Chapter 4

Proposed Methodology

4.1 Preamble

In this chapter, a method is proposed for the automatic correction of yoga poses using machine learning techniques. The approach processes images of yoga poses captured during practice. Key body alignment and positioning features are extracted using MediaPipe, allowing for accurate pose assessment and corrective feedback. This method ensures efficient and precise identification of incorrect postures, providing real-time suggestions to help users improve their practice.

4.2 Machine Learning Models for Pose Estimation

In the context of classifying yoga hand mudras using machine learning, pose estimation refers to detecting and analyzing the configuration of the hand in images or video. The goal is to identify and track the 21 keypoints on the hand, enabling machines to understand the specific gestures or mudras being performed. MediaPipe is an efficient tool for extracting these hand keypoints in real time. Unlike traditional pose estimation models that may rely on detecting the entire body, MediaPipe specializes in detailed hand tracking, making it particularly useful for this project.

Pose estimation models are commonly categorized as:

- **Top-down models:** These models detect a person first and then apply pose estimation to the detected region (e.g., OpenPose, PoseNet).
- **Bottom-up models:** These models directly detect keypoints without first detecting the person, which can be useful for multi-person tracking.

4.2.1 Introduction to Mediapipe

MediaPipe, developed by Google, is an advanced, open-source framework designed to simplify the creation of machine learning pipelines, particularly for real-time applications like hand tracking, pose estimation, facial recognition, and more. Its versatility stems from its modular design, which allows developers to customize various components according to their needs. MediaPipe excels in providing pre-trained models that are optimized for real-time, multimodal data processing, making it a go-to solution for projects requiring fast and efficient feature extraction, like yoga hand mudra classification.

One of the core strengths of MediaPipe is its cross-platform compatibility. It supports a wide range of devices, including mobile (Android, iOS), desktop, and even edge devices with limited computational power. This makes it ideal for applications that require on-device processing without the need for a powerful server or cloud-based infrastructure. For instance, in the case of classifying yoga hand mudras, users can interact with the system on their mobile devices, receiving real-time feedback on the accuracy of their mudra poses, all without significant latency.

MediaPipe's ability to process hand tracking data in real time is critical for yoga mudra classification. By detecting 21 keypoints in the hand, including the joints of all five fingers and key palm landmarks, MediaPipe allows for the precise tracking of hand gestures. This is essential for distinguishing between subtle variations in hand positions that define different mudras. The accuracy of these keypoints is further enhanced by MediaPipe's depth perception, which provides 3D coordinates, giving additional context to the orientation and position of the hand.

Another advantage of MediaPipe is its ease of integration. The framework is designed to be user-friendly, with APIs that allow developers to quickly implement hand tracking and pose estimation without having to build complex neural networks from scratch. Moreover, its modular structure means that developers can easily combine multiple models or features, such as hand tracking and facial recognition, into a single pipeline. This modularity is beneficial when expanding the yoga mudra classification system to include facial expressions or body posture for a more holistic understanding of the pose.

Performance-wise, MediaPipe is highly optimized for mobile and edge computing environments, ensuring low-latency processing even on devices with limited hardware capabilities. This real-time capability is a significant advantage for interactive applications like yoga mudra classification, where immediate feedback is crucial for learning and improvement. The framework's real-time processing, coupled with its efficient resource utilization, ensures that users experience smooth and responsive interactions, even on mobile platforms.

4.2.2 MediaPipe Hand Estimation: Extracting Hand Keypoints

For the task of classifying yoga hand mudras, MediaPipe's Hand Tracking solution efficiently extracts 21 keypoints from the hand. This system offers precise tracking of fingers, joints, and palm regions, enabling us to create models that classify mudras based on the relative positions and angles of these keypoints.

4.2.2.1 Key Components of MediaPipe Hand Estimation

- **Hand Tracking**: MediaPipe's Hand Tracking model is a powerful solution specifically designed to detect hand keypoints. It provides real-time 21-point tracking that identifies fingers and palm landmarks, even in challenging conditions like low light or hand occlusion.
- **Real-time Processing**: MediaPipe is optimized for real-time applications, providing low-latency keypoint extraction. This real-time capability is crucial for tasks such as yoga mudra classification, where live feedback is required.
- **Single Hand and Multi-hand Tracking**: The solution supports both single and multi-hand tracking, which is essential for classifying mudras involving one or both hands.
- **21 Keypoints**: MediaPipe Hand extracts 21 keypoints per hand, offering detailed information about the position of the fingers and palm, which is integral to differentiating various mudras.

