Application for Human Movement Analysis Based on Pose Estimation

Final Project Report

Team MCS11

Emily Chin Jing Yi
Project Manager
echi0029@student.monash.edu

Josh Hernett Tan
Quality Assurance
jtan0319@student.monash.edu

Lai Zhen Yoong
Tech Lead
zlai0012@student.monash.edu

Raveendran Paramesran
Supervisor
raveendran.paramesran@monash.edu

Table of Content

1. Introduction

Physical fitness and well-being have become an integral aspect of our lives alongside the increasing awareness of the importance of it. With the recent COVID19 pandemic, the number of people exercising at home has increased, hence the increase in the importance of having more accessible guidance through one's fitness journey. This demand is met by the numerous online exercise instruction videos made available to the public, however these systems are passive and are not specifically tailored to one (*AI Fitness Coach at Home Using Image Recognition*, n.d.). This report documents the journey and accomplishments of our team in developing an exercise form correction application by utilising the cutting-edge computer vision technology to provide timely and invaluable feedback to the users. This does not only help prevent the risk of injuries and ensure the productivity and effectiveness of one's workout, it would make the workout feel more interactive.

Our main focus as per our project topic would be on computer vision in human movement analysis. As mentioned before in the initial proposal, here is a reiteration of the general framework of human motion analysis as a refresher. Human motion analysis consists of three main components: Human Detection, Human Tracking and Behaviour Understanding ("Recent Developments in Human Motion Analysis," n.d.). Our project puts a large emphasis on human detection and human tracking, less on behaviour understanding. This is because the objective of our project is to extract information from the human body such as joint points, and the angles between the joints in order to classify the correctness of each action, we are not attempting to derive or recognise specific actions. This is where pose estimation comes into play.

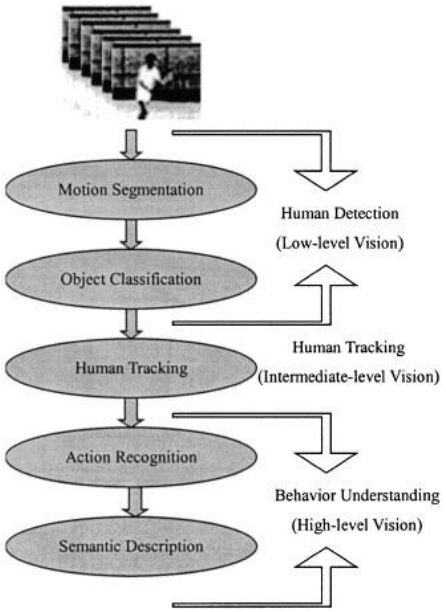

Fig. 1: General Framework. ("Recent Developments in Human Motion Analysis," n.d.)

In recent years, previous state-of-the-art machine learning techniques have been outperformed by deep learning methods in various fields, with computer vision being one of the more prominent cases (Voulodimos et al., 2018). One of the more common methods known to the public would be the Convolutional Neural Networks (CNN). CNN is an image recognition algorithm based on the visual perception process of biology (Liu, n.d.). It is composed of 3 main layers: Convolutional Layer, Pooling Layer and Fully Connected Layer, each with specific tasks that would be further elaborated in the later parts of this report.

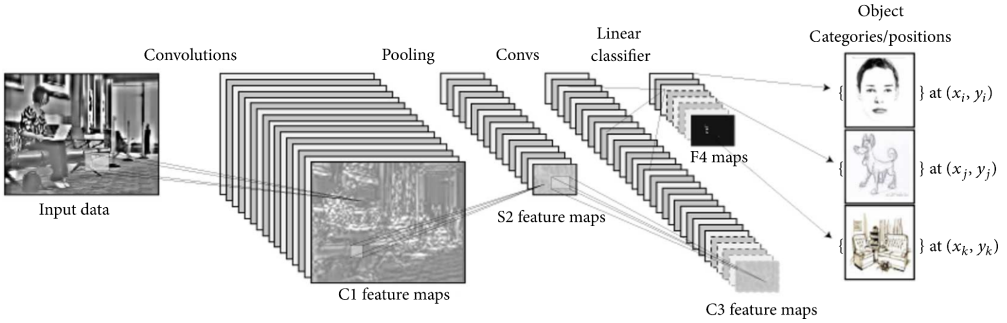

Fig. 2: Architecture of a Convolutional Neural Network. (Voulodimos et al., 2018)

2. Project Background (EM)

2.1 Background

2.1.1 Computer Vision for Human Movement Analysis

Human movement analysis with computer vision typically involves 4 main components, which are Human Motion Analysis, Tracking, Pose Estimation, and Recognition.

Human Motion Analysis

A famous experiment in 1887 called "Animal Locomotion" conducted by the pioneer in motion-capturing - Edward Murbridge (1830-1904) was a kickstarter to the growth of human motion analysis in various fields. The emergence of "computer vision" 50 years ago solves two main tasks:

1. Detecting correspondence between subsequent images,
2. And tracking of an object within a sequence of images.

There are multiple methodologies that follow a general framework composed of feature extraction, feature correspondence and feature extraction (Aggarwal & Cai, 1999), such as tracking 2D features at a local or global level, or tracking based on the projection of a generic 3D model (*Human Motion*, n.d.).

Fig. 3: Generic 3D Model of the Upper Human Body (Human Motion, n.d.)

Tracking

The definition of tracking in this content is "establishing coherent relations of the subject and/or limbs between frames" (Aggarwal & Cai, 1999). The main components are:

1. **Figure-Ground Segmentation**

The Various Characteristics and Subclasses of Figure–Ground Segmentation Approaches

Temporal data		Spatial data	
Subtraction	Flow	Threshold	Statistics
Two images	Points	Chroma-keying	Pixels
Three images	Features	Special clothes	Blobs
Background	Blobs	IR	Contours

Fig. 4: Figure-Ground Segmentation Approaches (Aggarwal & Cai, 1999b)

2. **Representation**

The Various Types of Data Representations

Object-based	Image-based
Point	Spatial
Box	Spatio-temporal
Silhouette	Edge
Blob	Features

Fig. 5: Types of Data Representations (Aggarwal & Cai, 1999b)

3. **Tracking Over Time**
   This represents the identification of objects of different representations in consecutive frames.

Pose Estimation

The tracking and calculating of the human skeleton during the movement of a subject is what pose estimation represents. It can play both active and passive parts in a tracking algorithm depending on the context.

It involves the use of sensor data, such as images or depth information, to compute the

3D position and orientation of objects or people relative to a reference frame. This technology has the potential to enhance navigation, object recognition, and spatial understanding in both industrial and consumer application (Schiele, n.d.-a).

Recognition

Recognition is the post processing in an attempt to achieve the classification of motions into one of several types of actions (Aggarwal & Cai, 1999b).

- **Static Recognition**
  Comparison of "known information" inclusive of templates, postures, silhouettes etc and the system with a given image. Spatial data are dealt frame by frame.

- **Dynamic Recognition**
  Dynamic recognition utilises temporal characteristics for low-level (typically based on spatio-temporal data) and high-level recognition (typically based on pos estimated data).

2.1.2 Deep Learning for Computer Vision

Deep learning for computer vision is a subfield of artificial intelligence that focuses on using deep neural networks to enable the understanding of machines and interpretation of visual information. There are a few key aspects of deep learning for computer vision:

1. **Convolutional Neural Networks (CNN)**
   CNNs are designed to automatically extract relevant features from images through a series of convolutional layers, pooling layers, and fully connected layers. They have achieved remarkable success in image classification and object detection tasks.

2. **Recurrent Neural Networks (RNNs)**
   Sequential data to represent time dependencies are generally dealt with by RNNs. Some examples of such tasks include translating natural languages, music, time-series data, video processing etc.

3. **Restricted Boltzmann Machines (RBMs)**
   RBMs are mainly used in unsupervised machine learning. It is a stochastic, generative, and energy-based model that excels at capturing complex patterns and dependencies within data (2017).

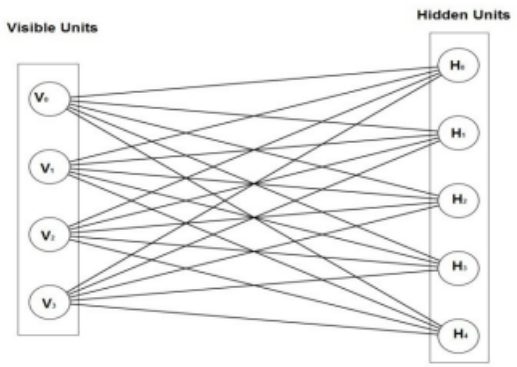

Fig 6: Architecture of RBMs (2017)

2.2 Rationale

2.2.1 Need for Computer Vision in Sports

Information technology, let alone the sports sector, has become inextricably linked. Technology in sports is used not just to improve one's performance, but it is also integrated into assessment systems to help with decision making in sports training or tournaments. *(Thomas et al., 2017)*

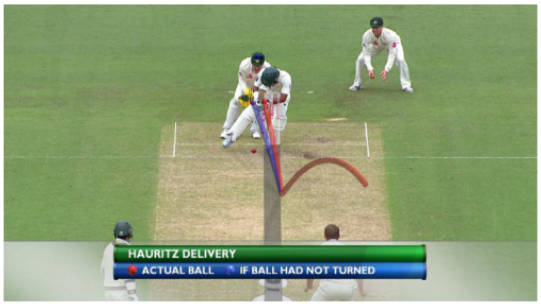

Fig. 7: Analysis and Prediction of cricket ball motion. (Thomas et al., 2017)

Computer vision has advanced to the point that it now plays an important part in the world of sports, from sports analysis for broadcast to training and coaching (Thomas et al., 2017). In this study, we are investigating the use of computer vision into kinematic analysis.

