A Major Project Report

On

HUMAN BEHAVIOUR RECOGNITION BASED ON MULTI-SCALE CONVOLUTIONAL NEURAL NETWORK

Submitted to CMREC (UGC Autonomous)

In Partial Fulfilment of the requirements for the Award of Degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (AI & ML)

Submitted By

MOHAMMAD ASHWAQ AHMED	**(228R1A66G6)**
NUNEE SREEJA	**(228R1A66H2)**
PADALA HEMANTH KUMAR	**(228R1A66H3)**
SINGAM LAXMI CHANDANA	**(228R1A66J2)**

Under the Esteemed guidance of

Dr. A. AMARA JYOTHI

Associate Professor,

Department of CSE(AI & ML).

Department of Computer Science and Engineering (AI&ML)

CMR ENGINEERING COLLEGE

(UGC AUTONOMOUS)

(Accredited by NAAC & NBA, Approved by AICTE, New Delhi, Affiliated to JNTU, Hyderabad)
(Kandlakoya, Medchal Road, Medchal-Malkajgiri Dist., Hyderabad-501 401)

(2025-2026)

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Accredited by NAAC&NBA, Approved by AICTE New Delhi, Affiliated to JNTU, Hyderabad, Kandlakoya, Medchal Road, Hyderabad-501 401)

Department of Computer Science & Engineering (AI & ML)

CERTIFICATE

This is to certify that the Major project entitled **"HUMAN BEHAVIOUR RECOGNITION BASED ON MULTISCALE CONVOLUTIONAL NEURAL NETWORK"** is a bonafide work carried out by

MOHAMMAD ASHWAQ AHMED	**(228R1A66G6)**
NUNEE SREEJA	**(228R1A66H2)**
PADALA HEMANTH KUMAR	**(228R1A66H3)**
SINGAM LAXMI CHANDANA	**(228R1A66J2)**

in partial fulfillment of the requirement for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING (AI&ML) from CMR Engineering College, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major Project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	**Major Project Coordinator**	**Head of the Department**
Dr. A. Pramod Kumar	Mr. G. Venkateswarlu	Dr. Madhavi Pingili
Associate Professor	Assistant Professor	Professor & HOD
Department of	Department of	Department of
CSE (AI & ML).	CSE (AI & ML).	CSE (AI & ML).

External Examiner:_____

DECLARATION

This is to certify that the work reported in the present Major project entitled "**HUMAN BEHAVIOUR RECOGNITION BASED ON MULTISCALE CONVOLUTIONAL NEURAL NETWORK**" is a record of bonafide work done by us in the Department of Computer Science and Engineering (AI & ML), CMR Engineering College. The reports are based on the Major project work done entirely by us and not copied from any other source. We submit our Major project for further development by any interested students who share similar interests to improve the Major project in the future.

The results embodied in this Major project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

MOHAMMAD ASHWAQ AHMED (228R1A66G6)
NUNEE SREEJA (228R1A66H2)
PADALA HEMANTH KUMAR (228R1A66H3)
SINGAM LAXMI CHANDANA (228R1A66J2)

ACKNOWLEDGEMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. Madhavi Pingili**, Professor & HOD, Department of CSE (AI & ML), CMR Engineering College for their constant support.

We are extremely thankful to **Dr. A. Amara Jyothi,** Associate Professor, Internal Guide, Department of CSE (AI&ML), for his constant guidance, encouragement and moral support throughout the Major project.

We will be failing in duty if we do not acknowledge with gratitude thanks to the authors of the references and other literature referred to in this Major Project.

We thank **Mr. G. Venkateswarlu,** Major Project Coordinator for his constant support in carrying out the Major project activities and reviews.

We express our thanks to all staff members and friends for all the help and co-ordination extended in bringing out this Major project successfully in time.

Finally, We are very much thankful to our parents who guided us for every step.

MOHAMMAD ASHWAQ AHMED (228R1A66G6)
NUNEE SREEJA (228R1A66H2)
PADALA HEMANTH KUMAR (228R1A66H3)
SINGAM LAXMI CHANDANA (228R1A66J2)

CONTENTS

ABSTRACT

This project presents an intelligent **Fall Detection and Activity Recognition system** designed to enhance the safety and independence of elderly individuals. The system leverages advanced deep learning techniques, specifically Convolutional Neural Networks (CNN) for spatial feature extraction and Long Short-Term Memory (LSTM) networks for temporal sequence analysis, to monitor human activities in real time through video streams. By analyzing both posture and motion patterns, the model accurately classifies activities such as walking, sitting, standing, bending, and falling. The integration of CNN and LSTM enables the system to effectively distinguish between normal daily movements and critical events like falls, ensuring high accuracy and reliability in real-world environments.

Upon detecting a fall, the system triggers immediate alerts through a connected dashboard, enabling caregivers or family members to respond quickly and reduce potential health risks. Designed to be non-intrusive, cost-effective, and scalable, the solution eliminates the need for wearable devices while providing continuous monitoring. Its modular architecture allows future enhancements such as integration with IoT devices, predictive analytics, and multi-camera support. Overall, the project demonstrates the potential of artificial intelligence and computer vision in modern healthcare by offering a practical, real-time solution that improves elderly care, minimizes emergency response time, and supports safer independent living.

Keywords

Fall Detection, Activity Recognition, Elderly Care, Convolutional Neural Networks, LSTM, Real-Time Monitoring, Deep Learning, Video Analysis, Alert System.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Introduction and Objectives

Falls are among the leading causes of accidental deaths and long-term disability among the elderly, often resulting in fractures, trauma, and reduced mobility. Global healthcare reports indicate millions of fall-related injuries annually, highlighting the urgent need for continuous and intelligent monitoring systems. Traditional approaches rely on wearable sensors such as accelerometers and gyroscopes **[2], [7]** to capture motion patterns, but these methods face limitations including user discomfort, inconsistent usage, and false alarms during normal activities. As a result, such systems become less effective in real-world scenarios where users may forget or refuse to wear devices. These challenges emphasize the need for non-intrusive and more reliable solutions capable of accurately analyzing complex human movement without depending on wearable technologies.

Vision-based methods provide a non-invasive alternative by enabling full-body motion monitoring through video analysis. Deep learning techniques play a crucial role in understanding human activities by combining spatial and temporal features **[1], [6], [9].** Convolutional Neural Networks (CNNs) extract spatial information such as posture and limb orientation, while Long Short-Term Memory (LSTM) networks capture temporal changes across video frames. Advanced architectures like YOLO-based detectors support real-time detection of sudden movements **[3], [11], [12]**, while 3D CNNs and bidirectional LSTMs enhance recognition in complex environments **[9], [10].** However, multimodal systems using audio, skeletal keypoints, or multiple cameras often face challenges such as occlusions, lighting variations, and increased computational cost **[6]**. To address these issues, the proposed Multiscale CNN-LSTM architecture processes spatio-temporal features from continuous video frames, enabling accurate differentiation between normal activities and falls. Integrated with IoT-based healthcare systems, it supports real-time alerts and efficient caregiving response **[7], [16].**

1.2PROJECTOBJECTIVES

The primary objective of this project is to design and develop a Human Behaviour Recognition system using a Multiscale Convolutional Neural Network (CNN) combined with a Long Short-Term Memory (LSTM) network, capable of accurately detecting human activities and identifying fall events from continuous video input. By leveraging deep learning–based spatial and temporal feature extraction techniques, the system eliminates the need for wearable sensors and complex manual monitoring, making it more practical and suitable for real-time surveillance environments. The project aims to transform raw video data into meaningful behavioural insights by analyzing posture variations, motion transitions, and activity patterns in a structured and automated manner.

Another important objective is to improve monitoring efficiency by generating reliable activity classification results that can support safety assistance systems, especially for elderly care environments. The system focuses on extracting both spatial posture features using CNN and temporal motion patterns using LSTM through sequential frame analysis. Additionally, the system generates real-time detection outputs with alert notifications and activity status visualization through a monitoring dashboard, enabling caregivers and administrators to quickly interpret the results and respond effectively to emergency fall situations.

Another important objective of the proposed system is to enhance the reliability and robustness of human activity recognition under different environmental conditions such as variations in lighting, background complexity, and camera placement. The system is designed to efficiently process continuous video streams and minimize false detections by integrating multiscale spatial feature extraction with temporal motion analysis. Furthermore, the implementation of a real-time monitoring interface supports continuous observation and quick decision-making by displaying detected activity status and alert notifications instantly. This improves the overall effectiveness of the system in providing timely assistance and ensures its applicability in practical environments such as elderly care centers, hospitals, smart homes, and surveillance-based safety monitoring systems.

Objectives:

- To develop a real-time, non-intrusive video-based monitoring system using a CNN-LSTM model for accurate recognition of elderly individuals' daily activities and fall detection.
- To ensure reliable differentiation between normal movements and falls, enabling immediate alerts to caregivers for timely assistance.
- To create a low-cost, scalable, and privacy-focused system capable of efficient data preprocessing, performance evaluation, and future real-time deployment.

1.3 Purpose of the Project

The purpose of this project is to provide an intelligent and reliable solution for detecting falls among elderly individuals and recognizing their daily activities in real time. Falls are a leading cause of injuries and emergency hospital visits among seniors, and delayed assistance can result in severe complications or death. This project aims to eliminate such risks by automatically detecting falls using deep learning and instantly notifying caregivers. The system ensures elderly independence, enhances safety, reduces caregiver workload, and offers a cost-effective alternative to wearable sensors and constant manual supervision. Its purpose also extends to long-term behavior monitoring and potential predictive healthcare insights.

Furthermore, the system enhances elderly safety by reducing the dependency on wearable sensor-based monitoring devices, which may be uncomfortable or inconvenient for long-term usage. The integration of Multiscale Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks enables accurate extraction of both spatial posture information and temporal motion patterns from continuous video streams, ensuring precise activity classification. The proposed solution also contributes to reducing caregiver workload by providing automated monitoring support through real-time detection alerts and activity status visualization on a monitoring dashboard. In addition to emergency detection, the system supports long-term behavior monitoring and activity pattern analysis, which can assist healthcare professionals in identifying potential health risks at an early stage and enable preventive medical intervention. Thus, the project provides a practical, scalable, and cost-effective intelligent monitoring solution suitable for smart healthcare environments, elderly care centers, hospitals, and home-based assisted living applications.

1.4 Problem Statement

Falls among elderly individuals remain one of the leading causes of serious injuries, disabilities, and fatal accidents, especially when immediate assistance is not available. Traditional fall-detection solutions relying on wearable sensors suffer from low user compliance, discomfort, and frequent false alarms. Manual monitoring through CCTV is also impractical as it requires continuous human supervision. Therefore, there is a need for a real-time, non-intrusive, intelligent fall detection system capable of automatically recognizing human activities and identifying falls with high accuracy. The main problem addressed in this project is to detect falls using a camera-based solution that applies CNN for spatial feature extraction and LSTM for temporal movement analysis. The system must classify actions such as walking, sitting, standing, and falling, and provide instant alerts to caregivers.

1.5 Existing System

The existing systems for fall detection and elderly monitoring mainly rely on wearable sensors such as accelerometers and gyroscopes, panic buttons, pressure sensors, or manual CCTV observation. These solutions often suffer from major limitations, including the need for users to consistently wear or charge the devices, which many elderly individuals forget or find uncomfortable. Panic buttons require manual activation, making them ineffective during sudden or unconscious falls, while sensor-based systems frequently generate false alarms due to normal activities like bending or sitting quickly. Manual monitoring through CCTV is labor-intensive, expensive, and not suitable for continuous real-time supervision. Overall, existing systems lack accuracy, convenience, and reliability, making them unsuitable for effective fall detection in real-world scenarios

Disadvantages

- Requires user compliance, as elderly individuals may forget or refuse to wear sensors or panic devices.
- High false alarms occur with wearable sensors when users sit, bend, or move quickly.
- Ineffective during unconscious or sudden falls, since panic buttons need manual activation.
- Uncomfortable or intrusive wearable devices may reduce regular usage.
- Manual CCTV monitoring is time-consuming, requires staff, and cannot ensure continuous observation.
- Limited activity recognition, as most existing solutions detect only falls and not overall movement patterns.
- Privacy concerns arise with some cloud-based systems storing raw video data.

1.6 Proposed System

The proposed system utilizes a Multiscale CNN–LSTM architecture to detect human falls from continuous video streams in a non-intrusive manner, as shown in Fig 1.6.1. The system is designed to capture both spatial (appearance-based) and temporal (motion-based) features of human activities. Initially, input video is divided into frames and preprocessed through resizing and normalization. These frames are passed through a Convolutional Neural Network (CNN), which extracts spatial features such as body posture, orientation, and structural patterns. The multiscale nature of CNN enables the model to capture both fine details (like limb movement) and global patterns (such as full-body position), improving the system's ability to identify fall-related postures.

After spatial feature extraction, the generated feature vectors are sequentially fed into a Long Short-Term Memory (LSTM) network, as illustrated in Fig 1.6.1. LSTM is responsible for learning temporal dependencies, meaning it analyzes how human posture changes over time. Since a fall is a sequence of rapid movements rather than a single frame event, LSTM effectively captures transitions such as standing to sudden collapse. Its memory cells and gating mechanisms (input, forget, and output gates) allow it to retain relevant motion patterns while ignoring unnecessary variations. This reduces false positives and helps distinguish falls from similar activities like sitting or bending.

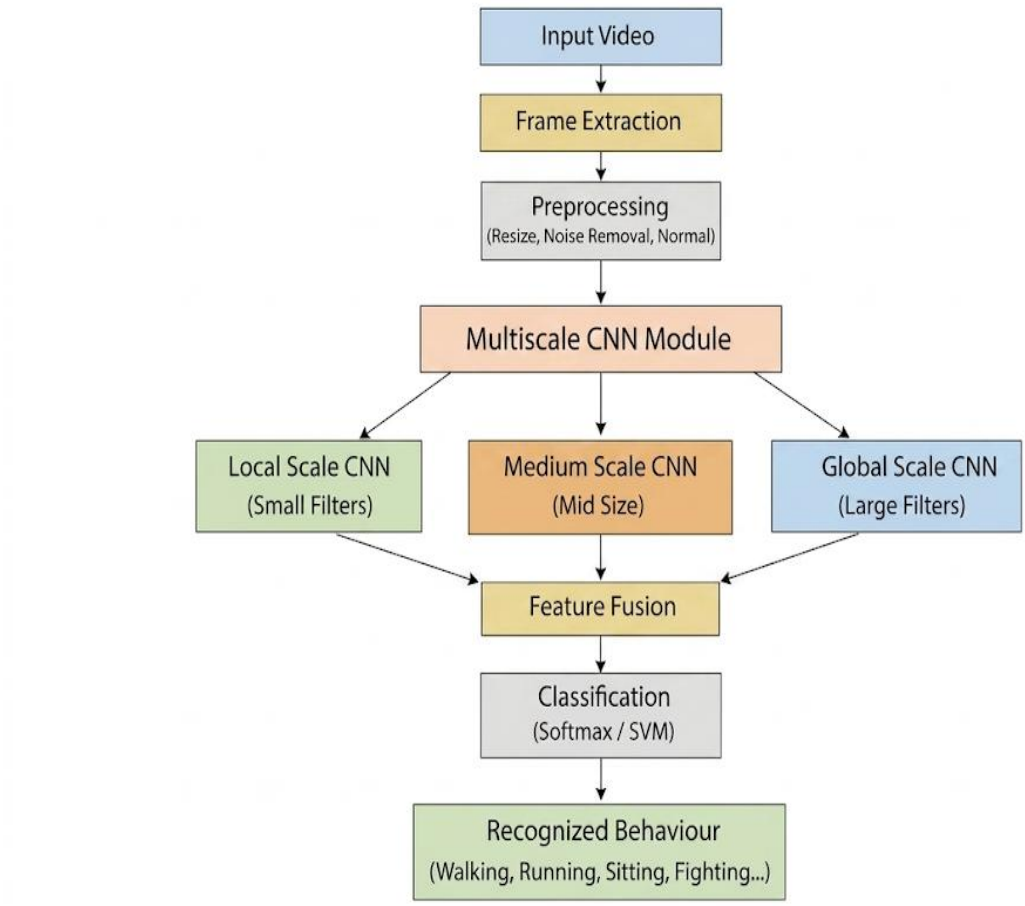

Figure 1.6.1: Block diagram of proposed system.

The overall approach of the system is based on spatio-temporal learning, where CNN and LSTM are integrated sequentially to form a unified deep learning pipeline, as represented in Fig 1.6.1. CNN answers "what is happening in each frame," while LSTM answers "how the movement evolves over time." This combination ensures higher accuracy compared to traditional or single-model approaches. The multiscale feature extraction further enhances performance by analyzing different levels of motion and posture simultaneously, making the system robust in real-world scenarios with varying environments, lighting conditions, and human behaviours.

In the project implementation, the CNN-LSTM model is developed using deep learning frameworks such as TensorFlow or Keras. Video frames are continuously processed, and their features are passed to the LSTM as sequences. The final output layer performs classification into **fall** or **non-fall** categories.

Key Points:

CNN (Spatial Features)

- Extracts posture, body orientation, and visual patterns
- Multiscale filters capture both fine and global features
- Converts video frames into feature vectors

LSTM (Temporal Features)

- Processes sequence of CNN features
- Captures motion changes over time
- Detects sudden transitions (fall events)
- Reduces false alarms

Approach

- Combines spatial + temporal learning
- Multiscale feature extraction improves accuracy
- End-to-end deep learning pipeline

Implementation (How Used in Project)

- Video → Frames → CNN → Feature sequences → LSTM
- Classification into fall / non-fall
- Integrated with Flask dashboard for alerts
- Supports real-time monitoring and IoT-based healthcare

Advantages

- **Non-Intrusive System** – No wearable sensors required, ensuring comfort and ease of use for elderly people.
- **High Accuracy** – Uses CNN (spatial) and LSTM (temporal) for precise fall detection.
- **Real-Time Monitoring** – Continuously analyzes video and detects falls instantly.
- **Reduced False Alarms** – Distinguishes falls from normal activities like sitting or bending.
- **Smart Alert Integration** – Automatically sends alerts through the dashboard for quick assistance.

