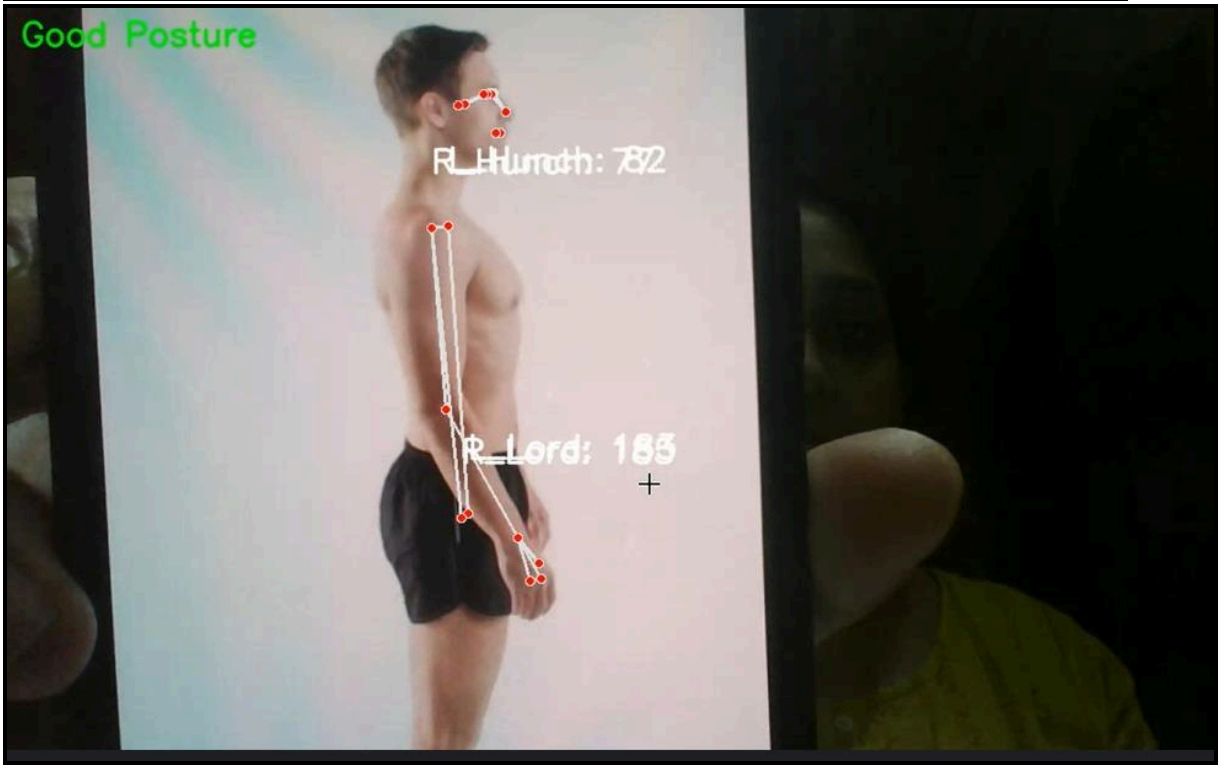



cobra


Kyphosis Detected
Recommended Yoga Pose: Cobra Pose
143


Lordosis Detected
Recommended Yoga Pose: Child's Pose
11:48
LUMBAR
Lower ba
curves in
the musc
the pelvi
Lordosis Detected
Recommended Yoga Pose: Child's Pose


child
Childs Pose Balasana - Ekhart
Visit ›


Good Posture

TEST CASES

Test Case ID	Input	Expected Output	Actual Output	Status Pass/ Fail
01	Image of person with Kyphosis	Kyphosis detected Recommended yoga pose-cobra pose	Kyphosis detected Recommended yoga pose-cobra pose	Pass
02	Image of person with Lordosis	Lordosis detected Recommended yoga pose-child's pose	Lordosis detected Recommended yoga pose-child's pose	Pass
03	Image of person with Kyphosis	Kyphosis detected Recommended yoga pose-cobra pose	Kyphosis detected Recommended yoga pose-cobra pose	Pass