There are already a few examples of computer vision being used to improve sports performance, one of which is video-based motion analysis systems. However, one of the key disadvantages of such technology, aside from its expense, is that it takes time to manually capture the films while also keeping the correctness of the coordinates of the joints and other factors crucial for the analysis in mind. Having said that, real-time analysis is not yet established (Liebermann et al., 2002).

2.3 Literature Review

2.3.1 Pose Estimation

The use of human models is a widely-known aspect of pose estimation. Pose estimation can be divided into 3 main classes (Aggarwal & Cai, 1999b):

Model-Free Class
The model-free class encompasses techniques that employ no priori models. Instead, a model is created to represent the stance; examples of pose representations include dots, basic forms, stick figures, and boundary boxes.

Indirect Model Use
The use of priori models as a reference or look-up table as guides to extract information for interpretation of measured data is referred to as indirect model use. Pose is typically represented by the head, hands, limbs, or a general description of the human body.

Direct Model Use
In contrast, direct model use implies that an a priori model is used directly as the model to represent the observed subject. A human model will significantly improve the ability to manage occlusion and integrate different kinematic limitations into a system. The number of joints and connecting bones in the human model represents its complexity.

By leveraging methodologies such as deep learning and RGB or depth sensors, pose estimation has the potential to revolutionise the way we understand and engage with sports. From tracking the precise movements of soccer players on the field to providing biomechanical insights for athlete training, the impact of pose estimation in sports is substantial and continues to expand (M. Manafifard, 2017).

2.3.2 Convolutional Neural Networks

CNNs have emerged as a powerful tool in sports analysis due to its competency as one of the most widely-known key aspects to deep learning. It enables comprehensive and detailed examination of movements, performance, and strategies, revolutionising various aspects of sports analysis (Mgaya et al., 2021).

It is specifically designed for the processing of grid-like structured data, such as images. An overview of the architecture is as follows:

Convolutional Layer

These layers are the core building blocks of CNNs. They consist of filters (also known as kernels) that are applied over the input image to detect local patterns and features, producing a feature map that highlights specific patterns (Albawi, Mohammed, & Al-Zawi, 2017).

Activation Function

The usage of an activation function is crucial to provide non-linearity to the model, allowing it to learn complex patterns. In the case of the absence of an activation function, hidden layers will have no effect as the output will always be linear. The more frequently used activation functions include: Sigmoid, RELU, tanh and so on (Li et al., 2022).

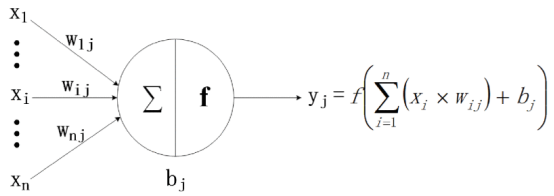

Fig. 8: General Activation Function Structure (Li et al., 2022)

Pooling Layer

Pooling layers are used to downsample the spatial dimensions of the feature maps while retaining the important information. This reduces the computational load of the model, decreases the sensitivity to small spatial variations and helps to provide more variation to the model by exposing it to small translations and rotations (Li et al., 2022).

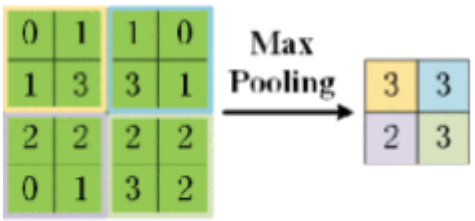

Fig. 8: Output of Pooling Layer (Li et al., 2022)

Fully Connected Layer

Fully connected layer literally represents a layer where each neuron is connected to every other neuron in the previous layer. These layers enable high-level feature combination and decision making (Li et al., 2022).

3. Outcome

The resulting product of this project consists of two main components:

1. Client (Web) Application
2. Prediction Model for Form Correction

Our project is relatively successful in the sense where we provided a feasible solution to our initial problem statement and met most part of the project requirements. A critical discussion stating the deviation from our initial proposal throughout the implementation will be included in a separate section of the report.

3.1 Deliverables

3.1.1 Client Application

A simple web application has been developed to allow users to interact with the prediction model implemented in the backend module through a user-friendly interface. The web application allows users to select a specific exercise, and grants them an option between uploading a pre-recorded footage for the classification model or to use the "real-time" feature for exercise form correction.

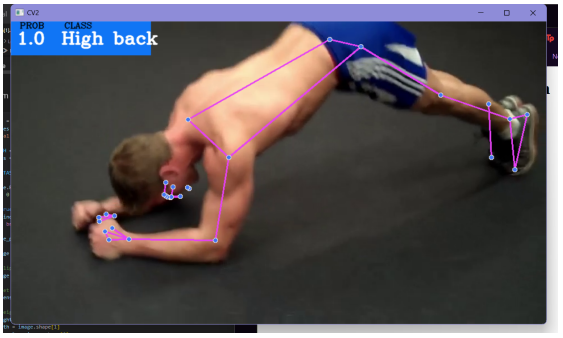

Fig. 9: Output of Pre-Recorded Footage Form Correction

Fig. 10: Output of "Real-Time" Form Correction

3.1.2 Model Evaluation

Metrics are utilised to gauge the effectiveness of a trained machine learning model. Assessing machine learning models or techniques is crucial for any endeavour. Numerous evaluation metrics exist to examine a model's performance. In this thesis, the following metrics were employed to assess the trained models:

1. Confusion matrix: This offers a comprehensive breakdown of the classification outcomes. For optimal model performance, it's essential to have high True Positive (TP) and True Negative (TN) values, while keeping False Negative (FN) and False Positive (FP) as low as possible.

    - True Positive (TP): Represents the count of positive samples predicted accurately.
    - False Positive (FP): Denotes the count of negative samples mistakenly identified as positive.
    - True Negative (TN): Indicates the count of negative samples predicted correctly.
    - False Negative (FN): Refers to the count of positive samples erroneously marked as negative.

2. Precision: Represents the proportion of true positives out of all the predicted positives.

$$precision = \frac{TP}{TP + FP}$$

3. Recall: Indicates the fraction of actual positives that the model correctly classifies. It is also known as Sensitivity.

$$recall = \frac{TP}{TP + FN}$$

4. F1 score: Calculated as the harmonic mean between precision and recall. A higher F1 score is achieved when both precision and recall are high.

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

5. F1 score curve: A graph illustrating the balance between the F1 score and varying thresholds. By monitoring the fluctuations in the F1 score, this curve assists in determining the best threshold for the model.

6. ROC curve (Receiver Operating Characteristic curve): A graphical representation that depicts the efficacy of a classification model across all potential classification thresholds. It plots the True Positive Rate (TPR) against the False Positive Rate (FPR).

$$TPR = \frac{TP}{TP+FN}$$
$$FPR = \frac{FP}{FP+TN}$$

7. AUC (Area Under the ROC Curve): Quantifies the total two-dimensional space below the entirety of the ROC curve. AUC offers a comprehensive evaluation of model performance over all conceivable classification thresholds.

1. Bicep Curl

The bicep curl is a well-known weightlifting exercise targeting the upper arm muscles, with minor effects on the lower arm muscles. It's typically done standing up, with the arms lifting and lowering a dumbbell or barbell. In this thesis, three common mistakes associated with the bicep curl are highlighted:

- **Loose/Unsteady Upper Arm**: Instead of remaining stationary, the upper arm shifts when one arm is raised during the exercise.
- **Weak/Incomplete Peak Contraction**: The arm doesn't rise high enough, preventing optimal bicep contraction.
- **Excessive Backward Lean**: The exercise leans back and forth, using a swinging motion during the curl.

Based on my study, key MediaPipe Pose landmarks essential for this exercise include the nose, left shoulder, right shoulder, right elbow, left elbow, right wrist, left wrist, right hip, and left hip.

a. **Loose/Unsteady Upper Arm**

This can be identified by measuring the angle formed between the elbow, shoulder, and the projection of the shoulder on the ground. Based on my findings, if this angle exceeds 40 degrees, the movement will be categorised as an "unsteady upper arm" mistake.

b. **Weak/Incomplete Peak Contraction**

This is discerned by determining the angle among the wrist, elbow, and shoulder as the exerciser's arm is raised. From my studies, if this angle goes beyond 60 degrees before the arm starts descending, it's labelled as a "suboptimal contraction at peak" error.

c. **Excessive Backward Lean**

Given the intricacy of identifying this error, machine learning techniques are employed. Training data is sourced from video recordings of individuals both exhibiting and not exhibiting this mistake. Figure 11 visually illustrates the distribution of frames extracted from these videos according to their classifications. The dataset comprises 15,372 images. Out of these, 8,238 images (or 53.6%) are categorised under the correct posture (C), while 7,134 (or 46.4%) are labelled as leaning back (L). Each entry in the dataset features 37 columns, structured based on its classification and pertinent landmarks.