1.7 Input and Output Design

1.7.1 Input Design

The input design of the system begins with continuous video acquisition through a camera placed in the monitoring environment. Each video stream is divided into frames, which undergo preprocessing steps such as resizing, normalization, and enhancement to ensure consistent input quality. This ensures that variations in lighting and environment do not significantly affect detection performance.

These processed frames are then structured as sequential inputs for the CNN-LSTM model. While the CNN identifies posture-related features from individual frames, the LSTM evaluates frame sequences to understand motion patterns over time. The input design ensures that both spatial and temporal information are efficiently captured to support highly accurate activity classification.

1.7.2 Output Designs

The output design focuses on delivering real-time classification results displayed on a web dashboard, where each detected activity—such as standing, walking, sitting, lying down, or falling—is continuously updated. When the system identifies a fall event, it highlights it distinctly to ensure immediate visibility. The output is clear, user-friendly, and designed to support quick assessment by caregivers or medical personnel.

The output design presents the real-time activity classification results clearly on a web dashboard, displaying whether the person is walking, sitting, standing, lying, or falling. The system updates the detected activity continuously as each frame sequence is processed by the CNN-LSTM model. When a fall is detected, the output section highlights the event with a distinct visual alert to immediately draw attention. Along with this, the system triggers an automatic alarm sound or notification to ensure quick caregiver response. The output design also includes an alert log section where all detected activities or fall incidents are recorded for future reference. This ensures that caregivers can review the history of movements and examine any repeated patterns. The output interface is simple, responsive, and designed to support fast decision-making in emergency scenarios.

When a fall is detected, the output section highlights the event with a distinct visual alert to immediately draw attention. Along with this, the system triggers an automatic alarm sound or notification to ensure quick caregiver response and reduce the delay in providing emergency assistance. The output design also includes an alert log section where all detected activities or fall incidents are recorded for future reference. This ensures that caregivers can review the history of movements and examine any repeated patterns for better monitoring and safety evaluation. The output interface is simple, responsive, and designed to support fast decision-making in emergency scenarios while improving the overall efficiency of real-time activity monitoring.

2. LITERATURE SURVEY

1.Alymani, Mofadal, et al. "Adaptive motion assisted human activity recognition for people with disabilities via osprey optimisation-based dimensionality reduction with recurrent neural network." Signal,Image and Video Processing (2026). The authors propose an adaptive human activity recognition system for people with disabilities using recurrent neural networks. The model uses dimensionality reduction and optimization techniques to improve feature selection. It enhances accuracy while reducing computational complexity.

2.Wang, Chuanchuan, et al. "Deep learning for 3D skeleton-based action recognition: a comprehensive review of methods, datasets, and future directions." International Journal of Machine Learning and Cybernetics (2026). This paper presents a survey of deep learning methods for 3D skeleton-based action recognition. It covers CNN, RNN, and graph-based models along with datasets. The study highlights the importance of spatial-temporal modeling and future research directions.

3.Parale, Mandar, et al. "A systematic review on human action detection and classification architectures using deep learning methodology." Multimedia Tools and Applications (2026). The authors provide a review of deep learning architectures for human action detection and classification. Various CNN and hybrid models are analyzed for performance. The paper discusses challenges like computational cost and dataset imbalance.

4.Ramu, Moola, et al. "Human activity recognition using a bagging-based deep learning framework with convolutional capsule networks and spiking neural networks." International Journal of Information Technology, 2025. This work proposes a bagging-based deep learning framework using capsule and spiking neural networks. It improves activity recognition accuracy through ensemble learning. The model enhances robustness and captures complex behavior patterns.

5. Alzahrani, Abdulrahman, et al. "Artificial Intelligence-based fine-tuning model for fall activity recognition in disabled persons within an IoT environment." Scientific Reports, 2025. The authors propose an AI-based fine-tuning model for fall detection in an IoT environment.

6.Han, H., et al. "Multilevel features cascade fusion network for infrared video human behavior recognition." Displays, 2025.This paper presents a multilevel feature cascade fusion network for behavior recognition. It uses infrared video data to improve performance in low-light conditions. The model combines multiple feature levels. It achieves better accuracy in challenging environments.

7.Wei, Y., et al. "MCNN-CMCA: A multiscale convolutional neural networks with cross-modal channel attention for physiological signal-based mental state recognition." Digital Signal Processing, 2025.The study proposes a multiscale CNN with cross-modal channel attention. It enhances feature extraction across multiple scales. The model improves classification accuracy using attention mechanisms. It is effective for complex signal-based recognition tasks.

8.Dianakamal, G., et al. "Recognition of human behaviour utilising multiscale convolutional neural networks." Taylor & Francis, 2025.This work focuses on human behavior recognition using multiscale CNNs. The model captures both fine and global features. It improves recognition accuracy in dynamic environments. The approach is efficient and scalable.

9.Cristina, Stefania, et al. "Audio-and Video-Based Human Activity Recognition Systems in Healthcare." IEEE Access, 2024.The authors review audio and video-based human activity recognition systems. The paper compares multimodal approaches for better accuracy. Challenges such as noise and data fusion are discussed. It highlights applications in healthcare monitoring.

10. Pereira, Gracile Astlin. "Fall detection for industrial setups using yolov8 variants." arXiv preprint arXiv:2408.04605(2024). The author proposes a fall detection system using YOLOv8 variants for industrial environments. The model focuses on real-time detection of human posture changes. It improves safety monitoring by quickly identifying fall events. The approach is efficient and suitable for high-risk workplace scenarios.

Focused Area / Title	Key Findings	Reference
Adaptive Human Activity Recognition (2026) **[1]**	Proposes an adaptive HAR system for people with disabilities using RNN. Uses dimensionality reduction and optimization for better feature selection. Improves accuracy while reducing computational complexity.	M. Alymani et al., "Adaptive Motion Assisted Human Activity Recognition for People with Disabilities via Osprey Optimisation-Based Dimensionality Reduction with Recurrent Neural Network," *Signal, Image and Video Processing*, 2026.
3D Skeleton-Based Action Recognition (2026) **[2]**	Presents a survey of deep learning models like CNN, RNN, and graph-based approaches. Highlights importance of spatial-temporal modeling. Discusses datasets and future research directions.	C. Wang et al., "Deep Learning for 3D Skeleton-Based Action Recognition: A Comprehensive Review of Methods, Datasets, and Future Directions," *International Journal of Machine Learning and Cybernetics*, 2026.
Action Detection Architectures Review (2026) **[3]**	Reviews CNN and hybrid deep learning models for action detection. Emphasizes feature extraction and optimization. Discusses challenges like computational cost and dataset imbalance.	M. Parale et al., "A Systematic Review on Human Action Detection and Classification Architectures Using Deep-Learning Methodology," *Multimedia Tools and Applications*, 2026.
Bagging-Based Deep Learning Framework (2025) **[4]**	Introduces ensemble model using capsule and spiking neural networks. Improves recognition accuracy and robustness. Captures complex behavior patterns effectively.	M. Ramu et al., "Human Activity Recognition Using a Bagging-Based Deep Learning Framework with Convolutional Capsule Networks and Spiking Neural Networks," *International Journal of Information Technology*, 2025.
AI-Based Fall Detection in IoT (2025) **[5]**	Proposes fine-tuned AI model for fall detection in IoT environments. Improves accuracy for disabled individuals. Supports real-time smart healthcare monitoring.	A. Alzahrani et al., "Artificial Intelligence-Based Fine-Tuning Model for Fall Activity Recognition in Disabled Persons within an IoT Environment," *Scientific Reports*, 2025.

Focused Area / Title	Key Findings	Reference
Infrared Behavior Recognition (2025) **[6]**	Uses multilevel feature fusion network for infrared video analysis. Improves performance in low-light conditions. Enhances accuracy in complex environments.	H. Han et al., "Multilevel Features Cascade Fusion Network for Infrared Video Human Behavior Recognition," *Displays*, 2025.
Multiscale CNN with Attention (2025) **[7]**	Proposes multiscale CNN with cross-modal channel attention. Enhances feature extraction and classification accuracy. Effective for complex signal-based recognition.	Y. Wei et al., "MCNN-CMCA: A Multiscale Convolutional Neural Networks with Cross-Modal Channel Attention for Physiological Signal-Based Mental State Recognition," *Digital Signal Processing*, 2025.
Multiscale CNN Behavior Recognition (2025) **[8]**	Applies Random Forest algorithms for detecting cyberbullying in microblog data. Effectively handles high-dimensional TF-IDF features and reduces overfitting.	G. Dianakamal et al., "Recognition of Human Behaviour Utilising Multiscale Convolutional Neural Networks," *Taylor & Francis*, 2025.
Audio-Video HAR in Healthcare (2024) **[9]**	Reviews multimodal HAR systems using audio and video data. Improves accuracy through data fusion. Discusses challenges like noise and system complexity. .	S. Cristina et al., "Audio-and Video-Based Human Activity Recognition Systems in Healthcare," *IEEE Access*, 2024.
YOLOv8-Based Fall Detection for Industrial Safety (2024) **[10]**	Proposes a fall detection system using YOLOv8 variants for real-time monitoring. Detects human posture changes efficiently to identify falls. Improves safety in industrial environments with fast and accurate detection.	G. A. Pereira, "Fall Detection for Industrial Setups Using YOLOv8 Variants," *arXiv preprint arXiv:2408.04605*, 2024.

Table 2.1: Literature Review Summary

3. SOFTWARE REQUIREMENTS ANALYSIS

3.1 Modules and Their Functionalities

3.1.1 Data Analysis

Falls among elderly individuals remain one of the leading causes of serious injuries, disabilities, and fatal accidents, especially when immediate assistance is not available. Traditional fall-detection solutions relying on wearable sensors suffer from low user compliance, discomfort, and frequent false alarms. Manual monitoring through CCTV is also impractical as it requires continuous human supervision. Therefore, there is a need for a real-time, non-intrusive, intelligent fall detection system capable of automatically recognizing human activities and identifying falls with high accuracy. The main problem addressed in this project is to detect falls using a camera-based solution that applies CNN for spatial feature extraction and LSTM for temporal movement analysis. The system must classify actions such as walking, sitting, standing, and falling, and provide instant alerts to caregivers. The goal is to reduce response time, prevent unattended injuries, and ensure safety for elderly individuals living alone or in assisted care environments.

3.1.2 Data Preprocessing

Data preprocessing ensures that all input video frames are standardized before being passed to the deep learning model. Frames are first extracted from the video stream and resized to a fixed resolution such as 224×224 pixels. Normalization is applied to scale pixel values for stable model performance. Additional steps like contrast enhancement or histogram equalization help in handling poor lighting conditions. Frames are then grouped into temporal sequences to maintain motion continuity. Augmentation techniques such as rotation, flipping, and scaling are used during training to improve model generalization. This preprocessing pipeline ensures that both CNN and LSTM receive clean and meaningful data.

3.1.3 Deep Learning Algorithm for Prediction

The system uses a hybrid CNN-LSTM architecture for accurate fall prediction. The CNN component processes each individual frame, extracting critical spatial features such as body posture, orientation, and movement direction. These extracted features are then passed sequentially into the LSTM network, which learns how these features evolve over time.

The algorithm predicts activities such as walking, sitting, standing, lying, or falling. This combination of CNN for spatial recognition and LSTM for motion analysis results in high accuracy and robust fall detection performance. The LSTM is responsible for capturing temporal dependencies and distinguishing between normal and abnormal activities. After temporal processing, the final feature vector is classified using a fully connected.

3.2 Functional Requirements

The system must be capable of continuously monitoring the environment using a camera and detecting human activities in real time. It should classify actions such as walking, sitting, standing, lying, and falling with high accuracy. When a fall is detected, the system must trigger an immediate alert through sound or notifications to caregivers. The dashboard should display real-time activity information and maintain logs of detected events. The system must operate efficiently under varying lighting and background conditions.

1. The system shall capture and process live video frames continuously.
2. The system shall classify activities using CNN-LSTM deep learning architecture.
3. The system shall trigger an alarm or notification when a fall is detected.
4. The system shall store activity logs for future monitoring and analysis.

3.3 Non-Functional Requirements

The system should be reliable, ensuring consistent performance without frequent false alerts or missed detections. It must deliver real-time results with minimum latency to support immediate response. The interface should be user-friendly, accessible, and simple for caregivers to interpret. The system must maintain privacy and security by ensuring that data is processed safely without misuse or unauthorized access. Additionally, the design should support scalability for deployment across multiple rooms or facilities.

1. The system shall provide high accuracy and low false-positive rates.
2. The system shall ensure fast response time for real-time fall detection.
3. The system shall be easy to use with a clear, intuitive interface.
4. The system shall ensuresecure handling of video data to protect user privacy.

3.4 Feasibility Study

The feasibility of the project is analyzed in this phase, and a business proposal is put forth with a general plan for the project and cost estimates. During system analysis, the feasibility study ensures that the proposed system is viable and not a burden to the company. Understanding the major system requirements is essential for feasibility analysis.

Key Considerations in Feasibility Analysis :

- Economic Feasibility
- Technical Feasibility
- Social Feasibility

3.4.1 Economic Feasibility

Economic feasibility focuses on the financial aspects of developing and implementing the fall detection system. Since the project uses low-cost hardware such as webcams and open-source software, the overall development cost is minimal. The long-term benefits, such as reduced medical emergencies and improved safety, outweigh the initial investment. Therefore, the system is economically justified and affordable for both families and organizations.

3.4.2 Technical Feasibility

Technical feasibility evaluates whether the required technologies, software tools, and hardware infrastructure are available to build the system. The CNN-LSTM model can be implemented using widely available frameworks like TensorFlow, Keras, and OpenCV. Cameras, GPUs, and local servers offer sufficient processing capability for real-time monitoring. Thus, the system is technically viable and supported by existing tools and resources.

3.4.3 Social Feasibility

From a social perspective, the system has strong feasibility because it provides direct benefits to the tennis community. It assists players and coaches in analyzing performance, movement patterns, and ball behavior without manual tagging. The system does not affect the normal gameplay or require athletes to wear sensors, making it convenient and user-friendly. The output visualizations are easy to interpret and help in performance improvement, injury prevention, and training strategy development. Therefore, the system promotes positive acceptance among its intended users.

4.SYSTEM SPECIFICATIONS

4.1 Software Requirements

The software requirements define the essential tools, libraries, and platforms necessary for developing and executing the fall detection and activity recognition system. This project utilizes a combination of deep learning frameworks, computer vision libraries, and web-based tools to ensure seamless model training, real-time video processing, and alert generation. The chosen software stack supports scalability, modularity, and compatibility for deployment across various environments.

- Programming Language: Python 3.x
- Deep Learning Frameworks: TensorFlow / Keras
- Computer Vision Library: OpenCV
- Numerical & Utility Libraries: NumPy, Scikit-learn
- Visualization Tools: Matplotlib / Seaborn
- Web Framework: Flask (for real-time dashboard and video streaming)
- Alert Mechanism: Pygame
- Development Environments: Jupyter Notebook, Visual Studio Code
- Environment Manager (Optional): Anaconda
- Version Control: Git
- Client-Side Interface: Modern Web Browser (Chrome / Firefox)
- Operating System: Windows 10 / Linux (Ubuntu recommended)

System Details

This section provides a detailed description of the components that make up the Fall Detection System using CNN and LSTM for Real-Time Action Classification in Elderly Care.

1. **Python 3.x (Programming Language)**

Python is the primary programming language used for developing the entire system. Its simplicity, rich library support, and vast ecosystem make it ideal for rapid development in the domains of computer vision and deep learning.

2. TensorFlow / Keras (Deep Learning Frameworks)

TensorFlow, with its high-level API Keras, is used to design and train the CNNLSTM model. CNN layers are responsible for extracting spatial features from image frames, while LSTM layers capture the temporal dynamics in a sequence of frames. Keras simplifies the implementation of these deep learning components with modular functions and automatic GPU acceleration (if available).

3. OpenCV (Computer Vision Library)

OpenCV (Open Source Computer Vision Library) is used for handling image and video input. It allows the system to interface with the webcam or video stream, process each frame (resizing, grayscale conversion, etc.), and pass it to the deep learning model. OpenCV also supports frame annotation, which can be used to visually indicate detected falls or activity classifications.

4. NumPy (Numerical Computation)

NumPy is a core Python library for numerical and matrix operations. It is used extensively for manipulating image arrays, reshaping model input/output, and performing mathematical calculations involved in preprocessing and model inference.

5. Matplotlib / Seaborn (Visualization Tools)

These libraries are used to visualize training metrics like accuracy and loss curves, confusion matrices, and class-wise performance analysis. They help in evaluating model behavior during experimentation and tuning.

6. Scikit-learn (Machine Learning Toolkit)

Scikit-learn provides tools for splitting datasets into training and testing sets, generating evaluation metrics like precision, recall, and F1-score, and displaying the confusion matrix. It complements TensorFlow by providing statistical and preprocessing utilities.

7. Flask (Web Framework for Real-Time Processing)

Flask is a micro web framework used to create a lightweight web interface. It handles the server-side logic for streaming video frames, routing them to the deep learning model for inference, and displaying results (such as detected activities or fall alerts) to the user in real time. It enables a clean separation between backend processing and frontend display

8.Pygame (Alert Mechanism)

Pygame is used to trigger sound-based alerts when a fall is detected. The sound alarm ensures that the user or caregiver is notified immediately. Pygame's asynchronous handling allows it to play audio while the rest of the system continues to process incoming video frames, maintaining real-time performance.