4.2.2.2 The 21 Keypoints Includes

- **Palm**: Wrist.
- **Thumb**: Thumb_CMC, Thumb_MCP, Thumb_IP, Thumb_Tip.
- **Index Finger**: Index_MCP, Index_PIP, Index_DIP, Index_Tip.
- **Middle Finger**: Middle_MCP, Middle_PIP, Middle_DIP, Middle_Tip.
- **Ring Finger**: Ring_MCP, Ring_PIP, Ring_DIP, Ring_Tip.
- **Little Finger**: Little_MCP, Little_PIP, Little_DIP, Little_Tip.

4.2.2.3 How MediaPipe Extracts Keypoints

- **Feature Extraction**: MediaPipe Hand uses a convolutional neural network (CNN) to extract features from an input image or video frame. These features capture the spatial arrangement of the hand, which is essential for mudra classification.
- **Hand Segmentation**: Before identifying keypoints, the model isolates the hand from the background, ensuring accurate keypoint detection in cluttered environments.
- **Keypoint Detection**: After segmentation, the model predicts the 21 keypoints on the hand. These keypoints are represented as (x, y, z) coordinates, where:
  - **x, y**: Correspond to the keypoint's position in 2D space.
  - **z**: Provides depth information, adding 3D context to the hand's posture.

This 3D data, particularly the z-coordinate, is crucial for understanding the orientation of the hand and differentiating between similar mudras. After extracting these keypoints, they can be used to create a skeleton representation of the hand, which is essential for tracking the movement and configuration of the fingers and palm.

4.3 Random Forest

Random Forest is an ensemble learning method that is widely used for classification, regression, and other tasks in machine learning. It is based on the concept of constructing multiple decision trees and combining their outputs to improve accuracy and reduce overfitting. The algorithm operates by building a "forest" of decision trees during training and then aggregating their predictions to make the final decision.

4.3.1 Key Concepts of Random Forest

- **Ensemble Learning**: Random Forest belongs to the ensemble learning category, which means it combines the predictions of multiple models to produce a better result. Instead of relying on a single decision tree, Random Forest builds several trees, each trained on a random subset of the data, and averages their predictions.
- **Decision Trees**: At its core, Random Forest is composed of decision trees. A decision tree is a flowchart-like model where each internal node represents a "test" on a feature (e.g., whether a feature's value is above or below a threshold), each branch represents the outcome of the test, and each leaf node represents a class label (in classification) or a value (in regression).
- **Random Subsets of Data**: Random Forest introduces two levels of randomness:
- **Bootstrap Sampling**: Each tree is trained on a random subset of the data (called a bootstrap sample). Some samples may be repeated, while others are excluded.
- **Random Feature Selection**: Instead of considering all the features when splitting nodes in a tree, Random Forest selects a random subset of features. This helps in reducing correlation between the trees and ensures that the trees are diverse.
- **Majority Voting for Classification**: In classification tasks, each tree in the forest makes its own prediction (votes for a class). The final prediction is based on a majority vote across all the trees. For example, if 60 trees vote for class A and 40 trees vote for class B, the final prediction will be class A.

- **Averaging for Regression**: In regression tasks, instead of voting, the final prediction is the average of all the trees' predictions. This averaging process helps in smoothing out any noise and reduces the risk of overfitting.
- **Out-of-Bag (OOB) Error**: Since each tree is trained on a bootstrap sample, about one-third of the original data is left out during the training of any given tree. This "out-of-bag" data can be used to evaluate the model's performance without needing a separate validation set. OOB error is a reliable estimate of model accuracy.
- **Feature Importance**: One of the most useful features of Random Forest is its ability to measure feature importance. After training, Random Forest provides a ranking of which features contribute the most to making predictions. This is helpful in understanding the underlying patterns in the data and for feature selection.
- **Parallelizable**: Since each tree in the Random Forest is independent of the others, the algorithm can be parallelized easily. This allows it to scale efficiently on large datasets and across distributed computing environments.

4.3.2 How Random Forest Works

- **Training Phase**:
    - Random Forest builds multiple decision trees using different random subsets of the training data and features.
    - Each tree is grown to its maximum depth, without pruning, to capture as much information as possible from the training set.
    - Trees are constructed in parallel, which speeds up the training process.
- **Prediction Phase**:
    - For a classification task, when a new input is given, each decision tree in the forest makes a prediction. The Random Forest aggregates the predictions from all trees using majority voting to make the final decision.

- For regression tasks, the model takes the average of all the individual tree predictions.

4.3.3 Advantages of Random Forest

- **Robustness**: By combining multiple trees, Random Forest reduces the variance of predictions and is less likely to overfit, especially when compared to individual decision trees.
- **High Accuracy**: The ensemble method generally yields better accuracy than using a single decision tree, especially on large datasets with high dimensionality.
- **Handling Missing Data**: Random Forest can handle missing values naturally by splitting the data based on available information. It can also provide reasonable estimates of missing values.
- **Works Well with Both Categorical and Continuous Data**: Random Forest can handle datasets that include both types of data, making it flexible for various types of problems.
- **Feature Importance**: It provides insight into which features are the most relevant, which can be used for feature selection in other models or for understanding the dataset better.