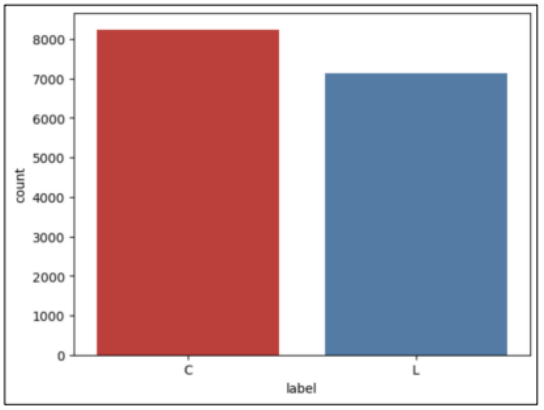

Fig. 11: Bicep Curl Dataset Count

Our training approach utilises a blend of machine learning algorithms from Scikit-learn and four distinct neural network architectures. The outcomes from training this combined model can be observed in *Table 1*.

Model	Neural Network	Precision	Recall	Accuracy
KNN	False	0.975	0.968	0.972
7 layers	True	0.972	0.962	0.967
5 layers	True	0.963	0.949	0.955
SVC	False	0.93	0.934	0.932
RF	False	0.947	0.925	0.931
7 layers with dropout	True	0.936	0.924	0.93
3 layers	True	0.939	0.92	0.929
LR	False	0.792	0.738	0.762
SGDC	False	0.712	0.715	0.715

Table 1: Model Training (Bicep Curls)

Based on the metrics presented in Table 1 from the training experiments for the Bicep Curls error, the K-Nearest Neighbors (KNN) model emerges as the most effective. As indicated below, this model produces commendable outcomes.

	Correct form	Lean too far back
Correct	338	1
Lean too far back	16	249

Table 2: Confusion matrix (Bicep Curl Error)

	Precision	Recall	F1 Score
Correct	0.955	0.997	0.975
Lean too far back	0.996	0.94	0.966
Average	0.976	0.968	0.971

Table 3: Evaluation results (Bicep Curl Error)

The selection of a confidence threshold impacts the model's evaluation and detection outcomes. Picking the right confidence threshold leads to improved results. Reviewing the F1 and ROC curves below aids in determining the ideal confidence threshold. As seen in the F1 curve graph, the lines representing different classes are closely aligned, and a threshold close to 0.5 appears to offer the best F1 score.

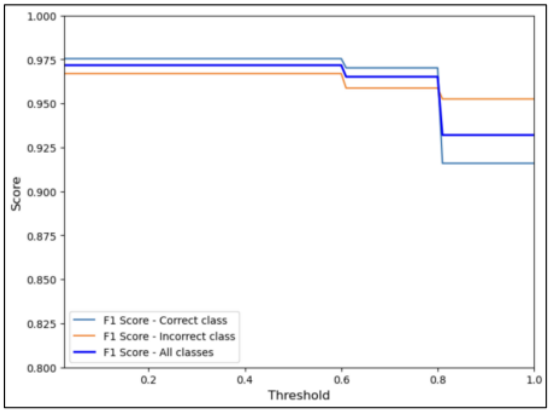

Fig. 12: - F1 curve (Bicep Curl Error)

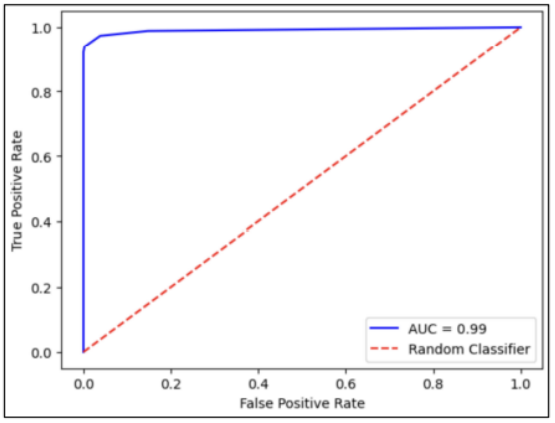

Fig. 13: - ROC curve (Bicep Curl Error)

2. Basic Planks

The plank, often referred to as planking, is an exercise focused on engaging core muscles, enhancing strength, balance, and stamina. To execute a plank, one should lie flat on the ground, ensuring the elbows align with the shoulders and the feet are spread shoulder-width apart. Then, elevate the body, placing the weight on the forearms and feet, ensuring the body remains in a straight line.

This thesis highlights three typical mistakes seen during a basic plank:

Elevated lower back: This error occurs when the lower back arches upwards and isn't maintained straight.
Sagging lower back: In this error, the lower back dips down excessively, rather than staying aligned.

Through my research, the pivotal MediaPipe Pose markers for this activity include: the nose, both shoulders, both elbows, both wrists, both hips, both knees, both ankles, both heels, as well as the index region of both feet.

Videos of participants demonstrating all three stages of the exercise (correct form, elevated lower back, and sagging lower back) were utilised to develop a machine learning model. Figure 14 showcases a graphical representation of the number of frames extracted from these videos and their corresponding categories. The dataset comprises a total of 28,623 images. Out of these, 9,630 samples (33.6%) are from the correct form category (C), 8,982 samples (31.4%) are from the elevated lower back category (H), and 10,011 samples (35%) are

from the sagging lower back category (L). Every entry in this dataset has 69 columns, derived from its categorization and the relevant landmarks.

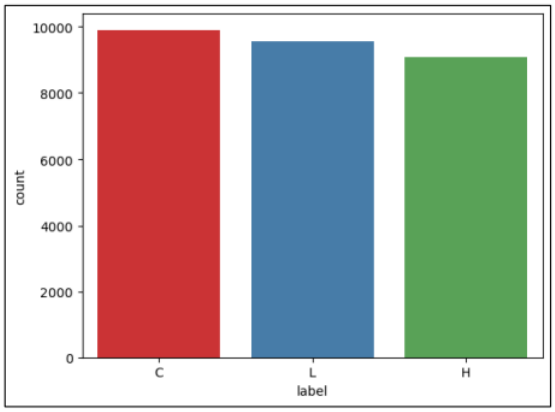

Fig. 14: Plank Dataset Count

The outcomes of the experiment for training the aforementioned model are displayed in Table 4

Model	Neural Network	Precision	Recall	Accuracy
LR	False	0.996	0.996	0.996
7 layers with dropout	True	0.994	0.994	0.994
SVC	False	0.987	0.987	0.987
SGDC	False	0.981	0.981	0.981
KNN	False	0.955	0.949	0.949
5 layers	True	0.934	0.93	0.93
7 layers	True	0.935	0.924	0.924
RF	False	0.922	0.899	0.899
3 layers	True	0.869	0.848	0.848

Table 4: Model Training (Planks)

Based on the metrics presented in Table 4 from the model training experiments focused on detecting errors in the Plank exercise, the Logistic Regression (LR) model emerges as the most effective. The results from this model, as detailed below, are commendable.

	Correct	High back	Low back
Correct	234	0	0
High back	1	240	0
Low back	2	0	233

Table 5: Confusion matrix (Plank Error)

	Precision	Recall	F1 Score
Correct	0.987	1.0	0.994
High back	1.0	0.996	0.998
Low back	1.0	0.991	0.996
Average	0.996	0.996	0.996

Table 5: Evaluation results (Plank Error)

Examining both the F1 curve and the ROC curve presented below assists in determining the best confidence threshold. As indicated by the F1 curve chart, the lines representing different classes are closely aligned, and the threshold that delivers the highest F1 score is approximately 0.6.

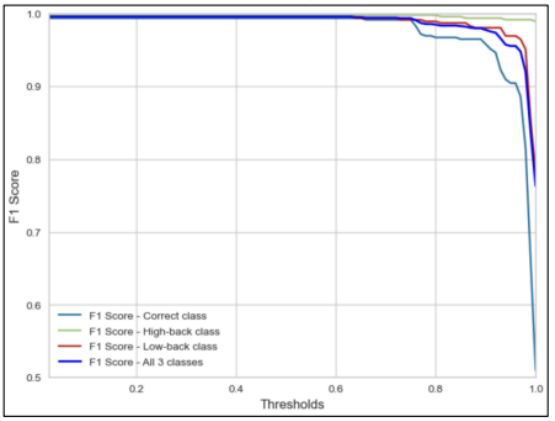

Figure 15 - F1 curve (Plank Error)

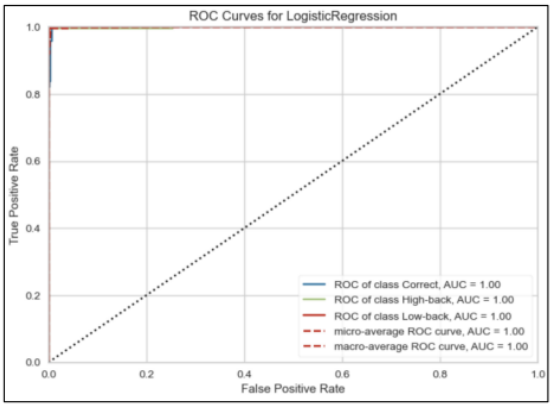

Figure 16 - ROC curve (Plank Error)

3.2 Requirements Met

Table 6 represents the functional and non-functional requirements for the project.