8. Jupyter Notebook / Visual Studio Code (IDE)

These are the primary development environments used to write, debug, and test code. Jupyter Notebook is particularly useful for visualizing intermediate results and model predictions step-by-step, while VS Code is preferred for modular development and Flask integration.

10. Anaconda (Optional Environment Manager)

Anaconda is optionally used to manage Python environments and dependencies efficiently. It helps maintain package compatibility and simplifies library installation through the conda package manager.

11. Operating System

The system can be run on both **Windows 10** and **Linux (Ubuntu)** environments. Linux is preferred for performance and compatibility in real-time deployments, especially when working with resource-constrained devices.

12. Git (Version Control)

Git is used for managing the source code, tracking changes, and collaborating with other developers if needed. It helps maintain a clean history of model improvements, experiments, and bug fixes.

13. Web Browser (Client-Side Interface)

A modern web browser (e.g., Google Chrome, Mozilla Firefox) is used to access the Flask web interface, where users can monitor real-time classification output and fall alert.

4.2 Hardware Requirements

The hardware requirements outline the components and computational resources essential for real-time fall detection and activity monitoring. These include both the computing hardware needed to run the deep learning model and optional sensory components for enhanced detection. The listed configuration supports current system functionality and remains scalable for future upgrades, including edge deployment or multi-sensor integration.

- Sensors (Optional for IoT-based systems): Pressure Sensors,Camera for video-based detection
- Processing Unit: Standard PC or Laptop for CNN–LSTM model execution
- Power Supply: Rechargeable Battery / Power Adapter
- Alert System: SMS or Email Notification Module
- Communication Module: Wi-Fi / Bluetooth for remote alerts and data transmission

5. SOFTWARE DESIGN

5.1 System Architecture

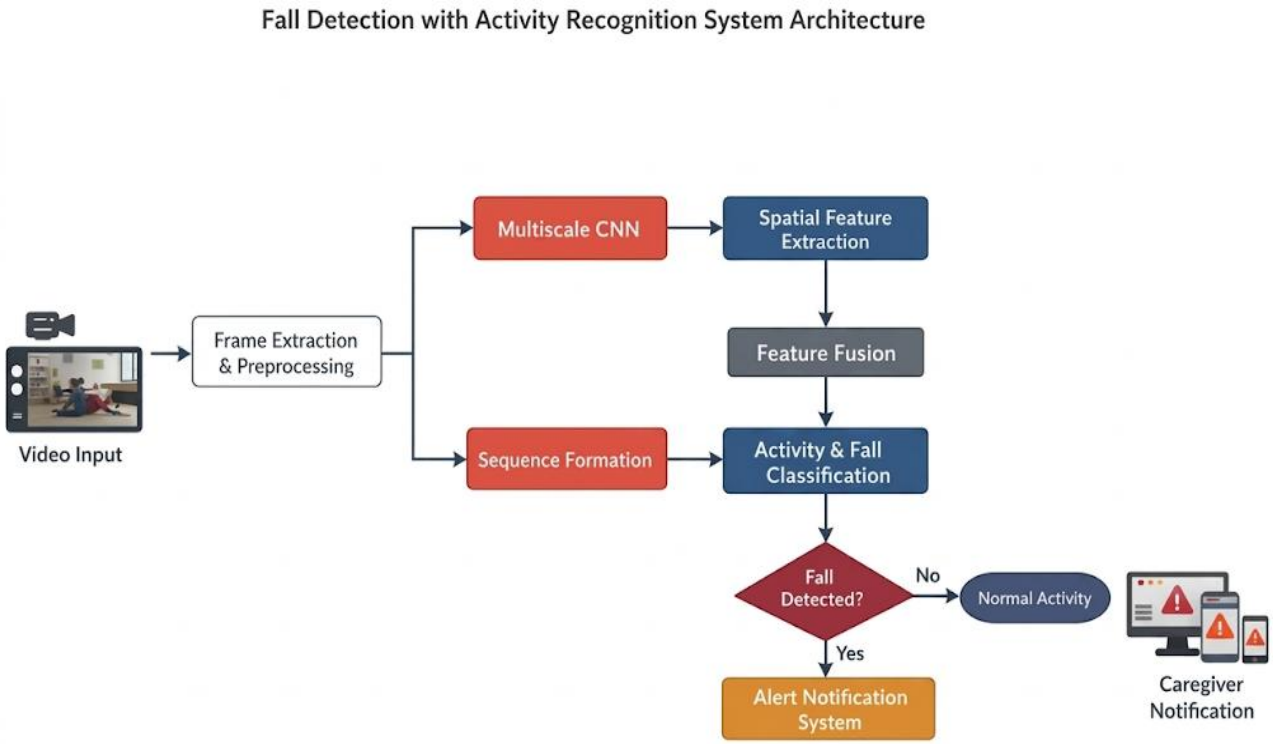

Fig. 5.1: System Architecture

The system architecture for Human Behaviour Recognition Based on Multiscale Convolutional Neural Network (CNN) is designed as a structured, end-to-end pipeline that processes video input, extracts hierarchical features, and classifies human activities with high accuracy, as shown in Fig. 5.1. The architecture integrates video preprocessing, multiscale feature extraction, temporal sequence modeling, and final activity classification to form a robust human behaviour recognition framework.

In the proposed fall detection system, video input captured from the camera is first converted into sequential frames. These frames undergo preprocessing operations such as resizing, normalization, and noise removal to improve data quality and reduce computational complexity. After preprocessing, the frames are passed into the Convolutional Neural Network (CNN) model, as illustrated in Fig. 5.1.

The CNN plays an important role in extracting spatial features from each frame. It detects important visual patterns such as body posture, movement orientation, and positional variations of the human subject. Multiple convolutional layers analyze the input at different receptive fields, enabling effective multiscale feature learning. These spatial features help distinguish between normal activities like walking, sitting, and standing, and abnormal activities such as falling.

Spatial Feature Extraction Layer

The spatial feature extraction stage focuses on identifying posture-related characteristics from individual frames. The convolution layers apply filters that capture:

- body alignment variations
- motion direction changes
- silhouette structure
- sudden posture imbalance patterns

These extracted features are converted into **feature vectors**, which serve as input for temporal sequence modeling in the next stage.

Temporal Feature Modeling using LSTM Network

After extracting spatial information, the feature vectors are forwarded to the **Long Short-Term Memory (LSTM)** network. Unlike CNN, which processes single frames independently, LSTM analyzes frame sequences over time.

This temporal modeling helps the system understand motion continuity and detect transitions between activities. The LSTM network effectively captures:

- sequential motion behavior
- sudden activity interruptions
- posture transition speed
- fall motion trajectory patterns

By analyzing time-dependent variations in movement, the system improves the accuracy of fall detection compared to traditional frame-based approaches.

The CNN plays an important role in extracting spatial features from each frame. It detects important visual patterns such as body posture, movement orientation, and positional variations of the human subject. Multiple convolutional layers analyze the input at different receptive fields, enabling effective multiscale feature learning. These spatial features help distinguish between normal activities like walking, sitting, and standing, and abnormal activities such as falling.

Spatial Feature Extraction Layer

The spatial feature extraction stage focuses on identifying posture-related characteristics from individual frames. The convolution layers apply filters that capture:

- body alignment variations
- motion direction changes
- silhouette structure
- sudden posture imbalance patterns

These extracted features are converted into **feature vectors**, which serve as input for temporal sequence modeling in the next stage.

Temporal Feature Modeling using LSTM Network

After extracting spatial information, the feature vectors are forwarded to the **Long Short-Term Memory (LSTM)** network. Unlike CNN, which processes single frames independently, LSTM analyzes frame sequences over time.

This temporal modeling helps the system understand motion continuity and detect transitions between activities. The LSTM network effectively captures:

- sequential motion behavior
- sudden activity interruptions
- posture transition speed
- fall motion trajectory patterns

By analyzing time-dependent variations in movement, the system improves the accuracy of fall detection compared to traditional frame-based approaches.

5.2 Dataflow Diagram

The system architecture for the Human Behaviour Recognition and Fall Detection system based on Multiscale Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) network is designed as a structured and efficient processing pipeline that captures video input, extracts meaningful spatial and temporal features, and performs activity classification with high reliability, as shown in Fig. 5.2. The architecture combines deep learning techniques with real-time monitoring capabilities to provide an intelligent and non-intrusive fall detection solution suitable for healthcare monitoring environments.

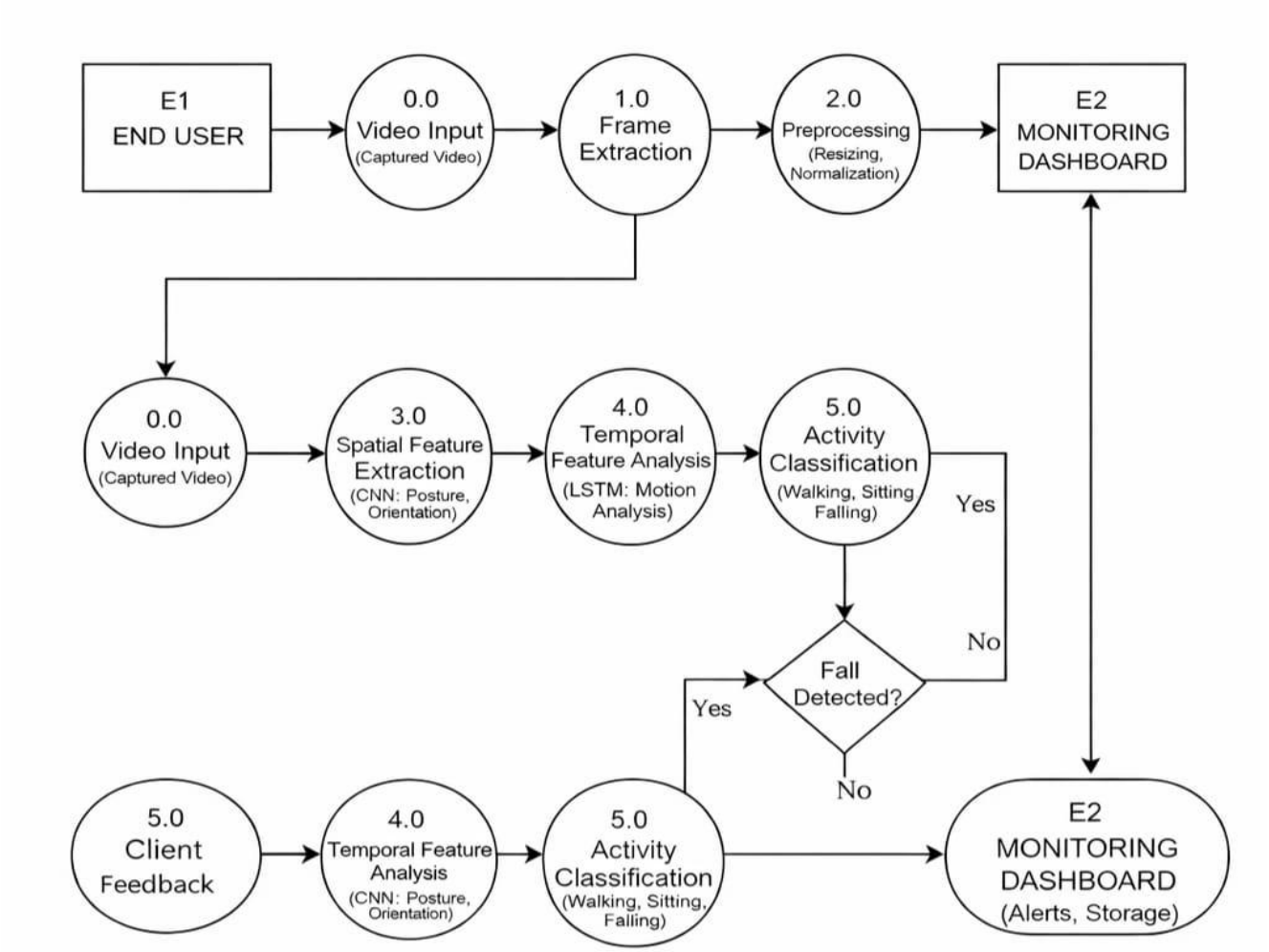

Fig. 5.2: Data Flow Diagram

In the proposed system, the architecture begins with continuous video acquisition through a camera placed in the monitoring environment. The captured video acts as the primary input source for behaviour recognition. The system continuously records human movements and converts the input video stream into sequential frames for further processing. This frame-level representation allows the system to analyze posture variations and movement transitions effectively over time.

Before feature extraction, the input frames undergo preprocessing operations to improve image quality and maintain uniformity across the dataset. These preprocessing steps include resizing the frames into fixed dimensions, normalization of pixel intensity values, and removal of unwanted noise that may affect detection accuracy. These steps ensure that the deep learning model receives consistent input data and improves computational efficiency during training and testing phases.

After preprocessing, the processed frames are forwarded to the Convolutional Neural Network (CNN), which performs spatial feature extraction. The CNN architecture consists of multiple convolutional layers that apply filters to detect posture-related information such as body orientation, structural alignment, and motion direction patterns. These convolutional layers operate at different receptive field sizes, enabling multiscale feature extraction. This helps the system identify both fine-level motion variations and overall body posture patterns that are essential for distinguishing fall events from normal daily activities.

Pooling layers are integrated within the CNN architecture to reduce feature dimensionality while preserving important spatial information. This dimensionality reduction improves computational performance and prevents overfitting during model training. Activation functions such as ReLU are applied to introduce non-linearity into the system, allowing the model to learn complex movement.

Following classification, the fall detection decision module verifies whether the predicted activity corresponds to a fall event. If a fall is detected, the system immediately activates the alert generation module. This module ensures quick response by notifying caregivers or monitoring authorities about the detected emergency situation. The decision-making process improves reliability by combining spatial posture information and temporal motion sequence analysis using the CNN–LSTM hybrid architecture, as represented in Fig. 5.2.

The alert generation module is integrated with a real-time monitoring dashboard developed using the Flask framework. The dashboard displays detection results along with activity status and timestamps. This monitoring capability improves system usability and supports continuous supervision of elderly individuals in smart healthcare environments, as shown in Fig. 5.2.

It also maintains prediction logs that help administrators monitor historical activity patterns and review previous fall events when required. This monitoring capability improves system usability and supports continuous supervision of elderly individuals in smart healthcare environments, as shown in Fig. 5.2.

Additionally, the system architecture supports integration with IoT-based monitoring infrastructure, enabling remote accessibility of detection results through connected devices. This integration enhances system scalability and allows caregivers to receive notifications even when they are located remotely. Therefore, the overall architecture provides an efficient, scalable, and intelligent solution for real-time human behaviour recognition and fall detection applications.

5.3 UML Diagrams

UML (Unified Modeling Language) is a standardized modeling language used to visualize, design, and document the components of a software system. In the context of the Human Behaviour Recognition Based on Multiscale Convolutional Neural Network project, UML diagrams play a significant role in representing system interactions, processing workflows, and data structures involved in video-based activity recognition. Developed by the Object Management Group (OMG), UML provides a structured and expressive way to model object-oriented systems nd supports both high-level system visualization and detailed component specification.

UML diagrams used in this project help in illustrating how the video input is processed, how the MSCNN extracts multiscale features, how classification modules interact, and how real-time alerts or outputs are generated. The diagrams also support better understanding of system behavior, modularity, and interactions between internal processes and external actors such as users or caregivers.

The Goals of UML are:

- To provide an expressive and standardized visual modeling language for designing software systems.
- To help developers represent system structure, behavior, and interactions clearly.
- To establish a formal basis for understanding and documenting software models.
- To simplify communication between developers, analysts, designers, and stakeholders.
- To support object-oriented analysis and design principles effectively.
- To reduce system design complexity by breaking the software into manageable components.
- To assist in identifying system requirements, modules, and relationships before coding begins.
- To improve software quality by enabling better planning and error detection during design.
- To encourage the growth and use of object-oriented tools and CASE tools.
- To provide reusable and scalable design models for future system enhancements.

Types of UML Diagrams:

1. Sequence Diagram.
2. Class Diagram.
3. Use Case Diagram.
4. Activity Diagram.

5.3.1 Sequence Diagram

The sequence diagram illustrates the step-by-step interaction among the User, Camera/Video Source, Preprocessing Module, MSCNN Model, Classifier, and optional Alert System during the human behaviour recognition process. The sequence begins when the user initiates the system or when the camera starts capturing live video from the monitoring environment. The camera continuously streams video frames into the system in real time, enabling continuous observation of human activities within the surveillance area. These frames act as the primary input for behaviour recognition and fall detection analysis, as shown in Fig. 5.3.

After capturing the live video stream, the incoming frames are forwarded to the preprocessing module, where necessary enhancement operations such as frame extraction, resizing, normalization, and noise reduction are performed. These preprocessing steps improve the visual quality of the frames and ensure uniform input formatting before feature extraction begins. Standardizing the frame structure helps the deep learning model process activity information efficiently and improves the overall prediction performance of the system, as illustrated in Fig. 5.3.

Once preprocessing is completed, the processed frames are supplied to the Multiscale Convolutional Neural Network (MSCNN) model for spatial feature extraction. The MSCNN architecture analyzes posture-related characteristics such as body orientation, structural alignment, and movement patterns using multiscale convolutional filters. These extracted spatial features are then forwarded to the classification module, where the classifier identifies the corresponding human activity category based on learned feature representations. This stage plays an important role in distinguishing normal daily activities from abnormal behaviour patterns such as fall events.

Finally, the recognized activity and any alert status are returned to the user through the system interface or monitoring dashboard for visualization and decision support. If a fall event or abnormal activity is detected, the optional alert system generates a notification to ensure immediate attention from caregivers or monitoring personnel. This structured interaction sequence demonstrates how the system processes live video data through organized stages to achieve accurate and real-time human behaviour recognition performance.