4.3.4 Limitations of Random Forest

- **Slow for Real-Time Predictions**: Because Random Forest aggregates predictions from multiple trees, it can be slower for making predictions, especially with a large number of trees.
- **Complexity**: While Random Forest is easier to tune than many other models, it can still be more complex to interpret than simpler models like decision trees or linear models. The final model is often considered a "black box."
- **Memory Intensive**: Since Random Forest builds multiple decision trees, it can consume a lot of memory and computational resources, especially with large datasets and many trees.

- **Not Suitable for Very High-Dimensional Data**: Although Random Forest can handle high-dimensional data, its performance may degrade if the number of features becomes extremely large, especially if the majority of features are irrelevant.

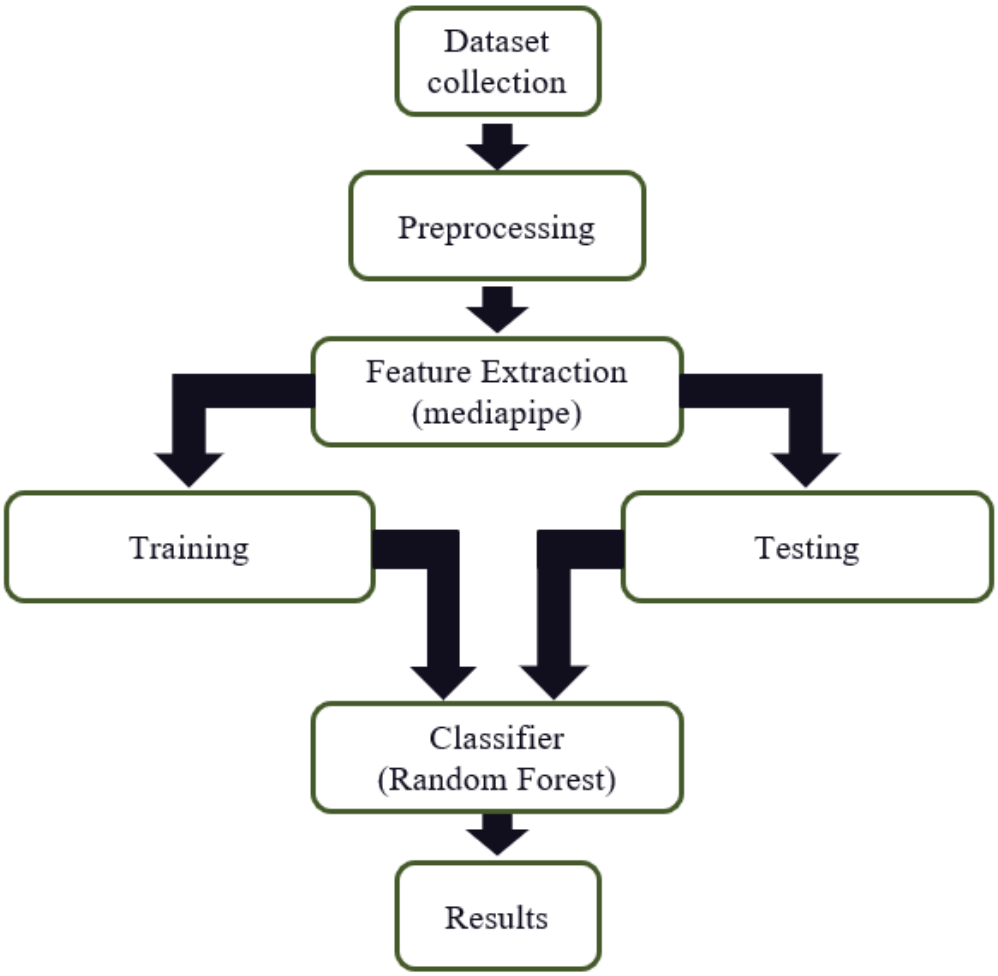

Fig 4.1 Architecture

The Yoga Hand Mudra Classification System guides users through a structured process of data collection, preprocessing, feature extraction, and classification. Initially, hand gesture data is collected and augmented during preprocessing to ensure a diverse set of training examples. MediaPipe is utilized to extract 21 keypoints from the hand, capturing the essential features for mudra recognition. The extracted features are then split into training and testing sets, ensuring the model is evaluated accurately. Finally,

a Random Forest classifier is applied to categorize the mudras based on the features. This systematic approach enhances the precision of mudra classification, promoting effective and accurate recognition during yoga practice.