Table 7 represents how the requirements are addressed.

ID	Requirements
01	Detection and classification accuracy that is reasonable high
02	Inference speed
03	In real time, the AI model can identify and analyse user postures
04	The model must maintain constant

	accuracy in a range of illumination situations

Table 6: Requirements to be met

ID	How Requirements are Met
01	Our final model reliably meets this criterion, consistently demonstrating an accuracy level above the 70% threshold for various exercises and movements. The accuracy can be seen in real-time on the corner of the video output (beside the feedback).
02	Our developed system consistently achieves and, in some cases, even exceeds the target frame rate of 15 fps. Such a speed ensures that the application provides real-time feedback, allowing users to make instantaneous corrections to their movements or postures.
03	The application successfully provides instant feedback to users, guiding them to maintain proper posture while performing exercises.
04	Our model exhibited strong performance in most lighting scenarios, particularly in daylight and under coloured lights. However, in darker conditions, the model's accuracy was somewhat compromised. This limitation is attributed to the inherent challenges of distinguishing subtle movement features in low light. While the model has been optimized for a wide range of conditions, achieving consistent results in darker settings remains an area for future improvement.

Table 7: How Requirements are Met

3.3 Justification of Decisions Made

3.3.1 Mediapipe as Pose Estimation Framework

When planning for which pose estimation framework to use, OpenPose was first considered. The decision to opt for MediaPipe over OpenPose for pose estimation was multifaceted, influenced by both technical and practical considerations. At the outset, MediaPipe's framework offers superior performance by presenting a model architecture that is both lighter and more efficient compared to OpenPose. This translates to notably faster inference times, a critical factor when the goal is to provide users with real-time feedback on their movements. The immediacy of feedback not only enhances user experience but is also vital for applications centred around physical activities where delays can lead to incorrect posture or even potential injuries. Another pivotal reason for choosing MediaPipe was its ease of integration. With its pre-built solutions and modular components, incorporating MediaPipe into various platforms and applications became a more streamlined process. Additionally, given its lightweight nature, MediaPipe demands considerably less computational power and memory. This ensures that the application remains agile and responsive even on devices with limited resources, widening its potential user base. Furthermore, the backing of Google for MediaPipe meant access to robust support, frequent updates, and a vibrant community. This assured us of the framework's reliability and future-proof nature, making it a strategic choice for the long-term viability and success of our project.

3.3.2 Flask for Web Development

Choosing Flask for our web development stemmed primarily from its harmonious compatibility with Python, our core programming language. This ensured easy integration between our backend and frontend operations. Flask's minimalist design allowed for rapid development and prototyping, enabling us to adapt and iterate quickly. Moreover, its modular structure meant we could easily expand functionalities with various plugins, ensuring the application's flexibility. In essence, Flask was selected for its technical advantages, adaptability, and streamlined development process.

3.5 Limitation of Project Outcome

3.5.1 Camera Quality and Environment

- Low-resolution video input; the accuracy of pose estimation algorithms significantly deteriorates with lower image quality. Lower-resolution images lack the detailed pixel information necessary for precise feature detection, leading to less accurate landmark detection and feedback.
- Poor lighting conditions compromise the visibility of the subject being monitored, leading to erroneous interpretations by the model. Shadows or overexposure can obscure landmark points of the person.

3.5.2 Multi-Person Recognition

Due to the reliance on MediaPipe, the application can only recognize and analyse movements from a single person at a time. This limitation narrows the usability scope of the application, especially for group fitness sessions or comparative studies of athletic performance.

3.5.3 Limited Exercise Coverage

The current architecture requires the development and deployment of separate models for each type of exercise, increasing the application's complexity, resource usage, and scaling difficulty. This approach restricts the number of exercises the application can assess and provide feedback on.

3.5.4 Dataset Constraints

The scarcity of publicly available, diverse, and labelled datasets for niche exercises limits the training material necessary for the model to learn effectively. It restricts the model's understanding of varied human anatomical structures, clothing, and exercise environments. A more comprehensive dataset could enhance the detection of subtle but important mistakes in exercise postures, such as head tilting during planks, potentially preventing strain or injury.

3.5.5 Specific User Demographic

The application was developed with a focus on individuals who perform exercises within a 'standard' range of motion and physical ability. It does not account for the unique challenges or specific needs of individuals with disabilities or pre-existing health conditions. This oversight means the application might not recognize or correctly interpret movements from people who perform exercises differently due to their conditions, potentially leading to incorrect feedback.

3.6 Possible Improvement and Future Works

3.6.1 Cloud Storage Implementation

By allowing users to upload and securely store their videos, they can easily access and review their past performances. This feature is not just a matter of convenience but a powerful motivational tool. The ability to visually track one's progress over time, by rewatching the videos and possibly sharing them with trainers or health professionals for further feedback, stands as a compelling reason for users to engage consistently with the application. This archival of videos serves as a repository of personal achievements, encouraging users to set new goals based on tangible proof of their progress.

From a technical standpoint, the scalability and reliability of cloud storage make it an excellent choice for an application experiencing growth. As the user base expands, so too does the volume of data. Cloud storage accommodates this growth with its scalable infrastructure, removing the need for developers to engage in the cumbersome

process of manual scaling. This ensures that the application remains stable and reliable, even as the stored videos' volume grows exponentially.

3.6.2 Expansion and Diversification of Dataset

A significant limitation of our model is the lack of inclusivity in datasets. People come in all shapes, sizes, and abilities. A dataset that lacks this diversity inadvertently leads to a model with built-in biases, limiting its applicability and fairness.

To further improve the model, implement techniques for synthetic data generation (like GANs) to augment existing datasets, allowing for richer and more diverse training data.

Different body types, clothing, lighting conditions, and environments ensure that the model is robust and reliable across various scenarios, not just the ones it was predominantly trained on.

3.6.3 Optimising Model Architecture

Moving from multiple models to a single, modular structure eliminates the need for separate maintenance and computational processes for each exercise type. This unified approach simplifies the system's backend, reducing computational load and streamlining updates, leading to a more responsive and stable application.

A modular model architecture offers seamless scalability. As we introduce new exercises or expand user interactions, a singular model can easily incorporate these enhancements without overcomplicating the system. This adaptability ensures the application's growth capacity isn't bottlenecked by its technical infrastructure.

3.6.4 Porting to Mobile

Smartphones are pervasive, and their usage continues to grow globally. Creating a mobile version of an application means tapping into a vast user base, allowing more people to access the application's benefits without the need for a computer or dedicated hardware.

Mobile devices come equipped with built-in cameras, which are crucial for the application’s functionality. Direct integration eliminates the need for external hardware, making the process of capturing motion for analysis significantly more seamless. They are also equipped with various sensors that can augment the application’s input data, such as accelerometers, gyroscopes, and even health data trackers. These can provide additional data points for more comprehensive feedback on user movements.

4. Methodology

4.1 Related Libraries and Software

1. MediaPipe

MediaPipe offers an open-source framework designed for constructing pipelines that carry out computer vision analysis on diverse sensory inputs, including video and audio. With MediaPipe, one can create a perception process in the form of a graph made up of modular elements. It caters to machine learning (ML) teams, software developers aiming for production-level ML applications, and academicians or researchers who share code and prototypes aligned with their studies.

Within computer vision processes, these modular elements encompass model predictions, media algorithm processing, data alterations, and more. Sensory inputs, like video feeds, are fed into this graph, yielding results such as object positioning or facial keypoint sequences. The object detection procedure using MediaPipe can be visualised in Figure 17.

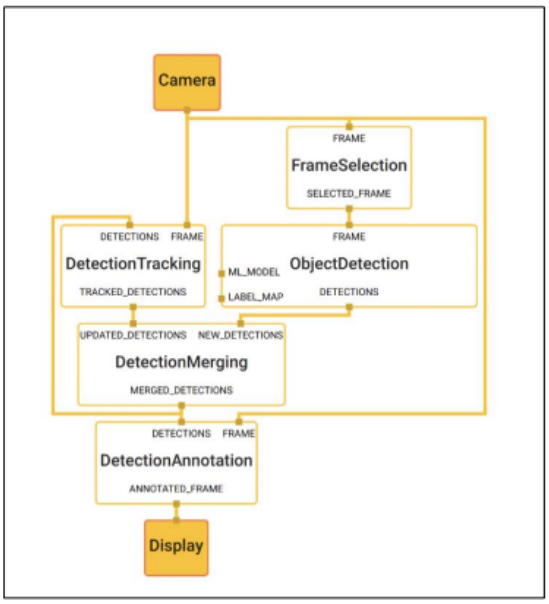

Fig. 17: MediaPipe Architecture

MediaPipe Pose stands as a machine learning tool designed for advanced body pose tracking. It can deduce 33 3D landmarks along with a background separation mask for the entire body using RGB video frames. While many contemporary leading methods depend mainly on robust desktop systems for inference, this technique ensures real-time operation across various platforms, including most up-to-date mobile phones, desktops, laptops, Python environments, and even web browsers.