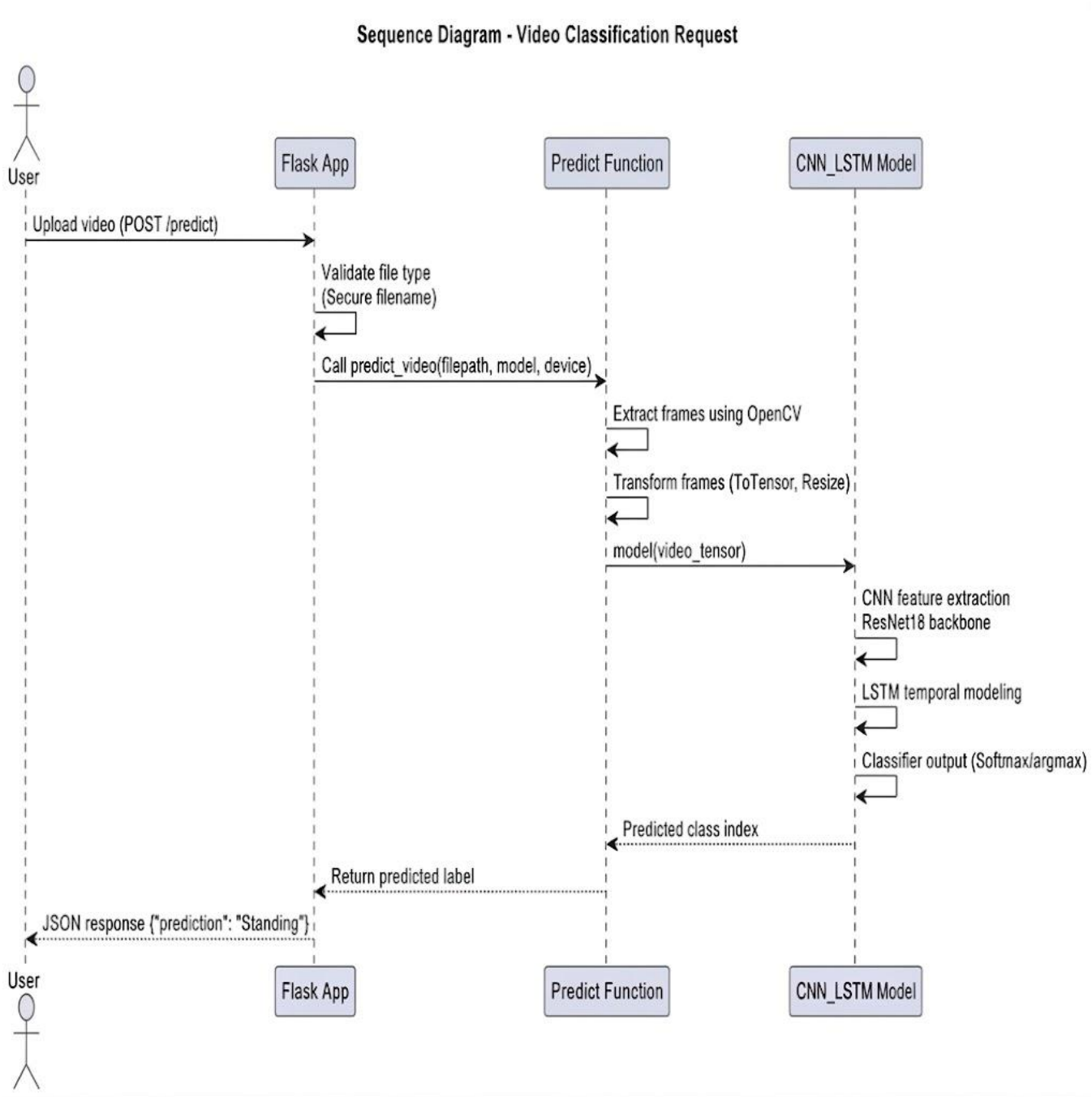

Fig. 5.3: Sequence Case Diagram of YOLO v5 System

List of actions

• User upload request

The sequence diagram begins with the user uploading a video file through the frontend interface of the Human Behaviour Recognition system. This uploaded video acts as the primary input for behaviour analysis and fall detection processing. The upload operation initiates interaction between the user interface and backend processing modules, as shown in Fig. 5.3.

• Request sent to Flask backend controller

After the video upload operation is completed, the frontend sends the request to the Flask backend server. The Flask framework acts as the central controller that manages communication between system components and coordinates execution of preprocessing, feature extraction, and prediction modules for behaviour recognition.

• File validation and security checking

Once the backend receives the uploaded video file, it performs validation procedures to ensure compatibility with the supported file format. The system also performs secure filename handling before initiating processing operations to prevent errors and maintain system reliability during prediction workflow execution.

• Prediction function invocation

After successful validation of the uploaded input, the Flask backend invokes the prediction function responsible for executing the behaviour recognition pipeline. This function coordinates frame extraction, preprocessing operations, spatial feature extraction, and temporal sequence analysis in an organized processing sequence.

• Frame extraction using OpenCV

Inside the prediction module, the uploaded video is converted into individual frames using OpenCV-based frame extraction techniques. Each extracted frame represents a snapshot of activity occurring within the video sequence and enables the system to perform detailed posture-level spatial analysis in later processing stages.

• Frame transformation and preprocessing

After frame extraction, preprocessing operations such as resizing, normalization, and tensor conversion are applied to standardize the input frame structure. These preprocessing operations improve compatibility with the CNN–LSTM architecture and ensure consistent feature learning performance.

• Spatial feature extraction using CNN

Following preprocessing, the transformed frames are forwarded to the Convolutional Neural Network for spatial feature extraction. The CNN extracts posture-related characteristics such as body alignment, structural orientation, silhouette variations, and motion direction indicators from individual frames, enabling accurate understanding of activity patterns within the monitored environment.

• Temporal sequence analysis using LSTM

After spatial feature extraction is completed, the generated feature vectors are passed sequentially to the Long Short-Term Memory network for temporal modeling. The LSTM captures motion continuity across consecutive frames and identifies posture transition patterns that help distinguish fall events from normal daily activities with improved prediction accuracy.

• Activity classification output generation

Once temporal modeling is completed, the classification layer generates probability scores for different human activity classes using softmax activation. The system selects the activity label with the highest probability value as the final prediction result representing the detected behaviour within the uploaded video sequence.

• Response generation by backend server

After prediction processing is completed, the backend converts the classification output into a readable activity label and prepares the structured response for visualization. This response is then transmitted back to the frontend interface through the Flask server communication channel.

• Prediction result display to user

Finally, the frontend interface receives the prediction result from the backend server and displays the detected activity label on the monitoring dashboard. If a fall event is identified, the system highlights the alert notification for immediate attention and monitoring support. This completes the interaction workflow represented in the sequence diagram, as illustrated in Fig. 5.3.

5.3.2 Class Diagram

The Class Diagram represents the structural components of the Human Behaviour Recognition system and the relationships between them, as shown in Fig. 5.4. The VideoCapture class handles real-time video streaming or video file inputs, providing extracted frames to the rest of the system. The Preprocessing class performs essential operations such as frame resizing, normalization, noise removal, and motion enhancement, ensuring the video frames are in optimal form before feature extraction.

The FeatureExtraction class is responsible for extracting meaningful spatial information from the processed frames using deep learning techniques such as Convolutional Neural Networks (CNN). This class identifies important visual patterns including body posture, movement orientation, and spatial characteristics required for behaviour understanding. The extracted spatial features are then forwarded to the SequenceModel class, which applies Long Short-Term Memory (LSTM) networks to capture temporal dependencies between consecutive frames and analyze motion transitions effectively.

Finally, the AlertSystem and OutputHandler classes manage the response generated after activity recognition. The AlertSystem class triggers notifications whenever abnormal behaviour such as fall detection is identified, enabling quick monitoring and safety response. The OutputHandler class manages result visualization, report generation, and storage of processed video outputs for further analysis. Together, these classes form a structured pipeline architecture that improves system performance, scalability, and ease of future enhancement, as illustrated in Fig. 5.4.

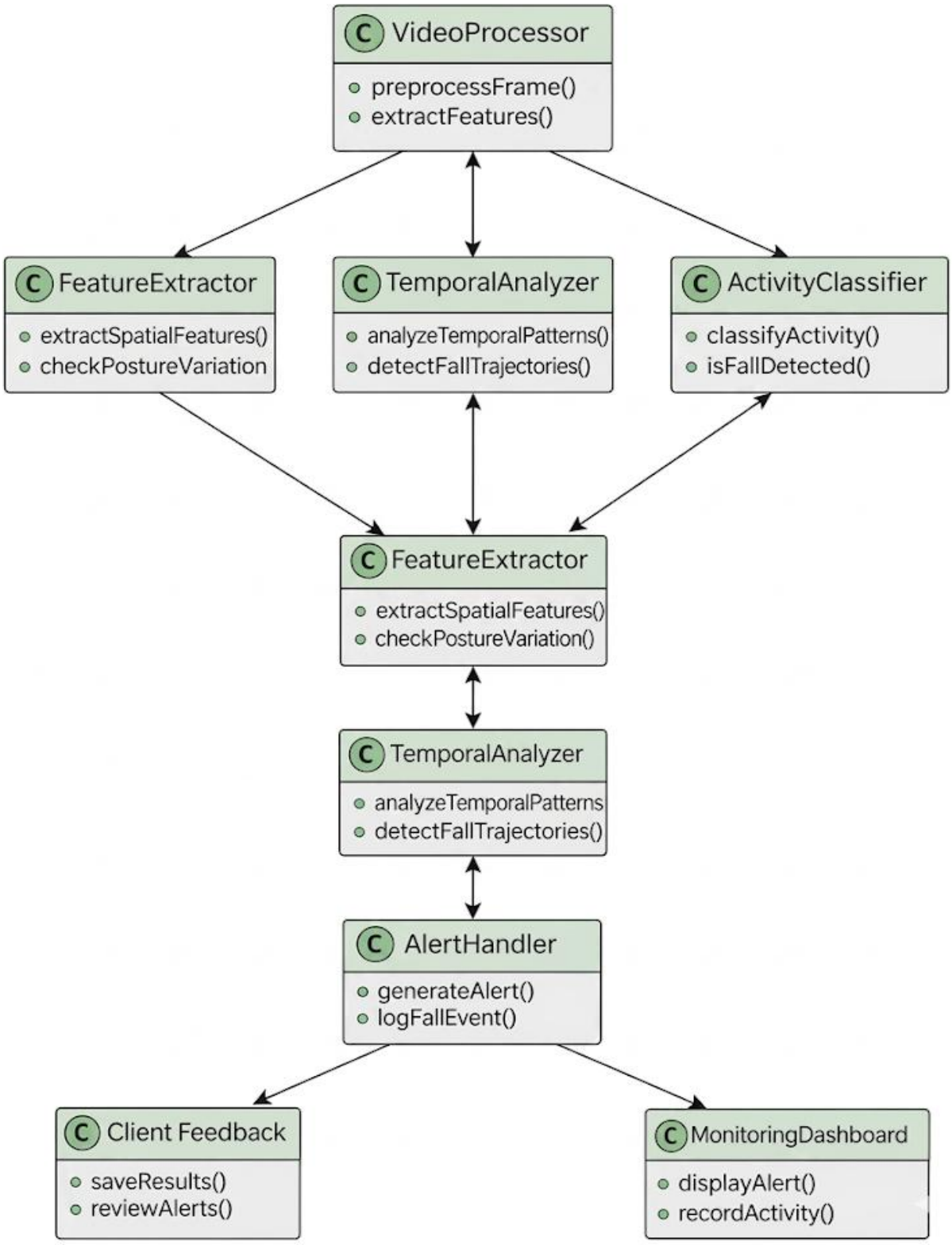

Figure 5.4 Class Diagram

5.3.2.1 Components of Class Diagram:

- **Camera Input Class**

The **Camera Input** class is responsible for capturing video data from the camera. It acts as the starting point of the system. The methods captureVideo() and sendFrames() are used to record video and forward frames to the next module. This class ensures continuous input for real-time monitoring.

- **Preprocessing Module Class**

The **Preprocessing Module** class handles all operations required to clean and prepare the video frames. It receives raw frames from the CameraInput class and processes them using methods like extractFrames(), resizeFrames(), removeNoise(), and normalizeFrames(). This step ensures that the data is consistent and suitable for deep learning models.

- **CNN Model Class**

The **CNN Model** class is responsible for extracting spatial features from the processed frames. It uses methods such as loadModel() to initialize the trained CNN and extractSpatialFeatures() to identify patterns like body posture and orientation. This class focuses on understanding what is happening in each frame.

- **LSTM Model Class**

The **LSTM Model** class processes the sequence of spatial features generated by the CNN. It uses loadSequenceModel() and analyzeTemporalPatterns() to study how activities change over time. This helps in identifying motion patterns and distinguishing between normal actions and abnormal events like falls.

- **Activity Recognition Class**

The **Activity Recognition** class takes the temporal features from the LSTM model and classifies them into specific activities. The method classifyActivities() is used to label actions such as walking, sitting, standing, or falling. This class converts model outputs into meaningful activity labels.

- **Fall Detection Class**

The **Fall Detection** class checks whether the recognized activity corresponds to a fall. Using the method detectFallEvent(), it identifies emergency situations by analyzing activity patterns.

- **Alert System Class**

The **Alert System** class is responsible for generating and sending alerts when a fall is detected. It uses methods like generateAlert() and sendNotification() to notify caregivers. This ensures quick response during emergencies.

- **Dashboard Class**

The **Dashboard** class provides a user interface for monitoring activities. It displays real-time status and activity updates using methods like displayStatus() and monitorActivities(). It helps administrators track system performance and user activity.

- **Caregiver Class**

The **Caregiver** class represents the external user who receives alerts. The method receiveAlert() allows the caregiver to get notifications when a fall is detected. This ensures timely assistance to the elderly person

- **Flow of Data**

The overall flow starts from **CameraInput**, moves through preprocessing, feature extraction (CNN), temporal analysis (LSTM), activity recognition, fall detection, and finally alert generation. The output is displayed on the dashboard and sent to the caregiver. This structured flow ensures efficient and real-time operation.

5.3.3 Use Case Diagram

The use case diagram illustrates the interaction between the User, Caregiver, Camera/Video Source, and the Human Behaviour Recognition System within the MSCNN-based activity monitoring framework. The user or elderly individual is monitored through a continuous video stream captured by the camera, while the caregiver or system operator interacts with the system for monitoring and receiving notifications as represented in Fig 5.5.

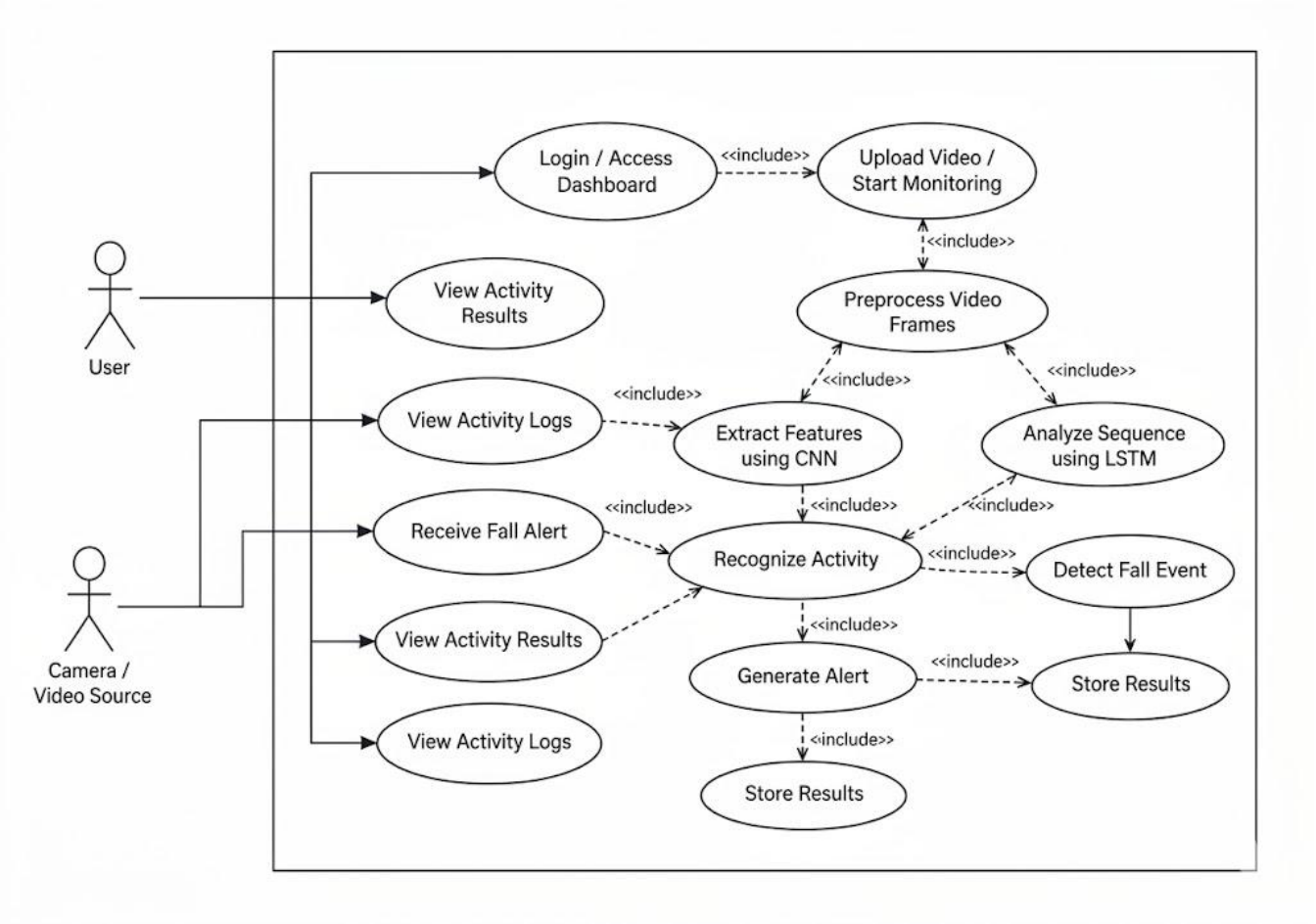

Fig. 5.5: Use Case Diagram

Actors in the System

- **Elderly Person**

The elderly person is the primary subject of monitoring. This actor does not directly interact with the system interface but is continuously observed through video input. Their movements and activities are captured automatically, making the system non-intrusive and user-friendly.