04	Image of person with good posture	Good posture	Good posture	Pass
05	Image of person with Lordosis	Lordosis detected Recommended yoga pose-child's pose	Lordosis detected Recommended yoga pose-child's pose	Pass
06	Image of person with a good posture	Good posture	Good posture	Pass
07	Image of person with Kyphosis	Kyphosis detected Recommended yoga pose-cobra pose	Kyphosis detected Recommended yoga pose-cobra pose	Pass
08	Image of person with Lordosis	Lordosis detected Recommended yoga pose-child's pose	Lordosis detected Recommended yoga pose-child's pose	Pass
09	Image of person with a good posture	Good posture	Good posture	Pass
10	Image of person with Kyphosis	Kyphosis detected Recommended yoga pose-cobra pose	Kyphosis detected Recommended yoga pose-cobra pose	Pass

6.CONCLUSION AND FUTURE SCOPE

“Posture perfect: Real time detection and Yoga Guidance for Healthy Spine" comes across a significant development in healthcare technology, it’s aim is to improve spinal health through real-time posture detection and personalized yoga guidance. This innovative system utilizes Openpose and Posenet algorithms to monitor user’s postures continuously,they provide immediate feedback and corrections to promote healthier spinal alignment.There are many benefits of Posture Perfect .It not only aids in preventing spinal disorders such as kyphosis and lordosis but also enhances overall muscle and skeleton health by promoting correct posture habits. By improving yoga guidance, the system encourages users to take part in targeted exercises that strengthen the spine and surrounding muscles, further reducing the risk of injury and chronic pain.

Looking forward, the future scope of Posture Perfect is promising. Continued advancements in Openpose and Posenet technology will likely enhance its accuracy and functionality, making it more accessible and user-friendly. Additionally, expanding its capabilities to include a broader range of posture-related issues and integrating with wearable devices could make it an appearing tool for daily spinal health management. Moreover, integrating data analytics could provide insights into users' long-term posture trends, enabling personalized recommendations for preventive care. Collaborations with healthcare providers and ergonomic specialists could also leverage Posture Perfect as a tool for rehabilitation and therapeutic interventions.In essence, Posture Perfect not only addresses the immediate need for better posture but also sets a foundation for future innovations in digital health. By combining real-time posture detection with yoga guidance, it empowers users to take proactive steps towards maintaining a healthy spine

and improving overall quality of life.

7.REFERENCES

[1] https://www.computer.org/csdl/proceedings-article/icaa/2021/373000a700/1zL1XsxKlGw

[2] https://sci-hub.se/https://ieeexplore.ieee.org/document/7975660

[3] https://www.computer.org/csdl/proceedings-article/cvpr/2017/0457f759/12OmNAKcNOh

[4]https://.computer.org/csdl/proceedingsarticle/icpr/2004/212830322/12OmNxEjY4x

[5] https://www.computer.org/csdl/proceedings-article/ssteps/2022/641400a343/1Nm6UlWWFhe

[6] https://www.computer.org/csdl/proceedings-article/ccat/2023/015400a080/1Ui38PQjW5q

[7] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10215866/

[8] https://www.computer.org/csdl/proceedings-article/cvprw/2022/873900d539/1G57kKE6IWQ

[9] https://www.computer.org/csdl/proceedings-article/iccst/2021/425400a290/1ziPmr2aCcg

[10]https://doi.ieeecomputersociety.org/10.1109/CSCE60160.2023.

00294

	LIST OF FIGURES	
Fig No	**Fig Title**	**Page No**
1	Overall Architecture Diagram	28
2	Use case Diagram	30
3	Class Diagram	32
4	Activity Diagram	34

	LIST OF TABLES	
Table No	**Table Name**	**Page No**
1	Literature Survey	8
2	Test Case	46