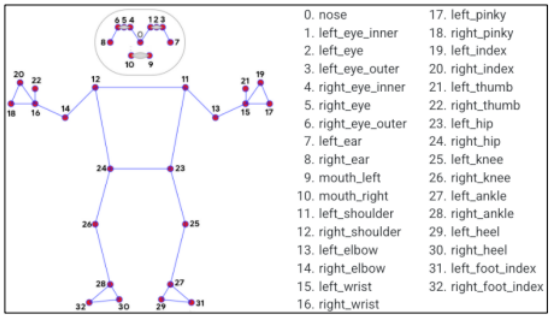

Fig. 18: MediaPipe Landmarks

MediaPipe Pose provides a set of 33 pose landmarks, as illustrated in Figure 18 above. Each of these landmarks possesses the subsequent attributes:

1. x and y: These are the landmark's coordinates, scaled between [0.0, 1.0] using the image's width and height, respectively.

2. z: This indicates the depth of the landmark, with the origin being the midpoint depth of the hips. A smaller value suggests the landmark is nearer to the camera. The range of z is approximately similar to that of x.

3. visibility: A measure between [0.0, 1.0], reflecting the probability that the landmark is both present and unobstructed in the picture.

2. OpenCV

OpenCV is a comprehensive library designed for computer vision tasks. With OpenCV, intricate processes such as facial recognition, object detection, human activity classification in videos, camera motion tracking, object movement tracking, 3D object model extraction, synthesis of high-resolution panoramic images, and much more, become straightforward to execute.

3. Scikit-learn

Scikit-learn is an open-source machine learning library for Python. It provides a range of supervised and unsupervised learning algorithms. From classification to regression, clustering, and dimensionality reduction, it's designed for practical and efficient data analysis.

4. Keras

Keras is a high-level deep learning framework in Python. It provides a clear and convenient way to build neural networks, whether that's a simple linear model or a complex multi-layer deep neural network. Keras is known for its user-friendly interface and ease of prototyping. The framework is underpinned by key tenets: modularity, simplicity, adaptability, and a design intrinsic to Python. Furthermore, Keras

offers ready-made solutions for standard neural architectures, ensuring a streamlined process to initiate convolutional neural networks.

5. Flask

Flask is a lightweight and flexible micro web framework written in Python. The term “micro” indicates the simplicity of Flask, only providing the essentials for a web application to be up and running while leaving the rest of its customizability to the developer. It easily maps URLs to Python functions, allowing dynamic web page generation, which in turn allows easy integration between the frontend and backend. It also has integrated support for unit testing ensuring the robustness and reliability of the application.

4.2 Choosing Exercises And Identifying Errors

By leveraging the strengths of each library, the team was able to streamline the development process. OpenCV ensured high-quality data preprocessing, Scikit-learn enabled efficient feature engineering, and Keras offered a straightforward way to design, train, and test a deep learning model. This multi-tool approach allowed the team to build a robust exercise posture analysis model that could provide real-time, accurate feedback to users, ensuring safer and more effective workouts.

The primary objective is to select specific exercises for the machine learning model to correct incorrect postures. Given the time constraints, only two exercises are considered.

The selected exercises should:

- Be commonly done by those working out alone or at home, increasing the chances of incorrect execution.
- Possess at least two frequent errors that diminish the exercise's effectiveness.

From these criteria, two exercises have been chosen: bicep curls and basic planks.

Upon finalising the exercises, it's essential to spot a minimum of two frequent mistakes per exercise and strategize their detection.

Identified mistakes include:

Bicep Curl: Moving upper arm, incomplete bicep contraction, and leaning back.

Basic Plank: Arching the lower back and raising the back.

4.3 Model Development

4.3.1 Data Gathering

1. Personal Data Collection

Due to a scarcity of online resources showcasing both correct and incorrect exercise forms, most videos were sourced from personal recordings, including those of friends and family. Four individuals contributed to this dataset.

For every exercise, participants were required to:

- Film two videos from diverse angles showcasing the correct form.
- Capture two videos from different perspectives for each identified mistake.

Every submitted video should:

- Be at least 30 seconds for stationary exercises like planks and contain a minimum of 15 reps for dynamic exercises like bicep curls.
- Be taken in a well-lit environment, displaying the entire body.
- Clearly showcase essential joints or movements related to the exercise.
- Keep the participant in view at all times.

Following the video collection, they are categorised based on the exercise. Figure 19 illustrates a sample video directory structure, including proper form and error form categories.

```
.
└── exercise/
    ├── proper-form/
    ├── error-form-1/
    └── error-form-2/
```

Fig. 19: Sample Video Directory

2. Kaggle's Public Dataset

For exercises like the plank with minimal motion, a dataset from Kaggle was used. In essence, Kaggle is a hub for data enthusiasts. The selected dataset covered various yoga poses. For this project, only the plank images were used. Out of 266 images, 30 accurately representing a basic plank were chosen and categorised under the proper form.

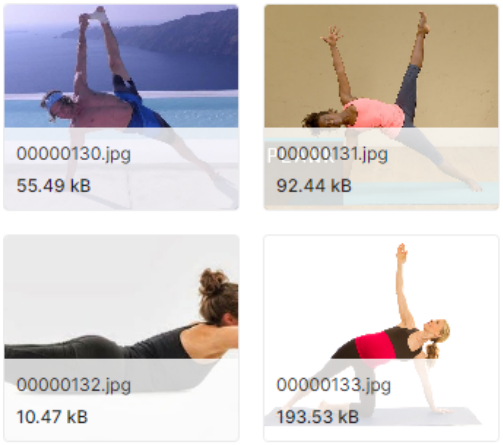

Fig. 20: Kaggle Dataset Example

4.3.2 Basic Error Recognition

For straightforward errors, detection involves gauging the distance or angle between joints using MediaPipe Pose outputs.

a. Distance Computation

The distance between two points, Point 1 (x1,y1) and Point 2 (x2,y2), is given by the formula:

$$distance = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

b. Angle Computation

The angle formed by three points, Point 1 (x1,y1), Point 2 (x2,y2), and Point 3 (x3,y3), is calculated as:

$$angle\_in\_radian = \arctan2(y3 - y2, x3 - x2) - arctan2(y1 - y2, x1 - x2)$$
$$angle\_in\_degree = (angle\_in\_rad * 180)/\Pi$$

4.3.2 Model Preprocessing

Prior to the actual development and training of the model, model preprocessing is a fundamental step in the machine learning pipeline. This ensures that data fed into the model is of high quality, relevant and properly formatted, which leads to better performance and better decision-making based on the model's prediction.

1. Data Preprocessing

Figure 21: Data processing below shows the main procedure for processing data from gathered movies for training models.

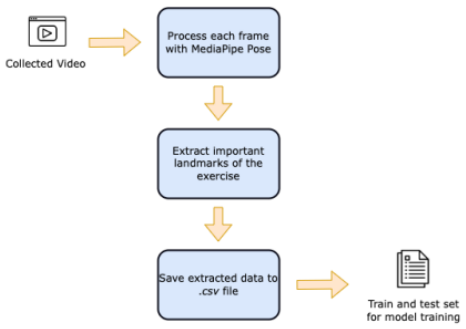

Figure 7: Data processing

Fig. 21: Data Processing Procedure

2. Detecting Significant Landmarks

There are several postures and body positions for every exercise, thus it's critical to recognise the body components (hips, shoulders, etc.)

that are involved in the activity. The location of the body components during the MediaPipe Pose exercise is extracted using the significant landmarks indicated for each exercise.

Fig. 22: Example Important Location of Body Components

In addition, the properties of each landmark, such as x, y, z, and visibility, are reduced to 4 headers for a csv file. For example, with the NOSE landmarks, the 4 headers would be as follows: NOSE_x, NOSE_y, NOSE_z and NOSE_v. Therefore, for each exercise, an empty csv file will be initialised. The first header is the "label" which contains a class for each datapoint, the rest of file's headers are all the important headers with their properties flatten.

3. Extract Data from Video to File

There are videos associated with each workout that are divided into distinct classes. Every frame in a video that corresponds to a certain workout type would be handled in this way:

OpenCV is used to process each frame, and MediaPipe Pose is used to provide a list of coordinate predictions for each keypoint position along with the prediction confidence associated with it.