- **System Admin**

The system administrator is responsible for managing and maintaining the system. This includes monitoring system performance, accessing dashboards, and managing stored data or logs. The admin ensures the system runs efficiently without errors.

- **Caregiver**

The caregiver is the end user who receives alerts and monitors the elderly person. This actor plays a critical role in responding to emergency situations such as falls and ensuring timely assistance.

- **System Boundary**

The system boundary defines the scope of the application, labeled as the **"Fall Detection and Activity Recognition System."** All internal processes such as video processing, activity recognition, and alert generation occur within this boundary, while actors interact from outside.

- **Capture Video Input**

This use case represents the initial step where the system captures live video from a camera. The camera continuously records the elderly person's movements. This data acts as the primary input for further processing and analysis.

- **Preprocess Frames**

Once the video is captured, it is divided into frames and processed to improve quality and consistency. Preprocessing includes resizing, normalization, and noise reduction. This step ensures that the data is suitable for deep learning models.

- **Extract Spatial Features using CNN**

In this stage, the Convolutional Neural Network (CNN) analyzes each frame to extract spatial features such as body posture, orientation, and appearance. These features help the system understand what the person is doing at a specific moment.

- **Analyze Temporal Patterns using LSTM**

The extracted features are passed to the LSTM model, which analyzes sequences of frames over time. This helps in understanding motion patterns and distinguishing between normal activities and abnormal events like falling.

- **Recognize Human Activities**

Based on spatial and temporal analysis, the system identifies the activity being performed. Activities may include walking, sitting, standing, bending, or lying down. This step is crucial for continuous monitoring.

- **Detect Fall Event**

After recognizing activities, the system specifically checks for fall events. It differentiates between normal actions and sudden falls using learned motion patterns. This ensures accurate detection and reduces false alarms.

- **Generate Alert Notification**

If a fall is detected, the system generates an alert. This alert may include a visual notification, sound alarm, or message indicating an emergency situation. This step ensures quick attention to critical events.

- **Send Alert to Caregiver**

The generated alert is sent to the caregiver through a notification system such as a web dashboard, SMS, or mobile alert. This enables the caregiver to take immediate action and provide assistance.

- **Display Monitoring Dashboard**

The system admin can access a dashboard that displays real-time activity information and system status. This helps in monitoring ongoing activities and reviewing past events.

- **Maintain System Records**

The system stores activity logs and alert history for future reference. The admin can manage these records for analysis, improving system performance, or tracking behavior patterns over time

5.3.4 Activity Diagram

The activity diagram illustrates the sequential workflow of the human behaviour recognition system. The process begins with capturing live video input from the camera or uploading an existing video. The incoming video stream is then sent to the preprocessing stage, where essential operations such as frame extraction, resizing, normalization, and noise reduction are applied. Once the frames are cleaned and standardized, they proceed to the feature extraction stage as represented in Fig 5.6.

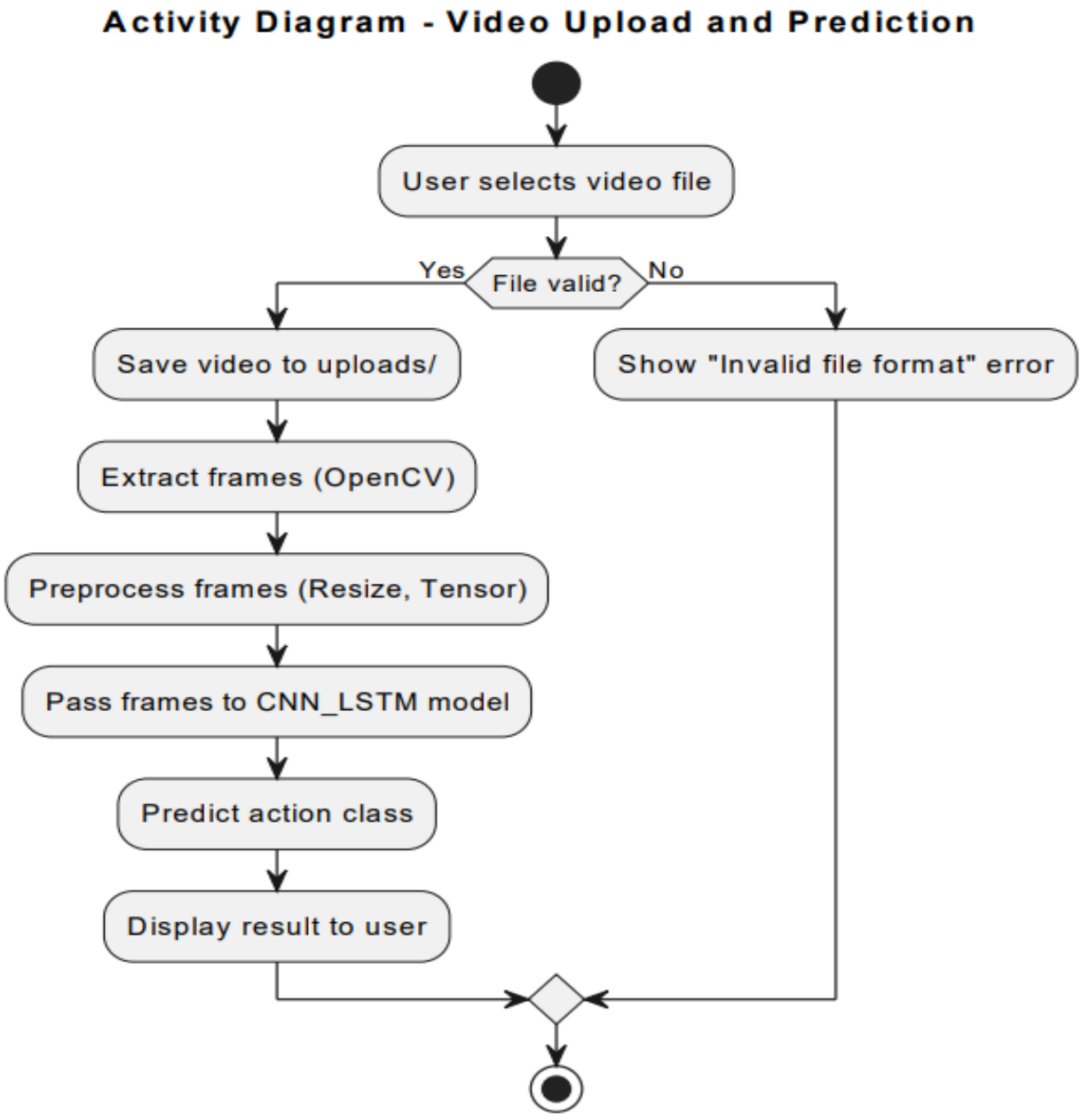

Fig. 5.6: Activity Diagram

- **Start of Process**

The workflow begins when the user uploads or selects a video file through the system interface. This step acts as the starting point of the activity diagram, where the system receives raw input for processing.

- **Video Input Acquisition**

After initiation, the system captures the video input provided by the user. This video may be a recorded file or a live stream. The input serves as the primary data source for activity recognition.

- **File Validation**

The system checks whether the uploaded video is in a valid format. Supported formats include standard video types such as MP4 or AVI. If the file is invalid, the system stops execution and displays an error message. If valid, the process continues to the next stage.

- **Video Storage**

Once validated, the video is stored in a designated location (such as an uploads folder). This ensures organized data handling and allows the system to access the file for further processing.

- **Frame Extraction**

The stored video is then broken down into individual frames using computer vision techniques like OpenCV. Each frame represents a snapshot of the video at a particular moment, allowing the system to analyze motion step-by-step.

- **Data Preprocessing**

The extracted frames undergo preprocessing to ensure consistency and improve model performance. This includes resizing frames to a fixed size, normalizing pixel values, and converting them into tensor format. Preprocessing reduces noise and prepares data for deep learning models.

- **Feature Extraction using CNN**

The preprocessed frames are passed to the Convolutional Neural Network (CNN). The CNN extracts spatial features such as body posture, position, and appearance from each frame. These features help the system understand what is happening in each image.

- **Temporal Analysis using LSTM**

The extracted features are then fed into the Long Short-Term Memory (LSTM) network. The LSTM analyzes sequences of frames over time, capturing motion patterns and transitions between actions. This helps in identifying dynamic activities accurately.

- **Activity Recognition**

Based on spatial and temporal analysis, the system classifies the activity being performed. It can recognize actions such as walking, sitting, standing, bending, or falling. This stage produces the main prediction output.

- **Fall Detection Decision**

After recognizing the activity, the system checks whether the detected action corresponds to a fall. This is an important decision point in the workflow. If a fall is detected, the system proceeds to alert generation.

- **Alert Generation**

When a fall event is identified, the system generates an alert notification. This alert indicates an emergency situation and ensures immediate attention.

- **Display Result**

If no fall is detected, the system simply displays the recognized activity to the user. The result is shown clearly on the interface for monitoring purposes.

- **Error Handling**

If the system encounters any issues such as invalid input or processing errors, it handles them gracefully by displaying appropriate messages. This ensures smooth user experience.

- **End of Process**

The workflow ends after displaying the result or generating an alert. The system is then ready to process the next input, supporting continuous monitoring

6. CODING AND IMPLEMENTATION

6.1 Source Code

Model.py:

 # Import Required Libraries

```
import os import cv2 import math import random
import numpy as np import tensorflow as tf
from collections import deque import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping from tensorflow.keras.utils import
plot_model
from IPython.display import Video
```

 # Check GPU Availability

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

 # Set Random Seed for Reproducibility

```
seed_constant = 27 np.random.seed(seed_constant)
random.seed(seed_constant) tf.random.set_seed(seed_constant)
```

 # Define Dataset Constants

```
IMAGE_HEIGHT , IMAGE_WIDTH = 256, 256 # Resize all video frames to this size
SEQUENCE_LENGTH = 10 # Number of frames per sequence for the LSTM model
```

 # Define Dataset Path

```
DATASET_DIR = "Human Activity Recognition - Video Dataset"
```

 # Extract Class Names (Activity Labels) from Dataset Directory

```
CLASSES_LIST = sorted([entry.name for entry in os.scandir(DATASET_DIR) if entry.is_dir()])
```

 # Print Extracted Class Labels

```
print("Detected Activity Classes:", CLASSES_LIST)
```

 # Function to Extract Frames from Videos

```
Defextract_frames_from_video(video_path,
sequence_length=SEQUENCE_LENGTH):frames_list = []
```

 # Capture the video

```
video_reader = cv2.VideoCapture(video_path)
total_frames = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
```

 # Select Frame Indices

```
frame_skip_interval = max(1, total_frames // sequence_length)
selected_frame_indices = [i * frame_skip_interval for i in range(sequence_length)]

for frame_index in selected_frame_indices
 video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_index)
 success, frame = video_reader.read()
 if not success:
 break
```

```
Num GPUs Available:  0
Detected Activity Classes: ['Clapping', 'Meet and Split', 'Sitting', 'Standing Still', 'Walking', 'Walking While Reading Book', 'Walking While Using Phone']
```

 # Resize and Normalize Frame

```
frame = cv2.resize(frame, (IMAGE_WIDTH, IMAGE_HEIGHT)) frame =
frame / 255.0  # Normalize pixels to range 0-1 frames_list.append(frame)
video_reader.release()
return frames_list if len(frames_list) == sequence_length else None
```

 # Function to Create Video Frame Sequences

```
def create_dataset(video_directory, classes_list,
sequence_length=SEQUENCE_LENGTH):X_data, y_data = [], []

for class_label, class_name in enumerate(classes_list):
class_path = os.path.join(video_directory, class_name)
video_files = [os.path.join(class_path, file) for file in os.listdir(class_path)
if file.endswith('.mp4')]
print(f"Processing Class: {class_name} ({len(video_files)} videos)")
for video_path in video_files:
frames_sequence = extract_frames_from_video(video_path, sequence_length)
if frames_sequence:
X_data.append(frames_sequence)
y_data.append(class_label)
return np.array(X_data), np.array(y_data)
```

 # Run Data Processing

```
X, y = create_dataset(DATASET_DIR, CLASSES_LIST)
```

 # Print Dataset Shape

```
print(f"Dataset Shape: {X.shape}, Labels Shape: {y.shape}")
```

```
Processing Class: Clapping (55 videos)
Processing Class: Meet and Split (55 videos)
Processing Class: Sitting (55 videos)
Processing Class: Standing Still (55 videos)
Processing Class: Walking (55 videos)
Processing Class: Walking While Reading Book (55 videos)
Processing Class: Walking While Using Phone (55 videos)
Dataset Shape: (385, 10, 256, 256, 3), Labels Shape: (385,)
```

 # Import required library

from tensorflow.keras.utils import to_categorical from sklearn.model_selection import train_test_split

 # Split dataset into Training & Testing sets (80% Train, 20% Test)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=seed_constant, stratify=y)

 # Convert Labels to One-Hot Encoding

y_train = to_categorical(y_train, num_classes=len(CLASSES_LIST)) y_test = to_categorical(y_test, num_classes=len(CLASSES_LIST))

 # Print dataset shapes

```
print(f"Training Data Shape: {X_train.shape}, Training Labels Shape: {y_train.shape}")
print(f"Testing Data Shape: {X_test.shape}, Testing Labels Shape: {y_test.shape}")
```

```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50 from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, TimeDistributed, LSTM, Dense, Dropout,
GlobalAveragePooling2D, Reshape
```

 # Define CNN Feature Extractor

```
def build_cnn():
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(IMAGE_HEIGHT,
IMAGE_WIDTH, 3))
base_model.trainable = False  # Freeze pretrained layers
model = Model(inputs=base_model.input, outputs=GlobalAveragePooling2D()(base_model.output))
return model
```

 # Build Hybrid CNN + LSTM Model

```
    def build_hybrid_model():
    cnn = build_cnn()
```

 # Define Input Shape

```
input_layer = Input(shape=(SEQUENCE_LENGTH, IMAGE_HEIGHT, IMAGE_WIDTH, 3))
```

 # Apply CNN Feature Extraction on Each Frame

```
cnn_output = TimeDistributed(cnn)(input_layer)
```

 # Reshape CNN Output to Fit LSTM Input

```
lstm_input = Reshape((SEQUENCE_LENGTH, 2048))(cnn_output)
```

 # LSTM for Temporal Learning

```
lstm_output = LSTM(256, return_sequences=False, dropout=0.3)(lstm_input)
```

 # Fully Connected Layers

dense_output = Dense(128, activation="relu")(lstm_output)

dense_output = Dropout(0.3)(dense_output)

 # Compile Model

model = Model(inputs=input_layer, outputs=output_layer)

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

return model

 # Instantiate Model

model = build_hybrid_model() model.summary()

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 10, 256, 256, 3)	0
time_distributed (TimeDistributed)	(None, 10, 2048)	23,587,712
reshape (Reshape)	(None, 10, 2048)	0
lstm (LSTM)	(None, 256)	2,360,320
dense (Dense)	(None, 128)	32,896
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 7)	903

Total params: 25,981,831 (99.11 MB)

Trainable params: 2,394,119 (9.13 MB)

Non-trainable params: 23,587,712 (89.98 MB)

```
import cv2

import numpy as np

from tensorflow.keras.utils import Sequence
```

 # Custom Data Generator

```
class VideoFrameGenerator(Sequence):

def __init__(self, video_paths, labels, batch_size=4, sequence_length=SEQUENCE_LENGTH,
 shuffle=True):
self.video_paths = video_paths
self.labels = labels
self.batch_size = batch_size
self.sequence_length = sequence_length
self.shuffle = shuffle
self.indices = np.arange(len(self.video_paths))

def __len__(self):
return int(np.floor(len(self.video_paths) / self.batch_size))

def __getitem__(self, index): indices = self.indices[index * self.batch_size:(index + 1) *
self.batch_size]  batch_videos  = [self.video_paths[i]

for i in indices] batch_labels = np.array([self.labels[i] for i in indices])
X, y = self._load_batch(batch_videos, batch_labels)
return np.array(X), np.array(y)
def _load_batch(self, batch_videos, batch_labels):
X_batch, y_batch = [], []
for video_path, label in zip(batch_videos, batch_labels): frames =
self._extract_frames(video_path)
if frames is not None:
X_batch.append(frames)
y_batch.append(label)
return np.array(X_batch), np.array(y_batch)
```

```
  def _extract_frames(self, video_path):
  frames_list = []
  video_reader = cv2.VideoCapture(video_path)
  total_frames = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
  frame_skip_interval = max(1, total_frames // self.sequence_length)
  for i in range(self.sequence_length): frame_idx =
  i * frame_skip_interval
 video_reader.set(cv2.CAP_PROP_POS_FRAMES, frame_idx) success,
 frame = video_reader.read()
 if not success: break
frame = cv2.resize(frame, (IMAGE_WIDTH, IMAGE_HEIGHT)) frame =
frame / 255.0  # Normalize pixel values frames_list.append(frame)
video_reader.release()
return np.array(frames_list) if len(frames_list) == self.sequence_length else None
def on_epoch_end(self):
if self.shuffle:
np.random.shuffle(self.indices)
```

 # Create Video Paths List

```
video_paths_train = []
video_paths_test = []


for class_name in CLASSES_LIST:
class_videos = os.listdir(os.path.join(DATASET_DIR, class_name))
for video in class_videos:
full_path = os.path.join(DATASET_DIR, class_name, video)
if np.random.rand() < 0.8:
video_paths_train.append(full_path)
else:
video_paths_test.append(full_path)
```