Options for setting up Medipipe Pose:

- `MIN_DETECTION_CONFIDENCE`: Minimum confidence value `([0.0, 1.0])` from the person-detection model for the detection to be considered successful. Default to `0.5`.
- `MIN_TRACKING_CONFIDENCE`: Minimum confidence value `([0.0, 1.0])` from the landmark-tracking model for the pose landmarks to be considered tracked successfully, or otherwise person detection will be invoked automatically on the next input image. Setting it to a higher value can increase robustness of the solution, at the expense of a higher latency. Default to `0.5`.

```
cap = cv2.VideoCapture("/path/to/video")

with mp_pose.Pose(min_detection_confidence=0.5,
min_tracking_confidence=0.5) as pose:
            while cap.isOpened():
                  ret, image = cap.read()
                  image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                  results = pose.process(image)
```

The significant landmarks for a corresponding exercise would be extracted for each frame including the list of anticipated landmarks and appended as a row to the csv file, where the label column corresponds to the video's class.

To train and test the model, the gathered data, which is recorded in a csv file, will be divided. Twenty percent of the data will be utilised for model evaluation and the remaining eighty percent for training the model.

4.3.3 Model Training

This thesis uses two different approaches to model training. The models trained for each approach will be compared for each exercise, and the optimal model will be selected.

1. Scikit-learn classification.
2. Using Keras to construct a neural network for classification.

1. Sci-kit Learn Classification

Decision Tree/Random Forest (RF), K-Nearest Neighbours (KNN), C-Support Vector (SVC), Logistic Regression classifier (LR), and Stochastic Gradient Descent classifier (SGDC) are the machine learning methods used for model training. Following the training session, measures including precision, recall, accuracy, and F1 score will be used to assess each algorithm's output. The algorithm that yields

the best outcomes will be chosen. The sample code for this part is shown below.

```
algorithms =  [("LR", LogisticRegression()),
               ("SVC", SVC()),
               ('KNN, KNeighborsClassifier()),
               ("SGDC", SGDClassifier()),
               ('RF', RandomForestClassifier()),]
for name, model in algorithms:
               trained_model = model.fit(X_train, y_train)
```

2. Keras Neural Network Classification

Building a neural network using Keras involves layer creation and interconnection. One kind of layer used in this research is referred to as a dense or completely linked layer. This is specified by the keras.layers.dense class. One parameter that may be selected when the layer is constructed is the number of neurons in a dense layer. Every neuron in the dense layer receives an edge, or connection, to every other neuron in the input and output layers when the layer is connected to its input and output layers.

Dropout layers are occasionally used in the construction of the model. In a neural network, "dropout" describes the removal of the input and hidden layer nodes. There is a p dropout probability for the nodes. The dropout layer is employed to address the overfitting issue. When a unit is overfitting, it might alter in a way that corrects the errors made by the other units. By employing dropout, it stops these units from correcting the errors made by the other units, limiting co-adaptation since a unit is extremely erratic in every iteration. As a result, it pushes the layers to adopt a probabilistic approach and assume some degree of responsibility for the input by arbitrarily discarding a few units (nodes).

There are four model architectures that would be experimented with throughout each training session. The model that produces the best results will be selected based on the assessment metrics for each model.

- 3 Dense Layers Neural Network
- 5 Dense Layers Neural Network
- 5 Dense Layers and 2 Dropout Layers Neural Network
- 7 Dense Layers Neural Network

4.4 Client Application Development

Our final implementation of the application largely deviates from our initial phases of planning and designing, from some of the initial requirements to the frameworks we decided to use. One really big difference was that we initially planned to work towards developing a mobile application instead of a web application. However due to multiple constraints which will be further elaborated in a separate section of the report, we were not able to achieve what we intended initially.

Fig. 23: Architecture of Client Application

Our implementation of the client web application consists of a few main processes:

1. Initial prototype of web application.
2. Connect to Flask for web server initialisation.
3. Proper integration of backend and frontend modules to update the web application to be responsive.

Before stepping foot into the development of the application, it is recommended to outline the requirements of the application and visualise the flow of the entire workflow. This ensures that we have a guideline to follow throughout the implementation process and prevents us from deviating from our initial objectives.

1. Initial Prototype of Web Application

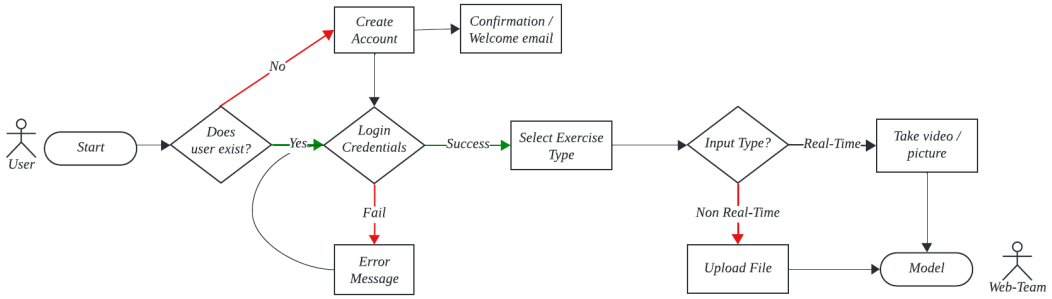

Fig. 24: Initial Prototype of Client Application Architecture

2. Connect to Flask for Web Server Initialisation

The HTML script is written and connected to Flask to initialise the web server for us to visualise the layout of the web application on a local port and address. The initialisation of the web server allows HTTP requests to be sent and received. This integrates the backend and frontend modules in such a way that when a user input is received from the application, it can be processed and returned to the application by the server.

For ease of debugging, we first develop a static web application by putting dummy values in place of actual values and processes in order to visualise the flow of the HTTP requests sent and received.

3. Proper Integration of Backend And Frontend Modules To Update The Web Application To Be Responsive

Dummy data used in place of actual processes are replaced by calling respective functions in the server to display the desired output. The machine learning models that are responsible for producing the results are integrated in the web application through the model_detection() function in app.py. The function takes in 4 parameters in which their values are dependent on the user input. A more detailed usage of the function will be illustrated in the appendix under “Sample Source Code”.

5. Software Deliverables

5.1 Summary of Software Deliverable

The end product of our project is an application that allows users to obtain feedback on their forms on specific exercises in real-time or as comments on pre-recorded footage. The list of deliverables that sums up our end product is as follows:

- A web application for inferencing pre-recorded footage AND live footage.
- Several machine learning models for the classification of the accuracy of form specific to its exercise.

This was achieved through a combination of computer vision to obtain information on the human body and a machine learning algorithm to output the probability of classification of the action. Users are able to interact with the backend modules through the user-friendly interface provided by the web application.

Upon launching the application by starting the server through the terminal console, users will be provided with a link to the corresponding web application.

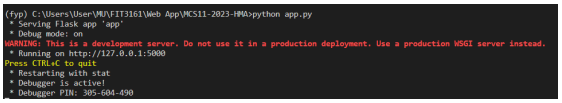

Fig. 25: Server Up and Running

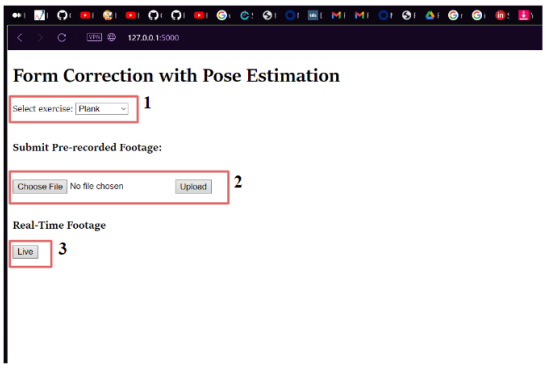

Fig. 26: Layout of Web Application

The web application is rather simple and only provides essential interactive sections which allows the user to:

1. Select exercise.
2. Choose to upload a pre-recorded footage,
3. Or choose to infer real-time footage.

The machine learning model will start producing the output by launching a video frame displaying the classification of form as feedback to the user once a video of the right format has been uploaded, or if the “Live” button is clicked.

5.2 Software Qualities

5.2.1 Robustness

A critical measure of robustness in our application pertains to its ability to consistently recognise and pinpoint landmarks on a person, irrespective of whether the entire body or just a segment is visible in the video feed. This capability exemplifies the software's resilience in handling varied user inputs, ensuring accurate pose estimation even under partial visibility conditions.

Furthermore, the application has been engineered with a sophisticated input validation mechanism. If no person is detected in the live video stream, the system intelligently pauses its operations. This not only optimises computational resources by eliminating redundant processing during periods of inactivity but also ensures that the system remains primed for immediate action once a person re-enters the frame.

5.2.2 Security

When addressing security within our application, it's essential to acknowledge certain design oversights. The application, in its current form, does not incorporate mechanisms for user data protection. Additionally, we had not actively integrated cloud storage or online connectivity services into the application's framework. However, a fortuitous byproduct of these omissions is that the application doesn't capture, store, or process data from input videos beyond its immediate operational requirements.

Given this design state, even though unintentional, the application does not retain any personal data of its users. Consequently, the inherent structure of the application effectively reduces potential risk vectors associated with data leaks or unauthorised disclosures. While this does not mitigate the

importance of intentional security design, the current configuration of the application remains devoid of avenues for unauthorised data access.

5.2.3 Usability

The "End-to-End Workflow" test was created to gauge the efficiency of the entire process from the end-user to the backend. Usability concepts were seriously considered during the designing of each component of the user interface in order to ensure a smooth workflow and an enjoyable experience for the users. The evaluation of the application from the perspective of the users and the developers is done through the conduct of a rather common usability test: **User Testing**.

We carried out the following test to obtain critical and constructive feedback from 5 anonymous users by instructing them to do the following on the application:

1. Attempt to upload a pre-recorded footage for form correction.
2. Attempt to use the real-time feature to correct their form of a specific exercise.

Table 8 shows the results of the test. Judging from the results, we were able to conclude that the flow of our application was decently implemented, however we could further improve the user interface to make it more interactive and interesting for better user experience.

Test ID	Question	Participant ID					Status
		1	2	3	4	5	
1	Completion of tasks above?	Yes	Yes	Yes	Yes	Yes	PASS
2	Difficulty of completing tasks above? (1-Very Easy, 5-Very Difficult)	1	1	1	1	1	PASS
3	Rate UI Design? (1-Worst, 5-Excellent)	3	4	3	5	3	PASS
4	Additional comments?	Layout can be more appealing to users.	-	Very easy to navigate.	Interesting application.	-	PASS

Table 8: User Feedback Summary

5.2.4 Scalability

Scalability was not taken into account in regards to the scope and the expectations of our project. This is because the resulting product is a web application that generates an output on the local computer and is not expected to handle an increasing number of users. Besides, the simplicity of the application only takes in a fixed amount of user input / information in which scalability is not necessarily of importance.

5.2.5 Documentation and Maintainability

To ensure the project's durability and simplicity of use, a constant commitment to documentation and maintainability was followed throughout the creation of the exercise form rectification web application. To begin, unit testing was used meticulously to check the functionality of each component, ensuring the system performed as intended and building the groundwork for a solid application. In addition, a strong emphasis on uniform naming conventions was maintained. This best practice ensured that each variable, function, and module name represented its purpose and functionality, making the codebase clear and intelligible for both the current development team and future maintainers. The usage of a version control system, namely Git, was critical to the project's success since it not only tracked and reported a clear history of development through consistent meaningful commits, it facilitates a safe collaborative environment for team members by safeguarding against potential code conflicts.

Conducted tests were well curated in a test report, detailing the logic, expected and actual results, to provide a clear picture of the testing procedures and respective outcomes. Moreover, a comprehensive user guide was crafted from the perspective of both end-users and future developers. End-users will be able to navigate through the application with ease while further extended work can be conducted with a clear understanding of the current implementation.

5.2 Sample Source Code

A few snippets of source code that provide the functionalities listed below are documented in the Appendix. Each figure is well labelled and properly illustrates the definition and usage of functions that aided in satisfying the project requirements.

1. Training of the machine learning model for plank exercise.
2. Definition of the model_detection() function.
3. Calling the model_detection() function

6. Critical Discussion

In the dynamic landscape of software development, it is not uncommon for the end product to diverge from the initial proposal. As we began implementing our project, little loopholes that were left unnoticed during the planning phase of the project became apparent which led our initial vision to be challenged by both practical and theoretical constraints. These deviations influenced by the unexpected hurdles prompted us to be flexible and adaptable to make relevant changes which indefinitely contributed to the overall productivity and success of the project.

6.1 Software Development

Requirement	Status of Completion
Real-time feedback on exercise form and technique.	Completed
Intuitive and well-designed user interface.	Completed
Machine learning algorithm for classification of exercise form.	Completed
Model is able to take in either pre-recorded footage or provide inference on real-time footage.	Completed
Deep learning algorithm for	Removed

classification of exercise form.	
Mobile application deployment.	Removed

Table 9: Requirements of Software

In the initial stages of our project, we were enthusiastic about leveraging the power of deep learning to drive the core of our application. However, the intricacies and complexities along with our lack of knowledge and understanding associated with deep learning methods limited our ability to integrate it into real-world applications. As mentioned in the earlier parts of the report, we did however manage to train a Keras classification algorithm, as the Keras framework provided a relatively user-friendly and beginner-friendly interface, which I think is a commendable milestone that we achieved.

A part of our objective was to make this application more accessible by deploying it as a mobile application, envisioning it as a handy tool for users on-the-go. Despite the seemingly simple transitioning from web to mobile platform, the complexities that come with mobile application integration coupled with computational feasibility presented hurdles that our team was not prepared for. Due to the time constraints caused in the early stages of model preprocessing, we had no choice but to deviate from this and develop a web application instead.

6.2 Application Development

Requirement	**Status of Completion**
Sign Up / Login	Removed
Select Exercise	Completed
Importing Videos	Completed
Displaying the Detection Results	
Video Playback	Removed
Previous Footage	Removed
Real-Time Inferencing	Added and Completed

Table 10: Requirements of Application

The intended design proposed during the planning phase of the project was done with a lack of better understanding of the complexities that might arise during the implementation of the actual product. Unforeseen challenges were met during the development phase, causing disruptions to our original timeline as well which had us operating under a tighter schedule, impacting the time allocated to the latter stages of the project.

During the development phase of the project, our focus was shifted towards a more user-centric design, emphasising on user testing and feedback, while initial designs were based more on assumptions.

6.3 Summary

As a brief summary of the comparison between the initial plans and the actual development of the project, the project can still be considered relatively successful as we managed to implement most of the major requirements that resolved our initial problem statement. Some of the additional features were opted out due to time constraints and lack of knowledge on respective topics. What could have been avoided was when we were faced with one challenge, we should not have put the entire project on hold to solve one problem, but instead continue to work on other parts of the project simultaneously in order to ensure that we are on par with our project timeline. While we made every effort to adhere to our initial roadmap, adjustments were not solely preferable, but necessary for the successful realisation of our goals.

We have also come to the realisation that iterative prototyping is very important in ensuring the success of our project. Initially, the plan was to develop a full-fledged product before testing, however this made the debugging process really complicated. Midway through the model development, we adopted this method by creating a very basic version of both the model and the user interface before the integration of additional features.

7. Conclusion

In our journey to develop an Application for Human Movement Analysis Based on Pose Estimation, we aimed to address the complexities and nuances inherent in capturing and analysing human movement. Our ambition was met with success in numerous areas, but not without facing challenges and learning valuable lessons.

A critical element of our success was our choice of using MediaPipe. This decision ensured a lightweight and efficient pose estimation, enabling real-time feedback for users.

However, like all developmental endeavours, our project presented certain limitations. While we managed robust detection even with partial visibility of a user, our model's performance was noticeably diminished in darker conditions. Additionally, considerations like scalability and security have been acknowledged, with room for improvement in future iterations. In terms of security, our unintentional design choices resulted in the fortuitous outcome of no data retention, thereby minimising risk vectors associated with unauthorised data access.

Looking ahead, numerous opportunities await to elevate our application. The potential integration of cloud storage services, like Firebase, can greatly enhance user experience, and porting the solution to a mobile application opens doors to a broader audience. Additionally, continued refinement in model architecture and the expansion of our dataset can further optimise accuracy and performance.

In sum, this final year project not only met a significant portion of the outlined requirements but also provided a foundation upon which future enhancements can be built. It stands as a testament to the importance of iterative development, continuous learning, and the adaptability required in the ever-evolving landscape of computer science.

8. References

Thomas, G., Gade, R., Moeslund, T. B., Carr, P., & Hilton, A. (2017). Computer vision for sports: Current applications and research topics. *Computer Vision and Image Understanding*, *159*, 3–18. https://doi.org/10.1016/j.cviu.2017.04.011

Liebermann, D. G., Katz, L., Hughes, M. D., Bartlett, R. M., McClements, J., & Franks, I. M. (2002). Advances in the application of information technology to sport performance. *Journal of Sports Sciences*, *20*(10), 755–769. https://doi.org/10.1080/026404102320675611

AI Fitness Coach at Home using Image Recognition. (n.d.). Research Square. https://www.researchsquare.com/article/rs-2047283/v1

Recent developments in human motion analysis. (n.d.). *Pattern Recognition*, *36*(3), 585–601. https://doi.org/10.1016/S0031-3203(02)00100-0

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, *2018*. https://doi.org/https://doi.org/10.1155/2018/7068349

Liu, J. (n.d.). Convolutional neural network-based human movement recognition algorithm in sports analysis. *Frontiers in Psychology*, *12*. https://doi.org/10.3389/fpsyg.2021.663359

Human motion. (n.d.). https://link.springer.com/content/pdf/10.1007/978-1-4020-6693-1.pdf

Aggarwal, J. K., & Cai, Q. (1999). Human motion analysis: A review. *Computer Vision and Image Understanding*, *73*(3), 428–440. https://doi.org/10.1006/cviu.1998.0744

Schiele, M. A., Leonid Pishchulin, Peter Gehler, Bernt. (n.d.-a). *2D human pose estimation: New benchmark and state of the art analysis*.

Rajat Kumar Sinha, Ruchi Pandey , & Rohan Pattnaik . (2017). *Deep Learning For Computer Vision Tasks: A review* . https://arxiv.org/ftp/arxiv/papers/1804/1804.03928.pdf

M. Manafifard. (n.d.). A survey on player tracking in soccer videos. *Computer Vision and Image Understanding*, *159*, 19–46. https://doi.org/10.1016/j.cviu.2017.02.002

Mgaya, G. B., Liu, H., & Zhang, B. (2021, January 1). *A survey on applications of modern deep learning techniques in team sports analytics*. Springer International Publishing. https://link.springer.com/chapter/10.1007/978-3-030-73689-7_42

S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), Antalya, Turkey, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.