 # Convert Labels to One-Hot Encoding

```
y_train_one_hot = to_categorical([CLASSES_LIST.index(os.path.basename(os.path.dirname(p))) for p in
video_paths_train], num_classes=len(CLASSES_LIST))
y_test_one_hot = to_categorical([CLASSES_LIST.index(os.path.basename(os.path.dirname(p))) for p in
video_paths_test], num_classes=len(CLASSES_LIST))
```

 # Create Generators

```
train_generator = VideoFrameGenerator(video_paths_train, y_train_one_hot, batch_size=4)
test_generator = VideoFrameGenerator(video_paths_test, y_test_one_hot, batch_size=4)
```

 # Define Callbacks

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model_checkpoint = ModelCheckpoint("best_model.h5", monitor='val_accuracy', save_best_only=True,
verbose=1)
```

 # Train the Model Using Efficient Generators

```
history = model.fit( train_generator,
validation_data=test_generator,
epochs=50,
callbacks=[early_stopping, model_checkpoint])
```

```
Epoch 1/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.3717 - loss: 1.4575
Epoch 1: val_accuracy improved from -inf to 0.48913, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy.
73/73 ━━━━━━━━━━━━━━━━━━━━ 956s 13s/step - accuracy: 0.3720 - loss: 1.4569 - val_accuracy: 0.4891 - val_loss: 1.3429
Epoch 2/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.3977 - loss: 1.5404
Epoch 2: val_accuracy did not improve from 0.48913
73/73 ━━━━━━━━━━━━━━━━━━━━ 964s 13s/step - accuracy: 0.3979 - loss: 1.5386 - val_accuracy: 0.4239 - val_loss: 1.3492
Epoch 3/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.3785 - loss: 1.4357
Epoch 3: val_accuracy did not improve from 0.48913
73/73 ━━━━━━━━━━━━━━━━━━━━ 965s 13s/step - accuracy: 0.3789 - loss: 1.4353 - val_accuracy: 0.3587 - val_loss: 1.2678
Epoch 4/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.3877 - loss: 1.4084
Epoch 4: val_accuracy did not improve from 0.48913
73/73 ━━━━━━━━━━━━━━━━━━━━ 956s 13s/step - accuracy: 0.3880 - loss: 1.4077 - val_accuracy: 0.4022 - val_loss: 1.4365
Epoch 5/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.3726 - loss: 1.3738
Epoch 6/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4774 - loss: 1.2208
Epoch 6: val_accuracy did not improve from 0.52174
73/73 ━━━━━━━━━━━━━━━━━━━━ 963s 13s/step - accuracy: 0.4768 - loss: 1.2215 - val_accuracy: 0.4457 - val_loss: 1.3712
Epoch 7/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.3915 - loss: 1.3446
Epoch 7: val_accuracy did not improve from 0.52174
73/73 ━━━━━━━━━━━━━━━━━━━━ 961s 13s/step - accuracy: 0.3916 - loss: 1.3447 - val_accuracy: 0.4348 - val_loss: 1.4027
Epoch 8/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4438 - loss: 1.3770
Epoch 8: val_accuracy improved from 0.52174 to 0.56522, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy.
73/73 ━━━━━━━━━━━━━━━━━━━━ 956s 13s/step - accuracy: 0.4439 - loss: 1.3761 - val_accuracy: 0.5652 - val_loss: 1.1513
Epoch 9/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4316 - loss: 1.3251
Epoch 9: val_accuracy did not improve from 0.56522
73/73 ━━━━━━━━━━━━━━━━━━━━ 957s 13s/step - accuracy: 0.4312 - loss: 1.3250 - val_accuracy: 0.5326 - val_loss: 1.1643
Epoch 10/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4306 - loss: 1.2892
Epoch 10: val_accuracy did not improve from 0.56522
73/73 ━━━━━━━━━━━━━━━━━━━━ 960s 13s/step - accuracy: 0.4309 - loss: 1.2891 - val_accuracy: 0.3261 - val_loss: 1.5988
```

```
Epoch 11: val_accuracy did not improve from 0.56522
73/73 ━━━━━━━━━━━━━━━━━━━━ 958s 13s/step - accuracy: 0.4722 - loss: 1.2805 - val_accuracy: 0.5543 - val_loss: 1.1275
Epoch 12/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4260 - loss: 1.3235
Epoch 12: val_accuracy did not improve from 0.56522
73/73 ━━━━━━━━━━━━━━━━━━━━ 947s 13s/step - accuracy: 0.4264 - loss: 1.3227 - val_accuracy: 0.4457 - val_loss: 1.1719
Epoch 13/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4972 - loss: 1.2067
Epoch 13: val_accuracy did not improve from 0.56522
73/73 ━━━━━━━━━━━━━━━━━━━━ 955s 13s/step - accuracy: 0.4971 - loss: 1.2073 - val_accuracy: 0.5000 - val_loss: 1.2128
Epoch 14/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.5064 - loss: 1.1671
Epoch 14: val_accuracy did not improve from 0.56522
73/73 ━━━━━━━━━━━━━━━━━━━━ 958s 13s/step - accuracy: 0.5064 - loss: 1.1671 - val_accuracy: 0.5435 - val_loss: 1.0706
Epoch 15/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4616 - loss: 1.1504
Epoch 15: val_accuracy improved from 0.56522 to 0.59783, saving model to best_model.h5
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy.
73/73 ━━━━━━━━━━━━━━━━━━━━ 957s 13s/step - accuracy: 0.4617 - loss: 1.1508 - val_accuracy: 0.5978 - val_loss: 1.0506
Epoch 16/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.5099 - loss: 1.1797
Epoch 17/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4982 - loss: 1.1179
Epoch 17: val_accuracy did not improve from 0.59783
73/73 ━━━━━━━━━━━━━━━━━━━━ 949s 13s/step - accuracy: 0.4985 - loss: 1.1180 - val_accuracy: 0.5978 - val_loss: 1.0122
Epoch 18/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.5222 - loss: 1.2238
Epoch 18: val_accuracy did not improve from 0.59783
73/73 ━━━━━━━━━━━━━━━━━━━━ 950s 13s/step - accuracy: 0.5222 - loss: 1.2228 - val_accuracy: 0.5000 - val_loss: 1.1930
Epoch 19/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.5044 - loss: 1.1114
Epoch 19: val_accuracy did not improve from 0.59783
73/73 ━━━━━━━━━━━━━━━━━━━━ 945s 13s/step - accuracy: 0.5044 - loss: 1.1114 - val_accuracy: 0.4239 - val_loss: 1.3009
Epoch 20/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.5513 - loss: 1.1948
Epoch 20: val_accuracy did not improve from 0.59783
73/73 ━━━━━━━━━━━━━━━━━━━━ 944s 13s/step - accuracy: 0.5514 - loss: 1.1945 - val_accuracy: 0.4783 - val_loss: 1.4732
Epoch 21/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.5190 - loss: 1.0990
Epoch 21: val_accuracy did not improve from 0.59783
73/73 ━━━━━━━━━━━━━━━━━━━━ 947s 13s/step - accuracy: 0.5192 - loss: 1.0990 - val_accuracy: 0.4674 - val_loss: 1.1668
Epoch 22/50
73/73 ━━━━━━━━━━━━━━━━━━━━ 0s 10s/step - accuracy: 0.4858 - loss: 1.1611
Epoch 22: val_accuracy did not improve from 0.59783
73/73 ━━━━━━━━━━━━━━━━━━━━ 956s 13s/step - accuracy: 0.4862 - loss: 1.1606 - val_accuracy: 0.5870 - val_loss: 1.0390
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy.
✅ Model Training Completed Successfully!
```

Backend:

App.py

```
from flask import Flask, render_template, request, jsonify import os
import cv2 import torch
import numpy as np
from werkzeug.utils import secure_filename from torchvision import transforms, models

# CNN+LSTM Model
class CNN_LSTM(torch.nn.Module):
def __init__(self, num_classes, hidden_dim=256, num_layers=1):
super(CNN_LSTM, self).__init__()
resnet = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
self.cnn = torch.nn.Sequential(*(list(resnet.children())[:-1]))
self.feature_dim=512
self.lstm = torch.nn.LSTM(input_size=self.feature_batch_first=True)
self.classifier = torch.nn.Linear(hidden_dim, num_classes)
def forward(self, x):
B, T, C, H, W = x.size()
x = x.view(B * T, C, H, W) features
= self.cnn(x)
features = features.view(B, T, self.feature_dim)
lstm_out, _ = self.lstm(features)
final_feature = lstm_out[:, -1, :] out =
self.classifier(final_feature) return out
```

 # Prediction Function

```
def predict_video(video_path, model, device,

num_frames=8):cap = cv2.VideoCapture(video_path)
if not cap.isOpened():

raise FileNotFoundError(f"Cannot open video file: {video_path}")

total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

cap.release()

indices = np.linspace(0, total_frames - 1, num_frames, dtype=int)

transform = transforms.Compose([transforms.ToPILImage(), transforms.ToTensor()])

frames = []
for idx in indices:

cap = cv2.VideoCapture(video_path)

cap.set(cv2.CAP_PROP_POS_FRAMES, idx) ret,

frame = cap.read()
cap.release()
if not ret:
continue

frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB) frame =

cv2.resize(frame, (224, 224))
frame = transform(frame)
frames.append(frame)
video_tensor = torch.stack(frames).unsqueeze(0).to(device)
model.eval()
with torch.no_grad():

outputs = model(video_tensor) _, pred = torch.max(outputs, 1)

return pred.item()
```

```
# Flask App Configuration UPLOAD_FOLDER = "uploads" ALLOWED_EXTENSIONS = {"mp4", "avi"}
app = Flask(__name__) app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER

# Load the trained model

MODEL_PATH = "cnn_lstm_action_recognition.pth" NUM_CLASSES = 7
NUM_FRAMES = 8


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model = CNN_LSTM(num_classes=NUM_CLASSES, hidden_dim=256, num_layers=1) state_dict = torch.load(MODEL_PATH, map_location=device)
new_state_dict = {k.replace("module.", ""): v for k, v in state_dict.items()}  # Fix potential multi-GPU prefix issue
model.load_state_dict(new_state_dict) model.to(device)
model.eval()



# Ensure upload folder exists

    if not os.path.exists(UPLOAD_FOLDER):

    os.makedirs(UPLOAD_FOLDER)


def allowed_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS


@app.route('/') def index():
return render_template('index.html')
```

```
@app.route('/predict', methods=['POST']) def predict():
if 'file' not in request.files:
return jsonify({"error": "No file part"})
file = request.files['file'] if
file.filename == '':
return jsonify({"error": "No selected file"})

if file and allowed_file(file.filename): filename =
secure_filename(file.filename)
filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename) file.save(filepath)
predicted_class = predict_video(filepath, model, device, num_frames=NUM_FRAMES)
class_labels = ["Sit down", "Lying Down", "Walking", "Stand up", "Standing", "Fall Down"]
result
= class_labels[predicted_class]
return jsonify({"prediction": result}) else:
return jsonify({"error": "Invalid file
format. Only MP4 and AVI allowed."})
if __name__== '__main__':
app.run(debug=True)
```

livepredict.py

```
import cv2
import numpy as np import tensorflow as tf import threading
from tensorflow.keras.models import load_mode


# Load the trained model


model = load_model("final_hybrid_model.keras")
```

 # Constants

IMAGE_HEIGHT, IMAGE_WIDTH = 256, 256

SEQUENCE_LENGTH = 10

CLASSES_LIST = ['Clapping', 'Meet and Split', 'Sitting', 'Standing Still', 'Walking', 'Walking While Reading Book', 'Walking While Using Phone']

 # Frame buffer

frame_buffer = [] predicted_action = "Detecting..." confidence_score = 0.0

 # Function to preprocess frames

def preprocess_frame(frame):

frame = cv2.resize(frame, (IMAGE_WIDTH, IMAGE_HEIGHT))
frame = frame / 255.0 # Normalize
return frame

 # Function to predict action in a separate thread

def predict_action():

global predicted_action, confidence_score
while True:
if len(frame_buffer) == SEQUENCE_LENGTH:
input_data = np.expand_dims(np.array(frame_buffer), axis=0)
predictions = model.predict(input_data)
predicted_action = CLASSES_LIST[np.argmax(predictions)]
confidence_score = np.max(predictions)
 # Remove oldest frame for smooth prediction
frame_buffer.pop(0)

```
# Start prediction thread

prediction_thread = threading.Thread(target=predict_action, daemon=True)
prediction_thread.start()
# Initialize webcam

cap = cv2.VideoCapture(0) if not cap.isOpened():
print("Error: Could not open webcam.")
exit()
while True:
ret, frame = cap.read() if not ret:
break

# Preprocess and store frame
processed_frame = preprocess_frame(frame)
frame_buffer.append(processed_frame)

# Keep only last SEQUENCE_LENGTH frames

if len(frame_buffer) > SEQUENCE_LENGTH:
frame_buffer.pop(0)
# Overlay detected action

cv2.putText(frame, f"Action: {predicted_action} ({confidence_score:.2f})", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)
cv2.imshow('Live Action Recognition', frame)

 # Press 'q' to exit

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

 # Release resources cap.release() cv2.destroyAllWindows()

Predict.py import os import cv2 import torch

import torch.nn as nn import numpy as np

from torchvision import transforms, models

IMG_SIZE = 224 # Must match training (ResNet18 input size) NUM_FRAMES = 8

Same as used during training NUM_CLASSES = 7

Number of action classes (adjust if needed)

MODEL_PATH = "cnn_lstm_action_recognition.pth" # Path to your saved model file VIDEO_PATH = r"dataset\train\Walking\video_20_flip.avi" # Update with your video file path

"""

Sit down: 0

Lying Down: 1

Walking: 2

Stand up: 3

Standing: 4

Fall Down: 5

Sitting: 6 """

transform = transforms.Compose([

transforms.ToPILImage(),

transforms.ToTensor(),])

def load_frame(path, frame_index=None):

if path.lower().endswith('.avi'):

cap = cv2.VideoCapture(path) if not

cap.isOpened():

raise FileNotFoundError(f"Cannot open video file: {path}")

```
total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) if
frame_index is None:
frame_index = total_frames // 2  if  frame_index >= total_frames:
frame_index = total_frames – 1
cap.set(cv2.CAP_PROP_POS_FRAMES, frame_index) ret,
frame = cap.read() cap.release() if not
ret: raise  ValueError(f"Could     not read frame {frame_index} from video: {path}") return
frame
else:
frame = cv2.imread(path) if frame is None:
raise FileNotFoundError(f"Image not found: {path}") return frame
class CNN_LSTM(nn.Module):

def __init__(self, num_classes, hidden_dim=256, num_layers=1): super(CNN_LSTM, self).__init
()
  resnet = models.resnet18(pretrained=True)

  self.cnn = nn.Sequential(*(list(resnet.children())[:-1])) self.feature_dim = 512
self.lstm = nn.LSTM(input_size=self.feature_dim, hidden_size=hidden_dim,
num_layers=num_layers, batch_first=True)
self.classifier = nn.Linear(hidden_dim, num_classes)

  def forward(self, x):

    B, T, C, H, W = x.size()
    x = x.view(B * T, C, H, W) features = self.cnn(x)
    features = features.view(B, T, self.feature_dim) lstm_out, _ = self.lstm(features)

    final_feature = lstm_out[:, -1, :] out = self.classifier(final_feature) return out
       def  predict_video(video_path,  model,  device,  num_frames=NUM_FRAMES):  cap  =
     cv2.VideoCapture(video_path)
    if not cap.isOpened():
```

```
raise FileNotFoundError(f"Cannot open video file: {video_path}") total_frames =
 int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) cap.release()
if total_frames < num_frames:

indices = list(range(total_frames)) + [total_frames - 1] * (num_frames - total_frames) else:
indices = np.linspace(0, total_frames - 1, num_frames, dtype=int) frames = []
for idx in indices:

frame = load_frame(video_path, frame_index=idx) frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB)
frame = cv2.resize(frame, (IMG_SIZE, IMG_SIZE))
frame  = transform(frame)
 frames.append(frame) video_tensor = torch.stack(frames)
video_tensor = video_tensor.unsqueeze(0) video_tensor = video_tensor.to(device)
 model.eval()
with torch.no_grad():
outputs = model(video_tensor)_, pred = torch.max(outputs, 1) return pred.item()
def main():

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = CNN_LSTM(num_classes=NUM_CLASSES, hidden_dim=256, num_layers=1)
model.to(device)

state_dict = torch.load(MODEL_PATH, map_location=device)

new_state_dict = {}

for k, v in state_dict.items():

new_key = k[7:] if k.startswith("module.")
else k new_state_dict[new_key] = v
model.load_state_dict(new_state_dict)
print(f"Loaded model from {MODEL_PATH}")
 predicted_class=predict_video(VIDEO_PATH,model,device,
 num_frames=NUM_FRAMES) print(f"Predicted class label: {predicted_class}")
 if ___name____ == "_main_": main()
```

Frontend:

Index.html:

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=          1.0">

<title>Video Action Recognition</title>
<script src="https://cdn.tailwindcss.com"></script>

</head>

<body class="bg-gray-100 flex flex-col items-center justify-center min-h-screen">

  <nav class="w-full bg-blue-600 p-4 text-white text-center text-xl font-bold">Action
  Recognition</nav>

<div class="bg-white shadow-1g rounded-1g p-6 mt-10 w-3/4 flex flex-col items-center">
<label class="block text-gray-700 text-sm font-bold mb-2">Upload a Video</label>
 <form action="/predict" method="POST" enctype="multipart/form-data" class="w-full flex
 flex-col
 items-center">
 <input id="videoInput" name="file" type="file" accept="video/*" class="mb-4 border p-2 w-
 full"
  required />
 <button type="submit" class="bg-blue-500 text-white px-4 py-2 rounded-1g hover:bg-blue-
 700">Predict</button></form>
<</form>
```