Li, Z., Liu, F., Yang, W., Peng, S., & Zhou, J. (2022). A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, *33*(12), 6999–7019. https://doi.org/10.1109/tnnls.2021.3084827

9. Appendix

```python
algorithms =[("LR", LogisticRegression()),
             ("SVC", SVC(probability=True)),
             ('KNN',KNeighborsClassifier()),
             ("DTC", DecisionTreeClassifier()),
             ("SGDC", CalibratedClassifierCV(SGDClassifier())),
             ("NB", GaussianNB()),
             ('RF', RandomForestClassifier()),]

models = {}
final_results = []

for name, model in algorithms:
    trained_model = model.fit(X_train, y_train)
    models[name] = trained_model

    # Evaluate model
    model_results = model.predict(X_test)

    p_score = precision_score(y_test, model_results, average=None, labels=[0, 1, 2])
    a_score = accuracy_score(y_test, model_results)
    r_score = recall_score(y_test, model_results, average=None, labels=[0, 1, 2])
    f1_score_result = f1_score(y_test, model_results, average=None, labels=[0, 1, 2])
    cm = confusion_matrix(y_test, model_results, labels=[0, 1, 2])
    final_results.append(( name,  round_up_metric_results(p_score), a_score, round_up_metric_results(r_score), round_up_metric_results(f1_score_result), cm))

```

Training of The Machine Learning Model for Plank Exercise

```python
with open("./model/all_sklearn.pkl", "wb") as f:
    pickle.dump(models, f)
```

```python
with open("./model/LR_model.pkl", "wb") as f:
    pickle.dump(models["LR"], f)
```

```python
with open("./model/SVC_model.pkl", "wb") as f:
    pickle.dump(models["SVC"], f)
```

```python
# Dump input scaler
with open("./model/input_scaler.pkl", "wb") as f:
    pickle.dump(sc, f)
```

Exporting and Saving the Model Using pickle.

```python
def model_detection(cap, model_path, input_scaler_path, VIDEO_PATH=None):

    IMPORTANT_LMS = [
        "NOSE",
        "LEFT_SHOULDER",
        "RIGHT_SHOULDER",
        "LEFT_ELBOW",
        "RIGHT_ELBOW",
        "LEFT_WRIST",
        "RIGHT_WRIST",
        "LEFT_HIP",
        "RIGHT_HIP",
        "LEFT_KNEE",
        "RIGHT_KNEE",
        "LEFT_ANKLE",
        "RIGHT_ANKLE",
        "LEFT_HEEL",
        "RIGHT_HEEL",
        "LEFT_FOOT_INDEX",
        "RIGHT_FOOT_INDEX",
    ]

    # Generate all columns of the data frame

    HEADERS = ["label"] # Label column

    for lm in IMPORTANT_LMS:
        HEADERS += [f"{lm.lower()}_x", f"{lm.lower()}_y", f"{lm.lower()}_z", f"{lm.lower()}_v"]


    # Drawing helpers
    mp_drawing = mp.solutions.drawing_utils
    mp_pose = mp.solutions.pose

    # Load model
    with open(model_path, "rb") as f:
        sklearn_model = pickle.load(f)

    # Dump input scaler
    with open(input_scaler_path, "rb") as f2:
        input_scaler = pickle.load(f2)

    if VIDEO_PATH is None:
        cap = cv2.VideoCapture(cap)
    else:
        cap = cv2.VideoCapture(VIDEO_PATH)

    current_stage = ""
    prediction_probability_threshold = 0.6

```

Defining model_detection() [1]

.

```
115
116     with mp_pose.Pose(min_detection_confidence=0.5, min_tracking_confidence=0.5) as pose:
117         print(cap.isOpened())
118         while cap.isOpened():
119             ret, image = cap.read()
120
121             if not ret:
122                 break
123
124             # Reduce size of a frame
125             image = rescale_frame(image, 100)
126             # image = cv2.flip(image, 1)
127
128             # Recolor image from BGR to RGB for mediapipe
129             image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
130             image.flags.writeable = False
131
132             results = pose.process(image)
133
134             if not results.pose_landmarks:
135                 print("No human found")
136                 continue
137
138             # Recolor image from BGR to RGB for mediapipe
139             image.flags.writeable = True
140             image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
141
142             # Draw landmarks and connections
143             mp_drawing.draw_landmarks(image, results.pose_landmarks, mp_pose.POSE_CONNECTIONS, mp_drawing.DrawingSpec(color=(244, 117, 66), thickness=2, circle_radius=2),
144                                       mp_drawing.DrawingSpec(color=(245, 66, 230), thickness=2, circle_radius=1))
145
146             # Make detection
147             try:
148                 # Extract keypoints from frame for the input
149                 row = extract_important_keypoints(results)
150                 X = pd.DataFrame([row], columns=HEADERS[1:])
151                 X = pd.DataFrame(input_scaler.transform(X))
152
153                 # Make prediction and its probability
154                 predicted_class = sklearn_model.predict(X)[0]
155                 prediction_probability = sklearn_model.predict_proba(X)[0]
156                 print(predicted_class, prediction_probability)
```

Defining model_detection() [2]

```
157
158             # Evaluate model prediction
159             # if predicted_class == "C" and prediction_probability[prediction_probability.argmax()] >= prediction_probability_threshold:
160             if predicted_class == 0 and prediction_probability[prediction_probability.argmax()] >= prediction_probability_threshold:
161                 current_stage = "Correct"
162             # elif predicted_class == "L" and prediction_probability[prediction_probability.argmax()] >= prediction_probability_threshold:
163             elif predicted_class == 2 and prediction_probability[prediction_probability.argmax()] >= prediction_probability_threshold:
164                 current_stage = "Low back"
165             # elif predicted_class == "H" and prediction_probability[prediction_probability.argmax()] >= prediction_probability_threshold:
166             elif predicted_class == 1 and prediction_probability[prediction_probability.argmax()] >= prediction_probability_threshold:
167                 current_stage = "High back"
168             else:
169                 current_stage = "unk"
170
171             # Visualization
172             # Status box
173             cv2.rectangle(image, (0, 0), (250, 60), (245, 117, 16), -1)
174
175             # Display class
176             cv2.putText(image, "CLASS", (95, 12), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
177             cv2.putText(image, current_stage, (90, 40), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
178
179             # Display probability
180             cv2.putText(image, "PROB", (15, 12), cv2.FONT_HERSHEY_COMPLEX, 0.5, (0, 0, 0), 1, cv2.LINE_AA)
181             cv2.putText(image, str(round(prediction_probability[np.argmax(prediction_probability)], 2)), (10, 40), cv2.FONT_HERSHEY_COMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
182
183         except Exception as e:
184             print(f"Error: {e}")
185
186         cv2.imshow("CV2", image)
187
188         # Press Q to close cv2 window
189         if cv2.waitKey(1) & 0xFF == ord('q'):
190             break
191
192     cap.release()
193     cv2.destroyAllWindows()
194
195     # (Optional)Fix bugs cannot close windows in MacOS (https://stackoverflow.com/questions/6116564/destroywindow-does-not-close-window-on-mac-using-python-and-opencv)
196     for i in range (1, 5):
197         cv2.waitKey(1)
```

Defining model_detection() [3]

```python
@app.route('/choose-action', methods=['POST'])
def choose_action():
    # Get the exercise choice from the form
    exercise_choice = request.form.get('exercise')
    button_pressed = request.form.get('submit_button')

    if button_pressed == "Upload":
        # Handle the file upload
        if 'file' not in request.files:
            flash('No file part')
            return redirect(request.url)

        file = request.files['file']

        if file.filename == '':
            flash('No selected file')
            return redirect(request.url)

        if file and allowed_file(file.filename):
            filename = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
            file.save(filename)

            # Call model_detection function
            if exercise_choice == "Plank":
                model_detection(cap=2, model_path="./models/plank/LR_model.pkl", input_scaler_path="./models/plank/input_scaler.pkl", VIDEO_PATH=filename)
            elif exercise_choice == "Bicep Curl":
                model_detection(cap=2, model_path="./models/bicep_curl/KNN_model.pkl", input_scaler_path="./models/bicep_curl/input_scaler.pkl", VIDEO_PATH=filename)

            clear_uploads()
            return redirect(url_for('index'))

        else:
            flash('Invalid video format!')
            return redirect(url_for('index'))

    elif button_pressed == "Live":
        if exercise_choice == "Plank":
            model_detection(cap=1, model_path="./models/plank/LR_model.pkl", input_scaler_path="./models/plank/input_scaler.pkl")
        elif exercise_choice == "Bicep Curl":
            model_detection(cap=1, model_path="./models/bicep_curl/KNN_model.pkl", input_scaler_path="./models/bicep_curl/input_scaler.pkl")

        return redirect(url_for('index'))
```

Calling model_detection() in app.py.