```
<div id="loading" class="hidden mt-4 text-blue-500">Processing...</div>

<div id="result" class="mt-4 text-lg font-semibold">Detected Action: {{ action }}  (: {{ confidence }})</div>
<video id="videoPlayer" class="mt-4" controls loop>
  <source src="{{ video_url }}" type="video/mp4"> Your browser does not support the video tag.
</video>
{% endif %}
</div>
</body>
</html>
```

6.2 Implementation

6.2.1 Front-End Implementation:

The front-end of the Human Behaviour Recognition and Fall Detection System is designed to provide a user-friendly, responsive, and monitoring-oriented interface. The main modules include Live Monitoring Dashboard, Activity Visualization, and Alert Notification Interface.

The Registration and Login modules enable secure authentication and controlled access to the system. Input validation mechanisms are implemented to ensure correct data entry and prevent unauthorized system access. These modules allow caregivers, administrators, or medical staff to securely log into the monitoring platform.

The Activity Visualization Module presents historical activity logs, behavioral statistics, and movement patterns of monitored individuals. This helps caregivers and healthcare professionals track long-term activity trends and identify possible health deterioration or abnormal movement patterns.

The front-end follows consistent UI design principles including clear navigation menus, structured layout, and responsive design to support accessibility across desktops, tablets, and mobile devices.

6.2.2 Backend Implementation:

The backend of the system is implemented using Python, which manages video processing, model execution, and activity classification operations. The backend integrates OpenCV for video stream handling and deep learning frameworks such as TensorFlow/Keras for behaviour recognition.

The backend workflow includes:

- Capturing video frames from webcam, CCTV camera, or stored video input
- Preprocessing captured frames for model compatibility
- Running frames through the trained Multi-Scale CNN model
- Performing activity classification and fall detection
- Generating alerts and updating monitoring interface

The backend system ensures continuous processing of video frames while maintaining computational efficiency. The architecture is modular, allowing separate handling of preprocessing, feature extraction.

6.2.3 Deployment and Reliability:

The system is deployed using a standard server configuration with environment-based security settings. The architecture supports deployment on both cloud servers and edge computing platforms such as and enabling flexible usage across smart homes, hospitals, and elderly care centers. Optimization techniques such as model pruning, quantization, and lightweight inference pipelines are implemented to improve processing efficiency on resource-constrained devices

6.2.4 Conclusion:

The implementation successfully integrates computer vision techniques, deep learning-based activity recognition, and real-time monitoring into a unified Human Behaviour Recognition system. The Multi-Scale CNN model enables accurate classification of daily human activities and reliable fall detection. The system supports continuous monitoring and instant alert generation, improving elderly safety and reducing response time during emergencies. The modular architecture allows future enhancements such as multi-person detection, IoT integration, and predictive health monitoring.

7. SYSTEM TESTING

System testing is a crucial phase that ensures the developed Human Activity Recognition (HAR) system using CNN and LSTM models performs accurately, consistently, and reliably under real-world operating conditions. The primary goal of system testing is to detect potential defects, validate system behavior, and confirm that all functional and non-functional requirements are satisfied.

In this project, system testing focuses on validating all major components of the HAR system, including video input handling, preprocessing pipelines, CNN-based spatial feature extraction, LSTM-based temporal sequence modeling, activity classification logic, and alert generation mechanisms. Testing ensures that user interactions, frame processing, model inference, and output visualization operate correctly without failures or inconsistencies.

System testing was conducted across multiple video sequences, activity types, and environmental conditions to ensure correctness, robustness, and usability. Special emphasis was placed on accurate activity classification, fall detection reliability, handling of edge-case movements, and system stability during continuous real-time monitoring.

7.1 Types of System Testing

7.1.1 Unit Testing:

Unit testing was performed to validate individual system modules independently before integrating them into the complete pipeline. Each functional component was tested using controlled inputs to verify whether the expected outputs were generated correctly. The key components tested during this phase included the video frame capture and extraction logic, image preprocessing operations such as resizing, normalization, and enhancement, CNN-based feature extraction layers, LSTM sequence processing and memory handling mechanisms, activity classification output layer, and the alert triggering module designed for fall detection. This testing ensured that each internal module functioned accurately in isolation, which made it easier to identify and resolve errors at an early development stage. As a result, unit testing contributed significantly to improving the reliability, stability, and overall performance of the proposed Human Behaviour Recognition system before proceeding to full system integration.

7.1.2 Integration Testing:

Integration testing verified the interaction between combined system modules after they were linked into a complete processing pipeline. During this phase, several important integrations were tested to ensure proper coordination between different components of the proposed system. The video input stream was successfully connected with the preprocessing pipeline to enable continuous frame extraction and normalization. The CNN-based spatial feature extraction module was effectively integrated with the LSTM temporal modeling component to ensure accurate learning of motion patterns across frame sequences. Furthermore, the activity classification output was properly linked with the alert generation and logging mechanisms so that fall events could be detected and recorded in real time. Additionally, the backend processing modules were integrated with the real-time monitoring dashboard to display activity status and detection results clearly. This phase ensured that data flowed correctly between modules and confirmed that the feature representations extracted by the CNN layers were accurately interpreted by the LSTM network without any mismatch or loss of temporal information, thereby improving the reliability and overall performance of the system.

7.1.3 Functional Testing:

Functional testing was conducted to validate that all system features performed according to the defined specifications and user expectations. During this phase, several test cases were designed to simulate real-world scenarios involving different human activities to ensure accurate system performance under practical conditions. The testing verified the correct classification of activities such as walking, sitting, standing, bending, lying, and falling, along with accurate detection of fall events accompanied by immediate alert generation. It also ensured proper handling of continuous video streams for uninterrupted monitoring and confirmed that the system could gracefully manage invalid or corrupted video inputs without failure. Additionally, the real-time display of detected activity labels on the monitoring interface was tested to ensure clarity and responsiveness. Overall, functional testing confirmed that the system operated efficiently, accurately, and in a user-friendly manner, making it suitable for real-time human behaviour monitoring applications.

7.1.4 System Testing :

System testing evaluated the Human Activity Recognition system as a fully integrated application with emphasis on overall reliability, consistency, and accuracy during operation under real-world conditions. This phase verified the coordinated execution of preprocessing, CNN-based feature extraction, and LSTM temporal modeling modules to ensure smooth interaction between all components of the system pipeline. The testing also confirmed stable performance during continuous real-time monitoring without interruption, along with accurate activity recognition under varied lighting conditions and different camera angles. In addition, the system demonstrated reliable fall detection capability without generating frequent false alarms, which is critical for practical monitoring environments. The testing results showed that the system consistently produced predictable and accurate activity recognition outcomes aligned with the defined project requirements, thereby confirming its effectiveness and robustness for real-time human behaviour monitoring applications.

7.1.5 White Box Testing:

White-box testing was applied to evaluate the internal processing components of the system, particularly the CNN–LSTM pipeline, by examining the internal operations and data flow within the model architecture. During this phase, internal feature maps, temporal sequences, and classification outputs were carefully observed to ensure correct execution at each stage of processing. The testing validated proper convolutional feature extraction by the CNN layers, accurate temporal dependency modeling by the LSTM layers, and correct propagation of feature vectors across different network layers without loss of information. Additionally, the threshold-based fall detection logic was verified to ensure reliable identification of fall events. Overall, white-box testing ensured transparency, correctness, and consistency of internal system operations, thereby improving the reliability and performance of the proposed Human Activity Recognition system.

7.1.6 Black Box testing:

Black-box testing evaluated the Human Activity Recognition system from an end-user perspective without examining the internal code structure or model architecture. During this phase, inputs were provided through video streams, and the outputs were observed directly through the system's user interface to verify overall functionality and usability. The testing focused on evaluating the accuracy of displayed activity labels, the responsiveness of the system to changes in human movements, and the proper generation of alerts during fall detection events. Additionally, the clarity and usability of the user interface were examined to ensure that users could easily interpret the monitoring results without confusion. The results of the black-box testing confirmed that the system behavior aligned well with user expectations and demonstrated reliable performance in real-time monitoring scenarios.

7.1.7 Acceptance Testing:

Acceptance testing was conducted to ensure that the proposed Human Activity Recognition system satisfied all documented requirements and was suitable for deployment in real-world monitoring environments. During this phase, stakeholders evaluated the system based on important factors such as accuracy, usability, responsiveness, and overall reliability during operation. The acceptance criteria included reliable real-time activity recognition across different human movements, accurate and timely fall detection with immediate alert support, and stable performance during long-duration continuous usage without interruptions.

7.2 Testing Strategies

A structured and systematic testing strategy was followed throughout the development of the project. Testing progressed from individual module validation to complete system verification.

7.2.1 Test Strategy and Approach

Testing was performed using both manual observation and dataset-driven evaluation methods to ensure the consistency and accuracy of the Human Activity Recognition system under different operating conditions. Multiple video samples containing varied human activities were used to validate the system's performance across different movement scenarios. During this phase, testing objectives included verifying the correctness of frame preprocessing and normalization processes to ensure proper input preparation for model execution. It also ensured that the CNN–LSTM integration effectively captured both spatial and temporal activity patterns for accurate classification results. Additionally, fall detection performance was validated under both rapid and slow movement scenarios to confirm reliable emergency identification. The testing further ensured uninterrupted real-time processing capability of the system during continuous monitoring operations, thereby confirming its stability and suitability for practical deployment in real-world environments.

7.2.2 Test Objectives:

The primary objectives of system testing were to ensure the correct classification of all supported human activities with high accuracy while maintaining real-time responsiveness without noticeable delay during continuous monitoring operations. The testing process also focused on verifying the system's ability to handle ambiguous or overlapping movements safely without misclassification. In addition, efforts were made to minimize false positives and false negatives, particularly in fall detection scenarios where reliability is critical for emergency response. Another important objective was to ensure smooth and coordinated operation across all system components, including preprocessing, feature extraction, temporal modeling, classification, and alert generation modules. Overall, these objectives helped confirm that the system performed efficiently and reliably under real-world operating conditions.

7.2.3 Features Tested:

The following features of the proposed Human Activity Recognition system were thoroughly tested to ensure reliable and accurate performance under real-time operating conditions. The testing process verified the proper functioning of real-time video capture and frame processing to support continuous monitoring without interruption. It also evaluated the activity recognition accuracy across multiple activity classes such as walking, sitting, standing, bending, lying, and falling to confirm consistent classification performance. In addition, the fall detection and alert triggering mechanism was carefully tested to ensure timely identification of emergency situations and immediate notification support. These validations confirmed that the system performed efficiently and met the required performance standards for real-world monitoring applications.

7.2.4 Integration Testing Strategy:

Integration testing emphasized the early detection of data flow and dependency-related issues between interconnected modules of the Human Activity Recognition system. During this phase, testing ensured the correct alignment of CNN feature extraction outputs with the corresponding LSTM inputs so that spatial features were accurately transferred for temporal sequence modeling. It also verified stable handling of temporal frame sequences to maintain consistency in activity recognition across continuous video streams. In addition, seamless communication between the processing pipeline and the user interface dashboard was validated to ensure real-time display of classification results and alerts without interruption.

7.2.5 Acceptance Criteria:

The system was considered acceptable when it successfully satisfied several important performance and functional conditions defined during the project development phase. These conditions included accurate recognition of human activities along with reliable fall detection capability to ensure effective monitoring performance. The system also demonstrated stable real-time operation during continuous usage without noticeable processing delays. In addition, error-free handling of video inputs was verified to ensure smooth execution of preprocessing and classification tasks. The activity recognition outputs displayed on the monitoring interface were clear, interpretable, and easy for users to understand in real-time situations. Overall, compliance with all defined project objectives confirmed that the system met the required standards for practical deployment in real-world human activity monitoring environments.

7.2.6 Overall Test Results:

All planned test cases were executed successfully. The system demonstrated stable operation, logical correctness, and high activity recognition accuracy. The CNN-LSTM model effectively captured both spatial and temporal features, resulting in reliable classification and fall detection performance.

7.2.7 Conclusion:

System testing confirmed that the Human Activity Recognition system using CNN and LSTM models meets functional expectations, operates reliably under diverse real-world conditions, and integrates all components effectively. Through rigorous testing, the project achieved robustness, usability, and dependable real-time activity recognition, making it suitable for deployment in healthcare, surveillance, and smart environment applications.

7.3 Sample Test Cases:

Table 7.3: Test Cases

S.No.	Test Case	Expected Result	Result	Remarks (if any)
01.	Landing Page Display	Human Activity Recognition landing page loads correctly	Pass	UI elements displayed properly
02.	Video Selection Interface	File selection option is visible on the screen	Pass	Accepts video input
03.	Prediction Button Functionality	Predict button is enabled and responsive	Pass	Triggers recognition process
04.	Activity Recognition Process	System processes input and performs activity recognition	Pass	CNN–LSTM model executed
05.	Prediction Result Display	Detected human activity is Displayed on screen	Pass	Output shown clearly
06.	Output Screen Update	Output screen updates after prediction	Pass	No page reload issues

Test Case 1:

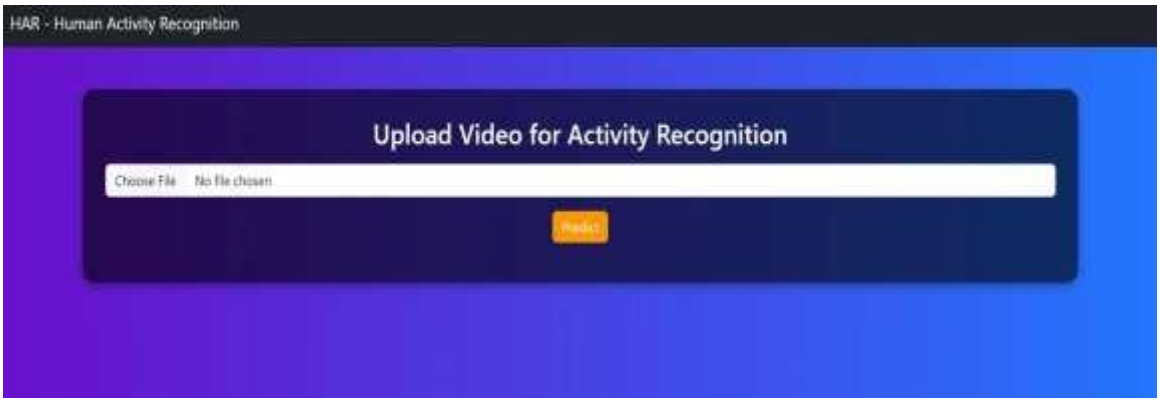

Fig 7.3.1 Landing Page Display

Description: This screen represents the initial landing page of the Human Activity Recognition system. It confirms that the application loads successfully and displays all essential user interface elements, including the system title and input area. The landing page serves as the primary access point for initiating the activity recognition process.

Test Case 2:

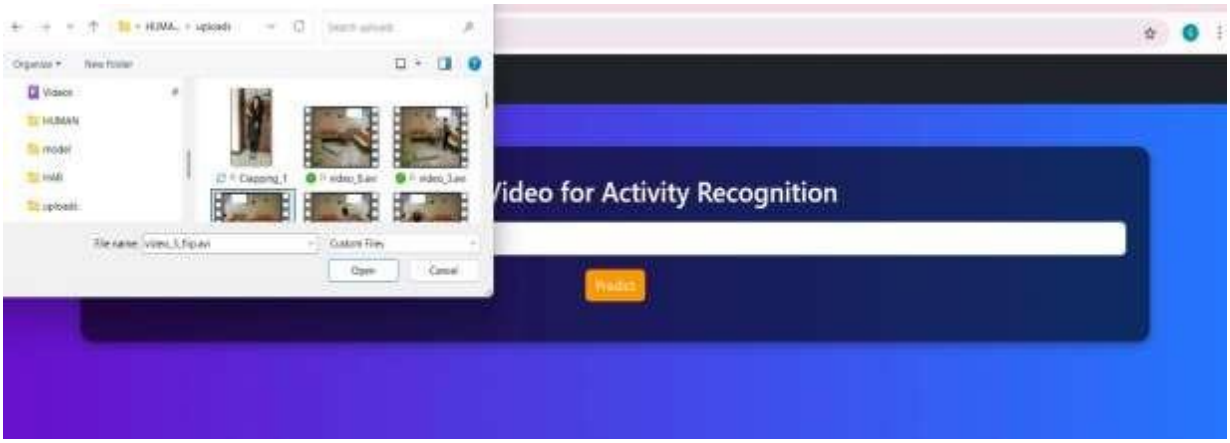

Fig 7.3.2 Video Selection Interface

Description: This output screen shows the file selection interface where users can choose a video file for activity recognition. The presence of the file input control verifies that the system is ready to accept video input for further processing and analysis.

Test Case 3:

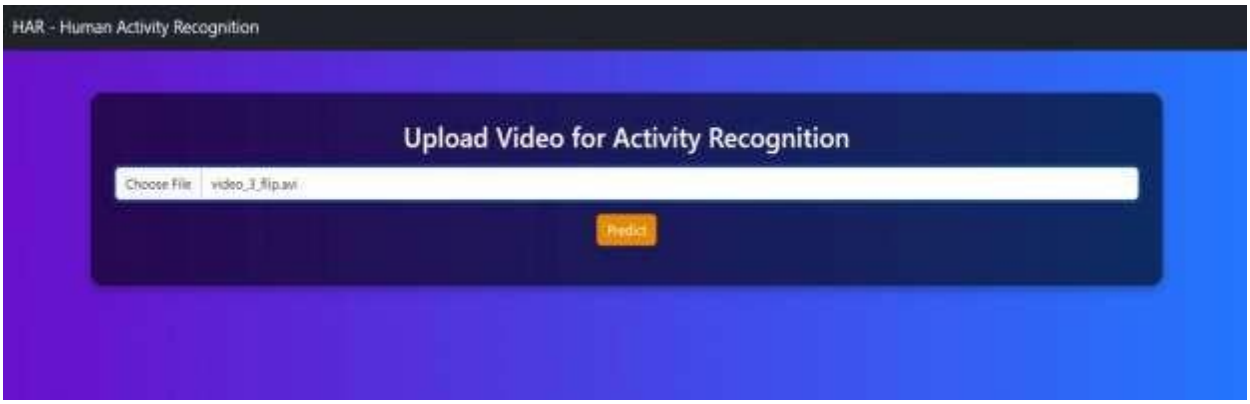

Fig. 7.3.3 Prediction Button Functionality

Description: This screen confirms that the "Predict" button is enabled and responsive. When activated, it initiates the activity recognition pipeline by sending the selected input to the backend for preprocessing and model inference.

Test Case 4:

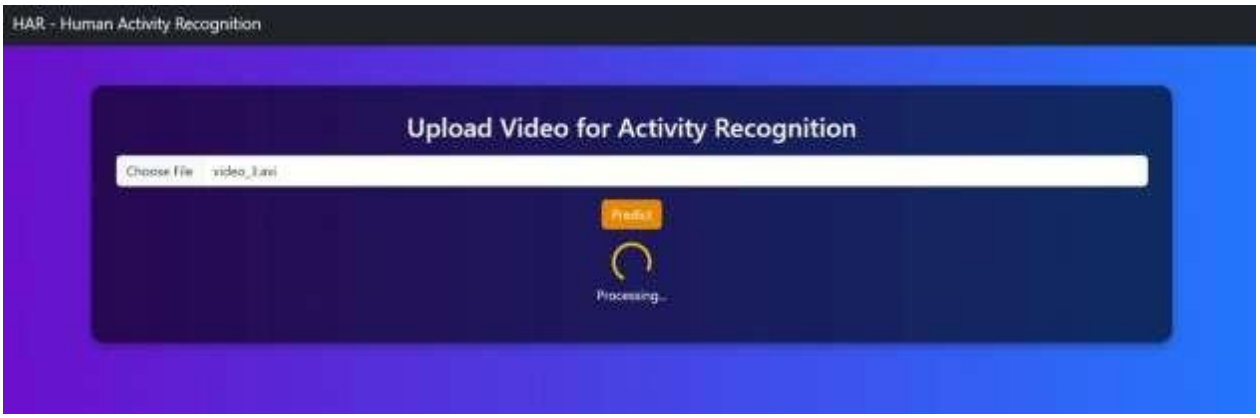

Fig. 7.3.4 Activity Recognition Process

Description: This screen demonstrates the execution of the Human Activity Recognition process. The input data is processed through the Multi-Scale Convolutional Neural Network (CNN) for spatial feature extraction, followed by a Long Short-Term Memory (LSTM) network for temporal pattern analysis.

Test Case 5:

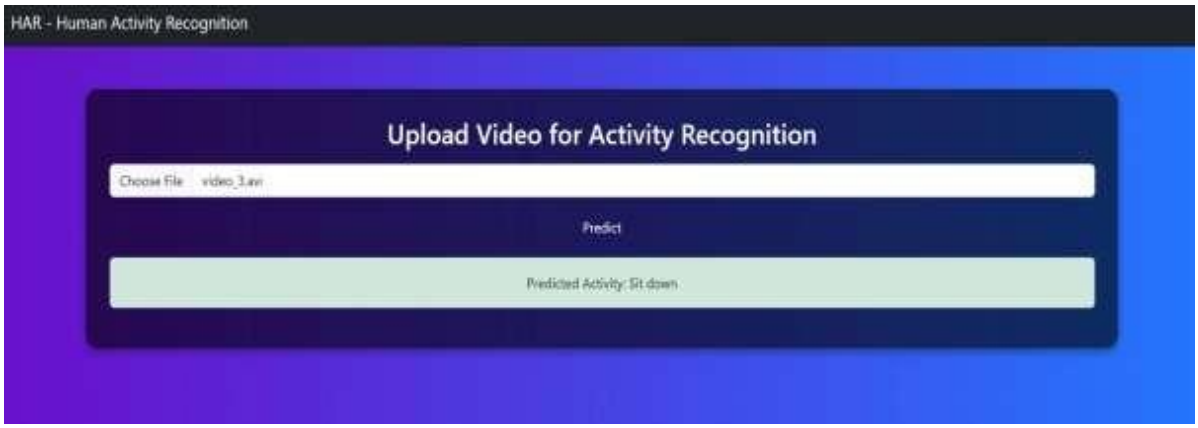

Fig. 7.3.5 Prediction Result Display

Description: This output screen displays the detected human activity after successful model inference. The predicted action is clearly shown on the interface, confirming that the system accurately recognizes and classifies human behaviour.

Test Case 6:

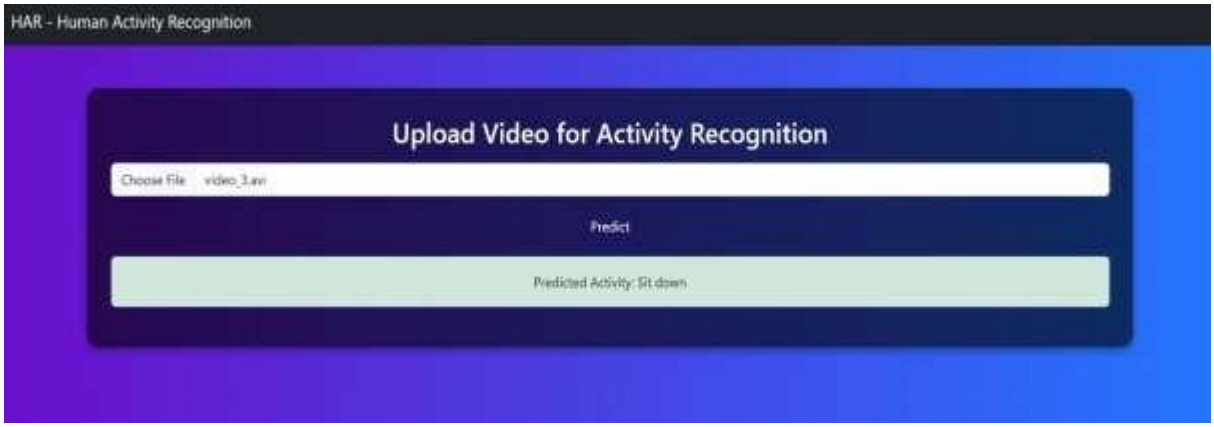

Fig. 7.3.6 Output Screen Update

Description: This screen verifies that the output section updates dynamically after the prediction process is completed. The system refreshes the displayed results without requiring a full page reload, ensuring a smooth and user-friendly experience.

8. RESULTS

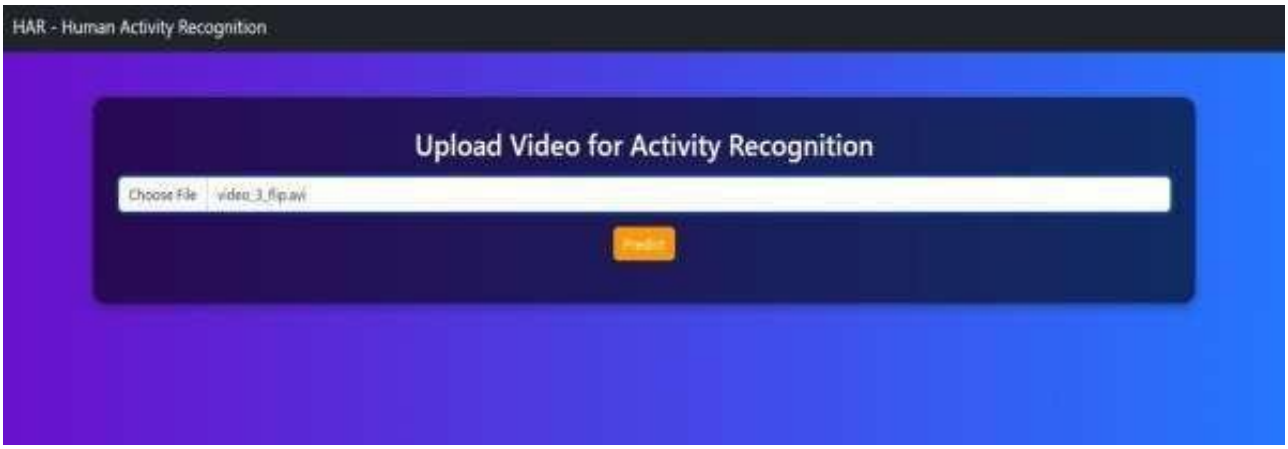

Fig 8.1 Human activity recognition landing page

Description: The primary interface of the application, titled "Human Behaviour Recognition using Multi-Scale CNN", serves as the main interaction point for monitoring and analyzing human activities in real time. The interface introduces the system as an intelligent activity recognition solution powered by a Multi-Scale CNN integrated with LSTM networks to capture both spatial and temporal features from video streams.

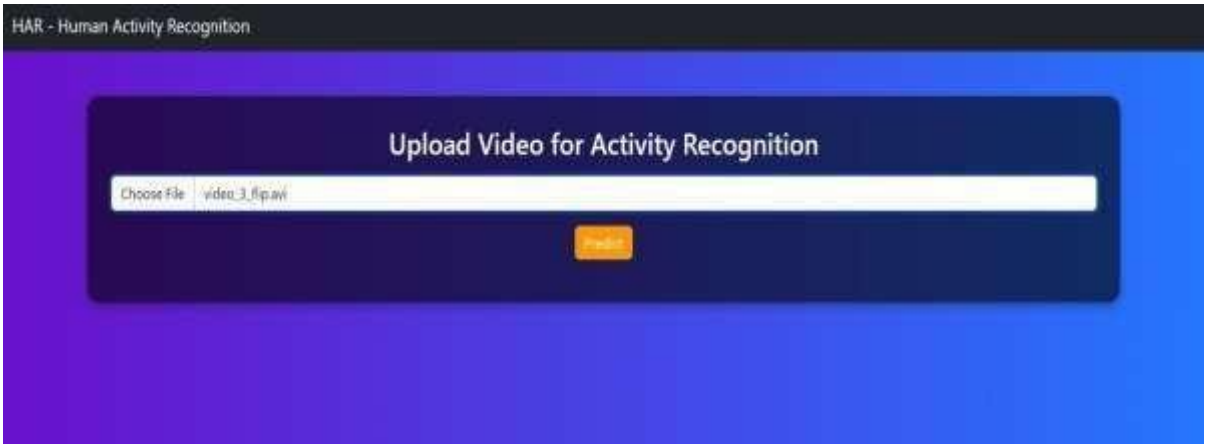

Fig 8.2 Uploading video

Description: The entry point of the application for "Human Behaviour Recognition using Multi-Scale CNN" provides secure access through Registration and Login modules. This interface ensures authorized usage of the system before granting access to real-time human activity recognition and monitoring features.

Fig 8.3 Human Behaviour Recognition

Description: This screen demonstrates the Human Activity Recognition module in operation. The input video stream is processed through a Multi-Scale Convolutional Neural Network (CNN) that extracts spatial features at different resolutions, followed by a Long Short-Term Memory (LSTM) network that models temporal motion patterns across consecutive frames. The combined spatiotemporal features enable the system to accurately recognize human activities such as walking, sitting, standing, bending, and falling.

9. CONCLUSION AND FUTURE SCOPE

9.1. Conclusion

The proposed Multiscale CNN–LSTM based human behavior recognition system effectively detects falls from continuous video streams by combining spatial and temporal learning. The integration of multiscale CNN enables detailed feature extraction, while LSTM captures motion patterns over time, allowing accurate differentiation between fall events and daily activities. Experimental results on benchmark datasets such as URFD and UP-Fall demonstrate improved performance in terms of accuracy, precision, recall, and F1-score when compared to traditional machine learning and single-stage CNN models **[3], [5], [12].** The hybrid architecture proves efficient in handling real-world challenges such as fast motion transitions, occlusions, and posture variations.

Furthermore, the system provides a non-intrusive and reliable solution for elderly care by eliminating the need for wearable devices, addressing issues like discomfort and inconsistent usage **[1], [7].** The integration of a real-time monitoring dashboard and automated alert system enhances its practical applicability in healthcare environments by ensuring timely caregiver response. The multiscale feature extraction and temporal modeling also help reduce false alarms caused by normal movements, making the system suitable for continuous surveillance applications **[6], [14].**

9.2. Future Scope

Future work can focus on integrating advanced deep learning models such as transformer-based architectures along with lightweight CNN variants to improve spatial-temporal representation while reducing computational complexity for edge deployment **[8], [16].** Enhancing system performance under challenging conditions such as low lighting, partial occlusions, and multi-camera environments is another important direction.

Additionally, incorporating pose estimation techniques with skeleton tracking and depth information can further improve motion understanding and detection accuracy. The system can also be extended into an IoT-enabled smart home framework with predictive analytics to enable proactive elderly care and continuous health monitoring **[9], [13].** These improvements will make the system more robust, scalable, and suitable for real-world healthcare applications

Finally, future improvements can focus on making the system more intelligent and efficient through predictive analytics and edge optimization**.** By analyzing long-term activity patterns, the system can predict potential fall risks and alert users before an incident occurs, enabling preventive care. Techniques such as model pruning and quantization can be applied to deploy the system on low-power edge devices like Raspberry Pi with faster performance. Additionally, incorporating **privacy-preserving methods** like pose estimation instead of raw video and expanding activity recognition capabilities will further enhance the system's usability and acceptance.

REFERENCES

1. Alymani, Mofadal, et al. "Adaptive motion assisted human activity recognition for people with disabilities via osprey optimisation-based dimensionality reduction with recurrent neural network." Signal, Image and Video Processing, 2026

2. Wang, Chuanchuan, et al. "Deep learning for 3D skeleton-based action recognition: a comprehensive review of methods, datasets, and future directions."International Journal of Machine Learning and Cybernetics, 2026.

3. Talari Swapna Integrated Dual Deep Convolutional Neural Network for Effective Content-Based Picture Retrieval. Conference, 2026.

4. Parale, Mandar, et al."A systematic review on human action detection and classification architectures using deep learning methodology."Multimedia Tools and Applications, 2026.

5. Ramu, Moola, et al. "Human activity recognition using a bagging-based deep learning framework with convolutional capsule networks and spiking neural networks."International Journal of Information Technology, 2025.

6. Alzahrani, Abdulrahman, et al."Artificial Intelligence-based fine-tuning model for fall activity recognition in disabled persons within an IoT environment."Scientific Reports, 2025.

7. Han, H., et al. "Multilevel features cascade fusion network for infrared video human behavior recognition."Displays, 2025.

8. Wei, Y., et al. "MCNN-CMCA: A multiscale convolutional neural networks with cross-modal channel attention for physiological signal-based mental state recognition." Digital Signal Processing, 2025.

9. Dianakamal, G., et al. "Recognition of human behaviour utilising multiscale convolutional neural networks." Taylor & Francis, 2025.

10. A. Srinivasula Reddy, Mandapati Raja An Implementation of Convolutional Neural Network-Based Architecture to Address Facial Sentiment Analysis. Conference, 2025.

11. B. Mamatha Accurate and improved MRI image quality assessment methods by using a CNN-based image quality approach. Conference, 2025.

12. Cristina, Stefania, et al. "Audio-and Video-Based Human Activity Recognition Systems in Healthcare." IEEE Access (2024).

13. Pereira, Gracile Astlin. "Fall detection for industrial setups using yolov8 variants." arXiv preprint arXiv:2408.04605 (2024).

14. Khekan, Ahlam R., Hadi S. Aghdasi, and Pedram Salehpoor. "Fast and High-Precision Human Fall Detection Using Improved YOLOv8 Model." IEEE Access (2024).

15. Papan, Vishal, and S. Maheswari. "Intelligent Fall Detection and Alert System for the Elderly Using Yolov8 and Cloud-Based Analytics." 2024 5th International Conference on Electronics and Sustainable Communication Systems (ICESC). IEEE, 2024.

16. Verma, Aanchal, Harsh Verma, and R. K. Saket. "Elderly Fall Detection for Smart Home Caring Using YOLOv8."

17. Gaud, Neha, Maya Rathore, and Ugrasen Suman. "MHCNLS-HAR: Multi-Headed CNN-LSTM Based Human Activity Recognition Leveraging a Novel Wearable Edge Device for Elderly Health Care." IEEE Sensors Journal (2024).

18. Avala Raji Reddy Mangalampalli Sesha Sai Lakshmi Lavanya, Botcha Kishore Kumar Development of image recognition teaching model by the AR technology. Journal, 2023.

19. Suman Mishra, Ramakishore Somala Deep Convolutional Neural Networks Based Image Fusion Method Using Pyramid Decomposition for Medical Applications. Conference, 2023.

20. Amarajyothi Aramanda, M. Kumaraswamy Improve the serendipity in recommender systems. Conference, 2023.

21. Mangalampalli Sesha Sai Lakshmi Lavanya, Botcha Kishore Kumar Development of image recognition teaching model by the AR technology. Journal, 2023.

22. Suman Mishra, Ramakishore Somala Deep Convolutional Neural Networks Based Image Fusion Method Using Pyramid Decomposition for Medical Applications. Conference, 2023.

23. Kumari Gubbala, M. Naveen Kumar, A. M. Sowjanya, et al. AdaBoost based Random Forest model for Emotion classification of Facial images. Journal, 2023.

24. Mrutyunjaya S. Yalawar An Approach for Detecting Drowsy Drivers in Vehicle using CNN Techniques. Journal, 2023.

25. Yhdego, Haben Girmay. Wearable Sensor Gait Analysis for Fall Detection Using Deep Learning Methods. Diss. Old Dominion University, 2023.